

De nos jours, les données sont devenues l'un des atouts majeurs qui constituent une richesse pour la recherche. Les informations présentes sont devenues aussi bien pour les entreprises que pour les instituts de recherche un facteur de compétitivité et d'innovation. En général, ces données permettent de découvrir et d'expliquer certains phénomènes existants ou bien d'extrapoler des nouvelles connaissances à partir des informations présentes. Cependant, dans le monde réel, et dans le contexte d'apprentissage supervisé, ces données brutes ne sont pas toutes faciles à traiter par les modèles d'apprentissage. On peut avoir :

- i. des données homogènes temporelles difficiles à traiter qui présentent un défi de par leur grande masse, leur grande dimensionnalité, et l'aspect de changement continu.
- ii. des données peu abondantes présentant un manque aigu de données étiquetées, ce qui entraîne généralement des difficultés de traitement ;
- iii. des données déséquilibrées contenant un dosage déséquilibré entre les observations fréquentes et les observations rares ou très peu représentées ;
- iv. des données contenant des observations aberrantes qui faussent généralement les méthodes traditionnelles d'analyse;

Cette thèse essaie de répondre à cette problématique en proposant des algorithmes qui s'appuient sur les réseaux de neurones. Dans cette thèse, on introduit de nouvelles approches, basées sur les réseaux de neurones artificiels ou sur l'apprentissage profond, qui produisent des résultats comparables ou meilleurs que ceux de l'état de l'art.

 Faculté des Sciences et Techniques
Settat
THÈSE DE DOCTORAT
Pour l'obtention de grade de Docteur en Informatique

Formation Doctorale: Mathématiques Appliquées et Informatique

Spécialité: Informatique

Sous le thème
**Deep Learning Applied to
Different Types of Data**
Présentée par :
Lamyaa SADOUK
Soutenu le: 16 février 2021

A la Faculté des Sciences et Techniques de Settat devant le jury composé de :

Pr. Mohamed ABOUFATAH	PES	FST Settat	Président
Pr. Othmane EL MESLOUHI	PH	ENSA Safi	Rapporteur
Pr. Mustapha HANOUNE	PES	FS Ben M'sik	Rapporteur
Pr. Youssef BALOUK	PH	FST Settat	Rapporteur
Pr. Hicham REDWANE	PES	FEG Settat	Examineur
Pr. El Hassan ESSOUFI	PES	FST Settat	Co-Directeur de thèse
Pr. Taoufiq GADI	PES	FST Settat	Directeur de thèse

Année Universitaire: 2020/2021

Acknowledgments

I would like to acknowledge a number of people that supported, in various ways, the creation of this work.

First and foremost I'd like to express gratitude towards my main academic advisor Prof. Taoufiq Gadi for his help and all the resources he made available for me to work on my Ph.D.. My deep gratitude goes to my second advisor, Prof. El Hassan Essoufi, from whom I learned a lot.

Furthermore, I would like to express my sincere appreciation to Prof. Hakim Al-lali, Prof. Taoufiq Gadi, Prof. Mohamed Aboulfatah and Prof. Abdellah Lakhouili who taught me a lot in the professional level and gave me the opportunity to gain a great working experience. I really enjoyed working with them.

I am grateful to my friends who influenced and supported me on my journey leading towards and through my doctoral studies.

I am also grateful to my uncle and aunt Badissy who supported me and helped me pursue my bachelor's degree. Without their support, I could not have completed my Ph.D. degree.

I would like to thank my husband for his great help, encouragements and precious moral support. Finally, I would like to send my warmest acknowledgments to my parents and my sisters for their constant care and encouragement.

The thesis is dedicated to all of you. Thank you.

Abstract

The research goal of this work is to develop learning methods and applications to interpret and learn features from various sources of information, such as images, time-series, as well as data from different fields. In this dissertation we introduce a series of artificial neural network and deep learning based approaches to make classification and regression tasks towards real-world data and applications. Throughout this dissertation, we show how these approaches are able to learn features from different types of data including homogeneous data and heterogeneous data such as imbalanced data, data with outliers and small datasets.

Faced with challenges that pose such types of data, we propose the following solutions:

- Homogeneous data. This part focuses on approaches for the classification of 1 dimensional input data (time-series) as well as 2 dimensional input data (images). Regarding time-series, in the data-level, we propose a method to convert time-series to frequencies based on the Stockwell Transform and, in the algorithm-level, an adaptable deep learning model for classifying data based on the input data variation. As for images, our contribution is the application of deep learning models for the recognition of Tifinagh handwritten characters.
- Lack of data. We attempt to address the challenge of lacking enough supervised data. We demonstrate how our framework can be further used within a multi-modal framework based on a novel transfer learning approach, in order to tackle the problem of lack of data in a particular classification task. Furthermore, we define three other multi-modal techniques for handling classification under lack of data: the first one relies on transfer learning with fine-tuning by using the appropriate source domain task, the second one on a deep neural network ensemble technique and the third one on a deep neural network with a novel augmentation technique and a voting technique at test time.
- Imbalanced data. We propose a cost-sensitive approach for classifying imbalanced data. Based on a cost-sensitive loss function, its objective is to correctly

classify the minority classes and favor them as much as the frequent ones by assigning a weighted misclassification cost based on the distribution of classes. We also show which loss functions are suitable for this approach and how this approach is efficient only for loss functions which ensure, via certain conditions, that the gradient of the loss is relatively large when misclassification occurs.

- Imbalanced data. We propose an efficient cost-sensitive regression algorithm for dealing with imbalanced data, suitable for learning with shallow and deep neural networks. The method uses an updated loss function which pushes gradients to be influenced by all data equally including rare and frequent ones during backpropagation.
- Imbalanced data. We propose novel evaluation strategies for regression models under imbalanced domains, which are shown to be more robust to the imbalance of data as they are able to reflect the performance of rare events as well as frequent ones. As such, we introduce new scalar measures, namely Geometric Mean Error (*GME*) and Class-Weighted Error (*CWE*), as well as graphical-based measures, namely REC_{TPR} , REC_{TNR} , REC_{G-Mean} and REC_{CWA} curves. These evaluation approaches are computed based on the estimate of the probability density function of given data and also on the concept of positives and negatives normally used in classification.
- Data with outliers. We introduce a deep regression model robust to outliers based on a novel loss function which does not require any hard threshold on the proportion of outliers in the training set. Our model has shown promising results and thus can generalize to any dataset with outliers.

Résumé

De nos jours, les données sont devenues l'un des atouts majeurs qui constituent une richesse pour la recherche. Les informations présentes sont devenues aussi bien pour les entreprises que pour les instituts de recherche un facteur de compétitivité et d'innovation. En général, ces données permettent de découvrir et d'expliquer certains phénomènes existants ou bien d'extrapoler des nouvelles connaissances à partir des informations présentes. Cependant, dans le monde réel, et dans le contexte d'apprentissage supervisé, ces données brutes ne sont pas toutes faciles à traiter par les modèles d'apprentissage. On peut avoir :

- des données homogènes temporelles difficiles à traiter qui présentent un défi de par leur grande masse, leur grande dimensionnalité, et l'aspect de changement continu.
- des données peu abondantes présentant un manque aigu de données étiquetées, ce qui entraîne généralement des difficultés de traitement ;
- des données déséquilibrées contenant un dosage déséquilibré entre les observations fréquentes et les observations rares ou très peu représentées ;
- des données contenant des observations aberrantes qui faussent généralement les méthodes traditionnelles d'analyse;

Cette thèse essaie de répondre à cette problématique en proposant des algorithmes qui s'appuient sur les réseaux de neurones. En effet, dans cette thèse, on introduit de nouvelles solutions adaptées à ces types de données :

- Données homogènes. On introduit de nouvelles approches au niveau des données et au niveau algorithmiques pour la classification des séries temporelles et des données à deux dimensions (images).
- Manque de données. On a développé un Framework d'apprentissage profond en se basant sur une nouvelle technique de transfert d'apprentissage (Transfer Learning). Par ailleurs, on propose d'autres nouvelles techniques à savoir : une approche associée à un transfert d'apprentissage avec "fine-tuning", une

technique d'un ensemble de réseaux de neurones profonds, et une approche basée sur une technique de vote appliquées sur un réseau de neurone profond durant la phase de test.

- Données déséquilibrées. On introduit une approche de reconnaissance optimisée pour la classification supervisée des données déséquilibrées. Cette approche consiste à modifier la fonction de coût en assignant des coûts de classification erronée pondérés selon la distribution des classes. On montre également les fonctions de coût adaptées à notre approche sont seulement celles dont le gradient est relativement large lorsque les classifications erronées surviennent.
- Données déséquilibrées. On propose un algorithme de régression de reconnaissance optimisée pour le traitement des données déséquilibrées, adaptées à l'apprentissage avec des réseaux de neurones artificielles et profonds. Cette méthode utilise une fonction de coût modifiée qui pousse les gradients à prendre en considération les événements rares autant que les événements fréquents et ce durant le processus de rétro-propagation du gradient.
- Données déséquilibrées. On propose de nouvelles stratégies d'évaluation des modèles de régression à partir de données déséquilibrées (à fréquences différentes). Ces stratégies sont plus robustes que les mesures existantes car elles permettent de faire apparaître la performance aussi bien des événements rares que les fréquents.
- Données avec intrus. Le nouveau modèle qu'on a développé est capable de traiter et classifier les bases de données bruitées.

Contents

I	Introduction	1
1	Introduction to machine learning	3
1.1	The Amazing Progress of Deep Learning	3
1.2	Towards Artificial Agents that Exist in the World	3
2	Contributions	5
3	Thesis Structure	7
	Bibliography	9
II	Neural Networks and Deep learning: background and basics	13
1	Deep Learning: definition, categories and prior work	17
1.1	Definition	17
1.2	Deep learning categories and their applications	18
1.2.1	Supervised Learning	18
1.2.2	Unsupervised Learning	20
1.2.3	Reinforcement Learning	21
	Bibliography	21
2	Feed-forward Neural Networks	27
2.1	Model hyper-parameters: feed-forward NN architecture	28
2.1.1	Linear functions	28
2.1.2	Non-linear functions	30
2.2	Optimization	32
2.2.1	Loss functions	32
2.2.2	Backpropagation	33
2.2.3	Gradient descent method	38
2.3	Types of feed-forward neural networks / Feed-forward Neural Network Architectures	42
2.3.1	Single-Layer Perceptron.	42

2.3.2	Multi-layer Perceptron (MLP).	44
2.3.3	Convolutional Neural Networks (ConvNet).	45
	Bibliography	48
3	Deep Belief Networks	55
3.1	Restricted Boltzmann Machines (RBMs)	55
3.2	Pre-training phase : Training each RBM	56
3.3	Forming the DBN and fine-tuning it	60
	Bibliography	61
III Learning from homogeneous data		67
1	Learning from time-series	69
1.1	Introduction	69
1.2	Related work	70
1.2.1	Classical methods	70
1.2.2	Automated methods	71
1.3	Overview of time series data	73
1.4	Methodology	74
1.4.1	Our Data-level approach	74
1.4.2	Algorithm-level approach: Adaptive convolutional layer filter	76
1.5	Experimental setup	77
1.5.1	Stereotypical Motor Movement Recognition task	77
1.5.2	Human Activity Recognition (HAR) task	80
1.6	Experiments	81
1.6.1	Experiment 1	81
1.6.2	Experiment 2	83
1.6.3	Experiment 3: Comparison with other SMM detection techniques	84
1.7	Results	84
1.7.1	Results of Experiment 1.	84
1.7.2	Results of Experiment 2.	87
1.7.3	Results of Experiment 3.	88
1.8	Conclusion	89
	Bibliography	90
2	Learning from images	97
2.1	Introduction	97
2.2	Related work	99

2.3	Methodology	100
2.3.1	Preprocessing and data augmentation phase	100
2.3.2	ConvNet model	100
2.3.3	DBN model	101
2.4	Experiments and results	101
2.4.1	Experiments using Convolutional Neural Networks	102
2.4.2	Experiments using Deep Belief Networks	103
2.4.3	ConvNet vs DBN results	106
2.4.4	Confusion errors and character similarity	107
2.4.5	Comparative results	107
2.5	Conclusion	109
	Bibliography	110
IV Learning from few data (with lack of data)		113
1	Multimodal learning: Transfer learning approach with fine-tuning	117
1.1	Overview of the Phoenician alphabet	118
1.2	Phoenician Handwritten Character Dataset	119
1.3	Handwritten Phoenician Character recognition	121
1.3.1	Phoenician ConvNet.	121
1.3.2	Experiments and Results.	122
1.4	Transfer Learning for recognizing Handwritten Alphabets with lack of annotated data	124
1.4.1	Methodology	124
1.4.2	Target datasets	125
1.4.3	Experiments and results	126
1.5	Conclusion	128
	Bibliography	129
2	Multimodal learning: Novel transfer learning approach	133
2.1	Methodology: Feature learning via knowledge transfer	134
2.2	Experiments	135
2.2.1	Experiment 1	135
2.2.2	Experiment 2: Comparison with other techniques	137
2.3	Results	137
2.4	Conclusion	140
	Bibliography	141

3	MultiModal Learning: Novel ensemble learning approach	145
3.1	Introduction	145
3.2	Related work	147
3.3	Methodology	148
3.4	Experimental setup	151
3.4.1	Dataset.	151
3.4.2	Deep learning Model architecture and training.	152
3.5	Experiments	154
3.6	Results	155
3.6.1	Results of training ConvNet models.	155
3.6.2	Results of the ensemble learning framework using different combination approaches.	156
3.6.3	Comparative results.	158
3.6.4	Running cost.	159
3.7	Conclusion	159
	Bibliography	160
4	Novel augmentation and voting approach at test time	165
4.1	Methodology	165
4.1.1	Data augmentation technique.	165
4.1.2	Multi-angle and multi-scale voting during test time	166
4.2	Experiments and results	168
4.2.1	Voting technique based on different variants.	168
4.2.2	Comparative study.	169
4.3	Conclusion	170
	Bibliography	171
V	Learning from imbalanced data	175
1	Approaches for Handling Classification under Imbalanced Domains	179
1.1	Introduction	179
1.2	Related work	180
1.2.1	Data-level methods	181
1.2.2	Algorithm-level methods	181
1.3	Methodology	183
1.3.1	Cost-sensitive learning approach	183
1.3.2	Efficiency of the cost-sensitive learning approach based on the nature of the loss function used	185
1.3.3	Faster convergence with cost-sensitive learning approach	195

1.4	Experimental study	196
1.4.1	Datasets	196
1.4.2	Training and Experimental setup	197
1.5	Experiments and results	199
1.5.1	Effect of our cost-sensitive approach on classification performance	199
1.5.2	Effect of our cost-sensitive approach on convergence speed	201
1.5.3	Comparison with other techniques	202
1.6	Conclusion	203
	Bibliography	204
2	Techniques for Handling Regression under Imbalanced Domains: algorithm and evaluation techniques	207
2.1	Introduction	207
2.2	Related work	208
2.2.1	Regression Strategies for Handling Imbalanced Domains	208
2.2.2	Evaluation approaches for imbalanced regression domains	210
2.3	Methodology	211
2.3.1	Cost-sensitive learning approach	212
2.3.2	Evaluation methods for regression under imbalanced domains	221
2.4	Experimental study	231
2.4.1	Datasets	231
2.4.2	Experimental setup	234
2.5	Experiments and results	235
2.5.1	Experiments	235
2.5.2	Results	236
2.6	Conclusion	240
	Bibliography	241
VI	Learning from outliers	245
1	Introduction	249
1.1	Robust Regression	249
1.2	Deep Learning for Regression	250
2	Methodology	253
2.1	Background on popular loss functions for regression	254
2.2	Our proposed loss function	255
2.3	Comparing our loss function to other loss functions	257

3	Experimental setup	259
4	Experiments and results	261
4.1	Baseline evaluation	261
4.2	Comparison with other techniques	261
5	Conclusion	263
	Bibliography	263
	Conclusion	267
1.1	Conclusion	267
1.2	Towards the Future	268
	List of publications	269
1.1	Journal papers	269
1.2	Books	269
1.3	Conference papers	270
	Notation	271
1.1	List of commonly used abbreviations	271
1.2	General math notation	272
1.3	Abbreviations for the DBN	272
	Appendix	273
1.1	History of Deep learning	273
1.2	Details of the Code	274
1.3	Supplementary material	275
1.3.1	Supplementary material for Part II	275
1.3.2	Supplementary material for Chapter 1	275
1.3.3	Supplementary material for Chapter 2	277
	Bibliography	288
	List of Figures	291
	List of Figures	291
	List of Tables	296

Part I

Introduction

Chapter 1

Introduction to machine learning

1.1 The Amazing Progress of Deep Learning

Deep learning is now regarded as the state-of-the-art solution in almost all machine learning tasks across various domains (Schmidhuber, 2015). The advantage these architectures provide over shallow architectures is the ability to extract hierarchies of meaningful abstracted features from the underlying input data. For instance, Convolutional Neural Networks (ConvNets), one type of deep learning networks, have yielded the highest accuracy on computer vision benchmarks such as ImageNet (Jia Deng et al., 2009; Krizhevsky et al., 2012).

Yet the breakthroughs of deep learning are not limited to computer vision but also to temporal sequences (RNN) such as natural language processing (Bahdanau et al., 2014) and speech recognition (Graves et al., 2013). Moreover, thanks to deep learning, new applications have emerged such as image captioning (by describing the picture in words) (Xu et al., 2015; Johnson et al., 2016), question answering (Weston et al., 2015) and image generation (Van Den Oord et al., 2016). Furthermore, deep reinforcement learning, which relies on deep learning and reinforcement learning principles, demonstrated a great success on robotics and video games (Lanctot et al., 2016).

1.2 Towards Artificial Agents that Exist in the World

However, machine learning and computer science are usually applied on datasets that are rather divorced from the real world. The current standard implementation for machine learning is to produce a dataset of tuples $\langle x, y \rangle$ in which every data sample is composed of an input x and a target y , and a dataset consists of aggregates of these tuples together. Datasets are typically cleaned and normalized, and in many cases, a pre-defined training and test split of the data is produced to equalize results

from many groups. This dataset formalization ignores many characteristics present in the real world. As these characteristics will be one of the main problematics of this thesis, it is worth noting them here:

- i First, in many real-world applications, the collected data follows an ‘imbalanced’ or ‘skewed’ distribution. Indeed, having data instances with some targets that are abundant (i.e., majority events) and others scarce (e.g., minority events), classification or regression algorithms tend to learn more from data belonging to the majority events and ignore minority events during the training process. Thus, minority events’ data tend are not adequately learned, which results in poor accuracy performance of these data. And, unfortunately, within many applications, the minority instances actually represent the concept of interest (such as fraud in banking operations, abnormal cell in medical data, dangerous activity in a a continuous surveillance task, object classification etc.), which makes the detection of these rare events even more important.
- ii Recently, with the increasing growth of data, several annotated (e.g., labeled) datasets have become publicly available. Nonetheless, some disciplines such as medicine have few annotated data either because it demands human expertise or because annotating data is time consuming. And, as a reminder, most of machine learning algorithms require a large amount of labeled data for proper training and for a proper feature representation.
- iii Third, noise is an inescapable feature of the natural environment which must be addressed.

These characteristics are often considered to be confounds on the true interesting problem of creating machines that learn, but it is far from obvious that these characteristics can be abstracted away without substantially modifying learning models or generating new ones.

This thesis will discuss new learning methods to face these challenges, and opens a door for substantial further research that bridges the domain of deep learning.

Chapter 2

Contributions

In this thesis, we introduce novel solutions to homogeneous data as well as problems of lack of data, imbalanced data and data with outliers. These solutions involve the proposal of new frameworks or approaches:

- Homogeneous data. We introduce a data-level and an algorithm-level approaches for classifying 1 dimensional input data (time-series) (Sadouk et al., 2018; Sadouk, 2018) (Part III - Chapter 1) as well as 2 dimensional input data (images) (Sadouk et al., 2017) (Part III - Chapter 2). Regarding time-series, we propose in the data-level a method to convert time-series to frequencies based on the Stockwell Transform and in the algorithm-level an adaptable deep learning model for classifying data based on the input data variation .
- Lack of data. We demonstrate how our framework can be further used within a multi-modal framework based on a novel transfer learning approach, in order to tackle the problem of lack of data in a particular classification task (Sadouk et al., 2018) (Part IV - Chapter 2)
- Lack of data. Furthermore, we define three other multi-modal techniques for handling classification under lack of data: the first one relies on transfer learning with fine-tuning by using the appropriate source domain task (Sadouk et al., 2020b) (Part IV - Chapter 1), the second one on a Deep Neural Network ensemble technique (Sadouk et al., 2016a) (Part IV - Chapter 3) and the last one on a Deep Neural Network voting technique at test time (Sadouk et al., 2016b) (Part IV - Chapter 4)
- Imbalanced data. We propose a cost-sensitive approach for classifying imbalanced data (Sadouk et al., 2020c) (Part V - Chapter 1). Based on a cost-sensitive loss function, its objective is to correctly classify the minority classes and favor them as much as the frequent ones by assigning a weighted

misclassification cost based on the distribution of classes. We also show which loss functions are suitable for this approach and how this approach is efficient only for loss functions which ensure, via certain conditions, that the gradient of the loss is relatively large when misclassification occurs.

- Imbalanced data. We propose an efficient cost-sensitive regression algorithm for dealing with imbalanced data (Sadouk et al., 2021) (Part V - Chapter 2), suitable for learning with shallow and deep neural networks. The method uses an updated loss function which pushes gradients to be influenced by all data equally including rare and frequent ones during backpropagation.
- Imbalanced data. We propose novel evaluation strategies for regression models under imbalanced domains (Sadouk et al., 2021) (Part V - Chapter 2), which are shown to be more robust to the imbalance of data as they are able to reflect the performance of rare events as well as frequent ones. As such, we introduce new scalar measures, namely Geometric Mean Error (*GME*) and Class-Weighted Error (*CWE*), as well as graphical-based measures, namely REC_{TPR} , REC_{TNR} , REC_{G-Mean} and REC_{CWA} curves. These evaluation approaches are computed based on the estimate of the probability density function of given data and also on the concept of positives and negatives normally used in classification.
- Data with outliers. We propose a robust deep regression model based on a novel loss function which does not require any hard threshold on the proportion of outliers in the training set (Sadouk et al., 2020a) (Part VI). Our model has shown promising results and thus can generalize to any dataset with outliers.

Chapter 3

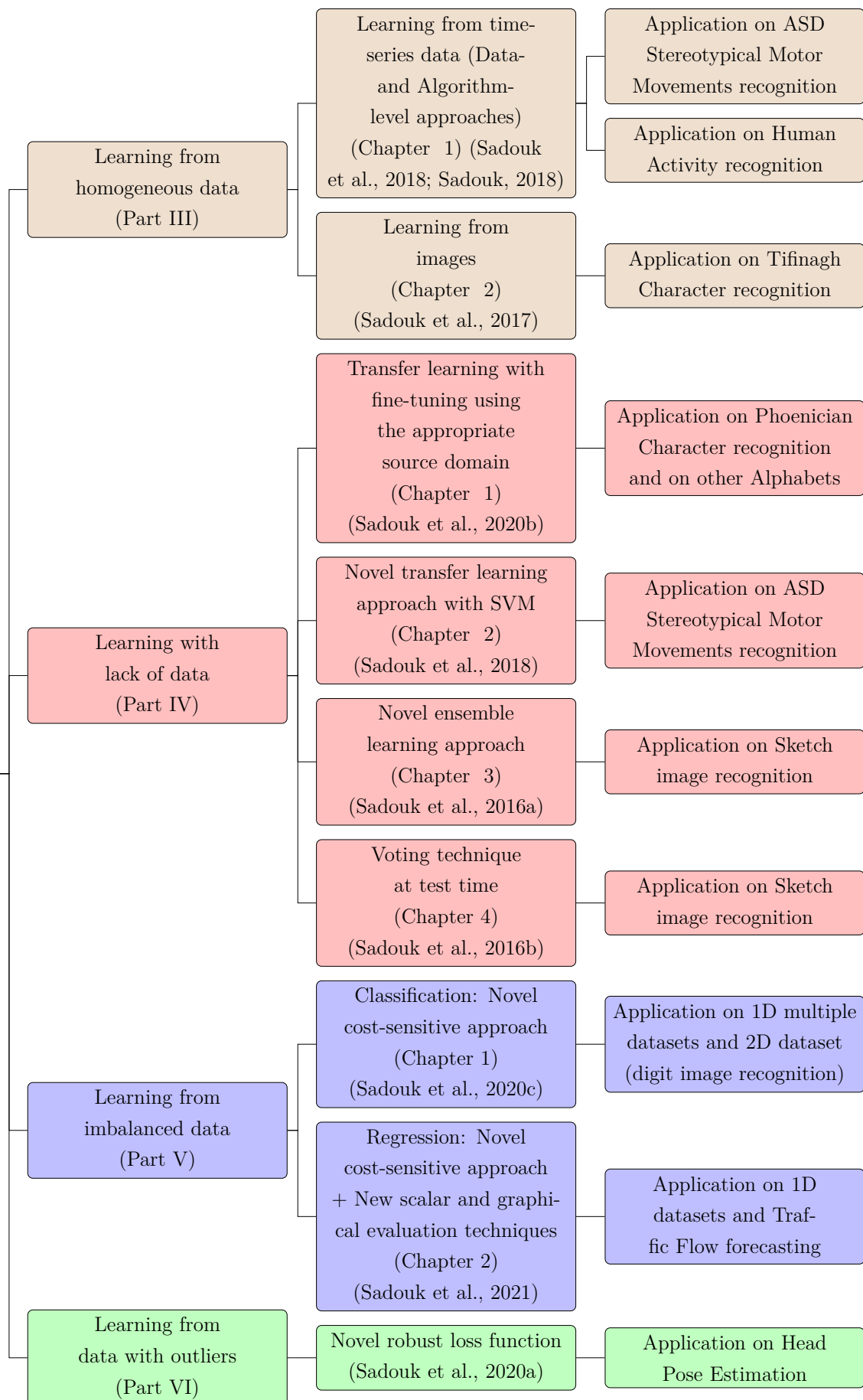
Thesis Structure

This dissertation is organized as follows:

- Part 2 gives an overview of Neural Networks and Deep learning to provide a background on recent works.
- Part 3 describes our research findings in the context of learning from homogeneous data including time-series and images.
- Part 4 is dedicated to the problem of lack of data and introduces new approaches allowing a good classification in real supervised tasks even with few input data.
- Part 5 approaches the issue of learning with imbalanced data and discusses learning data representations with imbalanced data for classification and regression tasks.
- Part 6 introduces a new regression approach in a data environment with outliers.
- Finally, Part 7 concludes the thesis by summarizing our contributions and findings and by discussing remaining challenges and future research directions.

Further details about the organization of the work is given the figure below.

Thesis Structure



Bibliography

- Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural Machine Translation by Jointly Learning to Align and Translate. arxiv.org, 2014. ISSN 0147-006X. doi: 10.1146/annurev.neuro.26.041002.131047.
- Alex Graves, Abdel Rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings, pages 6645–6649, 2013. ISBN 9781479903566. doi: 10.1109/ICASSP.2013.6638947.
- Jia Deng, Wei Dong, R Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A large-scale hierarchical image database. In IEEE Conference on Computer Vision and Pattern Recognition, pages 248–255, 2009. ISBN 978-1-4244-3992-8. doi: 10.1109/CVPRW.2009.5206848.
- Justin Johnson, Andrej Karpathy, and Li Fei-Fei. DenseCap: Fully convolutional localization networks for dense captioning. In Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, volume 2016-Decem, pages 4565–4574, 2016. ISBN 9781467388504. doi: 10.1109/CVPR.2016.494.
- Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In Advances in neural information processing systems, pages 1097–1105, 2012. ISBN 9780415468442. doi: 10.1061/(ASCE)GT.1943-5606.0001284.
- Marc Lanctot, Demis Hassabis, Thore Graepel, Veda Panneershelvam, Timothy Lillicrap, John Nham, Ioannis Antonoglou, David Silver, Chris J. Maddison, Arthur Guez, Ilya Sutskever, Aja Huang, Julian Schrittwieser, Nal Kalchbrenner, Dominik Grewe, George van den Driessche, Madeleine Leach, Laurent Sifre, Koray Kavukcuoglu, and Sander Dieleman. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, jan 2016. ISSN 0028-0836. doi: 10.1038/nature16961.
- Lamyaa Sadouk. CNN Approaches for Time Series classification. In *Time Series Analysis*. 2018. doi: 10.5772/intechopen.81170.
- Lamyaa Sadouk, Taoufiq Gadi, and El Hassan Essoufi. Sketch recognition using a hybrid model ensemble of deep CNNs. In *International conference on computing, wireless and communication systems*, 2016a.

- Lamyaa Sadouk, Taoufiq Gadi, and El Hassan Essoufi. A novel approach of deep convolutional neural networks for sketch recognition. In *Advances in Intelligent Systems and Computing*, pages 99–112, 2016b. ISBN 9783319529400. doi: 10.1007/978-3-319-52941-7_11.
- Lamyaa Sadouk, Taoufiq Gadi, and El Hassan Essoufi. Handwritten tfinagh character recognition using deep learning architectures. In *Proceedings of the 1st International Conference on Internet of Things and Machine Learning*, page 59, 2017. doi: 10.1145/3109761.3109788.
- Lamyaa Sadouk, Taoufiq Gadi, and El Hassan Essoufi. A Novel Deep Learning Approach for Recognizing Stereotypical Motor Movements within and across Subjects on the Autism Spectrum Disorder. *Computational Intelligence and Neuroscience*, 2018:1–16, 2018. ISSN 1687-5265. doi: 10.1155/2018/7186762.
- Lamyaa Sadouk, Taoufiq Gadi, and El Hassan Essoufi. Robust Loss Function for Deep Learning Regression with Outliers. In *Advances in Intelligent Systems and Computing*, volume 1076, pages 359–368. Springer, 2020a. ISBN 9789811509469. doi: 10.1007/978-981-15-0947-6_34.
- Lamyaa Sadouk, Taoufiq Gadi, El Hassan Essoufi, and Abdelhak Bassir. Handwritten Phoenician Character Recognition and its Use to Improve Recognition of Handwritten Alphabets with Lack of Annotated Data. *International Journal of Advanced Trends in Computer Science and Engineering*, 9(1)(1):171–181, 2020b. doi: 10.30534/ijatcse/2020/26912020.
- Lamyaa Sadouk, Taoufiq Gadi, El Hassan Essoufi, and Abdelhak Bassir. Handwritten phoenician character recognition and its use to improve recognition of handwritten alphabets with lack of annotated data. *International Journal of Advanced Trends in Computer Science and Engineering*, 9(1):171–181, 2020c. ISSN 22783091. doi: 10.30534/ijatcse/2020/26912020.
- Lamyaa Sadouk, Taoufiq Gadi, and El Hassan Essoufi. A novel cost-sensitive learning approach and metric for regression in imbalanced domains. *Expert Systems*, 2021, 2021. doi: 10.1111/exsy.12680.
- Jurgen Schmidhuber. Deep learning in neural networks: An overview. *Neural networks*, 61:85–117, 2015.
- Aäron Van Den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. Pixel recurrent neural networks. In *33rd International Conference on Machine Learning, ICML 2016*, volume 4, pages 2611–2620. International Machine Learning Society (IMLS), 2016. ISBN 9781510829008.

Jason Weston, Sumit Chopra, and Antoine Bordes. Memory networks. In 3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings. International Conference on Learning Representations, ICLR, 2015.

Kelvin Xu, Jimmy Lei Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhutdinov, Richard S Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. In 32nd International Conference on Machine Learning, ICML 2015, volume 3, pages 2048–2057, 2015. ISBN 9781510810587.

Part II

Neural Networks and Deep learning: background and basics

In recent years, machine learning has become more and more popular in research and a large number of applications, including multimedia concept retrieval, image classification, video recommendation, social network analysis, text mining, etc. Among various machine learning algorithms, “deep learning” (DL), also known as representation learning (Deng, 2014), is widely used in these applications nowadays. With great successes around many fields, deep learning now represent a huge step forward for machine learning and is regarded as one of the hottest research directions in the machine learning society.

Deep learning is a sub-field of machine learning. As such, it commits to a particular set of design choices, and explores ways of addressing the two challenges of machine learning in this self-imposed design space. In doing so, it has accumulated a set of methods that have proven to work well across a large range of tasks, sometimes outperforming other approaches by a large margin. In this chapter, a literature review on the related work based on the deep learning framework and automatic information retrieval is presented. In chapter 1, we define DL categories and recapitulate the prior work that has been accomplished in DL by discussing recent research directions, the most important methods and the most popular algorithms in these areas. This will serve as a starting point to begin investigating methods to come up with novel Deep Learning frameworks. Then, in chapters 2 and 3, we introduce components of deep neural networks (including design choices and their implications) for supervised learning tasks.

Chapter 1

Deep Learning: definition, categories and prior work

1.1 Definition

Similar to how medicine, energy, transportation, manufacturing, industrialization, and food production were revolutionized by electricity in the 20th century, deep learning (DL) is poised to set course in becoming part of the next century of revolutions. Moreover, the explosive growth and availability of data as well as the remarkable advancement in hardware technologies have led to the emergence of new studies in deep learning. Deep learning which has its root from conventional neural networks significantly outperforms its predecessors. It utilizes graph technologies with transformations among neurons to develop many layered learning models.

Traditionally, the efficiency of machine learning algorithms highly relied on the goodness of the representation of the input data. A bad data representation often leads to lower performance compared to a good data representation. Therefore, feature engineering has become an important research direction in machine learning for a long time, which focuses on building features from raw data and has led to lots of research studies. Furthermore, feature engineering is often very domain specific and requires significant human efforts. For example, in computer vision, different kinds of features have been proposed and compared such as SIFT (“Scale Invariant Feature Transform”) (Lowe, 1999), HOG (“Histogram of Oriented Gradients”) (Dalal and Triggs, 2005), and BoW (“Bag of Words”) (?). Once a new feature is proposed and performs well, it would become a trend for years. Similar situations happened in other domains including Speech Recognition and Natural Language Processing.

Comparatively, deep learning algorithms perform feature extraction in a quite automated way which allows the researchers to extract discriminative features with-

out domain knowledge and human input. These algorithms include a layered architecture of data representation, where high-level features can be defined in the top layers while low-level features are extracted from the bottom layers. Inspired by the way the human brain processes information and learns, the DL model consists of several levels of representation, in which every level uses information from the previous level to learn deeply. Each level corresponds, in this model, to a different area of the cerebral cortex, and every level abstract more the information in the same way of the human brain.

1.2 Deep learning categories and their applications

So far, deep learning has been actively applied in a wide range of applications in many areas and has covered all types of tasks including supervised, unsupervised and reinforcement learning tasks, as depicted in Figure 1.1.

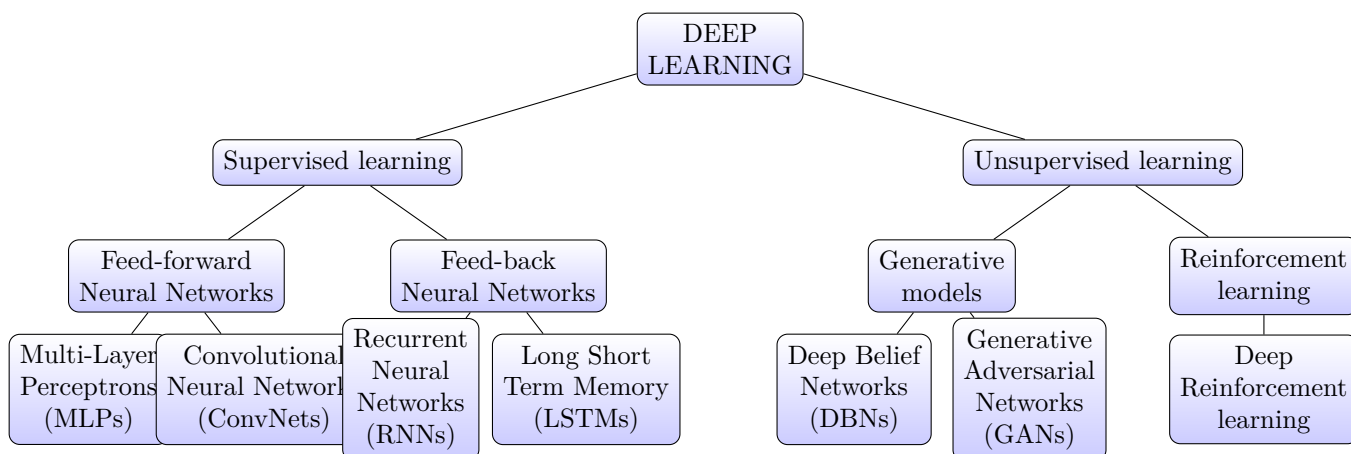


Figure 1.1: Overview of Deep Learning techniques.

1.2.1 Supervised Learning

The most straightforward deep learning tasks fall under the umbrella of supervised learning, whereby we have access to examples of correct input-output pairs that we can show to the neural network model during the training phase. In other words, each input is labeled with a ground-truth output value (e.g., target value) so that the learning system knows the real output when input is fed. Existing deep learning models for such learning can be split into two main categories: feedforward and feedback neural networks.

Feedforward neural networks. Feedforward neural networks are straightforward networks which associate inputs with outputs with no feedback i.e., the output of any layer does not affect that same layer (Figure 1.2.a). In other words, the information flow is unidirectional and a unit within a layer sends information to an

unit of another layer from which it does not receive any information. Accordingly, prediction at an instance i does not depend on the prediction at the previous instance $i - 1$ in the previous moment. This type of organization is also referred to as bottom-up or top-down. For example, if a network classifies the first image as a ship, then the next image classification will not be changed by the former classification and will therefore classify a “bus” image as a bus.

Feedforward neural network models include Multi-layer perceptrons (MLPs) and Convolutional Neural Networks (ConvNets) and have been used in many vision-related applications, such as face recognition (Taigman et al., 2014), object detection and semantic segmentation (Girshick et al., 2014), image retrieval (Babenko et al., 2014), self-driving cars (Chen, 2015), medical diagnostics (J and OG, 2015) and video surveillance (Kang et al., 2017). When first introduced in the 1980s, ConvNets were the preferable solution for small problems only (e.g. LeNet (Le Cun et al., 1990) for handwritten digit recognition). They did not gain popularity since they were restricted by the high computational cost due to the large network architecture and training data. It was not until the emergence of GPUs as well as new hyper-parameters such as ReLU activation neurons (instead of TanH), max-pooling (instead of average pooling) and dropout regularization, that ConvNets achieved very high performance.

Feedback neural networks. On the other hand, in a Feedback neural network, also denoted as recurrent or interactive neural network, the decision this network gets at this moment (at instance i) depends on the decision which the network got at the previous moment (at instance $i - 1$). Computations derived from earlier input are fed back into the network, thus introducing loops which produce a kind of memory, as shown in Figure 1.2.b. As a consequence, the current prediction of a recurrent neural network depends on both the previous prediction of the network (at instance $i - 1$) and the current one (at instance i). Thus, rather than learning a function that maps the input to the output (which is the case for feedforward networks), the Feedback network represents an internal state for the network that can cause the network’s behavior to change over time based on its input. Feedback networks are used in many applications including Image Captioning, Sentiment analysis, Machine Translation, etc. For example, in order to read a big text file and predict the next character, feedback networks can accumulate the knowledge through previous time-steps (previously read characters) and therefore be aware of the context.

The most common Feedback networks are Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTMs), which greatly improved the accuracy of Speech Recognition (Hinton, 2012), Natural Language Processing (Collobert et al., 2011), and Machine Translation (Bahdanau et al., 2014).

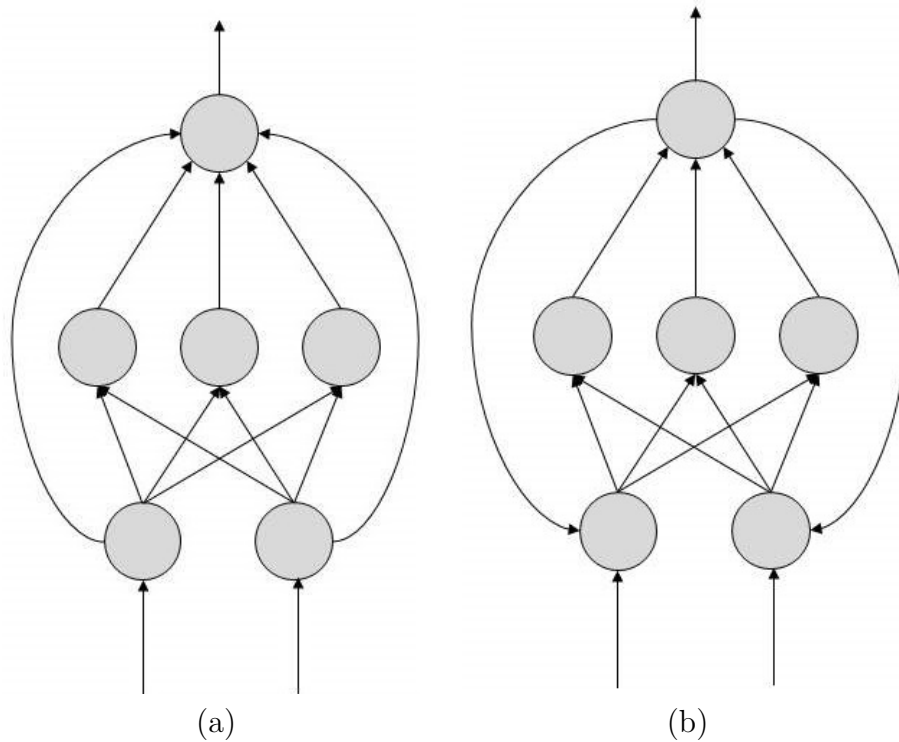


Figure 1.2: Example of: (a) a Feedforward neural network and (b) a Feedback neural network.

1.2.2 Unsupervised Learning

Some other deep learning tasks are axed on another common learning category/class referred to as Unsupervised Learning, where training instances are not labeled with the belonging class 1.2.b. The role of the unsupervised deep learning system is then to develop and organize the data, searching for common characteristics among them, and changing based on internal knowledge. Existing unsupervised deep learning models such as Deep Belief Networks (DBNs) and Deep Convolutional Generalized Adversarial Network are generative models whose goal is to imitate the process that generates the training data in order to produce new data that resembles the training data. These models have made significant advancements in generative models, which have improved the efficiency and quality of Compressed Sensing (Bora et al., 2017) and Super Resolution (Ledig et al., 2017). They even gave rise to new applications such as Image Style Transfer (Gatys et al., 2015), Visual Manipulation (Zhu et al., 2016) and Image Synthesis (Chen and Koltun, 2017). Furthermore, after being trained, unsupervised deep learning models such as DBNs can further serve as a supervised learning model by being fine-tuned using labeled data.

1.2.3 Reinforcement Learning

A newer type of learning problem that has gained a great deal of traction recently is called Reinforcement Learning (RL). In reinforcement learning, the system is not fed with examples of correct input-output pairs, but rather finds a method to quantify its performance in the form of a reward signal. Reinforcement learning methods resemble how humans and animals learn: the machine tries a bunch of different things and is rewarded when it does something well. Deep reinforcement learning have made progress in Game Playing (Lanctot et al., 2016), Visual Navigation (Mottaghi et al., 2017), Device Placement (Mirhoseini et al., 2017), Automatic Neural Network Architecture Design (Zoph and Le, 2016) and Robotic Grasping (Levine et al., 2018).

In our study, we focus on Supervised Learning tasks, especially non-sequential tasks (i.e., tasks whose current input does not depend on previous inputs). As such Feedforward networks will be discussed and their key concepts and formulation will be described (Sec. 2). Next, we will perform a general analysis of a generative model namely the “Deep Belief Network”, an unsupervised deep learning model that will be further used for supervised learning tasks through a fine-tuning process (chapter 3).

Bibliography

Artem Babenko, Anton Slesarev, Alexandr Chigorin, and Victor Lempitsky. Neural codes for image retrieval. In *Lecture Notes in Computer Science*, volume 8689 LNCS, pages 584–599. 2014. ISBN 9783319105895. doi: 10.1007/978-3-319-10590-1_38.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural Machine Translation by Jointly Learning to Align and Translate. arxiv.org, 2014. ISSN 0147-006X. doi: 10.1146/annurev.neuro.26.041002.131047.

Y. Bengio. Learning Deep Architectures for AI. *Foundations and Trends® in Machine Learning*, 2(1):1–127, 2009. ISSN 1935-8237. doi: 10.1561/22000000006.

Yoshua Bengio, Nicolas Boulanger-Lewandowski, and Razvan Pascanu. Advances in optimizing recurrent networks. In *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, pages 8624–8628, 2013. ISBN 9781479903566. doi: 10.1109/ICASSP.2013.6639349.

Ashish Bora, Ajil Jalal, Eric Price, and Alexandros G. Dimakis. Compressed

- Sensing using Generative Models. In *dl.acm.org*, pages 537—546, 2017. ISBN 9781510855144. doi: 10.5829/idosi.wasj.2013.23.04.368.
- Ken Chatfield, Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Return of the Devil in the Details: Delving Deep into Convolutional Nets. In *British Machine Vision Conference*, 2014. ISBN 1-901725-52-9. doi: 10.5244/C.28.6.
- Chenyi Chen. DeepDriving : Learning Affordance for Direct Perception in Autonomous Driving. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 2722–2730, 2015. doi: 10.1109/ICCV.2015.312.
- Qifeng Chen and Vladlen Koltun. Photographic Image Synthesis with Cascaded Refinement Networks. In *Proceedings of the IEEE International Conference on Computer Vision*, volume 2017-Octob, pages 1520–1529, 2017. ISBN 9781538610329. doi: 10.1109/ICCV.2017.168.
- Ronan Collobert, Jason Weston, Leon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. Natural Language Processing (almost) from Scratch. *Journal of machine learning research*, 12(Aug):2493—2537, 2011. ISSN 1532-4435. doi: 10.1145/2347736.2347755.
- George E. Dahl, Dong Yu, Li Deng, and Alex Acero. Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition. *IEEE Transactions on Audio, Speech and Language Processing*, 20(1):30—42, 2012. ISSN 15587916. doi: 10.1109/TASL.2011.2134090.
- Navneet Dalal and William Triggs. Histograms of Oriented Gradients for Human Detection. In *International Conference on computer vision & Pattern Recognition*, volume 1, pages 886–893, 2005. ISBN 0-7695-2372-2. doi: 10.1109/CVPR.2005.177.
- Li Deng. A Tutorial Survey of Architectures, Algorithms, and Applications for Deep Learning. *APSIPA Transactions on Signal and Information Processing*, 3, 2014.
- Stuart Dreyfus. The numerical solution of variational problems. *Journal of Mathematical Analysis and Applications*, 5(1):30–45, 1962. ISSN 10960813. doi: 10.1016/0022-247X(62)90004-5.
- Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. A Neural Algorithm of Artistic Style. *arXiv preprint arXiv:1508.06576*, aug 2015. ISSN 1935-8237. doi: 10.1561/22000000006.
- Ross Girshick, Jeff Donahue, Trevor Darrell, U C Berkeley, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In

IEEE Conference on Computer Vision and Pattern Recognition, pages 580–587, 2014. ISBN 978-1-4799-5118-5. doi: 10.1109/CVPR.2014.81.

Philippe Hamel and Douglas Eck. Learning Features from Music Audio with Deep Belief Networks. In Proc. of the 11th International Society of Music Information Retrieval, pages 339–344, 2010. ISBN 9789039353813.

Geoffrey E. Hinton. Training Products of Experts by Minimizing Contrastive Divergence. *Neural Computation*, 14(8):1771–1800, aug 2002. ISSN 0899-7667. doi: 10.1162/089976602760128018.

Geoffrey E. Hinton. A practical guide to training restricted Boltzmann machines. In *Neural networks: Tricks of the Trade*, pages 599–619. 2012.

Geoffrey E. Hinton, Simon Osindero, and Yee-Whye Teh. A Fast Learning Algorithm for Deep Belief Nets. *Neural Computation*, 18(7):1527–1554, jul 2006. ISSN 0899-7667. doi: 10.1162/neco.2006.18.7.1527.

Zhou J and Troyanskaya OG. Predicting effects of noncoding variants with deep learning–based sequence model. *Nature Methods*, 12(10):931, 2015.

Daniel Kang, John Emmons, Firas Abuzaid, Peter Bailis, and Matei Zaharia. NoScope: Optimizing Neural Network Queries over Video at Scale. *Proceedings of the VLDB Endowment*, 10(11):1586–1597, 2017. ISSN 16130073. doi: 10.14778/3137628.3137664.

Nitish Shirish Keskar, Jorge Nocedal, Ping Tak Peter Tang, Dheevatsa Mudigere, and Mikhail Smelyanskiy. On large-batch training for deep learning: Generalization gap and sharp minima. In *5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings. International Conference on Learning Representations, ICLR, 2017*.

Marc Lanctot, Demis Hassabis, Thore Graepel, Veda Panneershelvam, Timothy Lillicrap, John Nham, Ioannis Antonoglou, David Silver, Chris J. Maddison, Arthur Guez, Ilya Sutskever, Aja Huang, Julian Schrittwieser, Nal Kalchbrenner, Dominik Grewe, George van den Driessche, Madeleine Leach, Laurent Sifre, Koray Kavukcuoglu, and Sander Dieleman. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, jan 2016. ISSN 0028-0836. doi: 10.1038/nature16961.

Y. Le Cun, BE Boser, J. S. Js Denker, D. Henderson, R. E. Howard, E. Hubbard, L. D. Jackel, Bb Le Cun, J. S. Js Denker, and D. Henderson. Handwritten Digit Recognition with a Back-Propagation Network. In *Advances in Neural*

- Information Processing Systems (NIPS), pages 396–404. 1990. ISBN 1-55860-100-7. doi: 10.1111/dsu.12130.
- Yann LeCun, Marc Aurelio Ranzato, Christopher Poultney, Sumit Chopra, and Aurelio’Marc Ranzato. Sparse Feature Learning for Deep Belief Networks. In *Advances in neural information processing systems*, pages 1185—1192, 2008. ISBN 0018-9219. doi: 10.1109/5.726791.
- Yann A. LeCun, Léon Bottou, Genevieve B. Orr, and Klaus Robert Müller. Efficient backprop. In *Neural networks: Tricks of the trade*, volume 7700 LECTU, pages 9–48. 2012. ISBN 9783642352881. doi: 10.1007/978-3-642-35289-8-3.
- Christian Ledig, Lucas Theis, Ferenc Huszár, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, and Wenzhe Shi. Photo-realistic single image super-resolution using a generative adversarial network. In *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, volume 2017-Janua, pages 4681–4690, 2017. ISBN 9781538604571. doi: 10.1109/CVPR.2017.19.
- Sergey Levine, Peter Pastor, Alex Krizhevsky, Julian Ibarz, and Deirdre Quillen. Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. *International Journal of Robotics Research*, 37(4-5):421–436, apr 2018. ISSN 17413176. doi: 10.1177/0278364917710318.
- Seppo Linnainmaa. The representation of the cumulative rounding error of an algorithm as a Taylor expansion of the local rounding errors. Master’s Thesis (in Finnish), Univ. Helsinki, pages 6–7, 1970.
- D.G. Lowe. Object recognition from local scale-invariant features. In *Proceedings of the Seventh IEEE International Conference on Computer Vision*, volume 99, pages 1150–1157, 1999. ISBN 0-7695-0164-8. doi: 10.1109/ICCV.1999.790410.
- James Martens. Deep learning via Hessian-free optimization. In *ICML*, volume 27, pages 735–742, 2010. ISBN 9781605589077. doi: 10.1155/2011/176802.
- Azalia Mirhoseini, Hieu Pham, Quoc V. Le, Benoit Steiner, Rasmus Larsen, Yuefeng Zhou, Naveen Kumar, Mohammad Norouzi, Samy Bengio, and Jeff Dean. Device Placement Optimization with Reinforcement Learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 2430–2439, 2017.
- Abdel-rahman Mohamed, Dong Yu, and Li Deng. Investigation of full-sequence training of deep belief networks for speech recognition. In *Eleventh Annual Con-*

ference of the International Speech Communication Association, number September, 2010.

Abdel Rahman Mohamed, Tara N. Sainath, George Dahl, Bhuvana Ramabhadran, Geoffrey E. Hinton, and Michael A. Picheny. Deep belief networks using discriminative features for phone recognition. In ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing, pages 5060–5063, 2011. ISBN 9781457705397. doi: 10.1109/ICASSP.2011.5947494.

Roosbeh Mottaghi, Eric Kolve, Joseph J Lim, Abhinav Gupta, Li Fei-fei, and Ali Farhadi. Target-driven Visual Navigation in Indoor Scenes using Deep Reinforcement Learning. In IEEE international conference on robotics and automation (ICRA), pages 3357–3364, 2017.

V Nair and Geoffrey E. Hinton. 3D Object Recognition with Deep Belief Nets. In Advances in neural information processing systems, pages 1339–1347, 2009. ISBN 9781615679119.

Nesterov. A method of solving a convex programming problem with convergence rate. In Sov. Math. Dokl, volume 27, 1983.

David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, oct 1986. ISSN 0028-0836. doi: 10.1038/323533a0.

Tara N. Sainath, Brian Kingsbury, Bhuvana Ramabhadran, Petr Fousek, Petr Novak, and Abdel Rahman Mohamed. Making deep belief networks effective for large vocabulary continuous speech recognition. In 2011 IEEE Workshop on Automatic Speech Recognition and Understanding, ASRU 2011, Proceedings, pages 30–35, 2011. ISBN 9781467303675. doi: 10.1109/ASRU.2011.6163900.

Karen Simonyan and Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. In Proceedings of the 3rd International Conference on Learning Representations, sep 2015. ISBN 1097-0142 (Electronic)\n0008-543X (Linking). doi: 10.2146/ajhp170251.

Yaniv Taigman, Ming Yang, Marc’Aurelio Ranzato, and Lior Wolf. DeepFace: Closing the gap to human-level performance in face verification. In Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, pages 1701–1708, 2014. ISBN 9781479951178. doi: 10.1109/CVPR.2014.220.

Oriol Vinyals and Daniel Povey. Krylov Subspace Descent for Deep Learning. *Artificial Intelligence and Statistics*, pages 1261–1268, 2012.

S. Zhou, Q. Chen, and X. Wang. Active deep learning method for semi-supervised sentiment classification. *Neurocomputing*, 120:536–546, 2013.

Jun Yan Zhu, Philipp Krähenbühl, Eli Shechtman, and Alexei A. Efros. Generative visual manipulation on the natural image manifold. In *European Conference on Computer Vision*, volume 9909 LNCS, pages 597–613, 2016. ISBN 9783319464534. doi: 10.1007/978-3-319-46454-1_36.

Barret Zoph and Quoc V. Le. Neural Architecture Search with Reinforcement Learning. arXiv preprint arXiv:1611.01578., 2016.

Chapter 2

Feed-forward Neural Networks

Given a set of training data $\{(\mathbf{x}^s, \mathbf{y}^s)\}_{s=1}^S$ where S is the total number of instances (e.g., samples) in the training set, \mathbf{x}^s the input vector and \mathbf{y}^s the output vector, a feedforward neural network is trained using a function $\omega(\cdot)$. The goal of the network training is to obtain a mapping between \mathbf{x}^s and \mathbf{y}^s by adjusting or fine-tuning weights (also referred to as parameters) $\boldsymbol{\theta}$. The mapping is given by,

$$\hat{\mathbf{y}}^s = \omega(\mathbf{x}^s, \boldsymbol{\theta}) \quad (2.1)$$

where $\hat{\mathbf{y}}^s$ is the prediction or network output e.g., the estimated output value of the network.

Training consists of minimizing an objective function $L(\cdot)$ using backpropagation (Rumelhart et al., 1986) and stochastic gradient descent, as follows,

$$\min_{\boldsymbol{\theta}} L(\boldsymbol{\theta}, \omega) \quad (2.2)$$

where

$$L(\boldsymbol{\theta}, \omega) = \frac{1}{N} \sum_{i=1}^N L_i(\boldsymbol{\theta}, \omega), L_i(\boldsymbol{\theta}, \omega) = \ell(\mathbf{y}^i, \hat{\mathbf{y}}^i) \quad (2.3)$$

where N is the number of instances per batch, $\ell(\cdot)$ is the loss function, \mathbf{y}^i is the true label as one-hot encoding and $\hat{\mathbf{y}}^i = \omega(\mathbf{x}^i, \boldsymbol{\theta})$ is the network output vector for an instance i within the batch.

The rest of this chapter is structured as follows. In chapter 2.1, we start by defining the function $\omega(\cdot)$ and hyper-parameters of the feed-forward model (i.e., parameters responsible for mapping the input to the network output) are introduced. The next chapter (chapter 2.2) defines the backpropagation process where optimization is conducted to find the optimal $\boldsymbol{\theta}^*$. Then, types of feedforward neural networks are elaborated (chapter 2.3).

2.1 Model hyper-parameters: feed-forward NN architecture

A typical feed-forward neural network is a sequence of linear and non-linear functions. These linear and non-linear operators are often encapsulated into layers when designing neural networks. The output of the previous layer is served as the input of the next layer. Below we will introduce some commonly used operators.

2.1.1 Linear functions

Fully Connected layer. One of the principal linear function within feed-forward neural networks is the Fully Connected layer (also referred to as “FC”). It consists of a matrix-vector multiplication and is defined as,

$$\mathbf{y} = \mathbf{W}\mathbf{x} + b \quad (2.4)$$

where $\mathbf{y} \in \mathbb{R}^m$, $\mathbf{x} \in \mathbb{R}^d$ is an input vector, $\mathbf{W} \in \mathbb{R}^{m \times d}$ is the weight matrix, and $b \in \mathbb{R}^m$ is the bias vector. For FC, each output dimension y_j , $j = 1, \dots, m$, also called unit, is computed as a weighted sum of all the input dimensions.

Convolution layer. Another linear function referred to as Convolution layer constrains the size of this input range. The input to this layer has often three dimensions: rows (also denoted as “height”), columns (also denoted as “weight”), and depth (also denoted as “channels”). The first convolutional layer consists of convolving the input with a set of M_1 filters (also known as “kernels”) whose dimension has to have the same number of channels as the input.

Before addressing the convolution with a three-dimensional input, let’s first look at the convolution with a one-dimensional input such as a time-series $\mathbf{x} = \{x_t\}_{t=0}^{N-1}$. The goal of the first convolutional layer is to convolve each filter w_h^1 for $h = 1, \dots, M_1$ with the one-dimensional input to produce the following output feature map,

$$\mathbf{a}^1(i, h) = (w_h^1 * x)(i) = \sum_{p=-\infty}^{\infty} w_h^1(p)x(i-p) \quad (2.5)$$

where $w_h^1 \in \mathbb{R}^{1 \times k \times 1}$ and $\mathbf{a}^1 \in \mathbb{R}^{1 \times N-k+1 \times M_1}$, i is the feature map index at the second dimension ($i = 1, \dots, N - k + 1$) and h is the feature map index at the third dimension ($h = 1, \dots, M_1$). Having the dimension of the input channel set to 1, accordingly the dimension of the filter or weight matrix is also one, as shown in Figure 2.1.a.

Now, let’s consider a 3 – D input such as an image $x \in \mathbb{R}^{H \times W \times C}$ with C channels and spatial size $H \times W$. The convolution layer filter will have a weight $w_h^1 \in \mathbb{R}^{(2K+1) \times (2K+1) \times C}$. By convolving the filter w_h^1 for ($h = 1, \dots, M_1$) to x , we

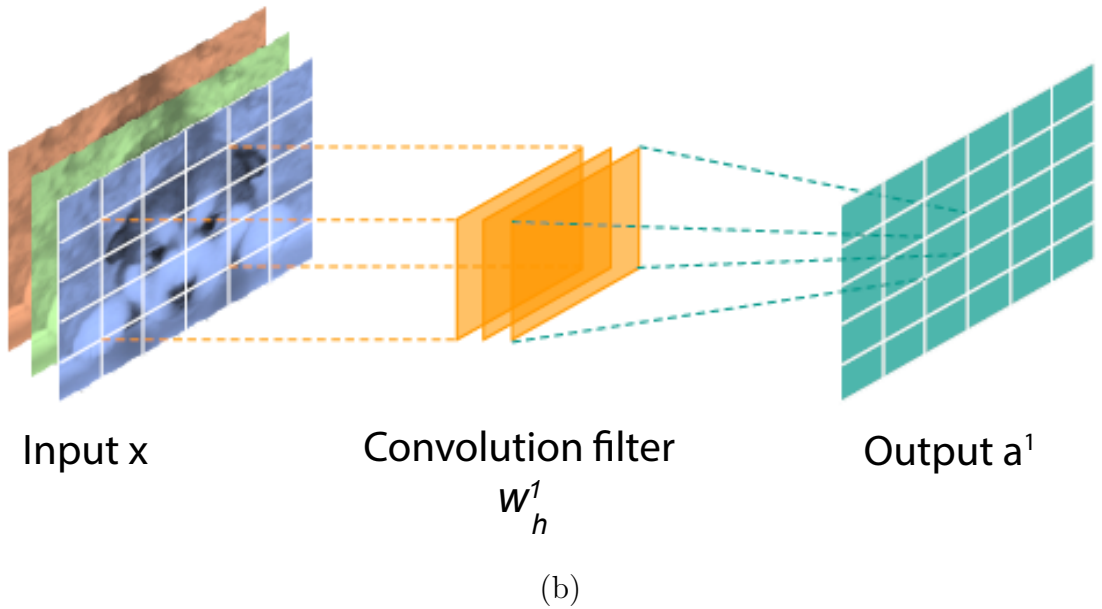
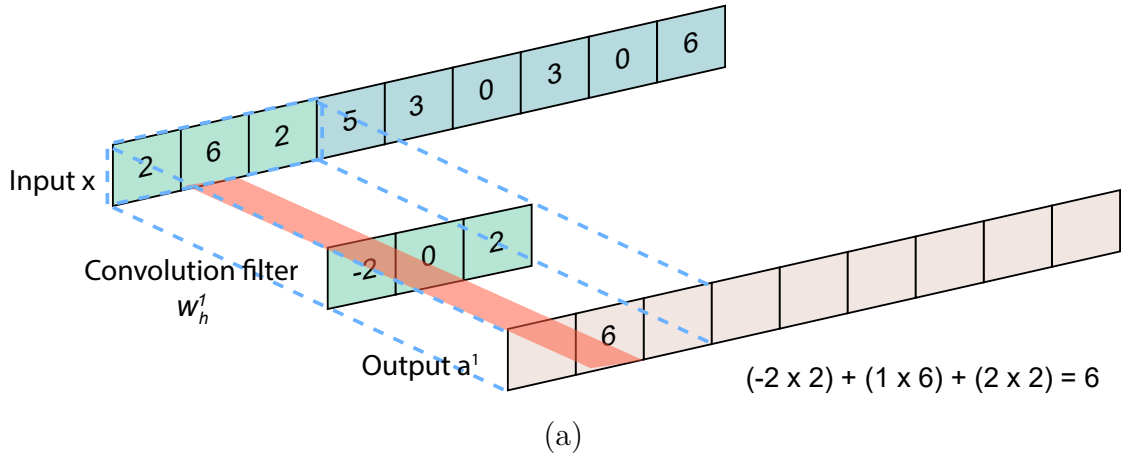


Figure 2.1: Illustration of a convolutional units with (a) a 1D input and (b) a 3D input.

obtain the output feature map below,

$$a^1(i, j, h) = (w_h^1 * x)(i, j) = \sum_{c=1}^C \sum_{p=-\infty}^{\infty} \sum_{q=-\infty}^{\infty} w_h^1(p, q, c)x(i + p, j + q, c) \quad (2.6)$$

For the next layers $l = 2, \dots, L$, for a convolution at the l th layer, we will have the following data:

- the output filter map from the previous convolution has the size of $N_{l-1} \times N_{l-1} \times M_{l-1}$, where $N_{l-1} = N_{l-2} - k + 1$, where $k = 2K + 1$,
- the input feature map is $f^{l-1} \in \mathbb{R}_{l-1}^N \times N_{l-1} \times M_{l-1}$,

- the convolution is done with a set of M_l filters $w_h^l \in \mathbb{R}^{k \times k \times M_{l-1}}$, $h = 1, \dots, M_l$,
- the obtained feature map $a^l \in \mathbb{R}^{N_l \times N_l \times M_l}$ is given by,

$$a^l(i, j, h) = (w_h^l * f^{l-1})(i, j) = \sum_{c=1}^C \sum_{p=-\infty}^{\infty} \sum_{q=-\infty}^{\infty} w_h^1(p, q, c) x(i+p, j+q, c) \quad (2.7)$$

This process is demonstrated in Figure 2.1.b. Each output unit can be seen as the inner-product between the filter and the input inside a local region, which represents their matching similarities. The convolution filter is shared across all the output units on the feature map. Thus it serves as a pattern detector that finds certain pattern on the input. As such, features will be captured in a invariant manner and weights to be learned by the network will be reduced.

2.1.2 Non-linear functions

Performing linear operations consecutively is useless, since they can be replaced by a single linear operation. For example, applying two FC layers as follows,

$$w^2(w^1x + b^1) + b^2 = (w^2w^1)x + (w^2b^1 + b^2) \quad (2.8)$$

is equivalent to a single FC layer with parameters $(w^2w^1; w^2b^1 + b^2)$. In order to enrich the function family that neural networks can represent, non-linear functions are often inserted between every two linear functions.

In neural networks, non-linear functions (denoted as $\phi(\cdot)$), which usually come after the linear function, apply a transformation to each input unit independently. They can serve as activation functions or as probability functions.

Activation functions are located in all layers except the last one. The most common are the *Sigmoid* unit, the *Tanh* unit and the rectified linear unit (ReLU). The sigmoid function is given by,

$$\phi(x) = \sigma(x) = \frac{1}{1 + e^{-x}} \quad (2.9)$$

, with $\sigma(x)$ being a non-linear function within the range $[0, 1]$.

The *Tanh* function has a range of $[-1, 1]$ and can be expressed as,

$$\phi(x) = \tanh(x) = 2\sigma(2x) - 1 \quad (2.10)$$

One drawback of the *Sigmoid* and *Tanh* units is that they take a real-valued number x and “squash” it into range $[0, 1]$ and $[-1, 1]$ respectively. For instance, given the

Sigmoid unit, when the feature map values x are very large (either positive or negative), $\phi(x)$ tends to be large too, pushing the output either to the extreme left (towards 0) or to the extreme right (towards 1) as seen in Figure 2.2.a. This extreme right and left are also referred to as “saturation regions”. Indeed, the gradient or derivative at these regions is very small (almost zero), resulting in a very small gradient of the weights (also called “vanishing gradient”) and thus in a very slow learning during backpropagation.

The ReLU (LeCun et al., 2012), which is a piece-wise linear function, is defined as,

$$\phi(x) = \text{ReLU}(x) = \max(0, x) = \begin{cases} x & \text{for } x \geq 0 \\ 0 & \text{otherwise} \end{cases} \quad (2.11)$$

, resolves the “vanishing gradient” problem and accelerates convergence thanks to its linear, non-saturating form. Moreover, as opposed to *sigmoid* and *tanh* units which involve heavy operations such as exponentials, the *ReLU* involves a simple operation of thresholding a matrix of activations at zero, as shown in Fig2.2.c.

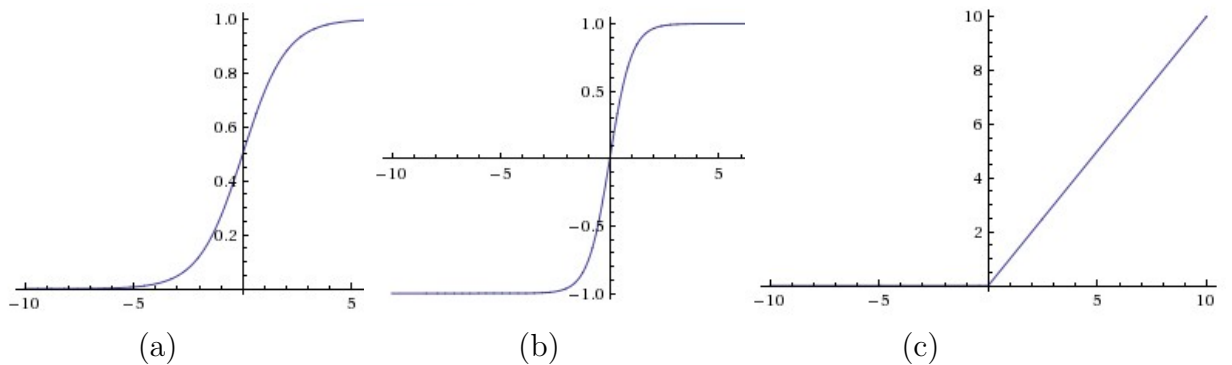


Figure 2.2: Typical activation functions for hidden layers with plots (a), (b) and (c) corresponding to *Sigmoid*, *Tanh* and *ReLU* activation units.

Probability functions, which are located at the last layer, turn network outputs into probability estimates by mapping them to $(0; 1)$. The sigmoid function also serves as a probability function. Another widely used probability function is the softmax function which maps a vector $x \in \mathbb{R}^C$ to a multinomial distribution,

$$\phi(x) = y_i = \text{softmax}(x_i) = \frac{e^{x_i}}{\sum_{c=1}^C e^{x_c}} \quad (2.12)$$

By having $y_i \geq 0$ and $\sum_{c=1}^C y_i = 1$, the goal of the *softmax* function is to cast network outputs into a categorical distribution over predefined classes.

2.2 Optimization

In the previous chapter, we have seen how a deep neural network is formed and what its model hyper-parameters are. And in order to find optimal parameters θ of this neural network, a suitable loss function and optimization method are required, which will be the topic of this chapter. To this matter, we first define the most popular loss functions (chapter 2.2.1). Next, we discuss properties of the used optimization method including the backpropagation principle (chapter 2.2.2) and the gradient descent method (chapter 2.2.3).

2.2.1 Loss functions

To optimize parameters θ of our model f , we need to define a measure that tells us how much a particular choice of model and parameters solves a given task. This formalization comes in the form of a loss function $L(\theta; w)$ (already mentioned in eq. 2.3). This loss function computes a scalar loss value, where the lowest values stand for the best solutions.

The role of the loss function is to compare the prediction \hat{y} of the model against a target value y^i for each input x^i within a set of N training instances. And, from eq. 2.3, we get,

$$L(\theta; \omega) = \sum_{i=1}^N \ell(\mathbf{y}^i, \omega(\mathbf{x}^i, \theta)) \quad (2.13)$$

The formalization of the loss depends on the discriminative learning in hand which can be split into two categories: classification and regression. Classification consists of categorizing data points (inputs) into one of C classes, via a function mapping the data to a vector of C numbers $\hat{y} \in \mathbb{R}^C$ giving the respective probabilities of the data point belonging to each of the classes. In this case, the target or the ground truth value can be expressed as a one-hot vector $y \in \mathbb{R}^C$ over C predefined classes ($y = [y_1, \dots, y_C]$) with $y_k = 1$ if the target is class k and $y_k = 0$ otherwise (all the other dimensions are zero). Meanwhile the goal of regression is to infer one or more scalar values from given data points via a function mapping the data to a scalar or vector. For instance, if we wish to build/train a predictive model for predicting the abalone (animal) age based on several attributes, we can either employ classification by predicting a categorical class $c = 1, 2, \dots, 28$ or use regression by directly predicting the age as a decimal number (such as 12.2).

2.2.1.1 Loss functions for regression.

The most common loss function for regression is called the squared loss (also denoted as the L_2 loss) which is given by,

$$\ell(\mathbf{y}^i, \hat{\mathbf{y}}^i) = \|\mathbf{y}^i - \hat{\mathbf{y}}^i\|_2^2 \quad (2.14)$$

2.2.1.2 Loss functions for binary classification.

One common loss function is the cross-entropy loss which is defined as,

$$\ell(\mathbf{y}^i, \hat{\mathbf{y}}^i) = -\mathbf{y}^i \log(\hat{\mathbf{y}}^i) - (1 - \mathbf{y}^i) \log(1 - \hat{\mathbf{y}}^i) \quad (2.15)$$

2.2.1.3 Loss functions for multi-class classification.

Several loss functions for multi-class classification exist. The most famous ones are listed in Table 2.1. Given an instance i , knowing that the corresponding true label \mathbf{y}^i is a one-hot vector, then y_k^i the true label at a certain node k can be either 0 or 1.

Table 2.1: List of loss functions for classification. \mathbf{y}^i and $\hat{\mathbf{y}}^i$ are the true label and network output vectors respectively. \cdot_k denotes the k th dimension (element) of a given vector, and $\sigma(\cdot)$ denotes a probability estimate (softmax function or the sigmoid function).

Symbol	Name	Equation $\ell(\mathbf{y}^i, \hat{\mathbf{y}}^i)$
L_2	L2 loss – Squared loss	$\ \mathbf{y}^i - \hat{\mathbf{y}}^i\ _2^2$
$L_2 \circ \sigma$	L2 expectation loss with $\sigma(\cdot) = \textit{sigmoid}$	$\ \mathbf{y}^i - \mathbf{p}^i\ _2^2$ where $\mathbf{p}^i = \sigma(\hat{\mathbf{y}}^i)$
Mshinge	Multi-class structured hinge loss (Crammer-Singer loss)	$\max\{0, (1 + \max_{q \neq t} \hat{y}_q^i - \hat{y}_t^i)\}, \hat{y}_t^i = 1$
$Mshinge_2$	Squared Multi-class structured hinge loss	$(1 + \max_{q \neq t} \hat{y}_q^i - \hat{y}_t^i)^2, \hat{y}_t^i = 1$
$Mshinge_3$	Cubed Multi-class structured hinge loss	$\max\{0, (1 + \max_{q \neq t} \hat{y}_q^i - \hat{y}_t^i)^3\}, \hat{y}_t^i = 1$
$\log \circ \sigma$	Cross entropy loss with $\sigma(\cdot) = \textit{softmax}$	$\sum_k^K y_k^i \log(p_k^i)$ where $p_k^i = \sigma(\hat{y}_k^i) = \frac{e^{\hat{y}_k^i}}{\sum_{j=1}^K e^{\hat{y}_j^i}}$ and K denotes the total number of neurons in the final output layer (i.e., K equals the number of classes)

2.2.2 Backpropagation

As mentioned in eq.2.2, the minimization of the objective function is performed based on the combination of the chosen model and the chosen loss function. We

can distinguish between the following cases:

- A convex function $L(\boldsymbol{\theta}, \omega)$. If the loss function and the model both generate a strictly convex function with respect to $\boldsymbol{\theta}$, then a single global optimum exist with no saddle point. This optimum can be easily computed (by using convex optimization for example).
- A Non-convex and differentiable function $L(\boldsymbol{\theta}, \omega)$. In other cases, the model and the loss function may form a non-convex function with respect to $\boldsymbol{\theta}$. But, if the model ω is differentiable with respect to $\boldsymbol{\theta}$ and the loss ℓ is differentiable with respect to the predicted model output $\hat{\mathbf{y}}$, then the chain rule makes $L(\boldsymbol{\theta}, \omega)$ differentiable with respect to $\boldsymbol{\theta}$. Thus, a local optimum can be found by using a gradient-based method such as gradient descent.
- A non-convex and non-differentiable objective function $L(\boldsymbol{\theta}, \omega)$. In other cases, the function $L(\boldsymbol{\theta}, \omega)$ is non-convex and non-differentiable with respect to $\boldsymbol{\theta}$, which calls for a stochastic search method to solve for $\boldsymbol{\theta}$ (such as evolutionary algorithms).

In our case, due to the stacked non-linearities present within neural networks, $L(\boldsymbol{\theta}, \omega)$ is not a convex function with respect to $\boldsymbol{\theta}$. Nonetheless, the loss function $\ell(\mathbf{y}^i, \hat{\mathbf{y}}^i)$ of each instance i is differentiable with respect to $\hat{\mathbf{y}}^i$, and the other neural network model hyper-parameters (such as the linear and non-linear functions) are also differentiable with respect to $\boldsymbol{\theta}$, making $L(\boldsymbol{\theta})$ differentiable, which make us fall within the second case.

Differentiating the loss function with respect to model parameters gives,

$$\begin{aligned} \frac{\partial L(\boldsymbol{\theta}, \omega)}{\partial \boldsymbol{\theta}} &= \sum_{i=1}^N \frac{\partial \ell(\mathbf{y}^i, \hat{\mathbf{y}}^i)}{\partial \boldsymbol{\theta}} \\ &= \sum_{i=1}^N \frac{\partial \ell(\mathbf{y}^i, \hat{\mathbf{y}}^i)}{\partial \hat{\mathbf{y}}^i} \frac{\partial f(\mathbf{x}^i, \boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \end{aligned} \quad (2.16)$$

Here, $\partial L(\boldsymbol{\theta}, \omega) \partial \boldsymbol{\theta}$ denotes a row vector of all partial derivatives of the scalar loss with respect to the different model parameters (a gradient). $\frac{\partial \ell(\mathbf{y}^i, \hat{\mathbf{y}}^i)}{\partial \hat{\mathbf{y}}^i}$ is the gradient with respect to the predicted network outputs. $\frac{\partial f}{\partial \boldsymbol{\theta}}$ denotes a matrix of all partial derivatives of the different network outputs with respect to model parameters (a Jacobian matrix). To further detail the backpropagation process, let's consider a neural network whose computational graph is depicted in Fig.2.3. We set $\hat{\mathbf{y}}^i = f(\mathbf{x}^i, \boldsymbol{\theta}) = \sigma_n(\mathbf{a}_n)$ and $\mathbf{a}_n = g(\mathbf{x}^i, \boldsymbol{\theta})$.

For the first term of the equation 2.16, partial derivatives of the loss with respect to network outputs are easily computed. For instance,

- In the case regression, the Mean Squared Error has the following partial derivative:

$$\frac{\partial \ell(\mathbf{y}^i, \hat{\mathbf{y}}^i)}{\partial \hat{\mathbf{y}}^i} = 2(\mathbf{y}^i - \hat{\mathbf{y}}^i) \quad (2.17)$$

- In the case of classification, the partial derivative of the Cross-Entropy loss is defined as,

$$\frac{\partial \ell(\mathbf{y}^i, \hat{\mathbf{y}}^i)}{\partial \hat{\mathbf{y}}^i} = (\mathbf{y}^i - \hat{\mathbf{y}}^i)^T \quad (2.18)$$

To compute the second term i.e., the Jacobian of $f(\mathbf{x}^i, \boldsymbol{\theta})$ with respect to $\boldsymbol{\theta}$, we employ the chain rule (see Fig.2.3). As discussed in the previous chapter, a deep neural network is a stack of linear and non-linear layers i.e., a parametrized linear transformation $\mathbf{a}_j = h_j(\mathbf{z}_{j-1}, \boldsymbol{\theta}_j)$ and an activation function $\mathbf{z}_j = \sigma_j(\mathbf{a}_j)$ for a given hidden layer j . Assuming that $\mathbf{x}^i = \mathbf{z}_0$ (as shown in Fig.2.3), $f(\mathbf{x}^i, \boldsymbol{\theta})$ can be written as,

$$\begin{aligned} f(\mathbf{x}^i, \boldsymbol{\theta}) &= h_n(\sigma_{n-1}(h_{n-1}(\cdots (\sigma_1(h_1(\mathbf{x}^i, \boldsymbol{\theta}_1)) \cdots), \boldsymbol{\theta}_{n-1})), \boldsymbol{\theta}_n) \\ &= (h_n \circ \sigma_{n-1} \circ h_{n-1} \circ \cdots \circ \sigma_1 \circ h_1)(\mathbf{x}^i, \boldsymbol{\theta}_1, \cdots, \boldsymbol{\theta}_{n-1}, \boldsymbol{\theta}_n) \end{aligned} \quad (2.19)$$

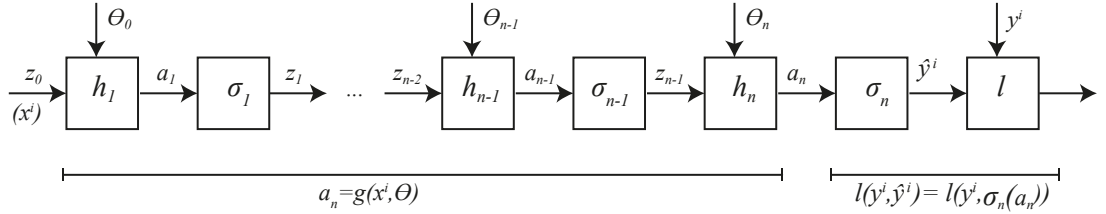


Figure 2.3: Computational graph of the minimization of the objective function $L(\mathbf{y}^i, \hat{\mathbf{y}}^i)$, divided into two components: $a_n = g(\mathbf{x}^i, \boldsymbol{\theta})$ and $l(\mathbf{y}^i, \hat{\mathbf{y}}^i)$.

First, let's consider the Jacobian of $f(\mathbf{x}^i, \boldsymbol{\theta})$ with respect to the parameters $\boldsymbol{\theta}_n$ located within the last hidden layer n ,

$$\begin{aligned} \frac{\partial f(\mathbf{x}^i, \boldsymbol{\theta})}{\partial \boldsymbol{\theta}_n} &= \frac{\partial \hat{\mathbf{y}}^i}{\partial \boldsymbol{\theta}_n} \\ &= \frac{\partial \hat{\mathbf{y}}^i}{\partial \mathbf{a}_n} \frac{\partial \mathbf{a}_n}{\partial \boldsymbol{\theta}_n} \\ &= \frac{\partial \sigma_n(\mathbf{a}_n)}{\partial \mathbf{a}_n} \frac{\partial h_n(\mathbf{z}_{n-1}, \boldsymbol{\theta}_n)}{\partial \boldsymbol{\theta}_n} \end{aligned} \quad (2.20)$$

Knowing from the graph 2.3 that \mathbf{z}_{n-1} is not a function of $\boldsymbol{\theta}_n$, this implies that the Jacobian $\frac{\partial h_n(\mathbf{z}_{n-1}, \boldsymbol{\theta}_n)}{\partial \boldsymbol{\theta}_n}$. Therefore, the gradient of the loss $\ell(\mathbf{y}^i, \hat{\mathbf{y}}^i)$ with respect to

θ_n can be written as,

$$\begin{aligned} \frac{\partial \ell(\mathbf{y}^i, \hat{\mathbf{y}}^i)}{\partial \theta_n} &= \delta_n \frac{\partial h_n(\mathbf{z}_{n-1}, \theta_n)}{\partial \theta_n} \\ \delta_n &= \frac{\partial \ell(\mathbf{y}^i, \hat{\mathbf{y}}^i)}{\partial \mathbf{a}_n} = \frac{\partial \ell(\mathbf{y}^i, \hat{\mathbf{y}}^i)}{\partial \hat{\mathbf{y}}^i} \frac{\partial \sigma_n(\mathbf{a}_n)}{\partial \mathbf{a}_n} \end{aligned} \quad (2.21)$$

Now, let's compute the Jacobian of $f(\mathbf{x}^i, \theta)$ with respect to parameters of the previous hidden layer θ_{n-1} ,

$$\begin{aligned} \frac{\partial f(\mathbf{x}^i, \theta)}{\partial \theta_{n-1}} &= \frac{\partial \hat{\mathbf{y}}^i}{\partial \theta_{n-1}} \\ &= \frac{\partial \hat{\mathbf{y}}^i}{\partial \mathbf{a}_n} \frac{\partial \mathbf{a}_n}{\partial \mathbf{z}_{n-1}} \frac{\partial \mathbf{z}_{n-1}}{\partial \mathbf{a}_{n-1}} \frac{\partial \mathbf{a}_{n-1}}{\partial \theta_{n-1}} \\ &= \frac{\partial \sigma_n(\mathbf{a}_n)}{\partial \mathbf{a}_n} \frac{\partial h_n(\mathbf{z}_{n-1}, \theta_n)}{\partial \mathbf{z}_{n-1}} \frac{\partial \sigma_{n-1}(\mathbf{a}_{n-1})}{\partial \mathbf{a}_{n-1}} \frac{\partial h_{n-1}(\mathbf{z}_{n-2}, \theta_{n-1})}{\partial \theta_{n-1}} \end{aligned} \quad (2.22)$$

, which means that this Jacobian depends on the input \mathbf{z}_{n-2} , the Jacobian of the activation function $\sigma_{n-1}(\mathbf{a}_{n-1})$ with respect to its input \mathbf{a}_{n-1} , Jacobian of the last layer's linear transformation h_n with respect to its inputs, and the Jacobian of the activation function $\sigma_n(\mathbf{a}_n)$ with respect to its input \mathbf{a}_n . Accordingly, the gradient of the loss $\ell(\mathbf{y}^i, \hat{\mathbf{y}}^i)$ with respect to θ_n can be expressed as,

$$\begin{aligned} \frac{\partial \ell(\mathbf{y}^i, \hat{\mathbf{y}}^i)}{\partial \theta_{n-1}} &= \delta_{n-1} \frac{\partial h_{n-1}(\mathbf{z}_{n-2}, \theta_{n-1})}{\partial \theta_{n-1}}, \\ \delta_{n-1} &= \frac{\partial \ell(\mathbf{y}^i, \hat{\mathbf{y}}^i)}{\partial \hat{\mathbf{y}}^i} \frac{\partial \sigma_n(\mathbf{a}_n)}{\partial \mathbf{a}_n} \frac{\partial h_n(\mathbf{z}_{n-1}, \theta_n)}{\partial \mathbf{z}_{n-1}} \frac{\partial \sigma_{n-1}(\mathbf{a}_{n-1})}{\partial \mathbf{a}_{n-1}} = \delta_n \frac{\partial h_n(\mathbf{z}_{n-1}, \theta_n)}{\partial \mathbf{z}_{n-1}} \frac{\partial \sigma_{n-1}(\mathbf{a}_{n-1})}{\partial \mathbf{a}_{n-1}} \end{aligned} \quad (2.23)$$

So, computing $\frac{\partial \ell(\mathbf{y}^i, \hat{\mathbf{y}}^i)}{\partial \theta_{n-1}}$ requires δ_{n-1} which in turn requires the already computed δ_n as well as computing the Jacobian $h_n(\mathbf{z}_{n-1}, \theta_n)$ with respect to \mathbf{z}_{n-1} and the partial derivative of $\sigma_{n-1}(\mathbf{a}_{n-1})$ with respect to \mathbf{a}_{n-1} .

Finally, let's compute the Jacobian of f with respect to parameters of the first layer (layer 1),

$$\begin{aligned} \frac{\partial f(\mathbf{x}^i, \theta)}{\partial \theta_1} &= \frac{\partial \hat{\mathbf{y}}^i}{\partial \theta_1} \\ &= \frac{\partial \hat{\mathbf{y}}^i}{\partial \mathbf{a}_1} \dots \frac{\partial \mathbf{a}_1}{\partial \theta_1} \\ &= \frac{\partial \hat{\mathbf{y}}^i}{\partial \mathbf{a}_n} \frac{\partial \mathbf{a}_n}{\partial \mathbf{z}_{n-1}} \frac{\partial \mathbf{z}_{n-1}}{\partial \mathbf{a}_{n-1}} \frac{\partial \mathbf{a}_{n-1}}{\partial \theta_{n-1}} \dots \frac{\partial \mathbf{z}_1}{\partial \mathbf{a}_1} \frac{\partial \mathbf{a}_1}{\partial \theta_1} \\ &= \frac{\partial \sigma_n(\mathbf{a}_n)}{\partial \mathbf{a}_n} \dots \frac{\partial \sigma_1(\mathbf{a}_1)}{\partial \mathbf{a}_1} \frac{\partial h_1(\mathbf{x}^i \theta_1)}{\partial \theta_1} \end{aligned} \quad (2.24)$$

Thus, this Jacobian with respect to parameters $\boldsymbol{\theta}_1$ requires the input x , and the Jacobian $f(\mathbf{x}^i, \boldsymbol{\theta})$ with respect to \mathbf{a}_1 , which in turn is the product of all Jacobians $\frac{\partial \sigma_j(\mathbf{a}_j)}{\partial \mathbf{a}_j}$ and $\frac{\partial h_j(\mathbf{z}_{j-1}, \boldsymbol{\theta}_j)}{\partial \mathbf{z}_{j-1}}$ for $j = 1, \dots, n$. Therefore, the gradient of the loss $\ell(\mathbf{y}^i, \hat{\mathbf{y}}^i)$ with respect to $\boldsymbol{\theta}_1$ is given by,

$$\begin{aligned} \frac{\partial \ell(\mathbf{y}^i, \hat{\mathbf{y}}^i)}{\partial \boldsymbol{\theta}_{n-1}} &= \delta_1 \frac{\partial h_1(\mathbf{x}^i, \boldsymbol{\theta}_1)}{\partial \boldsymbol{\theta}_1}, \\ \delta_1 &= \delta_2 \frac{\partial h_2(\mathbf{z}_1, \boldsymbol{\theta}_2)}{\partial \mathbf{z}_1} \frac{\partial \sigma_1(\mathbf{a}_1)}{\partial \mathbf{a}_1} \end{aligned} \quad (2.25)$$

By looking at equations 2.21, 2.23 and 2.25 and comparing them, we can see that the Jacobians with respect to the model parameters of the different layers have some features in common. This is the key to devise an algorithm that computes all of them at once. Indeed, we observe that the Jacobian of the loss with respect to $\boldsymbol{\theta}_j$ depends on the layer's input \mathbf{z}_{j-1} (since it depends on $\frac{\partial h_j(\mathbf{z}_{j-1}, \boldsymbol{\theta}_j)}{\partial \boldsymbol{\theta}_j}$ and on the Jacobian of the loss with respect to \mathbf{a}_j which is equivalent to the error δ_j). We also notice that the error δ_j at layer j is dependent on the error δ_{j+1} at the next layer. Thus, errors flow backward, from the last layer to the first layer.

Putting everything together, we obtain the algorithm below corresponding to the error backpropagation algorithm which has been introduced by (Dreyfus, 1962), implemented by (Linnainmaa, 1970), and popularized for training neural networks by (Rumelhart et al., 1986).

1. Given $\mathbf{x}^i = \mathbf{z}_0$, compute $\mathbf{a}_1 = h_1(\mathbf{z}_0, \boldsymbol{\theta})$ and $\mathbf{z}_1 = \sigma_1(\mathbf{a}_1)$. Proceed in this way up until \mathbf{a}_n and \mathbf{z}_n . This process is referred to as the forward pass, since it propagates data through the network from input to output.
2. Compute the gradient $\frac{\partial \ell(\mathbf{y}^i, \hat{\mathbf{y}}^i)}{\partial \hat{\mathbf{y}}^i}$ which is easy to compute, as discussed previously.
3. Use $\frac{\partial \ell(\mathbf{y}^i, \hat{\mathbf{y}}^i)}{\partial \hat{\mathbf{y}}^i}$ to compute δ_n , and compute $\frac{\partial h_n(\mathbf{z}_{n-1}, \boldsymbol{\theta}_n)}{\partial \boldsymbol{\theta}_n}$ (which depends on \mathbf{z}_{n-1}) to find the gradient $\Delta \boldsymbol{\theta}_n = \frac{\partial \ell(\mathbf{y}^i, \hat{\mathbf{y}}^i)}{\partial \boldsymbol{\theta}_n}$.
4. Compute the partial derivative $\frac{\partial \sigma_{n-1}(\mathbf{a}_{n-1})}{\partial \mathbf{a}_{n-1}}$ and the Jacobian $\frac{\partial h_n(\mathbf{z}_{n-1}, \boldsymbol{\theta}_n)}{\partial \mathbf{z}_{n-1}}$ in order to compute the error δ_{n-1} . Then use δ_{n-1} and $\frac{\partial h_{n-1}(\mathbf{z}_{n-2}, \boldsymbol{\theta}_{n-1})}{\partial \boldsymbol{\theta}_{n-1}}$ to compute $\Delta \boldsymbol{\theta}_{n-1}$.
5. Proceed in this way down until $\Delta \boldsymbol{\theta}_1$. This is called the backward pass, since it propagates gradients through the network from output to input.

2.2.3 Gradient descent method

The backpropagation algorithm allows us to compute the gradient of the objective function $L(\boldsymbol{\theta}, \omega)$ with respect to the model parameters $\boldsymbol{\theta}$. To reduce this cost function, we merely have to change each parameter value in the opposite direction of the gradient: A parameter value with positive gradient needs to be reduced, a value with negative gradient needs to be increased. As the dependency of the loss on the parameters is not actually linear, the linear approximation only holds locally at the position the gradient was evaluated. Thus, without further information, we can only modify the parameters by small amounts, and then have to re-evaluate the gradient at the new position. Repeating this, we can move through parameter space until we reach a point of zero gradient (a local extremum or saddle point). In this section, we describe common algorithms used to compute by how much and in which direction to change the parameters in each step. The main goal of these algorithms is to efficiently optimize the cost function without falling into saddle points or poor local minima.

2.2.3.1 Gradient Descent.

Given a nonzero gradient, there are infinitely many directions in parameter space the loss function decreases – for example all those that move each parameter against the sign of the corresponding gradient. Gradient Descent (Cauchy, 1847) chooses the direction the loss function decreases the fastest. As it turns out, this is simply the direction of the negative gradient (illustrated in Figure 2.17, derived in Goodfellow et al., 2016, Eq. 4.3 ff.). The step size is chosen proportional to the gradient magnitude, so the update for a parameter tensor $\theta \in \boldsymbol{\theta}$ becomes

$$\theta \leftarrow \theta - \alpha \Delta \theta \tag{2.26}$$

where α is referred to as the learning rate. It controls by how much to change parameters in each step, and has to be chosen outside of the optimization procedure – this is referred to as a hyperparameter. If α is too large, gradient descent can overshoot the minimum and it may fail to converge, cause oscillations and even diverge, as illustrated by the function $L(\theta) = \theta^2$ in Fig.2.4. On the other hand, if α is too small, gradient descent can be slow, thus slowing down optimization as shown in Fig.2.4.

2.2.3.2 Mini-Batch Gradient Descent.

For gradient descent, the objective function $L(\cdot)$ is defined as a sum of losses ℓ as given by eq.2.3 such that $N = S$ where S is the number of instances within the

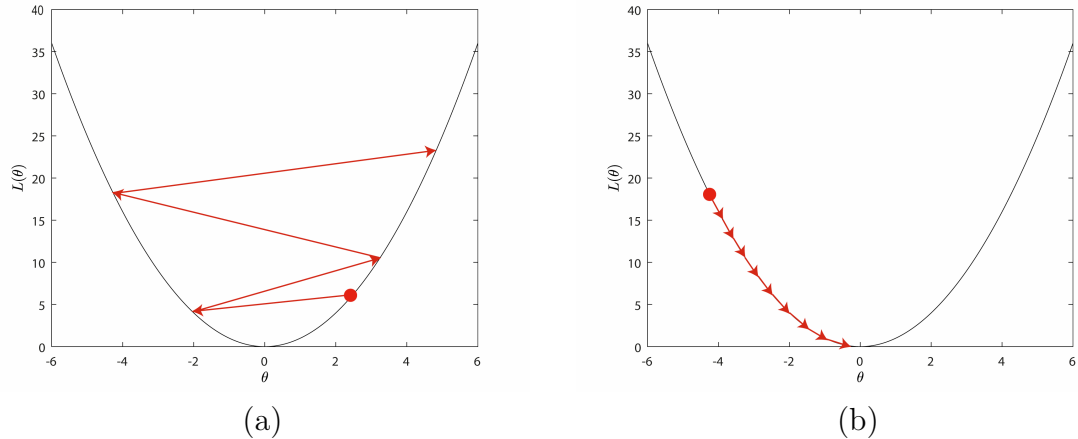


Figure 2.4: Visualization of: (a) a gradient descent with a large learning rate and, (b) a gradient descent with a small learning rate.

training set. In other words, the batch in the gradient descent method corresponds to the whole training set. Accordingly, each update step within backpropagation requires a forward and backward pass of all the training data to evaluate the gradient. So, for large training sets, this process becomes highly expensive. Stochastic Gradient Descent (SGD) alleviates this issue by defining the loss as a sum of penalties for a batch (e.g, N data points chosen randomly for each update step). N is also a hyperparameter to be defined: the larger, the faster the gradient evaluation, but the noisier the gradient estimate. Larger N thus generally requires smaller α to avoid taking a large step in an inaccurate direction. However, larger N can also help optimization escape poor local minima or saddle points (Keskar et al., 2017): Noisy gradients induce exploration of the parameter space around the trajectory that would be followed by Gradient Descent on the full training set.

2.2.3.3 Gradient Descent with Momentum.

The high variance oscillations in SGD makes it hard to reach convergence, so a technique called Momentum was invented which accelerates SGD by navigating along the relevant direction and softens the oscillations in irrelevant directions. Momentum uses the history of gradients to accelerate progress in stable directions, and reduce progress in directions the gradient changed. In other words all it does is adds a fraction “ η ” of the update vector of the past step to the current update vector. Specifically, updates are computed as:

$$\begin{aligned} v_{\theta} &\leftarrow \eta v_{\theta} - \alpha \Delta \theta \\ \theta &\leftarrow \theta + v_{\theta} \end{aligned} \tag{2.27}$$

where v_{θ} is an exponential moving average over the gradients $\Delta\theta$ or, in other words, the gradient that is retained from previous iterations, and η is the coefficient of Momentum which is the percentage of the gradient retained every iteration. This term is responsible for increasing updates for dimensions whose gradients point in the same directions and reduces updates for dimensions whose gradients change directions. This means it does parameter updates only for relevant examples. This reduces the unnecessary parameter updates which leads to faster and stable convergence and reduced oscillations. Here the momentum is the same as the momentum in classical physics: as we throw a ball down a hill it gathers momentum and its velocity keeps on increasing.

2.2.3.4 Nesterov Accelerated Gradient.

However, a ball that rolls down a hill, blindly following the slope, is highly unsatisfactory. We'd like to have a smarter ball, a ball that has a notion of where it is going so that it knows to slow down before the hill slopes up again. What actually happens is that as we reach the minima i.e the lowest point on the curve ,the momentum is pretty high and it do non't knows to slow down at that point due to the high momentum which could cause it to miss the minima entirely and continue moving up.

The Nestrov Accelerated Gradient (NAG) strategy, which was published by first by Nesterov (Nesterov, 1983) then reformulated by Bengio et al. (Bengio et al., 2013), solved this problem of momentum. In this method, the author suggested we first make a big jump based on our previous momentum then calculate the Gradient and them make a correction which results in an parameter update. Now this anticipatory update prevents us to go too fast and not miss the minima and makes it more responsive to changes.

In NAG, we know that we will use our momentum term ηv_{θ} to move the parameters θ . Computing $\theta + \eta v_{\theta}$ thus gives us an approximation of the “looked-ahead” position or the next position of the parameters which gives us a rough idea where our parameters are going to be (located at the tip of the green arrow in Fig.2.5). We can now effectively look ahead by calculating the gradient not with respect to our current parameters θ but with respect to the approximate future position of our parameters: Accordingly, NAG is computed as follows,

$$\begin{aligned} v_{\theta} &\leftarrow \eta v_{\theta} - \alpha \Delta\theta \\ \theta &\leftarrow \theta + \eta v_{\theta} - \alpha \Delta\theta \end{aligned} \tag{2.28}$$

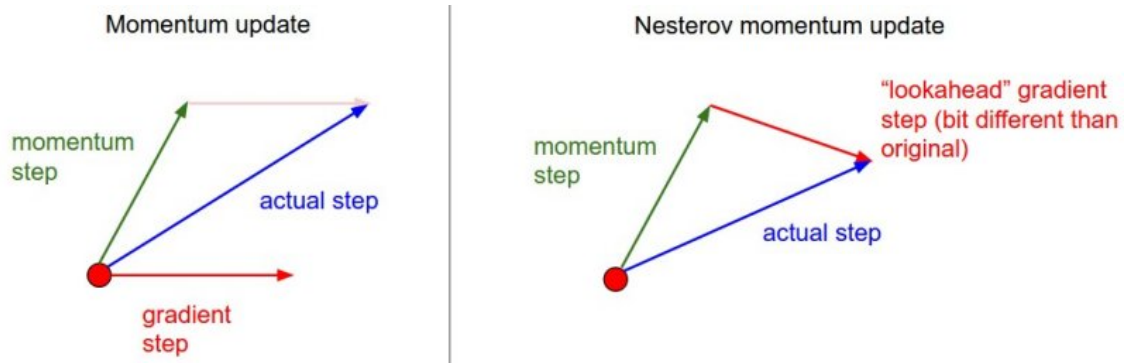


Figure 2.5: Nesterov momentum. Instead of evaluating the gradient at the current position (red circle), we know that our momentum is about to carry us to the tip of the green arrow. With Nesterov momentum we therefore instead evaluate the gradient at this “looked-ahead” position.

2.2.3.5 RMSPro.

RMSProp also tries to dampen the oscillations, but in a different way than momentum. RMS prop also takes away the need to adjust learning rate, and does it automatically. More so, RMSProp chses a different learning rate for each parameter.

In RMS prop, each update is done according to the equations described below. This update is done separately for each parameter.

$$\begin{aligned}
 v &\leftarrow \rho v + (1 - \rho)(\Delta\theta)^2 \\
 \theta &\leftarrow \theta - \frac{\alpha}{\sqrt{v + \epsilon}} \Delta\theta
 \end{aligned}
 \tag{2.29}$$

where α is the learning rate, $\Delta\theta$ is the gradient with respect to to the single parameter θ , v is the exponential average of squares of gradients (previous terms), and ρ is a hyperparameter controlling the weighing of this average with respect to the recent gradient. The reason why we use exponential average is because as we saw, in the momentum example, it helps us weigh the more recent gradient updates more than the less recent ones.

2.2.3.6 ADAM.

So far, we have seen RMSProp and Momentum take contrasting approaches. While momentum accelerates our search in direction of minima, RMSProp impedes our search in direction of oscillations. Adam or Adaptive Moment Optimization algorithms combines the heuristics of both Momentum and RMSProp. Here are the

update equations.

$$\begin{aligned} m &\leftarrow \beta_1 m + (1 - \beta_1) \Delta \theta \\ v &\leftarrow \beta_2 m + (1 - \beta_2) (\Delta \theta)^2 \\ \theta &\leftarrow \theta - \frac{\alpha}{\sqrt{v} + \epsilon} m \end{aligned} \tag{2.30}$$

where hyperparameters β_1 , β_2 and ϵ are generally kept around 0.9, 0.99 and 1e-10 respectively.

We have now discussed the most common optimization schemes in contemporary deep learning. All of these rely on first-order (gradient) information only, using sophisticated heuristics to determine how far to follow the negative gradient in each step. This begs the question why algorithms do not employ second-order (curvature) information – since low curvature implies a stable gradient, this could directly be used to determine an appropriate step size. In fact, around 2010, considerable effort was spent on efficiently using second-order derivatives in optimizing deep neural networks, e.g., Hessian-Free Learning (Martens, 2010) or Krylov Subspace Descent (Vinyals and Povey, 2012). However, this line of research became less important after progress in a previously neglected part of optimization we will discuss in the following chapter.

2.3 Types of feed-forward neural networks / Feed-forward Neural Network Architectures

2.3.1 Single-Layer Perceptron.

It is the simplest function of the family of artificial neural networks. It consists of one hidden layer, i.e., one fully-connected layer followed by a non-linear function and can be expressed as,

$$\hat{y} = f(\mathbf{x}, \boldsymbol{\theta}, \phi) = \phi\left(b + \sum_j w_j x_j\right) = \phi(b + \mathbf{w}^T \mathbf{x}) \tag{2.31}$$

where \mathbf{x} is the input vector of a particular instance i , \hat{y} is the network output which is a scalar, $\phi(\cdot)$ is a defined transfer function (non-linear function), and $\boldsymbol{\theta} = (b, \mathbf{w})$ are the tunable parameters composed of the weight vector \mathbf{w} the bias term b . The process consists of mapping \mathbf{x} to a scalar \hat{y} by computing a weighted sum of the input values x_j , expressed as a dot product $\mathbf{w}^T \mathbf{x}$, adding a scalar offset b and passing it through the transfer function $\phi(\cdot)$. This process, shown in Figure 2.6.a, is often denoted as a neuron or unit in artificial neural since it resembles a biological neuron

(Figure 2.6.b): connections or weights \mathbf{w} in artificial neural networks are represented by the dendrites in biological neurons; the function $\phi(b + \mathbf{w}^T \mathbf{x})$ (also referred to as nucleus using biological terms) accumulates non-linearly all incoming excitatory and inhibitory inputs (or signals), and fires with an output y which corresponds to the strength along the axon. To obtain a mapping from a vector \mathbf{x} to a vector $\hat{\mathbf{y}}$,

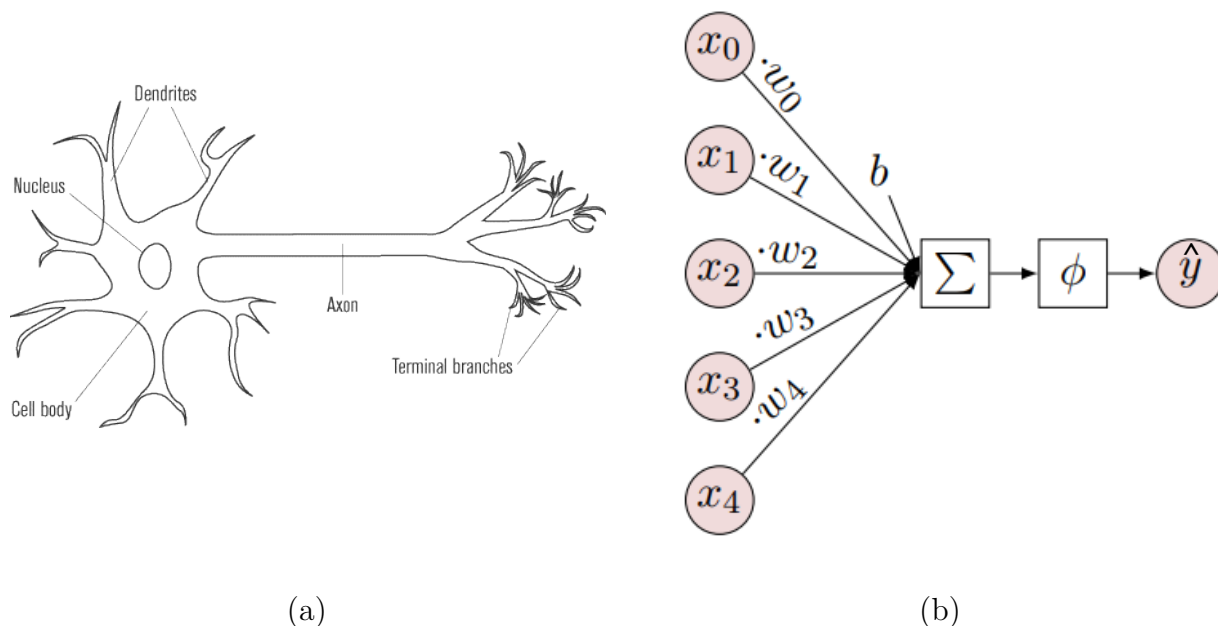


Figure 2.6: Visualization of: (a) a biological neuron and, (b) an artificial one (single-layer perceptron) $\phi(b + W^T \mathbf{x})$

we simply use multiple units of the same form, each with a separate bias and set of weights, as depicted in Figure 2.7. Since all these units share the same inputs \mathbf{x} , we can express the vector of weighted sums as a matrix product $W^T \mathbf{x}$. Using a vector addition for the biases, we arrive at,

$$\hat{\mathbf{y}} = f(\mathbf{x}^i \boldsymbol{\theta}, \phi) = \phi(\mathbf{b} + \mathbf{W}^T \mathbf{x}) \quad (2.32)$$

This model has some disadvantages: in this model, the layer's inputs are only linearly combined, and hence even adding a non-linearity to these combination(s) of inputs cannot produce the non-linearity necessary for finding all decision boundaries that separate between classes. Nonetheless, we can obtain a more complex model by stacking one hidden layer (composed of a fully-connected layer followed by a non-linear function) on top of the other. The first layer (Layer 1) can be thought of as encoding a set of features learned from the inputs, while the second layer (Layer 2) encodes a different (higher-level, more abstracted) set of features learned from the outputs of Layer 1, and so on. The multiple layers add levels of abstraction

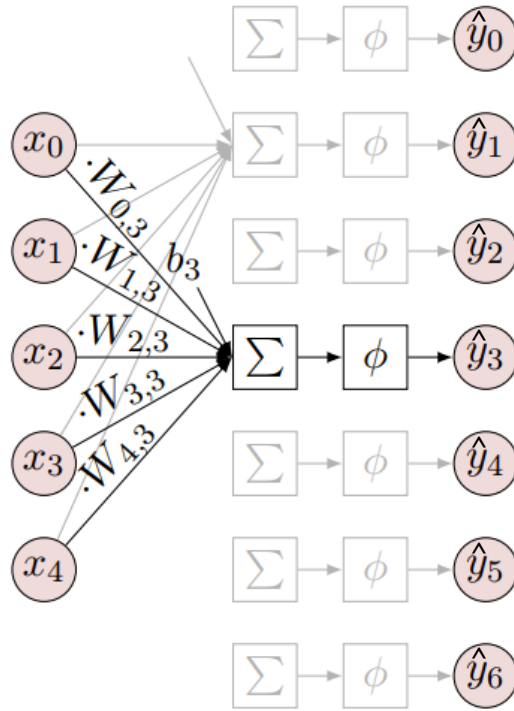


Figure 2.7: Visualization of $\phi(\mathbf{b} + \mathbf{W}^T \mathbf{x})$

that cannot be as simply contained within a single layer of the same number of parameters. This model is known as “Multi-Layer Perceptrons”.

2.3.2 Multi-layer Perceptron (MLP).

An MLP consists of multiple hidden layers composed of one fully-connected layer and a non-linear function each. In a MLP, each neuron from layer i is connected to layer $i + 1$, and the computation boils down to matrix-vector multiplication. Stacking two hidden layers results in,

$$\begin{aligned} \hat{\mathbf{y}} &= f(f(\mathbf{x}^i \boldsymbol{\theta}_1, \phi_1), \boldsymbol{\theta}_2, \phi_2) \\ &= \phi_2(\mathbf{b}_2 + \mathbf{W}_2^T \phi_1(\mathbf{b}_1 + \mathbf{W}_1^T \mathbf{x})) \end{aligned} \quad (2.33)$$

where each function f represents a hidden layer, and can be visualized through Figure 2.8. The units are not fully interconnected, but form three groups that are connected in sequence. More specifically, the input vector \mathbf{x} is referred to as the input, the output vector $\hat{\mathbf{y}}$ is referred to as the output, and functions f between the input and output are represented by the first and second hidden layer respectively.

Usually, the size of the output is usually fixed determined by the task, whereas the number of hidden layers and/or their individual sizes can be increased (by stacking more functions f and/or adapting the sizes of the weight matrices and bias

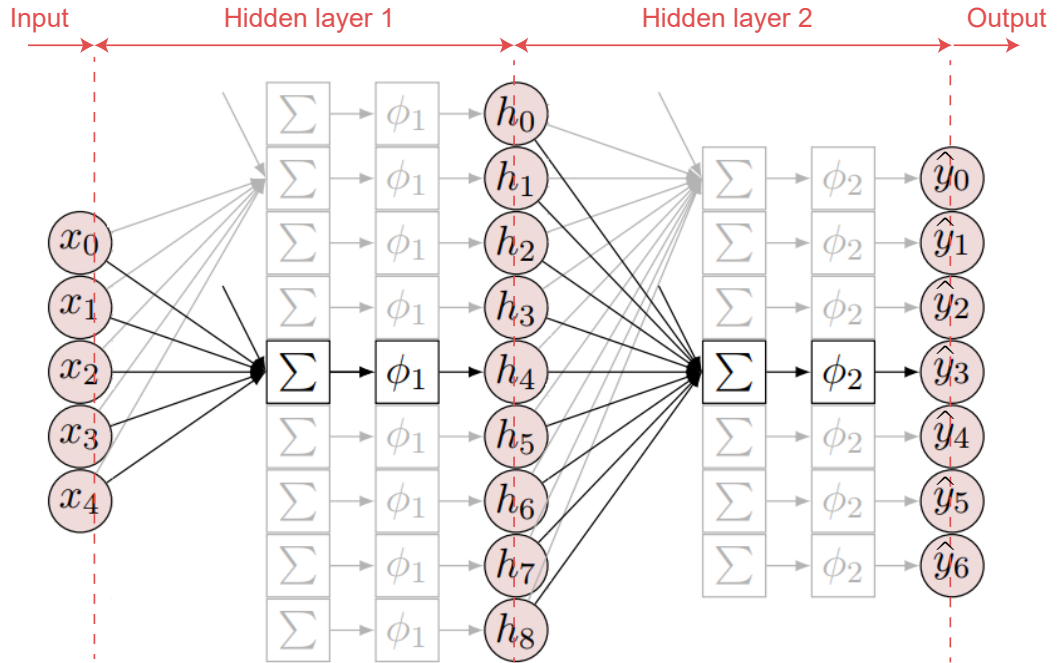


Figure 2.8: Visualization of $\phi_2(\mathbf{b}_2 + \mathbf{W}_2^T \phi_1(\mathbf{b}_1 + \mathbf{W}_1^T \mathbf{x}))$

vectors) or adapted depending on the complexity of features that should be detected within the task. The transfer function is set to one of the non-linear functions mentioned in section 2.1.2.

2.3.3 Convolutional Neural Networks (ConvNet).

The Convolutional Neural Network (ConvNet) takes advantage of using convolutional layers which offer spatial locality of the input signal (such as images) and share the weights in space, making it invariant to translations of the input. As mentioned in the previous chapter, the ConvNet architecture has a good data locality since the kernel can be reused across different places, making the ConvNet computation bounded.

The design space for ConvNets is similar to that of MLPs as we set the number of hidden layers as well as the size of each one of them and the activation function following it. Nonetheless, as for ConvNets, we do not only have to fix the number of convolutional units or filters for a layer, but also the size of the filter itself (e.g., height and width of the filter) which is responsible of determining the size of the output. For instance, a larger filter size increases the amount of spatial context per output pixel, increases the number of learnable parameters, and decreases the size of the output. Let's not that a particular case of convolution is when the filter size is set equal to the input size, which results in having a FC layer.

Nonetheless, stacking a couple of convolutional layers in a ConvNet makes the

number of parameters (or weights) grow quickly. Indeed, the first convolutional layer might process an input of only three channels or less, whereas the next layers have to process as many channels as the depth of the previous convolutional layers' filters. Furthermore, the ConvNet receptive fields – defined as the spatial extent (the width and height) of the convolutional layer filters – get larger throughout the network. To resolve the growing data issue (growing size of parameters), it is common to add a pooling layer (also known as subsampling or down-sampling) in-between successive convolutional layers in a ConvNet. The idea behind the pooling layer is to diminish the size of the output feature maps and thus lower the number of parameters and within the network, thereby resulting in less overfitting. The Pooling Layer operates independently on every depth slice of the input (i.e., operates on the channel of each input separately) and re-sizes it spatially, as shown in Figure 2.9. The most common forms of pooling are max-pooling or mean-pooling. For instance, a pooling layer with filters of size 2×2 applied with a stride of 2 down-samples every depth slice in the input by 2 along both width and height. In this case, the max-pooling layer will compute the max over 4 numbers (little 2×2 region in some depth slice) as the output. The depth dimension remains unchanged. From this example, we deduce that the amount of parameters is reduced by 75%.

In general, the pooling layer can be defined as an operation which requires the following,

- the input volume of size $W_1 \times H_1 \times Depth_1$,
- the following hyper-parameters:
 - spatial extent F ,
 - the stride S of the pooling layer.

and which produces an output with a volume of size $W_2 \times H_2 \times Depth_2$ where,

- $W_2 = (W_1 - F)/S + 1$
- $H_2 = (H_1 - F)/S + 1$
- $Depth_2 = Depth_1$

The main advantage of the pooling layer is to introduce zero parameters since it computes a fixed function of the input. If the input size is not evenly divisible by the pooling factors, part of the input will be ignored.

In general, no standard rule exists for constructing a ConvNet architecture. However, most famous ConvNets (Chatfield et al., 2014; Simonyan and Zisserman, 2015) agree on the following architecture: several layers composed of convolution,

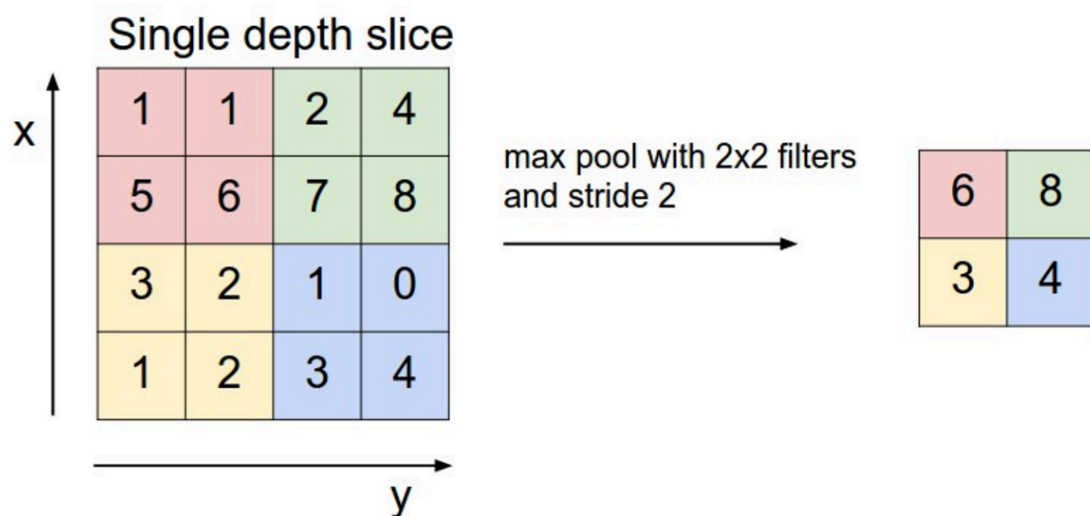


Figure 2.9: Visualization of max-pooling

ReLU and MaxPooling each, followed by connected layers with ReLUs. Indeed, a ConvNet will end in one or more fully-connected layers (FCs) that integrate information across all spatial locations and finally produce the prediction. This way the model is still oblivious to the ordering of target vector components, like an MLP, which is appropriate for most classification and regression tasks. To summarize, a ConvNet is composed of a stack of convolutional, activation and pooling layers, followed by one or multiple fully connected layers. Figure 2.10 shows a schematic architecture of a single layer of each of these types. It depicts feature maps as 3D volumes, with the number of channels forming the third dimension, and dense layers as columns. Labels denote the number and size of feature maps (top) and layer types (bottom). I will use this kind of visualization throughout the thesis.

2.3.3.1 Regularization

One common regularizer is the use of dropout layers. Dropout regularization technique in deep network (its weights cannot be updated, nor affect the learning of the other network nodes). With dropout, the learned weights of the nodes become somewhat more insensitive to the weights of the other nodes and learn to decide somewhat more by their own (and less dependent on the other nodes they're connected to). In other words, dropout removes individual activations at random while training the network, which makes the model more robust to the loss of individual pieces of evidence, and less likely to rely on particular idiosyncrasies of the training data.

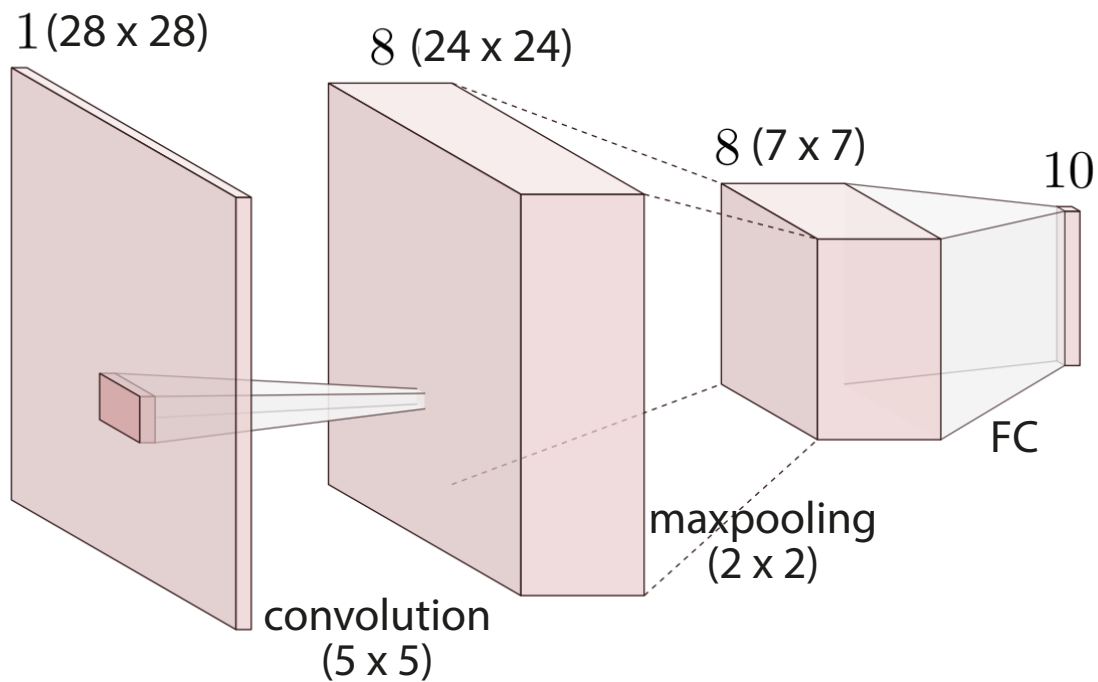


Figure 2.10: A simple ConvNet architecture with one set of convolutional, maxpooling and fully connected layers

Bibliography

Artem Babenko, Anton Slesarev, Alexandr Chigorin, and Victor Lempitsky. Neural codes for image retrieval. In *Lecture Notes in Computer Science*, volume 8689 LNCS, pages 584–599. 2014. ISBN 9783319105895. doi: 10.1007/978-3-319-10590-1_38.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural Machine Translation by Jointly Learning to Align and Translate. arxiv.org, 2014. ISSN 0147-006X. doi: 10.1146/annurev.neuro.26.041002.131047.

Y. Bengio. Learning Deep Architectures for AI. *Foundations and Trends® in Machine Learning*, 2(1):1–127, 2009. ISSN 1935-8237. doi: 10.1561/22000000006.

Yoshua Bengio, Nicolas Boulanger-Lewandowski, and Razvan Pascanu. Advances in optimizing recurrent networks. In *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, pages 8624–8628, 2013. ISBN 9781479903566. doi: 10.1109/ICASSP.2013.6639349.

Ashish Bora, Ajil Jalal, Eric Price, and Alexandros G. Dimakis. Compressed

- Sensing using Generative Models. In *dl.acm.org*, pages 537—546, 2017. ISBN 9781510855144. doi: 10.5829/idosi.wasj.2013.23.04.368.
- Ken Chatfield, Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Return of the Devil in the Details: Delving Deep into Convolutional Nets. In *British Machine Vision Conference*, 2014. ISBN 1-901725-52-9. doi: 10.5244/C.28.6.
- Chenyi Chen. DeepDriving : Learning Affordance for Direct Perception in Autonomous Driving. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 2722–2730, 2015. doi: 10.1109/ICCV.2015.312.
- Qifeng Chen and Vladlen Koltun. Photographic Image Synthesis with Cascaded Refinement Networks. In *Proceedings of the IEEE International Conference on Computer Vision*, volume 2017-October, pages 1520–1529, 2017. ISBN 9781538610329. doi: 10.1109/ICCV.2017.168.
- Ronan Collobert, Jason Weston, Leon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. Natural Language Processing (almost) from Scratch. *Journal of machine learning research*, 12(Aug):2493—2537, 2011. ISSN 1532-4435. doi: 10.1145/2347736.2347755.
- George E. Dahl, Dong Yu, Li Deng, and Alex Acero. Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition. *IEEE Transactions on Audio, Speech and Language Processing*, 20(1):30—42, 2012. ISSN 15587916. doi: 10.1109/TASL.2011.2134090.
- Navneet Dalal and William Triggs. Histograms of Oriented Gradients for Human Detection. In *International Conference on computer vision & Pattern Recognition*, volume 1, pages 886–893, 2005. ISBN 0-7695-2372-2. doi: 10.1109/CVPR.2005.177.
- Li Deng. A Tutorial Survey of Architectures, Algorithms, and Applications for Deep Learning. *APSIPA Transactions on Signal and Information Processing*, 3, 2014.
- Stuart Dreyfus. The numerical solution of variational problems. *Journal of Mathematical Analysis and Applications*, 5(1):30–45, 1962. ISSN 10960813. doi: 10.1016/0022-247X(62)90004-5.
- Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. A Neural Algorithm of Artistic Style. *arXiv preprint arXiv:1508.06576*, aug 2015. ISSN 1935-8237. doi: 10.1561/22000000006.
- Ross Girshick, Jeff Donahue, Trevor Darrell, U C Berkeley, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In

- IEEE Conference on Computer Vision and Pattern Recognition, pages 580–587, 2014. ISBN 978-1-4799-5118-5. doi: 10.1109/CVPR.2014.81.
- Philippe Hamel and Douglas Eck. Learning Features from Music Audio with Deep Belief Networks. In Proc. of the 11th International Society of Music Information Retrieval, pages 339–344, 2010. ISBN 9789039353813.
- Geoffrey E. Hinton. Training Products of Experts by Minimizing Contrastive Divergence. *Neural Computation*, 14(8):1771–1800, aug 2002. ISSN 0899-7667. doi: 10.1162/089976602760128018.
- Geoffrey E. Hinton. A practical guide to training restricted Boltzmann machines. In *Neural networks: Tricks of the Trade*, pages 599–619. 2012.
- Geoffrey E. Hinton, Simon Osindero, and Yee-Whye Teh. A Fast Learning Algorithm for Deep Belief Nets. *Neural Computation*, 18(7):1527–1554, jul 2006. ISSN 0899-7667. doi: 10.1162/neco.2006.18.7.1527.
- Zhou J and Troyanskaya OG. Predicting effects of noncoding variants with deep learning–based sequence model. *Nature Methods*, 12(10):931, 2015.
- Daniel Kang, John Emmons, Firas Abuzaid, Peter Bailis, and Matei Zaharia. NoScope: Optimizing Neural Network Queries over Video at Scale. *Proceedings of the VLDB Endowment*, 10(11):1586—1597, 2017. ISSN 16130073. doi: 10.14778/3137628.3137664.
- Nitish Shirish Keskar, Jorge Nocedal, Ping Tak Peter Tang, Dheevatsa Mudigere, and Mikhail Smelyanskiy. On large-batch training for deep learning: Generalization gap and sharp minima. In *5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings. International Conference on Learning Representations, ICLR, 2017*.
- Marc Lanctot, Demis Hassabis, Thore Graepel, Veda Panneershelvam, Timothy Lillicrap, John Nham, Ioannis Antonoglou, David Silver, Chris J. Maddison, Arthur Guez, Ilya Sutskever, Aja Huang, Julian Schrittwieser, Nal Kalchbrenner, Dominik Grewe, George van den Driessche, Madeleine Leach, Laurent Sifre, Koray Kavukcuoglu, and Sander Dieleman. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, jan 2016. ISSN 0028-0836. doi: 10.1038/nature16961.
- Y. Le Cun, BE Boser, J. S. Js Denker, D. Henderson, R. E. Howard, E. Hubbard, L. D. Jackel, Bb Le Cun, J. S. Js Denker, and D. Henderson. Handwritten Digit Recognition with a Back-Propagation Network. In *Advances in Neural*

Information Processing Systems (NIPS), pages 396–404. 1990. ISBN 1-55860-100-7. doi: 10.1111/dsu.12130.

Yann LeCun, Marc Aurelio Ranzato, Christopher Poultney, Sumit Chopra, and Aurelio’Marc Ranzato. Sparse Feature Learning for Deep Belief Networks. In *Advances in neural information processing systems*, pages 1185—1192, 2008. ISBN 0018-9219. doi: 10.1109/5.726791.

Yann A. LeCun, Léon Bottou, Genevieve B. Orr, and Klaus Robert Müller. Efficient backprop. In *Neural networks: Tricks of the trade*, volume 7700 LECTU, pages 9–48. 2012. ISBN 9783642352881. doi: 10.1007/978-3-642-35289-8-3.

Christian Ledig, Lucas Theis, Ferenc Huszár, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, and Wenzhe Shi. Photo-realistic single image super-resolution using a generative adversarial network. In *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, volume 2017-Janua, pages 4681–4690, 2017. ISBN 9781538604571. doi: 10.1109/CVPR.2017.19.

Sergey Levine, Peter Pastor, Alex Krizhevsky, Julian Ibarz, and Deirdre Quillen. Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. *International Journal of Robotics Research*, 37(4-5):421–436, apr 2018. ISSN 17413176. doi: 10.1177/0278364917710318.

Seppo Linnainmaa. The representation of the cumulative rounding error of an algorithm as a Taylor expansion of the local rounding errors. Master’s Thesis (in Finnish), Univ. Helsinki, pages 6–7, 1970.

D.G. Lowe. Object recognition from local scale-invariant features. In *Proceedings of the Seventh IEEE International Conference on Computer Vision*, volume 99, pages 1150–1157, 1999. ISBN 0-7695-0164-8. doi: 10.1109/ICCV.1999.790410.

James Martens. Deep learning via Hessian-free optimization. In *ICML*, volume 27, pages 735–742, 2010. ISBN 9781605589077. doi: 10.1155/2011/176802.

Azalia Mirhoseini, Hieu Pham, Quoc V. Le, Benoit Steiner, Rasmus Larsen, Yuefeng Zhou, Naveen Kumar, Mohammad Norouzi, Samy Bengio, and Jeff Dean. Device Placement Optimization with Reinforcement Learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 2430–2439, 2017.

Abdel-rahman Mohamed, Dong Yu, and Li Deng. Investigation of full-sequence training of deep belief networks for speech recognition. In *Eleventh Annual Con-*

ference of the International Speech Communication Association, number September, 2010.

Abdel Rahman Mohamed, Tara N. Sainath, George Dahl, Bhuvana Ramabhadran, Geoffrey E. Hinton, and Michael A. Picheny. Deep belief networks using discriminative features for phone recognition. In ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing, pages 5060–5063, 2011. ISBN 9781457705397. doi: 10.1109/ICASSP.2011.5947494.

Roosbeh Mottaghi, Eric Kolve, Joseph J Lim, Abhinav Gupta, Li Fei-fei, and Ali Farhadi. Target-driven Visual Navigation in Indoor Scenes using Deep Reinforcement Learning. In IEEE international conference on robotics and automation (ICRA), pages 3357–3364, 2017.

V Nair and Geoffrey E. Hinton. 3D Object Recognition with Deep Belief Nets. In Advances in neural information processing systems, pages 1339–1347, 2009. ISBN 9781615679119.

Nesterov. A method of solving a convex programming problem with convergence rate. In Sov. Math. Dokl, volume 27, 1983.

David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, oct 1986. ISSN 0028-0836. doi: 10.1038/323533a0.

Tara N. Sainath, Brian Kingsbury, Bhuvana Ramabhadran, Petr Fousek, Petr Novak, and Abdel Rahman Mohamed. Making deep belief networks effective for large vocabulary continuous speech recognition. In 2011 IEEE Workshop on Automatic Speech Recognition and Understanding, ASRU 2011, Proceedings, pages 30–35, 2011. ISBN 9781467303675. doi: 10.1109/ASRU.2011.6163900.

Karen Simonyan and Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. In Proceedings of the 3rd International Conference on Learning Representations, sep 2015. ISBN 1097-0142 (Electronic)\n0008-543X (Linking). doi: 10.2146/ajhp170251.

Yaniv Taigman, Ming Yang, Marc’Aurelio Ranzato, and Lior Wolf. DeepFace: Closing the gap to human-level performance in face verification. In Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, pages 1701–1708, 2014. ISBN 9781479951178. doi: 10.1109/CVPR.2014.220.

Oriol Vinyals and Daniel Povey. Krylov Subspace Descent for Deep Learning. *Artificial Intelligence and Statistics*, pages 1261–1268, 2012.

- S. Zhou, Q. Chen, and X. Wang. Active deep learning method for semi-supervised sentiment classification. *Neurocomputing*, 120:536–546, 2013.
- Jun Yan Zhu, Philipp Krähenbühl, Eli Shechtman, and Alexei A. Efros. Generative visual manipulation on the natural image manifold. In *European Conference on Computer Vision*, volume 9909 LNCS, pages 597–613, 2016. ISBN 9783319464534. doi: 10.1007/978-3-319-46454-1_36.
- Barret Zoph and Quoc V. Le. Neural Architecture Search with Reinforcement Learning. arXiv preprint arXiv:1611.01578., 2016.

Chapter 3

Deep Belief Networks

Deep Belief Networks is another deep learning architecture that has attracted a lot of attention and has been applied to many problems so far such as image processing (Nair and Hinton, 2009), natural language processing (Zhou et al., 2013), automatic speech recognition (Dahl et al., 2012; Sainath et al., 2011; Mohamed et al., 2010, 2011), and feature extraction and reduction (Hamel and Eck, 2010), to name a few.

A DBN (Hinton et al., 2006) can be defined as a multi-layered probabilistic generative model based on multiple Restricted Boltzmann Machines (RBMs). Training of a Deep Belief Network is divided into two phases: (i) the first phase being a greedy, unsupervised, layer-by-layer pre-training phase which is designed to initialize the weights of each RBM to values in the neighborhood of a good local optimum of the error surface. This pre-training phase allows the DBN to make use of unlabeled data, which is often very desirable since most data is unlabeled. The second phase of training is a supervised phase where the DBN is trained as a perceptron using label data. Accordingly, this chapter is organized as follows: we start by defining components of an RBM (chapter 3.1). Then, we describe the pre-training process of these RBMs (chapter 3.2). Finally, the training of the overall DBN is explained (chapter 3.3).

3.1 Restricted Boltzmann Machines (RBMs)

An RBM is an energy-based generative model that consists of a bottom layer of I binary visible units (observed variables), $\mathbf{v} \in \{0, 1\}^I$, and a top layer of J binary hidden units (explanatory factors), $\mathbf{h} \in \{0, 1\}^J$, which are fully and bidirectionally connected with symmetric weights (Hinton, 2012), as depicted in Figure 3.1.

RBM's neurons form two disjoint sets, satisfying the definition of bipartite graphs. Neuron connections in a RBM may be directed or undirected; in the latter case, the network forms an auto-associative memory which is characterized by

bi-directional information flow due to feedback connections (Hinton et al., 2006). RBMs follow the encoder–decoder paradigm (LeCun et al., 2008) where both the encoded representation and the (decoded) reconstruction are stochastic by nature. The encoder–decoder architecture is useful because: (i) after training, the feature vector can be computed in a very fast way and (ii) by reconstructing the input we can assess how well the model was able to capture the relevant information from the data (LeCun et al., 2008).

In RBMs, units within the same layer are not connected which makes inference and learning within this graphical model tractable.

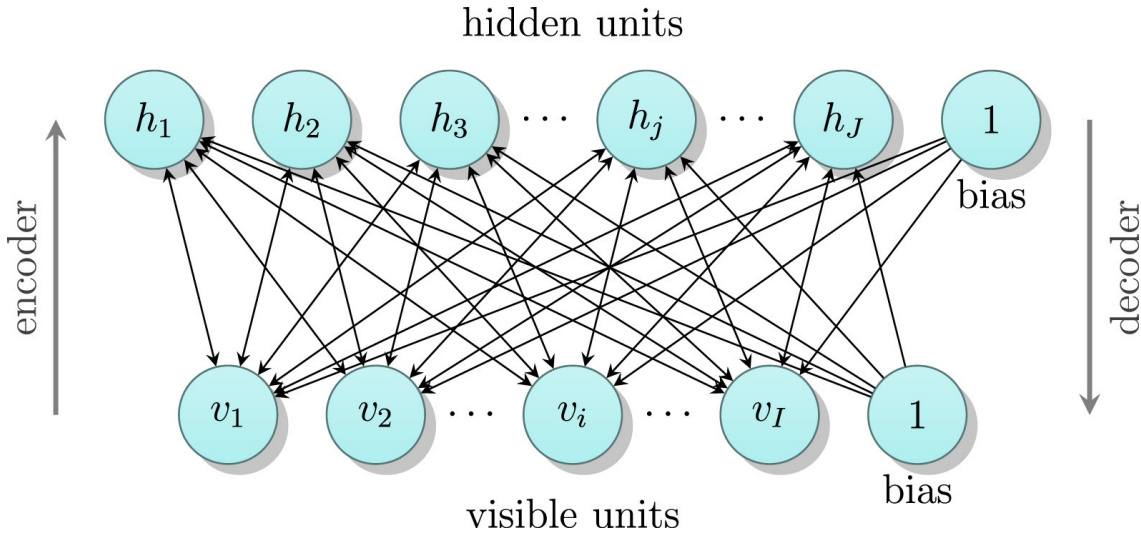


Figure 3.1: Schematic representation of a Restricted Boltzmann Machine (RBM).

3.2 Pre-training phase : Training each RBM

In a binary RBM, the units stochastically take on states 0 or 1, depending on their inputs from the other layer. Denoting the states of visible units with v_i , the states of hidden units with h_j , the weights connecting these units with W_{ij} , and the biases of visible and hidden units with c_i and b_j respectively, a RBM encodes a joint probability distribution $P(\mathbf{v}, \mathbf{h}|\boldsymbol{\theta})$, defined via the energy function

$$\begin{aligned} E(\mathbf{v}, \mathbf{h}; \boldsymbol{\theta}) &= -\mathbf{c}\mathbf{v}^T - \mathbf{b}\mathbf{h}^T - \mathbf{v}^T \mathbf{W} \mathbf{h} \\ &= -\sum_{i=1}^I c_i v_i - \sum_{j=1}^J b_j h_j - \sum_{i=1}^I \sum_{j=1}^J W_{ij} v_i h_j \end{aligned} \quad (3.1)$$

where $\boldsymbol{\theta} = (\mathbf{W}, \mathbf{c}, \mathbf{b})$, $\mathbf{c} \in \mathbb{R}^I$ represents the bias of the visible units, $\mathbf{b} \in \mathbb{R}^J$ denotes the bias of the hidden units and $\mathbf{W} \in \mathbb{R}^{J \times I}$ is a matrix containing all connection weights W_{ij} .

The RBM assigns a probability for each configuration (\mathbf{v}, \mathbf{h}) , using

$$P(\mathbf{v}, \mathbf{h}|\boldsymbol{\theta}) = \frac{1}{Z} e^{-E(\mathbf{v}, \mathbf{h}; \boldsymbol{\theta})} \quad (3.2)$$

where \mathbf{v} and \mathbf{h} are the visible and hidden units respectively, and Z is the normalization constant called “partition” function by analogy with physical systems, given by the sum of all energy configurations.

Since there are no connections between any two units within the same layer, given a particular random input configuration \mathbf{v} , all the hidden units are independent of each other and the probability of \mathbf{h} given \mathbf{v} becomes

$$P(\mathbf{h}|\mathbf{v}) = \prod_j P(h_j = 1|\mathbf{v}) \quad (3.3)$$

where $P(h_j = 1|\mathbf{v})$, the probability that a hidden unit h_j is activated given a visible vector \mathbf{v} , is given by 2.18

$$P(h_j = 1|\mathbf{v}) = \sigma(b_j + \sum_{i=1}^I W_{ij}v_i) \quad (3.4)$$

and $\sigma(x)$ is the sigmoid function $1/(1 + e^{-x})$. For implementation purposes, h_j is set to 1 when $P(h_j = 1|\mathbf{v})$ is greater than a given random number (uniformly distributed between 0 and 1) and to 0 otherwise.

Similarly, given a specific hidden \mathbf{h} , the probability of \mathbf{v} given \mathbf{h} is given by 3.5

$$P(\mathbf{v}|\mathbf{h}) = \prod_i P(v_i = 1|\mathbf{h}) \quad (3.5)$$

where $P(v_i = 1|\mathbf{h})$, the probability that a visible unit v_i is activated given a hidden vector \mathbf{h} , is defined as,

$$P(v_i = 1|\mathbf{h}) = \sigma(c_i + \sum_{j=1}^J W_{ij}h_j) \quad (3.6)$$

When using eq.2.20, the hidden states are to be binary so that the input vector could be reconstructed.

The marginal probability assigned to a visible vector \mathbf{v} can be expressed as,

$$P(\mathbf{v}) = \sum_{\mathbf{h}} p(\mathbf{v}, \mathbf{h}) = \frac{1}{Z} \sum_{\mathbf{h}} e^{-E(\mathbf{v}, \mathbf{h})} \quad (3.7)$$

During learning, the visible units are clamped to the actual inputs, which are seen as samples from the “data distribution”. The task for learning is to adapt the parameters $\boldsymbol{\theta}$ such that the marginal distribution $P(\mathbf{v}, |\boldsymbol{\theta}) = \sum_{\mathbf{h}} P(\mathbf{v}, \mathbf{h}|\boldsymbol{\theta})$ becomes

maximally similar to the true observed data distribution $p^*(v)$. Indeed, given a specific training vector \mathbf{v} , its probability can be raised by adjusting weights and biases of the network so as to make the energy of that particular vector smaller and the energy of the other vectors higher. To this end, we can perform a stochastic gradient ascent on the log-likelihood manifold obtained from the training data vectors, by computing the derivative of the log probability with respect to the network parameters $\boldsymbol{\theta} \in \{b_j, c_i, W_{ij}\}$ which can be computed as follows,

$$\begin{aligned}
\frac{\partial \log P(\mathbf{v})}{\partial \boldsymbol{\theta}} &= \frac{\partial \log(\frac{1}{Z} \sum_{\mathbf{h}} e^{-E(\mathbf{v}, \mathbf{h})})}{\partial \boldsymbol{\theta}} \\
&= \frac{\overbrace{\partial \log(\sum_{\mathbf{h}} e^{-E(\mathbf{v}, \mathbf{h})})}^{\text{positive phase}}}{\partial \boldsymbol{\theta}} - \frac{\overbrace{\partial \log(Z)}^{\text{negative phase}}}{\partial \boldsymbol{\theta}} \\
&= \frac{\partial \log(\sum_{\mathbf{h}} e^{-E(\mathbf{v}, \mathbf{h})})}{\partial \boldsymbol{\theta}} - \frac{\partial \log(\sum_{\mathbf{v}, \mathbf{h}} e^{-E(\mathbf{v}, \mathbf{h})})}{\partial \boldsymbol{\theta}} \\
&= - \frac{\sum_{\mathbf{h}} e^{-E(\mathbf{v}, \mathbf{h})} \frac{\partial E(\mathbf{v}, \mathbf{h})}{\partial \boldsymbol{\theta}}}{\sum_{\mathbf{h}} e^{-E(\mathbf{v}, \mathbf{h})}} + \frac{\sum_{\mathbf{v}, \mathbf{h}} e^{-E(\mathbf{v}, \mathbf{h})} \frac{\partial E(\mathbf{v}, \mathbf{h})}{\partial \boldsymbol{\theta}}}{\sum_{\mathbf{v}, \mathbf{h}} e^{-E(\mathbf{v}, \mathbf{h})}}
\end{aligned} \tag{3.8}$$

Using eq.3.2 and knowing that

$$p(\mathbf{h}|\mathbf{v}) = \frac{p(\mathbf{h}, \mathbf{v})}{p(\mathbf{v})} = \frac{p(\mathbf{h}, \mathbf{v})}{\sum_{\mathbf{h}} p(\mathbf{h}, \mathbf{v})} = \frac{1}{\sum_{\mathbf{h}} \frac{e^{-E(\mathbf{v}, \mathbf{h})}}{Z}} \frac{e^{-E(\mathbf{v}, \mathbf{h})}}{Z} = \frac{\sum_{\mathbf{h}} \mathbf{h} e^{-E(\mathbf{v}, \mathbf{h})}}{e^{-E(\mathbf{v}, \mathbf{h})}} \tag{3.9}$$

then eq.3.8 can be rewritten as,

$$\frac{\partial \log P(\mathbf{v})}{\partial \boldsymbol{\theta}} = - \overbrace{\sum_{\mathbf{h}} p(\mathbf{h}|\mathbf{v}) \frac{\partial E(\mathbf{v}, \mathbf{h})}{\partial \boldsymbol{\theta}}}_{\text{positive phase}} + \overbrace{\sum_{\mathbf{h}} p(\mathbf{h}, \mathbf{v}) \frac{\partial E(\mathbf{v}, \mathbf{h})}{\partial \boldsymbol{\theta}}}_{\text{negative phase}} \tag{3.10}$$

As in the maximum likelihood learning procedure, we aim at finding the set of network parameters for which the probability of the (observed) training dataset $P(\mathbf{v})$ is maximized. Computing $\frac{\partial E(\mathbf{v}, \mathbf{h})}{\partial \boldsymbol{\theta}}$ is straightforward. Thus, in order to obtain an unbiased stochastic estimator of the log-likelihood gradient, we need a procedure to sample from $p(\mathbf{h}|\mathbf{v})$ and another to sample from $p(\mathbf{h}, \mathbf{v})$. To do so, in the “positive phase”, we first clamp \mathbf{v} to the observed input vector \mathbf{x} and then sample \mathbf{h} from the clamped \mathbf{v} . On the other hand, in the “negative phase”, we sample both \mathbf{v} and \mathbf{h} from the model (Bengio, 2009). Sampling can be done by setting up a Markov chain Monte Carlo (MCMC) using alternating Gibbs sampling (AGS) (Bengio, 2009; Hinton, 2012). Each iteration of the AGS consists of updating all of the hidden units through sampling from the probability $p(\mathbf{h}|\mathbf{v})$ using equation 2.18, followed by updating all of the visible units through sampling from the distribution/probability $p(\mathbf{v}|\mathbf{h})$ using equation 2.20. This process is visualized in Figure 3.2. As a result,

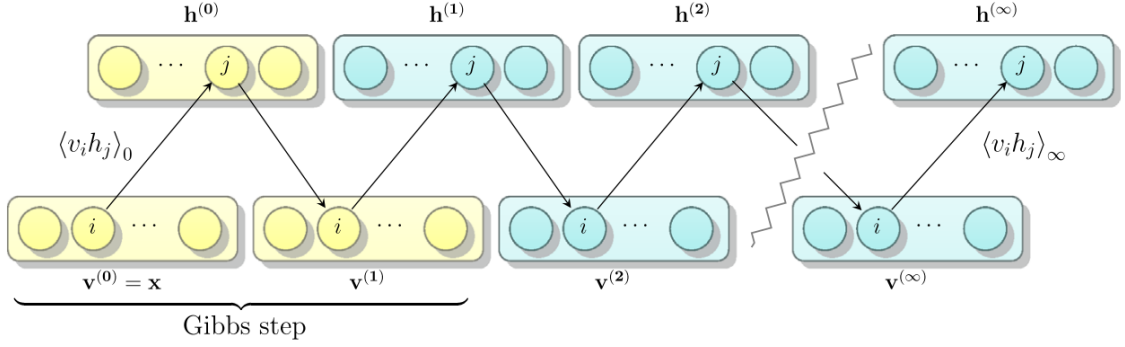


Figure 3.2: Markov chain Monte Carlo using alternating Gibbs sampling in a restricted Boltzmann machine (RBM). The chain is initialized with the data input vector \mathbf{x} .

equation 3.10 becomes,

$$\frac{\partial \log P(\mathbf{v})}{\partial \boldsymbol{\theta}} = - \overbrace{\left\langle \frac{\partial E(\mathbf{v}, \mathbf{h})}{\partial \boldsymbol{\theta}} \right\rangle_0}^{\text{positive phase}} + \overbrace{\left\langle \frac{\partial E(\mathbf{v}, \mathbf{h})}{\partial \boldsymbol{\theta}} \right\rangle_\infty}^{\text{negative phase}} \quad (3.11)$$

where $\langle \cdot \rangle_0$ denotes the expectations for the data distribution ($p_0 = p(\mathbf{h}|\mathbf{v})$) (e.g., an average over samples with visible units clamped to actual inputs) and $\langle \cdot \rangle_\infty$ denotes the expectations under the model distribution ($p_\infty(\mathbf{v}, \mathbf{h}) = p(\mathbf{v}, \mathbf{h})$) (e.g., an average over samples when the network is allowed to sample all units freely) (Hinton, 2002). Unfortunately, using a sampling approximation by computing $\langle v_i h_j \rangle_\infty$ (which is the negative phase e.g., the second term of eq.2.25) normally requires creating enough samples such that the network can settle into an equilibrium. So this approximation requires performing AGS for a very long time (Hinton, 2012; Bengio, 2009) in order to draw unbiased samples from the model distribution, thus making the problem intractable. To solve this issue, Hinton proposed a much faster learning procedure which performs very well in practice: the contrastive divergence (CD- k) algorithm (Hinton, 2012, 2002) which consists of fixing k to a small value (such as 1) and replacing $\langle \cdot \rangle_\infty$ with $\langle \cdot \rangle_k$.

For example, at $k = 1$, only a single sample for the data and model distribution is used. Indeed, CD- k first samples new values for all hidden units in parallel, conditioned on the current input, which gives a complete sample $(\mathbf{v}^{(0)}, \mathbf{h}^{(0)})$ for the data distribution. It then generates a sample for the visible layer, conditioned on the hidden states $\mathbf{h}^{(0)}$ sampled in the first step, and then samples the hidden layer again, conditioned on this new activity in the visible layer. This generates a sample $(\mathbf{v}^{(1)}, \mathbf{h}^{(1)})$ from the model distribution.

Accordingly, by applying CD- k into eq.3.11, the following update rules are ob-

tained,

$$\Delta W_{ij} = \alpha(\langle v_i h_j \rangle_0 - \langle v_i h_j \rangle_k) \quad (3.12)$$

$$\Delta b_j = \alpha(\langle h_j \rangle_0 - \langle h_j \rangle_k) \quad (3.13)$$

$$\Delta c_i = \alpha(\langle v_i \rangle_0 - \langle v_i \rangle_k) \quad (3.14)$$

where α stands for the learning rate.

The main steps of the CD- k algorithm are summarized in Algorithm 1.

Algorithm 1: CD- k algorithm.

Data: \mathbf{x} is an input vector of the training dataset.

Result: update of parameters θ

$\mathbf{v}^0 = \mathbf{x}$;

Compute the binary states of the hidden units, \mathbf{h}^0 , using \mathbf{v}^0 and Eq.3.4 ;

for ($n = 1$; $n < k$; $n = n + 1$) {

 Compute the “reconstruction” states for the visible units \mathbf{v}^n , using
 \mathbf{h}^{n-1} and Eq.3.6 ;

 Compute the binary features (states) for the hidden units \mathbf{h}^n , using \mathbf{v}^n
 and Eq.3.4 ;

}

Update the weights W_{ij} using eq.3.12 ;

Update the biases b_j using eq.3.13 ;

Update the biases c_i using eq.3.14 ;

3.3 Forming the DBN and fine-tuning it

As illustrated by Figure 3.3, DBNs are constructed by stacking Restricted Boltzmann Machines (RBMs). The visible layers of RBMs at the bottom of a DBN are clamped to the actual inputs when data is presented. When RBMs are stacked to form a DBN, the hidden layer of the lower RBM becomes the visible layer of the next higher RBM, as explained in 3.4. Through this process, higher level RBMs can be trained to encode more and more abstract features of the input distribution. In Figure 3.3, visible and hidden units of the 3 RBMs 1, 2 and 3 are $\{\mathbf{v}, \mathbf{W}_1\}$, $\{\mathbf{W}_1, \mathbf{W}_2\}$ and $\{\mathbf{W}_2, \mathbf{W}_3\}$ respectively. The supervised phase consists of a global fine-tuning process that is very similar to traditional neural network training and can use normal gradient or conjugate gradient descent (Rumelhart et al., 1986). After training a stack of RBMs, the bottom-up recognition weights of the result-

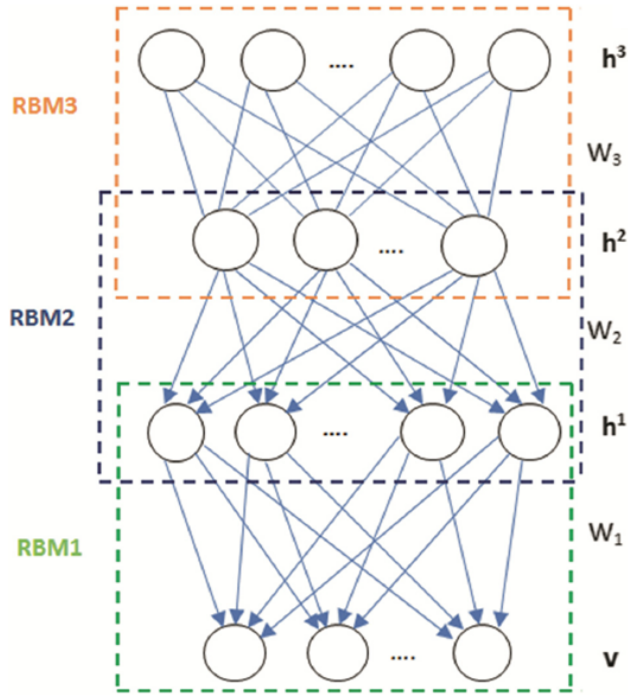


Figure 3.3: Structure of a DBN with three hidden layers.

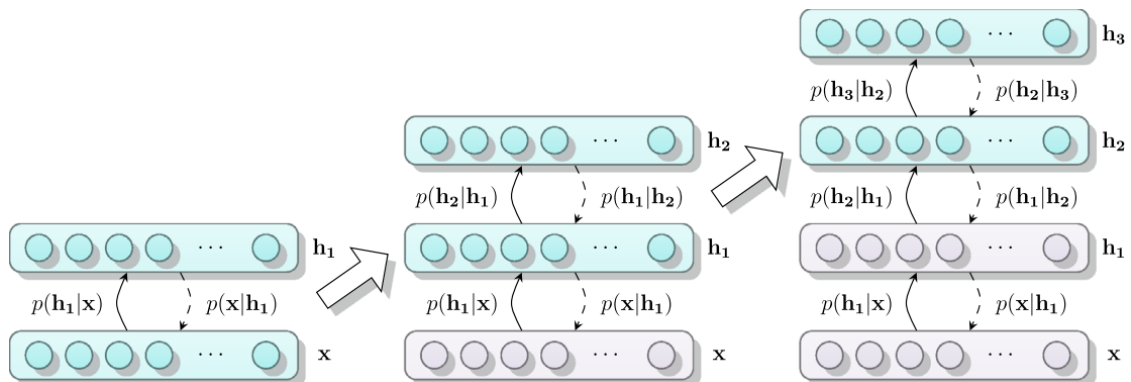


Figure 3.4: DBN training with one input layer \mathbf{x} , and three hidden layers $\mathbf{h1}$, $\mathbf{h2}$, $\mathbf{h3}$. From left to right, purple color represents layers already trained, while cyan the RBM being trained.

ing DBN (from the unsupervised phase) can be used to initialize the weights of a Multi-Layer Feed-Forward Neural Network with as many neurons at the top end as the number of classes (e.g., labels). This feed-forward neural network can then be discriminatively fine-tuned using labeled data by back propagating error derivatives.

Bibliography

Artem Babenko, Anton Slesarev, Alexandr Chigorin, and Victor Lempitsky. Neural codes for image retrieval. In Lecture Notes in Computer Science, vol-

ume 8689 LNCS, pages 584–599. 2014. ISBN 9783319105895. doi: 10.1007/978-3-319-10590-1_38.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural Machine Translation by Jointly Learning to Align and Translate. arxiv.org, 2014. ISSN 0147-006X. doi: 10.1146/annurev.neuro.26.041002.131047.

Y. Bengio. Learning Deep Architectures for AI. Foundations and Trends® in Machine Learning, 2(1):1–127, 2009. ISSN 1935-8237. doi: 10.1561/22000000006.

Yoshua Bengio, Nicolas Boulanger-Lewandowski, and Razvan Pascanu. Advances in optimizing recurrent networks. In ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings, pages 8624–8628, 2013. ISBN 9781479903566. doi: 10.1109/ICASSP.2013.6639349.

Ashish Bora, Ajil Jalal, Eric Price, and Alexandros G. Dimakis. Compressed Sensing using Generative Models. In dl.acm.org, pages 537—546, 2017. ISBN 9781510855144. doi: 10.5829/idosi.wasj.2013.23.04.368.

Ken Chatfield, Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Return of the Devil in the Details: Delving Deep into Convolutional Nets. In British Machine Vision Conference, 2014. ISBN 1-901725-52-9. doi: 10.5244/C.28.6.

Chenyi Chen. DeepDriving : Learning Affordance for Direct Perception in Autonomous Driving. In 2015 IEEE International Conference on Computer Vision (ICCV), pages 2722–2730, 2015. doi: 10.1109/ICCV.2015.312.

Qifeng Chen and Vladlen Koltun. Photographic Image Synthesis with Cascaded Refinement Networks. In Proceedings of the IEEE International Conference on Computer Vision, volume 2017-October, pages 1520–1529, 2017. ISBN 9781538610329. doi: 10.1109/ICCV.2017.168.

Ronan Collobert, Jason Weston, Leon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. Natural Language Processing (almost) from Scratch. Journal of machine learning research, 12(Aug):2493—2537, 2011. ISSN 1532-4435. doi: 10.1145/2347736.2347755.

George E. Dahl, Dong Yu, Li Deng, and Alex Acero. Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition. IEEE Transactions on Audio, Speech and Language Processing, 20(1):30—42, 2012. ISSN 15587916. doi: 10.1109/TASL.2011.2134090.

- Navneet Dalal and William Triggs. Histograms of Oriented Gradients for Human Detection. In International Conference on computer vision & Pattern Recognition, volume 1, pages 886–893, 2005. ISBN 0-7695-2372-2. doi: 10.1109/CVPR.2005.177.
- Li Deng. A Tutorial Survey of Architectures, Algorithms, and Applications for Deep Learning. APSIPA Transactions on Signal and Information Processing, 3, 2014.
- Stuart Dreyfus. The numerical solution of variational problems. Journal of Mathematical Analysis and Applications, 5(1):30–45, 1962. ISSN 10960813. doi: 10.1016/0022-247X(62)90004-5.
- Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. A Neural Algorithm of Artistic Style. arXiv preprint arXiv:1508.06576, aug 2015. ISSN 1935-8237. doi: 10.1561/22000000006.
- Ross Girshick, Jeff Donahue, Trevor Darrell, U C Berkeley, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In IEEE Conference on Computer Vision and Pattern Recognition, pages 580–587, 2014. ISBN 978-1-4799-5118-5. doi: 10.1109/CVPR.2014.81.
- Philippe Hamel and Douglas Eck. Learning Features from Music Audio with Deep Belief Networks. In Proc. of the 11th International Society of Music Information Retrieval, pages 339–344, 2010. ISBN 9789039353813.
- Geoffrey E. Hinton. Training Products of Experts by Minimizing Contrastive Divergence. Neural Computation, 14(8):1771–1800, aug 2002. ISSN 0899-7667. doi: 10.1162/089976602760128018.
- Geoffrey E. Hinton. A practical guide to training restricted Boltzmann machines. In Neural networks: Tricks of the Trade, pages 599–619. 2012.
- Geoffrey E. Hinton, Simon Osindero, and Yee-Whye Teh. A Fast Learning Algorithm for Deep Belief Nets. Neural Computation, 18(7):1527–1554, jul 2006. ISSN 0899-7667. doi: 10.1162/neco.2006.18.7.1527.
- Zhou J and Troyanskaya OG. Predicting effects of noncoding variants with deep learning–based sequence model. Nature Methods, 12(10):931, 2015.
- Daniel Kang, John Emmons, Firas Abuzaid, Peter Bailis, and Matei Zaharia. NoScope: Optimizing Neural Network Queries over Video at Scale. Proceedings of the VLDB Endowment, 10(11):1586–1597, 2017. ISSN 16130073. doi: 10.14778/3137628.3137664.

Nitish Shirish Keskar, Jorge Nocedal, Ping Tak Peter Tang, Dheevatsa Mudigere, and Mikhail Smelyanskiy. On large-batch training for deep learning: Generalization gap and sharp minima. In 5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings. International Conference on Learning Representations, ICLR, 2017.

Marc Lanctot, Demis Hassabis, Thore Graepel, Veda Panneershelvam, Timothy Lillicrap, John Nham, Ioannis Antonoglou, David Silver, Chris J. Maddison, Arthur Guez, Ilya Sutskever, Aja Huang, Julian Schrittwieser, Nal Kalchbrenner, Dominik Grewe, George van den Driessche, Madeleine Leach, Laurent Sifre, Koray Kavukcuoglu, and Sander Dieleman. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, jan 2016. ISSN 0028-0836. doi: 10.1038/nature16961.

Y. Le Cun, BE Boser, J. S. Js Denker, D. Henderson, R. E. Howard, E. Hubbard, L. D. Jackel, Bb Le Cun, J. S. Js Denker, and D. Henderson. Handwritten Digit Recognition with a Back-Propagation Network. In *Advances in Neural Information Processing Systems (NIPS)*, pages 396–404. 1990. ISBN 1-55860-100-7. doi: 10.1111/dsu.12130.

Yann LeCun, Marc Aurelio Ranzato, Christopher Poultney, Sumit Chopra, and Aurelio’Marc Ranzato. Sparse Feature Learning for Deep Belief Networks. In *Advances in neural information processing systems*, pages 1185—1192, 2008. ISBN 0018-9219. doi: 10.1109/5.726791.

Yann A. LeCun, Léon Bottou, Genevieve B. Orr, and Klaus Robert Müller. Efficient backprop. In *Neural networks: Tricks of the trade*, volume 7700 LECTU, pages 9–48. 2012. ISBN 9783642352881. doi: 10.1007/978-3-642-35289-8-3.

Christian Ledig, Lucas Theis, Ferenc Huszár, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, and Wenzhe Shi. Photo-realistic single image super-resolution using a generative adversarial network. In *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, volume 2017-Janua, pages 4681–4690, 2017. ISBN 9781538604571. doi: 10.1109/CVPR.2017.19.

Sergey Levine, Peter Pastor, Alex Krizhevsky, Julian Ibarz, and Deirdre Quillen. Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. *International Journal of Robotics Research*, 37(4-5):421–436, apr 2018. ISSN 17413176. doi: 10.1177/0278364917710318.

- Seppo Linnainmaa. The representation of the cumulative rounding error of an algorithm as a Taylor expansion of the local rounding errors. Master's Thesis (in Finnish), Univ. Helsinki, pages 6–7, 1970.
- D.G. Lowe. Object recognition from local scale-invariant features. In Proceedings of the Seventh IEEE International Conference on Computer Vision, volume 99, pages 1150–1157, 1999. ISBN 0-7695-0164-8. doi: 10.1109/ICCV.1999.790410.
- James Martens. Deep learning via Hessian-free optimization. In ICML, volume 27, pages 735–742, 2010. ISBN 9781605589077. doi: 10.1155/2011/176802.
- Azalia Mirhoseini, Hieu Pham, Quoc V. Le, Benoit Steiner, Rasmus Larsen, Yuefeng Zhou, Naveen Kumar, Mohammad Norouzi, Samy Bengio, and Jeff Dean. Device Placement Optimization with Reinforcement Learning. In Proceedings of the 34th International Conference on Machine Learning-Volume 70, pages 2430–2439, 2017.
- Abdel-rahman Mohamed, Dong Yu, and Li Deng. Investigation of full-sequence training of deep belief networks for speech recognition. In Eleventh Annual Conference of the International Speech Communication Association, number September, 2010.
- Abdel Rahman Mohamed, Tara N. Sainath, George Dahl, Bhuvana Ramabhadran, Geoffrey E. Hinton, and Michael A. Picheny. Deep belief networks using discriminative features for phone recognition. In ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing, pages 5060–5063, 2011. ISBN 9781457705397. doi: 10.1109/ICASSP.2011.5947494.
- Roosbeh Mottaghi, Eric Kolve, Joseph J Lim, Abhinav Gupta, Li Fei-fei, and Ali Farhadi. Target-driven Visual Navigation in Indoor Scenes using Deep Reinforcement Learning. In IEEE international conference on robotics and automation (ICRA), pages 3357–3364, 2017.
- V Nair and Geoffrey E. Hinton. 3D Object Recognition with Deep Belief Nets. In Advances in neural information processing systems, pages 1339–1347, 2009. ISBN 9781615679119.
- Nesterov. A method of solving a convex programming problem with convergence rate. In Sov. Math. Dokl, volume 27, 1983.
- David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, oct 1986. ISSN 0028-0836. doi: 10.1038/323533a0.

Tara N. Sainath, Brian Kingsbury, Bhuvana Ramabhadran, Petr Fousek, Petr Novak, and Abdel Rahman Mohamed. Making deep belief networks effective for large vocabulary continuous speech recognition. In 2011 IEEE Workshop on Automatic Speech Recognition and Understanding, ASRU 2011, Proceedings, pages 30–35, 2011. ISBN 9781467303675. doi: 10.1109/ASRU.2011.6163900.

Karen Simonyan and Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. In Proceedings of the 3rd International Conference on Learning Representations, sep 2015. ISBN 1097-0142 (Electronic)\n0008-543X (Linking). doi: 10.2146/ajhp170251.

Yaniv Taigman, Ming Yang, Marc’Aurelio Ranzato, and Lior Wolf. DeepFace: Closing the gap to human-level performance in face verification. In Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, pages 1701–1708, 2014. ISBN 9781479951178. doi: 10.1109/CVPR.2014.220.

Oriol Vinyals and Daniel Povey. Krylov Subspace Descent for Deep Learning. Artificial Intelligence and Statistics, pages 1261–1268, 2012.

S. Zhou, Q. Chen, and X. Wang. Active deep learning method for semi-supervised sentiment classification. Neurocomputing, 120:536–546, 2013.

Jun Yan Zhu, Philipp Krähenbühl, Eli Shechtman, and Alexei A. Efros. Generative visual manipulation on the natural image manifold. In European Conference on Computer Vision, volume 9909 LNCS, pages 597–613, 2016. ISBN 9783319464534. doi: 10.1007/978-3-319-46454-1_36.

Barret Zoph and Quoc V. Le. Neural Architecture Search with Reinforcement Learning. arXiv preprint arXiv:1611.01578., 2016.

Part III

Learning from homogeneous data

Chapter 1

Learning from time-series

Time-series classification is a challenging task in data mining. With the rise of accessible time series datasets, many approaches have been proposed so far. However, despite all efforts made in this field, time-series classification is still challenging due to the following factors: (i) the large amount of data, (ii) the high-dimensionality of data and, (iii) the continuously changing aspect within this type of data. Furthermore, it is worth mentioning that several proposed approaches do not take into consideration that features or details held by time-series often have different time scales. Recently, new approaches based on Deep Learning frameworks have been proposed to tackle such challenges and issues. To this matter, we introduce two approaches for handling time series classification including: (i) a data-level method which turns time-series signals into frequency-domain ones using the Stockwell Transform technique, and (ii) an algorithm-level method which trains a ConvNet using an adaptive convolutional layer filter suiting the time-series input. These approaches are further run on classifying human activities including normal movements of typical subjects and abnormal movements of atypical subjects especially Stereotypical Motor Movements performed by children on the autism spectrum. Experimental results prove the effectiveness of our approaches in dealing with time-series classification.

1.1 Introduction

A time series is a set of regular time-ordered observations taken at successive and regular intervals (e.g., points of time). Examples of time series include Weather records, economic indicators, patient health evolution metrics, etc. Extensive research has been conducted so far on the classification of these time series. And still, time series classification presents some challenges: the large volume and high dimensionality of data, as well as the continuously updating scheme of time series. And recently, time series classification has attracted great interests and initiated

various researches related to Deep Learning. Indeed, time series classification with Deep Learning has been widely applied in different fields such as computer science for speech recognition or signature verification (Abdel-hamid et al., 2012; Abdel-Hamid et al., 2013), in medical science to identify human movements (Sadouk and Gadi, 2017; Sadouk et al., 2018) or to track cardiac anomaly (Liu et al., 2013). To this matter, proposes a ConvNet approach for classifying time series data. Our contributions are:

- i we show that the Stockwell Transform can be used as a data-level approach to improve performance of our ConvNet framework,
- ii we introduce a novel algorithm-level approach based on an adaptive convolutional layer filter that shows better performance than the standard ConvNet.

The remainder of this chapter is organized as follows. In section 1.2, we present an overview of existent techniques for time-series classification including including handcrafted and automated methods. Then, we present some preliminaries about the nature of the time series (Section 1.3). Afterward, we introduce and describe our data level and algorithm level approaches (Section 1.4). Implementation details are provided in Sec. 1.5. Experiments and results are summarized in Sec. 1.5 and the paper concludes in Sec. 1.6.

1.2 Related work

1.2.1 Classical methods

To address the task of time series classification, several approaches have been introduced so far. These approaches can be either model based, distance based or feature based.

Model based methods. In this category, a model is built for each and every class by fitting its parameters to that class, ending up with as many models as the number of classes. Then, at test time, each time series of the validation set is compared to the models to determine which class it belongs to. The auto-regressive (AR) model has been commonly used for time series analysis (Venkataramana and Sekhar, 2013), but it requires that the time series satisfy a stationary assumption, which is unfeasible in practice. The Markov model (MM) and the hidden Markov model (HMM) approaches were applied to model non-stationary time series and to perform time series classification (Antonucci et al., 2015), but they are limited to symbolic time series only. Besides, in general, generative models (including AR, MM and HMM) are unable to extract the best representations out of time series.

Distance based methods. This category consists of two steps: (i) building a distance function to measure the similarity (or dissimilarity) between two time series and, (ii) selecting the proper classifier such as K nearest neighbor (KNN) (Fix and Hodges, 1951), support vector machines (SVM) (Boser et al., 1992), etc.. However, these approaches have limitations: they are sensitive to distortions in time dimension and the length of the two series inputs must be equal. To deal with these limitations, the concept of dynamic time warping (DTW) distance was introduced (Rakthanmanon et al., 2013; Jeong Y S et al., 2011). Nonetheless, the DTW distance does not satisfy the triangle inequality, and is computationally expensive.

Feature based methods. In this category, the main goal is to capture relevant features from time series data. This is achieved via dimension reduction by using a set of features representing the time series. One proposed approach was to use discrete Fourier transform (DFT) (Schäfer, 2015), a spectral method for converting a time series into frequency domain signals, and to apply some of the few Fourier coefficients as features. However, with DFT, many local features of the original data may be lost. To alleviate this problem, short-time Fourier transform (STFT) (also known as windowed Fourier transform) was suggested (Bailly et al., 2015), but it requires determining the optimal window size. An alternative is the discrete wavelet transform (DWT) which is commonly used for time series analysis and which is designed for time-frequency multi-resolution analysis of time series (of length 2k only) (Jeong Y S et al., 2011; Rakthanmanon et al., 2013). There are other approaches for feature extraction which rely on eigenvalues such as principal component analysis (PCA), singular value decomposition (SVD) and sparse coding (Chen et al., 2015). In addition, other approaches based on shapelets have been proposed for time series classification (Hills et al., 2014).

Ensemble algorithms also yield state-of-the-art performance with time series classification problems. Three of the most successful ensemble algorithms that integrate various features of a time series are Elastic Ensemble (PROP) (Lines and Bagnall, 2015), a model that integrates 11 time series classifiers using a weighted ensemble method, Shapelet ensemble (SE) (Cetin et al., 2015), a model that applies a heterogeneous ensemble onto transformed shapelets, and a flat collective of transform based ensembles (COTE) (Bagnall et al., 2016), a model that fuses 35 various classifiers into a single classifier.

1.2.2 Automated methods

Automated approaches based deep learning models have shown to be successful in classifying time series, especially ConvNets. In (Abdel-hamid et al., 2012; Abdel-

Hamid et al., 2013), authors proposed a solution to the issue of speech recognition which consists of having speech signals sharing similar patterns within different frequency band locations which convey a different meaning. One solution is to use a limited weight sharing ConvNet architecture (Abdel-hamid et al., 2012) which consists of limiting weight sharing only to local filters which are close to each other and which are pooled together in the subsampling layer. Another solution suggested by (Wang and Oates, 2015) is composed of a tiled ConvNet architecture with a pre-training stage. A tiled ConvNet (Le et al., 2010) is an extension of ConvNets which unties weights locally and does not require that adjacent hidden units share identical weights. This type of ConvNet rather learns k separate convolution kernels within the same layer.

Meanwhile, authors of (Cui et al., 2016) introduced a multi-scale convolutional neural network (MConvNet) where each of the three transformed versions of the input is fed into a branch (e.g., a set of consecutive convolutional and pooling layers), thus ending up with three outputs which are further concatenated and fed into more convolutional and pooling layers, fully connected layers and a softmax layer to obtain the final output. Another work (Wang et al., 2016) also makes use of a ConvNet architecture with multiple branches which are fed by the same time series signal. Each of these branches has a different convolutional filter size so as to detect all multi-scale characteristics within the input. Another attempt to improve time series classification was made by (Wang et al., 2017) who proposed two variants of ConvNets: (i) a fully convolutional network (FCN) without subsampling layers, with batch normalization layers and with a global pooling layer instead of fully connected layers. Let's note that FCNs (Long et al., 2015) are networks with convolutional layers only and no fully-connected layers, (ii) a residual network (ResNet) with batch normalization layers added. As a definition, a ResNet (He et al., 2016) is a type of network which solves the "vanishing gradient" problem within very deep networks (i.e., networks with several stacked layers) by using residual blocks which have the property of preserving inputs. These two variants achieve comparable or better results than MConvNet (Cui et al., 2016). An ensemble method of deep learning models named LSTM-FCN was introduced by (Fazle Karim, Houshang Darabi, Somshubra Majumdar, 2018). This method, the same input is fed into the FCN and the Long Short Term Recurrent Neural Network (LSTM) block (Hochreiter and Schmidhuber, 1997), thus generating two outputs which are further concatenated and passed onto a softmax classification layer. Meanwhile, Guennec et al. (Le Guennec et al., 2016) suggested to use data-augmentation techniques and to pre-train each layer in an unsupervised manner (by an auto-encoder) using unlabeled training time series from different datasets. Regarding multivariate time series, only few research papers based on ConvNets

were published. The work of (Zheng et al., 2014) introduced a multi-channels deep convolution neural network (MC-DConvNet) composed of multiple branches. Each one of these branch is fed by a single dimension of the multivariate time series and learns features individually. Then the learned features of each branch are embedded and fed into a fully connected layer for classification. Other works including (Sadouk and Gadi, 2017; Sadouk et al., 2018; Zhao et al., 2017) handled multivariate time series differently by respectively treating the 3-, 12-, and 9-variate time series inputs as a 3-, 12-, and 9-channel inputs and convolving them as a whole instead of convolving each channel separately as illustrated by (Zheng et al., 2014). This method seems more plausible than the former (Zheng et al., 2014) since separating multivariate time series into univariate makes us lose the interrelationship between different univariate time series.

In this chapter, our ConvNet approaches for time series classification are laid out. This chapter is structured as follows. Section 1.2 starts by defining the main concepts of time series. In Section 1.3, a summary of existent data-level techniques for time series classification is provided, our data-level approach is explained, then corresponding experiments and results are presented. The next section (Section 1.4) gives an overview of algorithm-level approaches for time series classification, then introduces our algorithm-level approach and lays out conducted experiments and obtained results. Finally, the last section within this chapter (Section 1.5) concludes our paper with future perspectives.

1.3 Overview of time series data

Types of time-series. There exists two types of time-series data:

- Univariate Time-series involves the analysis of a single variable. The input will then have one channel,
- Multivariate Time-series is an extension of the univariate case and involves two or more input variables. The resulting input consists of multiple channels.

Most time-series predicting models work with univariate data and show promising results. However, existing multivariate time-series models do not perform as well as univariate ones due to the complexity in modeling simultaneously multiple time-series.

Raw vs extracted time-series signals. A raw time-series is the original time-series (e.g., a sequence of discrete-time data taken at successive equally spaced points in time). In time-series classification tasks, some authors choose to train their predicting models using raw time-series data (Sadouk and Gadi, 2017; Sadouk et al.,

2018; Abdel-Hamid et al., 2013; Abdel-hamid et al., 2012; Wang and Oates, 2015; Zheng et al., 2014; Yang et al., 2015; Rad et al., 2018). On the other hand, others use extracted time-series data which consist of raw time-series already segmented and converted into a set of fixed-length signals. Examples of such models using ConvNet include (Cui et al., 2016; Wang et al., 2016, 2017; Fazle Karim, Houshang Darabi, Somshubra Majumdar, 2018; Le Guennec et al., 2016; Zheng et al., 2014; Hatami et al., 2017) whose simulations are conducted using extracted signals from the UCR time-series classification archive (Keogh et al., 2011). However, this benchmark is composed of relatively small datasets (with a small number of instances), which makes the ConvNet less efficient knowing that ConvNets require large training sets for training. Furthermore, in most of the cases, fixed-length signals cannot be further encoded into new representations, as opposed to raw time-series. These issues have led us to use raw time-series data instead (Sadouk and Gadi, 2017; Sadouk et al., 2018).

1.4 Methodology

1.4.1 Our Data-level approach

In this section, we start by describing pre-processing methods to turn time-series data into input vectors (which will be further used for training deep learning models), including the basic Sliding Window method and more advanced methods (Sec.1.4.1.1). Then, we introduce our pre-processing method (Sec.1.4.1.2).

1.4.1.1 Background on pre-processing methods for time-series

Basic pre-processing method: the Sliding Window. This Sliding Window approach is the basis of how to turn any time series dataset into a supervised learning problem. The Sliding Window is a temporary approximation over the actual value of the time series data (BenYahmed et al., 2015) by using prior time steps to predict the next time step. The Sliding Window method steps are summarized in Algorithm 2. Indeed, after selecting the first segment, the next segment taken at multiple time steps is selected. The process is repeated until all time series data are segmented. Each segment represents an input vector, whereas the output corresponding to the label of this vector is generally annotated by an expert. The size or width of the sliding window L can change to include more or less previous time steps depending on the dataset characteristics and on the user choice.

Other pre-processing methods applied to ConvNets. Several pre-processing approaches have been conducted so far on ConvNets to improve classification or prediction of time series.

Algorithm 2: Sliding window algorithm.

Data: The number of frames or windows: n , the original time-series data/sequence: T , the size of sliding window: L , the step: s .
 Result: The set of extracted frames/windows: F
 initialization;
 instructions;
 $i = 0$;
 $n = 0$;
 $F = []$;
 while $i + L \leq \text{length}(T)$ do
 $F[n] = T[i \cdots (i + L - 1)]$;
 $i = i + s$;
 $n = n + 1$;
 end

One category of these approaches consists of turning raw time-series to a matrix representation, such as the Markov Transition Field (MTF) (Wang and Oates, 2015), the Gramian Angular Field (GAF) (Wang and Oates, 2015), stacked time-series signals (Yang et al., 2015; Wolfgang Grob, Sascha Lange, Joschka Boedecker, 2017) and Recurrence Plots (RP) (Hatami et al., 2017).

Another category of data pre-processing approaches involves data augmentation via transformations applied to data, which ensure a better ConvNet convergence. One example of such category is the window slicing method (Le Guennec et al., 2016) which trains the ConvNet using slices of the time-series input, classifies at test time each slice of the test time-series using ConvNet, and then performs majority voting to output the predicted label. Another example is the window warping method (Le Guennec et al., 2016) whose goal is to warp a randomly selected slice of a time-series (by speeding it up or down), which produces a transformed raw time-series. The resulting time-series is further converted into fixed-length input signals (e.g., instances) via window slicing. Another way of augmenting time-series is to apply small noise or smoothing on raw time-series (Sadouk and Gadi, 2017). Other transformations were also considered in (Cui et al., 2016) such as down-sampling to generate versions of a time-series at different time scales, and spectral transformations in the frequency domain by adopting low frequency to remove noise from time-series inputs.

One drawback of the first two categories is the presence of high-frequency perturbations and noise within time-series data inputs, which alters the ConvNet learning performance. As such, one solution is to turn time-series into frequency-domain signals some works (Abdel-Hamid et al., 2013; Rad et al., 2018) by applying the Fast Fourier Transform (FFT) and converting raw time-series into a set of frequency-domain signals which are then fed to the ConvNet.

1.4.1.2 Our data-level approach: The Stockwell Transform.

One drawback of FFT is that it extracts frequency signals by using a predefined fixed window length. A solution to this problem is to opt for the Stockwell Transform (ST) as our data pre-processing technique (Sadouk and Gadi, 2017; Sadouk et al., 2018), which has the property of adaptively capturing spectral changes over time without data windowing, thus ending up with a better time-frequency resolution for non-stationary signals (Stockwell et al., 1996).

The continuous S-transform of a function $h(t)$ is,

$$S(t, f) = \int_{-\infty}^{+\infty} h(t) \frac{|f|}{\sqrt{2\pi}} e^{-i2\pi ft} dt \quad (1.1)$$

A voice $S(\tau, f_0)$ can be defined as a one-dimensional function of time for a constant frequency f_0 , which highlights how amplitude and phase change over time at f_0 .

In the discrete case, the equivalent frequency domain definition of the S-transform is given by

$$S[jT, \sqrt{n}NT] = \sum_{m=0}^{N-1} H[\sqrt{m+n}NT] e^{-\frac{2\pi^2 m^2}{n^2}} e^{\frac{i2\pi mj}{N}}, n \neq 0 \quad (1.2)$$

Further details and steps about the Discrete Stockwell Transform can be found in our paper (Sadouk et al., 2018).

1.4.2 Algorithm-level approach: Adaptive convolutional layer filter

Several ConvNet models have been developed to capture the most meaningful features out of time-series signals. (Cui et al., 2016; Le Guennec et al., 2016) proposed to apply transformations to time-series such as down-sampling, slicing, or warping, in order to allow convolution filters at the 1st layer detect entire fluctuations (or peaks) within signals. Meanwhile, other studies chose to modulate the ConvNet hyper-parameters, especially model hyper-parameters such as the number hidden layers, number of feature maps per layer, the filter size of convolutional layers, the regularizer (dropout), etc. In general, there is no design rule for choosing ConvNet's hyper-parameters and these latter are either set based on the literature or on the trial-and-error process. Nonetheless, some authors believe that some model hyper-parameters could be chosen based on the nature of inputs. For instance, (Wang et al., 2016) suggest to feed the same time-series signal input to 3 branches, each having a 1st convolutional filter size different from the other so that input signal peaks with multiple sizes (lengths) could be identified. In a similar way, our proposal is to be based on designing an adaptive 1st convolutional layer filter whose size is the best at capturing most of entire peaks present within input signals. In this

section, the overall concept of our adaptive 1st convolutional layer filter is laid out.

Usually, the filter size of the 1st convolutional layer is chosen to be 3×3 , 5×5 , 7×7 or 10×10 , based on whether fine details need to be detected out of the image or large details. In other words, the size of the receptive field size is chosen so that complete features (e.g., entire variations) within an object are captured. Features can include edges, colors or textures within images and peaks or fluctuations within signals. Selecting a very small 1st layer filter may be best at capturing short variations or peaks within a time-series input signal, but may capture only a slice of a large variation of another input signal by convolving only part of it, thus failing to detect the whole fluctuation within the signal. Conversely, a relatively large filter may convolve multiple signal peaks at once and therefore no fluctuation is detected either. Therefore, in order to maximize the recognition performance, we need an optimal 1st layer filter that suits most of input signals and which convolves most of entire signal peaks within the input signals. To this end, we apply sampling whose goal is to select a subset (a statistical sample) of individuals from within a statistical population to estimate characteristics of the whole population. In other words, samples (a number of observations) are taken from a larger population to represent the population in question. In our study, the goal is to take sample peak lengths taken from randomly selected signals in order to estimate the optimal peak length of all signals. The sample can take the form of the sample mean, sample median or mode. After reviewing advantages and drawbacks of each sample form (see our paper (Sadouk et al., 2018), we deduced that the sample median is the best at estimating the population. In other words, by computing the sample median of signal peak lengths, we can obtain the optimal filter size of the 1st convolutional layer.

1.5 Experimental setup

1.5.1 Stereotypical Motor Movement Recognition task

Stereotypical Motor Movements (SMMs) are rhythmic, repetitive and predictable movements made by children with Autism Spectrum Disorders (ASD). Different SMMs have been identified, the most prevalent among them being body-rocking, complex hand movements and mouthing (LaGrow and Repp, 1984). Unfortunately, most of ASD research is centered around social and communication deficits within subjects with ASD and neglect the high prevalence of restricted and repetitive behaviors. Indeed, identifying SMM behaviors is very important in the screening, monitoring and therapy of ASD, thus potentially improving the lives of children on the spectrum.

Dataset. SMM recognition for subjects with ASD is conducted on Goodwin et al.’s dataset (Goodwin et al., 2014). This dataset is composed of raw time-series of accelerometer signals recorded from 6 subjects with ASD in a longitudinal study. After wearing three 3-axis wireless accelerometer sensors on the left and right wrist as well as on the torso, subjects started to perform SMM and non-SMM behaviors. SMMs include hand flapping, body rocking and simultaneous body rocking and hand flapping. In order to label data, subject movements were recorded with a video camera and passed on to an expert to be labeled as SMM or non-SMM. Two data collections were performed: one referred to as “Study1” which was recorded at a sampling frequency of 60Hz, and the other denoted by “Study2” which was collected on the same subjects three years later with a sampling frequency of 90Hz. So, to equalize the data, Study1 data are resampled to 90Hz (as explained in the next subsection). Each subject within each study has a two to three recorded observation sessions, where each session lasts between 9 and 39 minutes. The only exception is subject 6 within Study 2 who had only one session.

This dataset is further referred to as “SMM dataset”.

Pre-processing.

Resampling: Since each study has a different sampling frequency, the 60Hz signals of Study1 are resampled and interpolated to 90Hz (as in Study2). Then, noise within data is deleted by passing data of both studies into a high pass filter with a cut-off frequency of 0.1Hz. Next, data is segmented either into time-domain vector signals (via the Sliding Window method) or into frequency-domain one (via the Stockwell Transform).

Extraction of time-domain signals: Time-domain samples are obtained by segmenting raw data using a one second window (e.g., $L = 90$, see algorithm 2) and 88.9% overlap between consecutive data segments (e.g. $s = 10$, see algorithm 2), resulting in 90 time-point samples. And, knowing that three 3-axis acceleration signals are measured per accelerometer, an input instance will be a $90 \times 1 \times 9$ matrix with 9 denoting the number of channels ($9 = 3$ accelerometers $\times 3$ coordinates). The number of instances (N) per subject and per study is recorded in Table 1.1.

Extraction of frequency-domain signals: Frequency-domain samples are obtained by deriving ST for every other 10th sample, which at 90Hz is equal to 1/9th of a second. The next step is to choose the optimal frequency range. According to (Grobekathofer et al., 2017), human activity frequencies are between 0 and 20Hz, and 98% of the FFT amplitude is contained below 10Hz. As such, choosing the frequency range to be 0 to 10Hz did not yield good results when it comes to recognizing SMMs. Nonetheless, after considering Goodwin’s observation that frequencies in the range of 1-3Hz covered almost all SMMs (Goodwin et al., 2014), we set the new frequency range to be 0 to 3Hz, which produced satisfying classification

rates. So, for every input signal, ST is computed and a feature vector of length 50 is obtained, resulting into a frequency-domain instance of size $50 \times 1 \times 9$. The final input matrix is $90 \times 1 \times 9 \times N$ matrix for the SMM dataset, where N is the number of samples per subject and study (as shown in Table 1.1).

Table 1.1: Number of instances (N) per subject per study after data extraction in time and frequency domains.

Subject	S1	S2	S3	S4	S5	S6
Study1						
Frequency-domain	27134	17314	34814	20994	24133	30111
Time-domain	27117	17296	34796	20976	24115	30093
Study2						
Frequency-domain	30652	27594	41004	47239	29802	13642
Time-domain	30625	27576	40986	47212	29784	13633

ConvNet training and architecture.

As mentioned before, the goal is to analyze intersession variability of SMMs. As such, we train one ConvNet for each domain (time or frequency domain) for every subject per study. This way, each ConvNet will be able to identify SMMs across multiple sessions of a given subject i at a study j . A k -fold cross-validation for this subject is performed such that k is the number of sessions a participant was observed within each study, and every fold consists of data from a specific session.

The time domain ConvNet architecture consist of three sets of convolution, ReLU and pooling layers respectively stacked on top of one another with the number of convolutional filters set to $\{96, 192, 300\}$, followed by a fully connected layer with 500 neurons. On the other hand, the frequency domain ConvNet architecture is composed of two sets of convolution, ReLU and pooling layers respectively stacked on top of one another with the number of convolutional filters set to $\{96, 192\}$, followed by a fully connected layer with 500 neurons. The overall framework for the frequency-domain ConvNet is given in Figure 1.1. The $50 \times 1 \times D$ input signals (D referring to the number of channels or signals recorded from sensors) are convolved with the convolutional layer $C1$ which has 96 kernels of size 10×1 , then are followed by 3×1 subsampling $P1$ (with a stride of 2). The outputs of $P1$ are then fed to the second convolutional layer $C2$ composed of 192 filters of size 7×1 , followed by the 3×1 pooling layer $P2$. The resulting output is then vectorized and passed on to the first and second fully connected layer $F1$ and $F2$, followed by a softmax layer.

As for optimization parameters, training is performed for 10 to 40 epochs with a dropout of 0.5, a momentum of 0.9, a learning rate of 0.01, a weight decay of 0.0005, and a mini-batch size of 150.

Index of performance. Since the SMM dataset is unbalanced with normal ac-

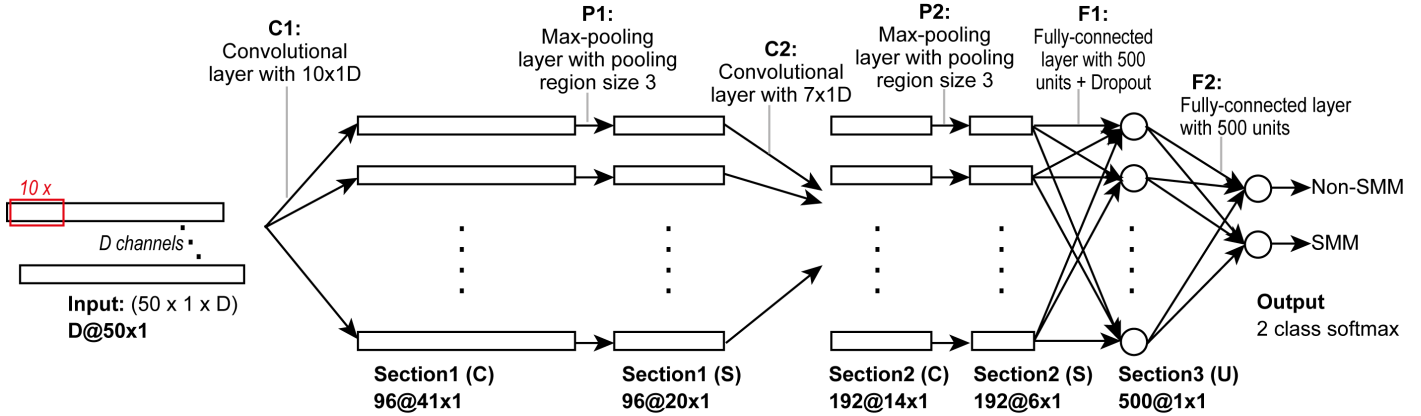


Figure 1.1: The overall frequency-domain ConvNet architecture. Symbols “C”, “S”, “U” in the parentheses of the layer tags refer to convolution, subsampling and unification operations respectively. Numbers before and after “@” refer to the feature map size. Note that *ReLU* layers located after $C1$, $C2$ and $F1$ are not showed due to the limitation of space.

tivities being much more abundant than SMM ones, the accuracy measure will not reflect a correct performance for the SMM recognition task. As such, the F1-score is used instead as our evaluation metric.

1.5.2 Human Activity Recognition (HAR) task

Dataset. The PUC dataset (Ugulino, Wallace and Cardador, D\{e\}bora and Vega, Katia and Velloso, Eduardo and Milidi\{u\}, Ruy and Fuks, 2012) is the selected dataset for training the ConvNet on the HAR task. The PUC data was collected for 8 hours of activities from 4 tri-axial ADXL335 accelerometers respectively positioned in the waist, left thigh, right ankle, and right arm. The recorded activities of labels of this data are: “sitting”, “sitting down”, “standing”, “standing up”, and “walking”. The data is further sampled at a frequency of 8Hz.

Pre-processing. Raw time-series taken from the PUC dataset is turned into either time or frequency domain signals. In time-domain, we use the Sliding Window method with the time window set to 1 second (e.g., $L = 8$) and the overlapping to 125 ms (e.g., $s = 1$). The resulting instances have the size of 8×1 . Nonetheless, an 8×1 input vector is too small to be fed to the ConvNet. As such, we chose to re-sample signals from 8 to 50 using an anti-aliasing FIR low-pass filter and compensating for the delay introduced by the filter. The final time domain input instances are of size $50 \times 1 \times 12$, 12 representing the number of channels (3 coordinates \times 4 accelerometers). In frequency domain, we re-sample raw time-series from 8 to 16 Hz before feeding them to the Stockwell Transform. This results in frequency-domain input samples of size $50 \times 1 \times 12$, where each sample contains the power of 50 frequencies in the range of 0 - 8 Hz.

ConvNet Training and hyper-parameters. In this experiment, we train a ConvNet for time domain signals and another ConvNet for frequency domain ones, using a 10-fold cross-validation. Time and frequency domain ConvNet hyper-parameters are similar to ConvNet hyper-parameters for the SMM recognition task (see section 1.5.1

Index of performance. The performance measure used for the HAR task is the accuracy, since the PUC dataset has a balanced distribution.

1.6 Experiments

1.6.1 Experiment 1

We run experiments with our data-level approach (on our frequency domain ConvNet using both PUC and SMM datasets) and our algorithm-level approach (on our time and frequency domain ConvNets using the SMM dataset), according to settings mentioned in the previous section. The inputs used are either time-domains acceleration signals of size $90 \times 1 \times 9$ for time-domain ConvNet training or frequency-domain signals of size $50 \times 1 \times 9$ for frequency-domain ConvNet training.

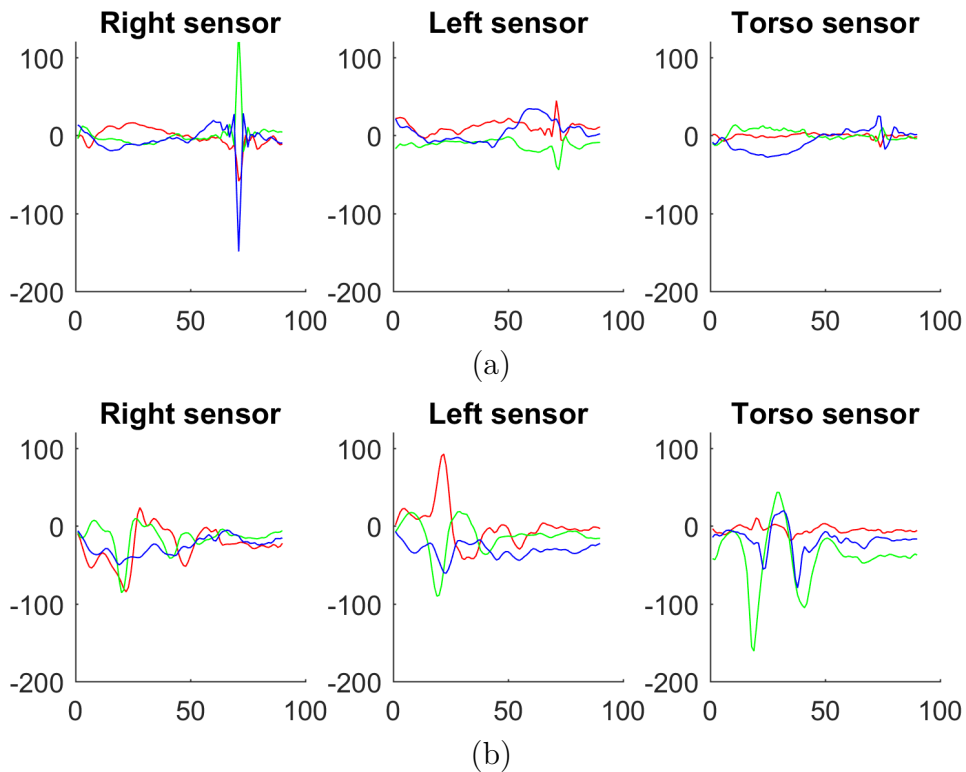


Figure 1.2: Plots (a) and (b) display time domain acceleration signal samples taken from the SMM dataset. The red, green and blue curves stand for x, y and z signals respectively. The size of samples is 90×1 (e.g., 1 second long).

Examples of signals are shown in Figures 1.2 and 1.3. Figure 1.2 illustrates

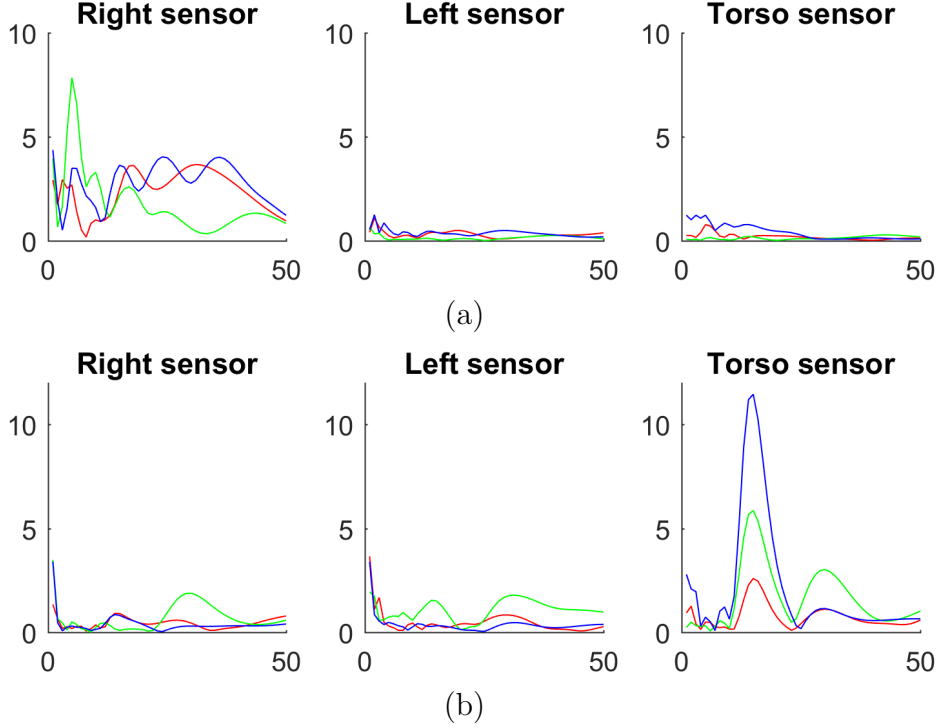


Figure 1.3: Plots (a) and (b) represent frequency domain acceleration signal samples taken from the SMM dataset. The red, green and blue curves stand for x, y and z signals respectively. The size of samples is 50×1 (e.g., 50 frequency points).

2 time-domain acceleration signal samples while Figure 1.3 displays 2 frequency-domain signal samples. Every sample is represented by x, y and z signals (corresponding to red, green and blue curves respectively) of the right wrist, left wrist and torso sensors (corresponding to the left, middle and right plots). For each signal, the x-axis represents the duration of either the time signal (in Fig.1.2) or the frequency signal (in Fig.1.3). The y-axis represents the acceleration value of that signal.

Samples illustrated in both figures are annotated as SMM behaviors. For instance, a flap-rock SMM appears in Fig.1.2.b because of high peaks in almost all three axes of right wrist, left wrist and torso signals. Meanwhile, flap SMMs are present within Fig.1.2.a and Fig.1.3.a due to the presence of fluctuations within axes of the right wrist signal only. Finally, Fig.1.3.b shows a variation in torso signals only, conveying that the sample is a rock SMM movement.

Frequency-domain samples (a) and (b) whose total length is 50 display several amplitude fluctuations (e.g., variations) whose length varies between 10 and 15 points (a frequency span of 0.2Hz). Meanwhile, the time-domain samples, which are 90 points long, show peaks contained within an interval of 7 to 20 time-points. By observing these time and frequency-signal samples, we have a rough idea about the range of the peak duration but we cannot determine the optimal peak length

that will cover most of the peaks within the ‘‘SMM dataset’’. As a consequence, the sampling method is needed.

Accordingly, for each domain, 30 random signals of atypical subjects are collected from signals containing at least one peak. Then, 30 peaks or fluctuations are selected from these signals in order to compute the sample median e.g, the optimal peak length of signals which will correspond to the optimal 1st convolutional layer filter size. The resulting medians in time and frequency domain are illustrated in histograms (a) and (b) respectively. These histograms correspond to the frequency distribution of the 30 peak lengths present within the 30 randomly selected signals. The obtained medians are 9 in time-domain and 10 in frequency-domain. These values will represent the size of our adaptive 1st convolution layer filters in time and frequency domain.

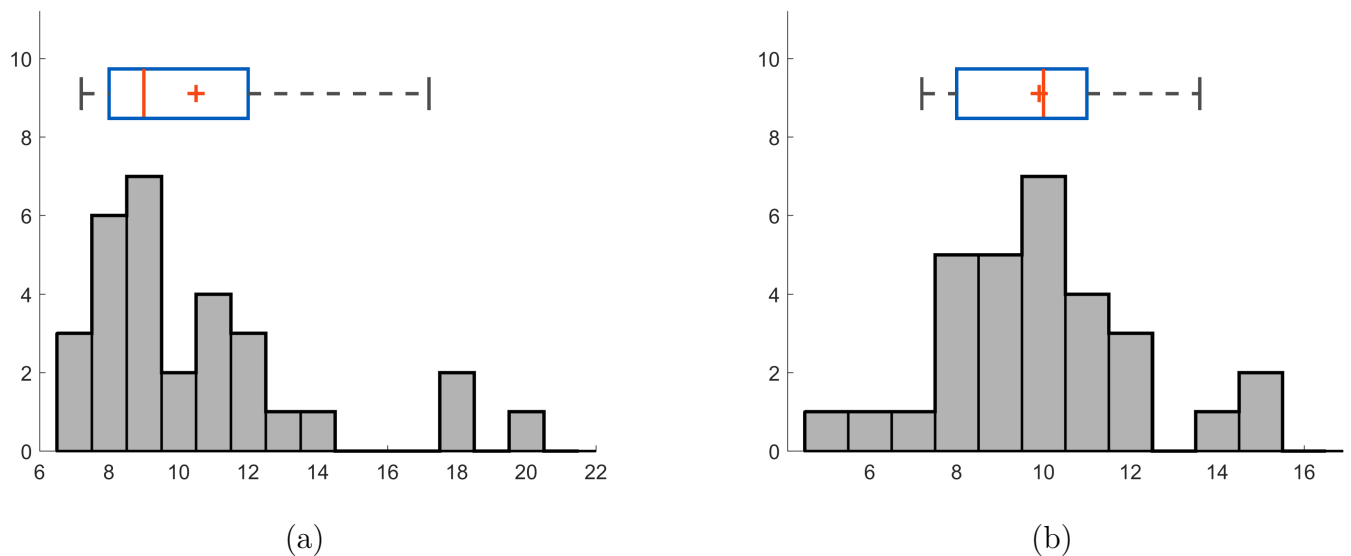


Figure 1.4: Figures (a) and (b) show histograms and box plots which represent the frequency distribution of 30 peak lengths present within 30 randomly selected time domain signals and 30 randomly selected frequency domain signals respectively.

1.6.2 Experiment 2

The purpose of this experiment is to prove that the computed sample median is indeed the best 1st convolutional layer filter size. To do so, we run experiments on signals of one atypical subject (Subject 1 of Study 1), by varying the size 1st convolution filter, resulting in different time and frequency domain ConvNet architectures. The size of the 1st convolutional layer filter ranges from 7 to 11 for both time and frequency domain ConvNets.

1.6.3 Experiment 3: Comparison with other SMM detection techniques

Moreover, to further show the efficiency of our data-, and algorithm-level approaches, our framework is compared to other SMM detection techniques, namely:

- i Support Vector Machines (Cortes and Vapnik, 1995) trained using both baseline and Stockwell features (referred to as “SVM-C”),
- ii Random Forest with Recurrence Quantification Analysis (Grobekathofer et al., 2017) (denoted as “RF-RQA”),
- iii Some deep learning approaches proposed by Rad et al. (Rad et al., 2018) namely a standard ConvNet model (referred to as “ConvNet-Rad”), a ConvNet pre-trained using a transfer learning technique (referred to as “ConvNet-TL-Rad”), and an ensemble of LSTM models (referred to as “LSTM-Rad”),
- iv Deep Belief Networks (DBNs).

SVM-C, RF-RQA and ConvNet-Rad apply the same dataset as ours and results are already available. As for the DBN, training needs to be conducted. The DBN is first pre-trained in an unsupervised manner then an additional feed-forward layer is used for the supervised learning phase to read out the top-level internal representations of the hierarchical generative model, as explained in the background section. In the unsupervised phase, learning parameters are set as follows: 450 and 810 input vector in frequency and time domains respectively, 25 hidden units per layer, 0.001 and 0.0001 as the learning rates of the unsupervised (pre-training) and supervised (training) phase respectively, 0.7 as the momentum in pre-training, 0.3 as the dropout in the supervised phase, 100 as the mini-batch size, 100 and 250 epochs for the unsupervised (pre-training) and supervised (training) phase respectively. As indicated in (Hinton, 2012), a 1-step contrastive divergence learning is applied. The trained time and frequency domain DBNs are referred to as “Time-domain DBN” and “Frequency-domain DBN” respectively.

1.7 Results

1.7.1 Results of Experiment 1.

Table 1.2 lays out results of time and frequency domain ConvNets trained on both the SMM recognition and HAR tasks. Data-level approach. For the SMM recognition task, we notice that frequency-domain ConvNets (of all subjects in all studies)

Table 1.2: Performance rates of time-, and frequency-domain ConvNets for the SMM recognition (in terms of F1-score, denoted as “F1 sc.”) and Human Activity Recognition referred to as “HAR” (in terms of accuracy). Highest rates are in bold.

SMM Recognition (F1-sc.)	Study1						Study2					
	S1	S2	S3	S4	S5	S6	S1	S2	S3	S4	S5	Mean
Time-domain ConvNet	91.23	76.76	84.95	93.38	86.41	95.11	95.97	75.67	60.17	91.68	82.55	84.90
Frequency-domain ConvNet	96.54	78.41	93.62	96.46	95.74	98.58	96.07	95.27	85.03	98.03	93.88	93.42
HAR (Accuracy)												
Time-domain ConvNet	99.90											
Frequency-domain ConvNet	95.98											

achieve higher F1-scores than time-domain ConvNets by an average of 8.52%, implying that the Stockwell Transform does a good job at suppressing noise and unnecessary details and thus contributes to detecting more relevant features. However, training time and frequency domain ConvNets on the HAR task gives us opposite results, with the time-domain ConvNets having higher accuracies than frequency domain ones by an average 3.92% (as shown in Table 1.2). These contradictory results can be explained by the difference in the chosen ST frequency range for SMM recognition and that of HAR. Indeed, in SMM recognition, the frequency range of the ST was carefully chosen to cover almost all SMMs (0-3Hz), resulting in optimal frequency-domain samples (containing full and noise-free information), which gave rise to better ConvNet learned parameters. Meanwhile, the ST frequency range for HAR (0 to 3 Hz) may be a short/small range which generated frequency-domain samples that may have lost relevant information. Indeed human activity frequencies fall between 0 and 20Hz (with 98% of the FFT amplitude contained below 10Hz). So, a proper analysis of raw time-series needs to be conducted in order to find the proper ST frequency range and thus obtain the best performance from our approach,.

Algorithm-level approach. From Table 1.2, it is hard to prove the efficiency of our adaptive 1st convolutional layer filter. Nonetheless, the efficiency of our approach can be seen through the feature representations learned by the ConvNets trained on the SMM recognition task. Indeed, we randomly choose feature representations learned by a ConvNet trained on one of the subjects within the SMM dataset.

First, we examine the type of features learned by filters of the 1st convolutional layer within the trained frequency-domain ConvNet by looking at the learned weights. Figure 1.5 displays some of these sample filters of the 1st convolutional layer. Plots (a) and (b) of this figure represent the best 2 out of the 96 1st convolutional layer filters based on their highest activations (weights across x, y and z-axis combined). Each of these plots has nine subplots: the three subplots on the

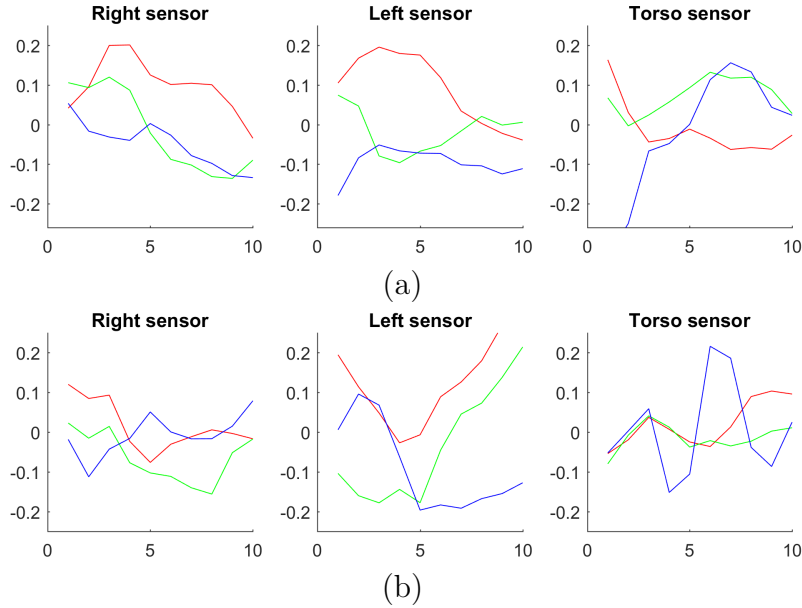


Figure 1.5: Plots (a) and (b) display weights of two filters within 1st convolutional layer of the frequency-domain ConvNet model. The red, green and blue curves stand for weights of x, y and z signals respectively.

top, middle and bottom standing for the weights of the x, y and z axis of the right, left and torso sensor respectively. We can clearly see sharp fluctuations (edges) in plots, conveying that, during the training phase, filters have learned such variations by detecting peaks and subtle changes in the input acceleration signals. Moreover, we notice that weights in the z-axis of the torso sensor show great variations compared to other weights. In the instance (a), we can see a high peak in the z-axis of the torso sensor while other weights stagnate or change with smaller peaks. Thus, weights of filter (a) must probably detect rock SMM activities. On the other hand, in filter (b), weights in all axes show variations especially weights in the z-axis of the torso and left sensors and in the x and y axis of the left sensor, which means that this filter is probably for detecting flap-rock SMMs. We repeat the same process by visualizing some 1st convolutional layer filters learned from a trained time-domain ConvNet that have the highest activations across all axes x, y and z (1.6). We can clearly see the differences between weights learned in time-domain and those learned in frequency-domain: (i) time-domain weights are less prominent (e.g, less intense in amplitude) than frequency-domain weights, (ii) the time-domain weights has different shapes from the frequency-domain weights with smoother curves, less fluctuations, more downhill (as seen through signals within the y-axis of the right and left sensor in filter –a- and signals within the z-axis of the left sensor in filter –b-), and more uphill (as seen through signals within the z-axis of the right sensor in filter –a- and signals within the y-axis of the left and torso sensor in filter –b-). From this comparison, we conclude that the ConvNet learns feature repre-

sentations specific to each domain (time or frequency). In addition, the smaller intensities, the reduced number of fluctuations and the smoother curves within the time-domain weights may explain why the time-domain ConvNet performs less in average (mean F1-score = 87.97%) than the frequency-domain ConvNet (mean F1-score = 93.23%). Furthermore, let's note that it is hard to know from 1.6 whether these filters are responsible for detecting flap, rock or flap-rock movements. Indeed, we notice small variations in weight intensities in almost all axes within each sensor.

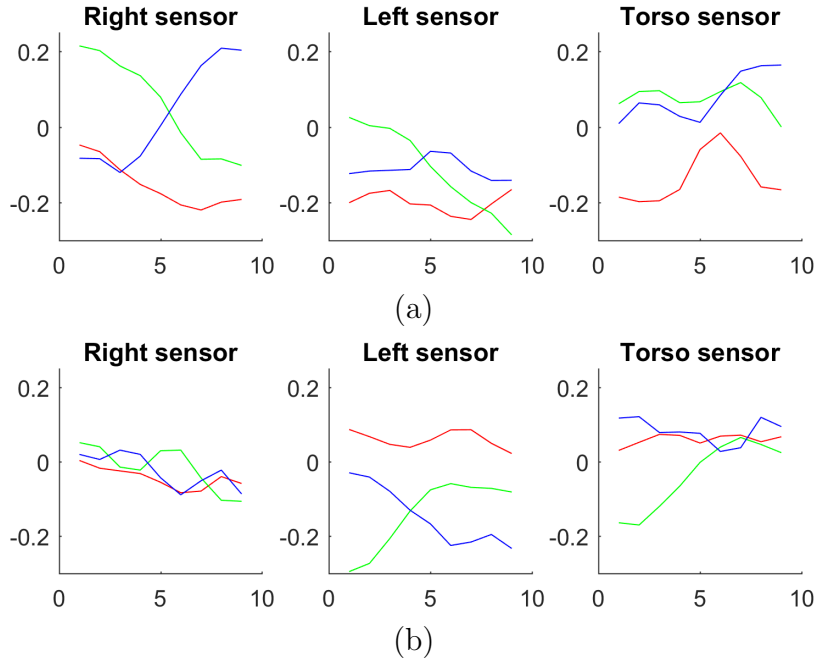


Figure 1.6: Plots (a) and (b) represent weights of two filters contained in the 1st convolutional layer of the time-domain ConvNet model. The red, green and blue curves represent weights of x, y and z signals respectively.

1.7.2 Results of Experiment 2.

Results of this experiment are reported in Figure 1.7. In time-domain, by increasing the size of the 1st convolutional layer filter from 7 (a time span of 0.078 seconds) to 9 (a time span of 0.1 seconds), the obtained classification rate rises from 87.97 to 91.23%, which implies that larger filters seem to detect more low-level details from the input signal. Meanwhile, applying larger filters such 10 (a time span of 0.11 seconds) and 11 (a time span of 0.12 seconds) seems to decrease the performance of the network. Hence, the best 1st convolutional layer time span able to retrieve the best acceleration changes is 0.1, which corresponds to an optimal peak length of 9 (e.g., $M_e(x) = 9$). On the other hand, for frequency domain signals, increasing the size of the 1st convolutional layer kernel from 7 (a frequency span of 0.14Hz) to 10 (a frequency span of 0.2Hz) seems to increase the performance rate. Also,

the performance of the framework seems to decrease when the kernel is too large, suggesting that the optimal kernel size for capturing the whole amplitude peaks is the sample median 10 (e.g., $M_e(x) = 10$).

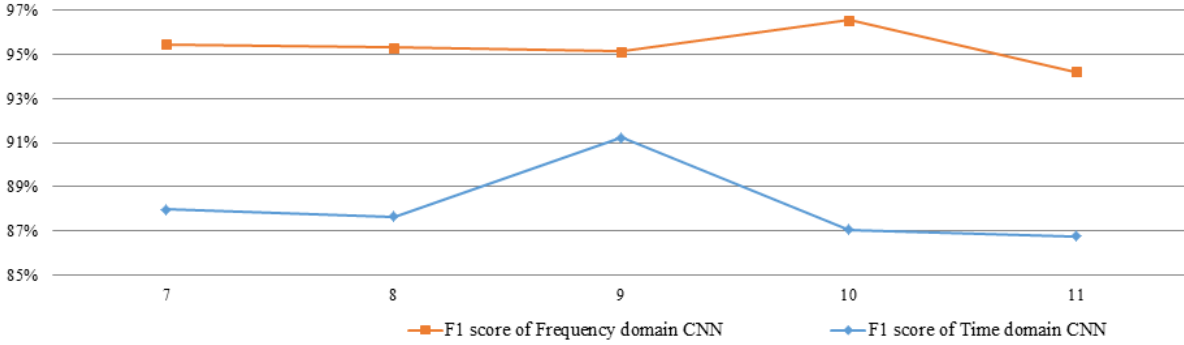


Figure 1.7: Effect of the size of 1st convolutional layer kernel on SMM recognition performance.

1.7.3 Results of Experiment 3.

Table 1.3 summarizes accuracy and F1-score results of existent works (SVM-C, RF-RQA, ConvNet-Rad, ConvNet-Rad, ConvNet-TL-Rad and LSTM-Rad), Time-domain DBN, Frequency-domain DBN, as well as our time and frequency domain ConvNet models. As shown in Table 1.3, we notice that there is a large difference between accuracy values and F1-scores which is due to the highly imbalanced distribution in the test set. The following observations are made:

Table 1.3: Results of our models and other models on 6 subjects of Study1 and Study2 datasets. For each study, two to three video sessions were performed per subject except for Subject6 (in Study2) who had only one session recorded; therefore, experiments could not be performed, which is indicated by “_”.

Experiments	S1		S2		S3		S4		S5		S6		Mean	
	Acc.	F1-sc.	Acc.	F1-sc.	Acc.	F1-sc.	Acc.	F1-sc.	Acc.	F1-sc.	Acc.	F1-sc.	Acc.	F1-sc.
Study 1														
SVM-C [17]	85.90	73.00	85.30	36.00	94.00	50.00	66.50	73.00	75.10	44.00	87.30	46.00	82.35	53.67
ConvNet-Rad [18]	_	74	_	75	_	68	_	92	_	51	90	_	74	_
ConvNet-TL-Rad [18]	_	71.00	_	73.00	_	70.00	_	92.00	_	68.00	_	94.00	_	78.00
LSTM-Rad [18]	80	_	74	_	72	_	93	_	51	_	91	_	77	_
RF-RQA [23]	83.00	_	89.00	_	93.00	_	91.00	_	80.00	_	88.00	_	87.33	_
Time-domain ConvNet	96.55	91.23	88.51	76.76	97.19	84.95	93.34	93.38	92.51	86.41	94.20	95.11	93.71	87.97
Frequency-domain ConvNet	98.80	96.54	88.93	78.41	98.89	93.62	96.56	96.46	97.77	95.74	98.33	98.58	96.55	93.23
Frequency-domain DBN	91.50	82.41	87.10	78.06	93.73	71.50	93.55	93.63	89.31	81.69	93.72	94.65	91.49	83.66
Study 2														
SVM-C [17]	71.00	43.00	79.00	26.00	99.00	3.00	90.00	86.00	73.00	72.00	_	_	82.40	46.00
ConvNet-Rad [18]	_	61	_	20	_	2	_	72	_	21	_	_	_	35
ConvNet-TL-Rad [18]	_	68	_	22	_	2	_	77	_	75	_	_	_	48.80
LSTM-Rad [18]	_	59	_	29	_	2	_	87	_	43	_	_	_	53
RF-RQA [23]	80	_	69	_	99	_	95	_	85	_	_	_	85.60	_
Time-domain ConvNet	96.88	95.97	89.53	75.67	99.10	60.17	96.88	91.68	91.69	82.55	_	_	94.81	81.21
Frequency-domain ConvNet	96.95	96.07	98.28	95.27	99.79	85.03	99.31	98.03	97.11	93.88	_	_	98.29	93.66
Frequency-domain DBN	88.60	85.57	88.84	74.09	99.11	62.70	96.91	91.65	94.04	87.92	_	_	93.50	80.39

- i Our time and frequency domain ConvNets perform better than traditional hand-crafted feature approaches (SVM-C and RF-RQA). Hence, our approach is able to detect a better feature representation from time and frequency domain signals.
- ii Our time domain ConvNets surpass the time-domain ConvNet of “ConvNet-Rad” by 30.09%, implying that regulating the ConvNet hyper-parameters (including convolutional layer filter sizes) plays a great role in increasing the SMM recognition rate.
- iii Our time and frequency domain ConvNets perform better than existent deep learning models ConvNet-Rad (by 30.09% and 38.95% respectively), ConvNet-TL-Rad (by 29.82% and 38.68% respectively), and LSTM-Rad (by 19.59% and 28.45% respectively), suggesting that a properly trained ConvNet performs better than the transfer learning with fine-tuning framework and the LSTM framework.
- iv The time-domain DBN achieves a poor score and does not even converge to a local minima, which is why the corresponding results were not displayed in Table 1.3. This is can be explained by the nature of the DBN whose each hidden layer neurons are able to signal variations only and cannot learn the place of these variations within the signal. As a reminder, DBNs do not have the concept of “local signal patch” inside weights and the “weight sharing” concept as in ConvNets. Thus, DBNs are unable to perform well on data that is not aligned by means of size and translation such as time-domain signals.
- v Nonetheless, SMM acceleration signals which are turned into frequency-domain are translation-invariant across frequency. As a result, frequency-domain DBNs yield a satisfying F1-score average of 82.03% (see Table 1.3). Furthermore, this deep neural network is able to detect better SMM feature representations (within each atypical subject) than those extracted by more complicated approaches such as RF-RQA and those extracted from computationally heavy approaches such as SVM-C. Nevertheless, our time and frequency domain ConvNet models are still better at recognizing SMMs than the frequency-domain DBN by an average of 2.57% and 11.42% respectively.

1.8 Conclusion

This chapter involves developing new approaches for time-series classification tasks. The two proposed approaches are: (i) a data-level approach which applies the Stockwell Transform to turn time-series data into frequency-domain signals, thereby

reducing noise within input signals and improving the performance of the deep learning model, (ii) an algorithm-level approach relying on an adaptive convolutional layer filter whose size is determined based on the nature of variations present within input time-series signals. Indeed, choosing the proper 1st layer filter generates features maps which are more informative about the input signals and which capture the whole peaks within input signals. Experiments were implemented on the recognition of human activities, including normal activities performed by typical subjects (i.e., the HAR task) and disorder-based activities performed by atypical subjects such as the recognition of SMM behaviors within subjects. Those experiments showed the efficiency of our time-series approaches.

Bibliography

Ossama Abdel-hamid, Hui Jiang, and Gerald Penn. Applying Convolutional Neural Networks concepts to hybrid NN-HMM model for speech recognition. In 2012 IEEE international conference on Acoustics, speech and signal processing, pages 4277–4280, 2012. ISBN 9781467300469. doi: 10.1109/ICASSP.2012.6288864.

Ossama Abdel-Hamid, Li Deng, and Dong Yu. Exploring convolutional neural network structures and optimization techniques for speech recognition. In Proceedings of the Annual Conference of the International Speech Communication Association, INTERSPEECH, pages 1173–5, 2013. ISBN 9781479903566. doi: 10.1.1.703.648.

Alessandro Antonucci, Rocco De Rosa, Alessandro Giusti, and Fabio Cuzzolin. International Journal of Approximate Reasoning Robust classification of multivariate time series by imprecise hidden Markov models. International Journal of Approximate Reasoning, 56:249–263, 2015. ISSN 0888-613X. doi: 10.1016/j.ijar.2014.07.005.

Anthony Bagnall, Jason Lines, Jon Hills, and Aaron Bostrom. Time-series classification with COTE: The collective of transformation-based ensembles. IEEE Transactions on Knowledge and Data Engineering, 27(9):1548–1549, 2016. ISSN 10414347. doi: 10.1109/ICDE.2016.7498418.

Adeline Bailly, Simon Malinowski, Romain Tavenard, Laetitia Chapel, and Thomas Guyet. Dense bag-of-temporal-SIFT-words for time series classification. In International Workshop on Advanced Analysis and Learning on Temporal Data, volume 9785 LNCS, pages 17–30, 2015. ISBN 9783319444116. doi: 10.1007/978-3-319-44412-3_2.

- Yahyia BenYahmed, Azuraliza Abu Bakar, Abdul RazakHamdan, Almahdi Ahmed, and Sharifah Mastura Syed Abdullah. Adaptive sliding window algorithm for weather data segmentation. *Journal of Theoretical and Applied Information Technology*, 80(2):322–333, 2015. ISSN 18173195.
- Bernhard E. Boser, Isabelle M. Guyon, and Vladimir N. Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on Computational learning theory*, pages 144–152, 1992. ISBN 089791497X. doi: 10.1145/130385.130401.
- Mustafa S Cetin, Abdullah Mueen, and Vince D. Calhoun. Shapelet Ensemble for Multi-dimensional Time Series. In *Proceedings of the 2015 SIAM International Conference on Data Mining*, pages 307–315, Philadelphia, PA, jun 2015. Society for Industrial and Applied Mathematics. ISBN 978-1-61197-401-0. doi: 10.1137/1.9781611974010.35.
- Zhihua Chen, Wangmeng Zuo, Qinghua Hu, and Liang Lin. Kernel sparse representation for time series classification. *Information Sciences*, 292:15–26, 2015. ISSN 00200255. doi: 10.1016/j.ins.2014.08.066.
- Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, sep 1995. ISSN 0885-6125. doi: 10.1007/BF00994018.
- Zhicheng Cui, Wenlin Chen, and Yixin Chen. Multi-Scale Convolutional Neural Networks for Time Series Classification. *arxiv.org*, 2016. ISSN 1000324X. doi: 10.3724/SP.J.1077.2009.00909.
- Shun Chen Fazle Karim, Houshang Darabi, Somshubra Majumdar. LSTM fully convolutional networks for time series classification. *IEEE Access*, 6:1662–1669, 2018.
- E Fix and JL Hodges. Nonparametric discrimination: consistency properties. Technical report, 1951.
- Matthew S Goodwin, Marzieh Haghighi, Qu Tang, Murat Akcakaya, Deniz Erdogmus, and Stephen Intille. Moving towards a real-time system for automatically recognizing stereotypical motor movements in individuals on the autism spectrum using wireless accelerometry. In *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, pages 861–872, 2014. doi: 10.1145/2632048.2632096.
- Ulf Grobekathofer, Nikolay V Manyakov, Vojkan Mihajlovic, Gahan Pandina, Andrew Skalkin, Seth Ness, Abigail Bangerter, and Matthew S Goodwin. Auto-

- mated Detection of Stereotypical Motor Movements in Autism Spectrum Disorder Using Recurrence Quantification Analysis. *Frontiers in Neuroinformatics*, 11: 9, feb 2017. ISSN 1662-5196. doi: 10.3389/fninf.2017.00009.
- Nima Hatami, Yann Gavet, and Johan Debayle. Classification of Time-Series Images Using Deep Convolutional Neural Networks. In *Tenth International Conference on Machine Vision*, volume 10696, page 106960Y, 2017. ISBN 9781510619418. doi: 10.1117/12.2309486.
- Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. ISBN 978-1-4673-8851-1. doi: 10.1109/CVPR.2016.90.
- Jon Hills, Jason Lines, Edgaras Baranauskas, James Mapp, and Anthony Bagnall. Classification of time series by shapelet transformation. *Data Mining and Knowledge Discovery*, 28(4):851–881, jul 2014. ISSN 1384-5810. doi: 10.1007/s10618-013-0322-1.
- Geoffrey E. Hinton. A practical guide to training restricted Boltzmann machines. In *Neural networks: Tricks of the Trade*, pages 599–619. 2012.
- Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, nov 1997. ISSN 0899-7667. doi: 10.1162/neco.1997.9.8.1735.
- Jeong Y S, Jeong Myong K, and Omitaomu Olufemi A. Weighted dynamic time warping for time series classification. *Pattern Recognition*, 44(9):2231–2240, 2011. ISSN 00313203.
- E. Keogh, Q. Zhu, B. Hu, Hao. Y., X. Xi, L. Wei, and C. A. Ratanamahatana. The UCR time series data mining archive. The UCR Time Series Classification/Clustering Homepage: www.cs.ucr.edu/~eamonn/time_series_data/, 2011.
- Steven J. LaGrow and Alan C. Repp. Stereotypic responding: a review of intervention research. *American Journal of Mental Deficiency*, 88(6):595–609, 1984. ISSN 0002-9351(Print).
- Quoc V. Le, Jiquan Ngiam, Zhenghao Chen, Daniel Chia, Pang W. Koh, and Andrew Y. Ng. Tiled convolutional neural networks. In *Advances in Neural Information Processing Systems 23 (NIPS 2010)*, pages 1279–1287, 2010.
- Arthur Le Guennec, Simon Malinowski, and Romain Tavenard. Data Augmentation for Time Series Classification using Convolutional Neural Networks. In

ECML/PKDD Workshop on Advanced Analytics and Learning on Temporal Data, 2016.

Jason Lines and Anthony Bagnall. Time series classification with ensembles of elastic distance measures. *Data Mining and Knowledge Discovery*, 29(3):565–592, may 2015. ISSN 13845810. doi: 10.1007/s10618-014-0361-2.

Ping Liu, Saeid Nahavandi, Abbas Kouzani, Jin Wang, and Mary F.H. She. Bag-of-words representation for biomedical time series classification. *Biomedical Signal Processing and Control*, 8(6):634–644, 2013. ISSN 17468094. doi: 10.1016/j.bspc.2013.06.004.

J Long, E Shelhamer, and T Darrell. Fully Convolutional Networks for Semantic Segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015. ISBN 9781467369640. doi: 10.1109/CVPR.2015.7298965.

Nastaran Mohammadian Rad, Seyed Mostafa Kia, Calogero Zarbo, Twan van Laarhoven, Giuseppe Jurman, Paola Venuti, Elena Marchiori, and Cesare Furlanello. Deep learning for automatic stereotypical motor movement detection using wearable sensors in autism spectrum disorders. *Signal Processing*, 144: 180–191, 2018. ISSN 01651684. doi: 10.1016/j.sigpro.2017.10.011.

Thanawin Rakthanmanon, Bilson Campana, Abdullah Mueen, Gustavo E.A.P.A. Batista, Brandon Westover, and Qiang Zhu. Addressing big data time series: Mining trillions of time series subsequences under dynamic time warping. *ACM Transactions on Knowledge Discovery from Data*, 7(3):10, 2013. ISSN 1556-472X. doi: 10.1145/2500489.

Lamyaa Sadouk and Taoufiq Gadi. Convolutional Neural Networks for Human Activity Recognition in Time and Frequency-Domain. In *Advances in Intelligent Systems and Computing*, pages 485–496, 2017. ISBN 9783319913360. doi: 10.1007/978-3-319-91337-7_43.

Lamyaa Sadouk, Taoufiq Gadi, and El Hassan Essoufi. A Novel Deep Learning Approach for Recognizing Stereotypical Motor Movements within and across Subjects on the Autism Spectrum Disorder. *Computational Intelligence and Neuroscience*, 2018:1–16, 2018. ISSN 1687-5265. doi: 10.1155/2018/7186762.

Patrick Schäfer. The BOSS is concerned with time series classification in the presence of noise. *Data Mining and Knowledge Discovery*, 29(6):1505–1530, nov 2015. ISSN 1384-5810. doi: 10.1007/s10618-014-0377-7.

- R. G. Stockwell, L. Mansinha, and R. P. Lowe. Localization of the complex spectrum: The S transform. *IEEE Transactions on Signal Processing*, 44(4):998–1001, 1996. ISSN 1053587X. doi: 10.1109/78.492555.
- Hugo Ugulino, Wallace and Cardador, D{\'}e}bora and Vega, Katia and Velloso, Eduardo and Milidi{\'}u}, Ruy and Fuks. Wearable Computing: Accelerometers' Data Classification of Body Postures and Movements. In *Advances in Artificial Intelligence-SBIA*, pages 52–61, 2012. doi: 10.1007/978-3-642-34459-6_6.
- Kini B. Venkataramana and C. Chandra Sekhar. Large margin mixture of AR models for time series classification. *Applied Soft Computing*, 13(1):361–71, 2013. doi: 10.1109/icpr.2008.4761719.
- Wenlin Wang, Changyou Chen, Wenqi Wang, Piyush Rai, and Lawrence Carin. Earliness-Aware Deep Convolutional Networks for Early Time Series Classification. *arxiv.org*, 2016.
- Zhiguang Wang and Tim Oates. Encoding Time Series as Images for Visual Inspection and Classification Using Tiled Convolutional Neural Networks. In *AAAI Workshop*, pages 40–46, 2015. ISBN 9781577357254. doi: 10.1111/j.1533-8525.1993.tb00118.x.
- Zhiguang Wang, Weizhong Yan, and Tim Oates. Time Series Classification from Scratch with Deep Neural Networks : A Strong Baseline. In *2017 International Joint Conference on Neural Networks*, pages 1578–1585, 2017. ISBN 9781509061822.
- Manuel Blum Wolfgang Grob, Sascha Lange, Joschka Boedecker. Predicting time series with space-time convolutional and recurrent neural networks. In *Proceeding of European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning*, pages 71–76, 2017. ISBN 9782875870391.
- Jian Bo Yang, Minh Nhut Nguyen, Phyo Phyo San, Xiao Li Li, and Shonali Krishnaswamy. Deep Convolutional Neural Networks On Multichannel Time Series For Human Activity Recognition. *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015. ISSN 10450823. doi: 10.3897/zookeys.77.769.
- B Zhao, H Lu, S Chen, J Liu *Journal of Systems ...*, and Undefined 2017. Convolutional neural networks for time series classification. *Journal of Systems Engineering and Electronics*, 28(1):162–169, 2017.
- Yi Zheng, Qi Liu, Enhong Chen, Yong Ge, and J. Leon Zhao. Time series classification using multi-channels deep convolutional neural networks. In *International*

Conference on Web-Age Information Management, volume 8485 LNCS, pages 298–310, 2014. ISBN 9783319080093. doi: 10.1007/978-3-319-08010-9_33.

Chapter 2

Learning from images

Image classification is an important computer vision task which has had recently an outstanding success. The idea behind image classification is to extract meaningful features from the raw pixels, and then match these features to known, labeled ones in order to achieve recognition. Thank to deep learning, the highest benchmarks accuracy rates has been reached in image recognition tasks.

In this context, we propose to conduct deep learning models for a new task, namely “Handwritten Tifnagh Character” recognition. The reason why we chose the Tifnagh-IRCAM alphabet (the Amazigh language which is widely used in North Africa) is that it remains a young alphabet and recognizing its characters is still a young field of research.

As such, the proposed models are: randomly initialized Convolutional Neural Networks (ConvNets), Deep Belief Networks (DBNs) and transfer learning with ConvNets. The first and second approach achieve satisfactory results with an accuracy of 98.95% and 95.47% respectively while the third method outperforms state-of-the art methods with an accuracy of 99.05%.

2.1 Introduction

Handwritten character recognition has been a popular research field since it has been applied for several tasks, including analysis of documents, processing of bank checks reading postal codes and reading different forms. Several techniques of handwriting recognition have been developed and refined, especially for Latin scripts, Chinese/Japanese/Hindi scripts and Arabic/Farsi scripts (Steinherz et al., 1999), (Bazzi et al., 1999), (Zhang et al., 2009), (Agrawal et al., 2009), (Amin, 1998), (Ebrahimpour et al., 2011). Recently, with the growth of means of communication, the Tifnagh alphabet has been integrated into the information systems. And only few works have been introduced to address the handwriting recognition task of such

a language. In literature, some articles dealing with offline recognition of handwritten Tifinagh character have been conducted using neural networks (Ait ouguengay and Taalabi, 2009; EL Ayachi et al., 2011; Bencharef et al., 2011; Kessab et al., 2011; Gounane et al., 2011), Support Vector Machines (Oujaoura et al., 2013), Fuzzy K-NN (Gounane et al., 2011), Finite Automata (Es Saady et al., 2011a), Hidden Markov Models (Amrouch et al., 2009, 2010, 2012a; Kessab et al., 2011) and Fuzzy K-NN combined to Bigram language model (Steinherz et al., 1999). The Tifinagh alphabet is regarded as the writing system of the Amazigh language. An ancient version of Tifinagh existed from the 3rd century BC to the 3rd century AD and was more widely used in North Africa (from the oasis of Siwa in Egypt, to Morocco passing through Libya, Tunisia, Algeria, Niger, Mali, Burkina Faso and Mauritania). Since then, several variations have been applied to the original Tifinagh. The Royal Institute of the Amazigh Culture adopted “Tifinagh-IRCAM” as the latest version of Amazigh alphabet, which is officially recognized by the International Organization of Standardization (ISO) as the basic multilingual plan (Zenkouar, 2004). Figure 1.6 shows Tifinagh characters used in Morocco along with their corresponding Latin characters. This alphabet consists of 33 phonetic entities. However, Unicode codes only 31 letters plus a modifier letter that forms the two phonetic units: ⵍ (gw) and ⵎ (kw).

The Tifinagh script is written from left to right and it employs the same punctuation marks as Latin alphabet. However, capital letters do not exist within the Tifinagh script, meaning that the notion of upper and lowercase characters is not present within this language. Regarding the shape of the characters, the majority of graphic models of the characters consist of a combination of horizontal, vertical or diagonal segments. In addition, one advantage of the Tifinagh alphabet is that it is not cursive, which makes the Tifinagh handwriting recognition an easier task. Throughout this study, we propose to learn features for the Tifinagh handwritten characters classification using two deep learning approaches: Convolutional Neural Networks (ConvNets) and Deep Belief Networks (DBNs). Recently, ConvNets and DBNs have been successfully applied into pattern recognition, whereby features are no longer hand-crafted but rather learned automatically from images. Both approaches provide an efficient feature extraction by detecting relevant all level features (low-, mid-, and high-level features). In this study, we attempt to apply ConvNets and DBNs on Tifinagh handwritten characters the same way they were applied to Latin script. Our contribution is to: train different ConvNet and DBN architectures with Tifinagh characters taken from the AMHCD database (Zenkouar, 2004) and discuss their corresponding feature representations. In the next sections, we review related works on Tifinagh Handwritten character recognition (Section 2.2). Then, we define the structure of our ConvNet and DBN architectures (Sec-

tion 2.3). In Section 2.4, experiments are laid out and corresponding results are compared to existent techniques. Finally, Section 2.5 concludes our work.

2.2 Related work

Several studies have been conducted on the Tifinagh Handwritten character recognition task. Among the first studies is the work of (Oulamara and Duvernoy, 1988) which provides a statistical approach for Berber character classification whereby straight-line segments are extracted using the Hough transform. The features are defined and measured in the parameter space obtained by the Hough transforming of the character shape (Oulamara and Duvernoy, 1988). Djematene et al. alleviate the problem of incorrect segmentation due to straight-line segments (Oulamara and Duvernoy, 1988) by proposing curved strokes instead (Djematene et al., 1997). Authors in (Ait ouguengay and Taalabi, 2009) introduce neural networks (NNs) for classifying Tifinagh characters. The database used for training the network includes Amazigh spelling patterns of different fonts and sizes. Meanwhile, the study (EL Ayachi et al., 2011) implements multilayer neural perceptrons (MLPs). Works (Amrouch et al., 2009, 2010, 2012a; Kessab et al., 2011) apply Hidden Markov Models for Tifinagh handwritten characters recognition. In (Amrouch et al., 2009), features generated by applying the Hough transform on Tifinagh characters are used as input. In (Kessab et al., 2011), neural networks are combined with Hidden Markov Models to produce a better recognition performance. Finally, (Amrouch et al., 2012a) uses a combination of continuous Hidden Markov Models and directional features. The work of (El Yachi et al., 2010) employs dynamic programming for the recognition of Tifinagh scripts. In (Es Saady et al., 2011b), authors adopt a horizontal centerline of writing as a new approach for classifying Amazigh handwriting. Positions of baselines of characters (a central line, upper and lower line of writing) are used to derive a subset of baseline independent and dependent features (Es Saady et al., 2011b). The same authors add another contribution in (Es Saady et al., 2014), where parameters such as horizontal and vertical baselines of the character are estimated, then used to derive a subset of baseline dependent features. These features are extracted on characters using the sliding window technique. Another approach that contributed to the increase of Tifinagh handwriting recognition performance is a Syntactic Approach using Finite Automata (Es Saady et al., 2011a). (Gounane et al., 2011; Steinherz et al., 1999) present the k-nearest neighbor algorithm that proposes candidates (classes) weighted by their membership degree so as to conduct the Tifinagh classification task.

2.3 Methodology

In this section, we define the main components for training our deep learning models. First, we describe the preprocessing and data augmentation phases before training 2.3.1. Next, we discuss the architecture of our ConvNet Section:Fapp3.2. Then, we go through the architecture of our DBN Section:Fapp3.3.

2.3.1 Preprocessing and data augmentation phase

First, the training dataset needs to be preprocessed then increased via multiple transformations.

2.3.1.1 Preprocessing phase

Each image of the dataset is resized to 30×30 , 60×60 or 100×100 based on the configuration of the architecture used. The ratio between height and width of images is kept the same. Then it is converted to a grayscale format. For ConvNet training, it is binarized using the Otsu threshold method (Otsu, 1979). For DBN training, we increase the contrast of images by adjusting their intensity values.

2.3.1.2 Data augmentation phase

Generally, deep neural networks yield satisfying classification rates when trained with enough training instances. In other words, with a lack in training samples, these networks overfit and provide misleading (incorrect) classifications at the testing phase. The best way to alleviate this problem is by enlarging the training dataset via data augmentation. Data augmentation is applied during ConvNet training and during DBN supervised training. In this context, our characters are replicated with several transformations. To this end, we randomly select half the instances of the batch and distort them by a random rotation between -20 and 20 degrees and by rescaling the image width and height independently with a random resizing coefficient ranging from 0.7 to 1.

2.3.2 ConvNet model

As depicted in Table 2.1, our ConvNet is composed of seven layers: four convolutional layers and three fully connected. The selected activation function is the rectified linear unit (ReLU). The ConvNet takes a grayscale image (one channel) as input. Each convolutional layer convolves the output of its previous layer with a set of learned kernels, followed by the ReLU, and Max-pooling as an optional layer. The first and fourth convolutional layers are followed by a max-pooling layer. The

fourth convolutional layer is then followed by two fully connected layers (FC1, FC2) each of which has 500 neurons as output. Finally, one last fully connected layer is added (FC3); and its output is fed into a softmax function which computes the probability distribution over the different classes. The final layer has 31 output units corresponding to the 31 Tifinagh character labels.

Table 2.1: ConvNets architecture.

Layer	Type	Kernel size	Number of kernels	Stride	Pad	Output Size
	Input	-				30×30
L1	Conv	4×4	20	1	0	27×27
	ReLU	-	-	-	-	27×27
	Maxpool	2×2	-	2	0	13×13
L2	Conv	3×3	50	1	0	11×11
	ReLU	-	-	-	-	11×11
L3	Conv	3×3	150	1	1	11×11
	ReLU	-	-	-	-	11×11
L4	Conv	3×3	150	1	1	11×11
	ReLU	-	-	-	-	11×11
	Maxpool	2×2	-	2	0	5×5
L5	Conv (FC1)	5×5	500	1	0	1×1
	ReLU	-	-	-	-	1×1
	Dropout	-	-	-	-	1×1
L6	Conv (FC2)	1×1	500	1	0	1×1
	ReLU	-	-	-	-	1×1
	Dropout	-	-	-	-	1×1
L7	Conv (FC3)	1×1	32	1	0	1×1

As depicted in Table 2.1, the filter size of the first convolutional layer is chosen to be 4×4 , which is relatively large compared to the input image size 30×30 . Indeed, unlike photo images that have texture information and require a very small filter size (relative to input image size) to capture small shapes, we are dealing with grayscale character images with no texture. Accordingly, applying smaller filters is not adequate and adopting large filters (relative to the input image) detects more structured context than textured information.

2.3.3 DBN model

2.4 Experiments and results

We conduct our experimental studies on handwritten Tifinagh character recognition using ConvNet and DBN methods. These two deep learning approaches are

evaluated on AMHCD database (Es Saady et al., 2011c). Results are detailed and discussed in the next subsections.

2.4.1 Experiments using Convolutional Neural Networks

Simulation details. In our work, we used the stochastic (mini-batch) gradient descent as our gradient-based optimization method. The learning rate was chosen to be 0.01 for the first 5 epochs and 0.001 for the 8 remaining epochs. The momentum is fixed to 0.9 while the dropout rate is set to a high value 0.5 to avoid overfitting. Our batch size is set to 50.

2.4.1.1 Experimental setup

Using k -cross validation with $k = 3$, we randomly divide the dataset into three equal parts, and choose two parts as training data and the remaining part as testing/validation data. We take the average accuracy of the validation set as the final evaluation.

2.4.1.2 Results

After training the ConvNet model explained in section 3.2 with 30×30 input images, we achieve a classification accuracy of 97.99% (Table 2.2). Furthermore, we investigate the performance of different ConvNet architectures on the AMHCD dataset by a parametrization of the size and number of layers as well as kernel sizes. For the 60 by 60 input images, we apply an architecture that consists of 8 layers (5 convolutional and 3 fully connected) as follows: 5×5 Conv., ReLU, Maxpool, 4×4 Conv., ReLU, Maxpool, 3×3 Conv., ReLU, 3×3 Conv., ReLU, 3×3 Conv., ReLU, Maxpool, 6×6 Conv., ReLU, Dropout, 1×1 Conv., ReLU, Dropout, 1×1 Conv. Since the input image size is now 60 by 60, the filter size of the first convolutional layer is set to be 5×5 instead of 4×4 and an extra layer is added (a convolution, a ReLU and Maxpool). This more complex architecture yields an accuracy of 98.25%, which is slightly better than the result we got for the previous architecture. Due to the fact that 5^{th} layer filters adopted for 60 by 60 images are proportionally smaller w.r.t the input image size than 5^{th} layer filters adopted for 30 by 30 images, they may be susceptible to capture more entire edges. Therefore, one way to improve recognition of images is to : (i) choose a larger input image size which results in keeping more details about the image, and (ii) choose a 5^{th} layer filter size that is proportionally smaller w.r.t the input image so that the finest details (edges) of the image could be detected. Furthermore, we set input images to be 100×100 and use the same ConvNet architecture as the one applied to the 60×60 image, except that the filter size of the first convolutional layer is chosen to be 7×7 . Note

that st layer filters adopted in this architecture are proportionally smaller w.r.t the input image size than st layer filters adopted for 60 by 60 images. The accuracy of this architecture is surprisingly lower (97.95%) compared to the two previous architectures. This means that, even if we adopt a larger input image size and thus an image with more details, choosing a st layer filter size that is too small w.r.t the input image leads to a failure in capturing the entire edge, catching only a part of the edge, therefore losing valuable information about the image.

As a result, in order to increase the accuracy of a ConvNet model, one way is to keep the resolution/frame of the input image as big as possible and keep the st layer filter size as proportionally small w.r.t the input image as possible (but not too small to lose edges within images).

These results can be explained by the dimension of images present in the AMHCD dataset. The original input image sizes are neither uniform nor fixed; they vary from 20×30 pixels to 150×180 pixels. However most of the images are around the mean size of 60×60 . That is why the architecture adapted for 60×60 images gave the best results, followed by the architecture for 30×30 images, then followed by the architecture for 100×100 images, knowing that we lose quality and information about the image when decreasing or increasing its size.

Table 2.2: Comparative results using different Transfer Learning ConvNets.

Input image size	Accuracy at test time (%)
30×30	97.99
60×60	98.25
100×100	97.95

2.4.1.3 Feature representations

We analyze the type of visual features learned by neurons of the first convolutional layer by plotting the corresponding weight matrices for the three architectures mentioned above (Figure 2.1). We notice that, the larger the first layer filters are, the clearer the edges and contours become. Indeed, going from 4 by 4 filter weights in (a) to 7 by 7 filter weights in (b) makes edges stronger and look like Gabor filters.

2.4.2 Experiments using Deep Belief Networks

We further investigate the performance on recognizing Tifinagh characters of AMHCD database using the unsupervised feature learning approach DBN. After preprocessing images and expanding the size of the training set by the augmentation technique mentioned in section 2.4.1, the DBN is first pre-trained in an unsupervised manner. Next, an additional feed-forward layer is used to read out the top-level internal

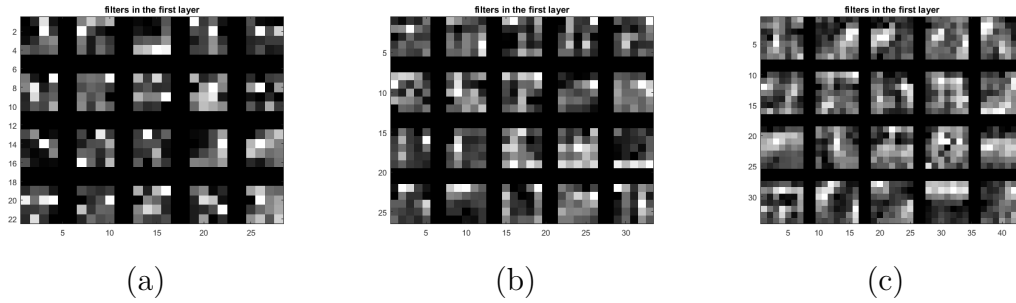


Figure 2.1: Filter weights on the first layer for the three ConvNet architectures: (a) 4 by 4 filter weights for the architecture adopted for 30×30 images, (b) 5 by 5 filter weights for the architecture adopted for 60×60 images, (c) 7 by 7 filter weights for the architecture adopted for 100×100 images.

representations of the hierarchical generative model (Fig. 2.2).

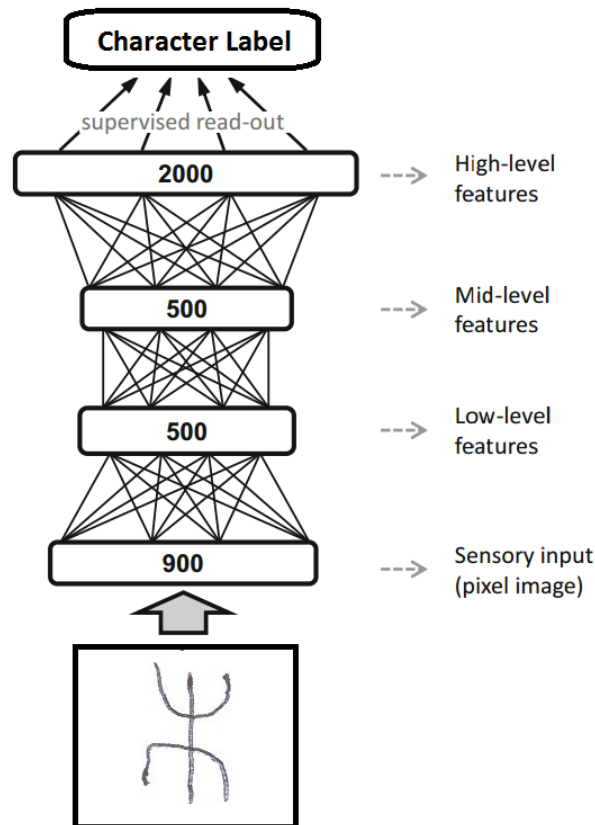


Figure 2.2: Structure of the chosen DBN architecture (500-500-2000-31). Unidirectional connections stand for unsupervised learning, whereas directed arrows on the top layer entail supervised learning. The number on each layer denotes the layer size.

2.4.2.1 Architecture and Simulation details

We consider four different deep architectures, with a varying number of hidden neurons. The first architecture of DBN is 500-500-2000-31, which consists of three

RBM, with 500 hidden neurons in the first and second layers, and 1000 hidden neurons in the third layer. The second architecture is 500-1000-2000-31, the third 1000-1000-2000-31, the fourth 1000-1000-1000-2000-31. The size of the first and second hidden layer is varied between 500 and 1000 neurons across different architectures. In addition, the number of RBMs is varied between 3 and 4 hidden layers respectively. For the unsupervised learning, learning parameters are set according to suggestions published by Hinton (Hinton et al., 2006). The 1-step contrastive divergence learning is used with a learning rate of 0.0001, a momentum coefficient of 0.7, and a dropout of 0.3. The pre-training of the RBMs –unsupervised learning– is performed for 200 epochs using a mini-batch of size 200. On the other hand, parameters of the supervised learning are set to 0.0001 for the learning rate and 200 for the batch size. The number of epochs varied according to the minimum value of the error rate.

2.4.2.2 Results

Classification accuracy for all the considered architectures, measured by correct classification rates, is reported in Table 2.3. For instance, the architecture 500-500-2000-31 yields an accuracy of 93.68%. Next, we observe that, the larger the size of the hidden layers, the better the recognition accuracy. Indeed, increasing the number of hidden units in the first layer from 500 to 1000 improves the classification performance by 0.72, suggesting that having learned more low-level features helps classifying the Tifinagh characters. Similarly, going from 500 to 1000 units in the second hidden layer increases the accuracy by 1.07%, which implies that the more hidden units we have in the second layer, the more mid-level features we capture. Therefore, increasing low and mid-level features results in better high-level features and a better character recognition.

Table 2.3: Results of different DBN architectures.

Architecture (# units per layer)	Character recognition accuracy	
	Training data (%)	Testing data (%)
500-500-2000-31	98.55%	93.68%
500-1000-2000-31	99.31%	94.75%
1000-1000-2000-31	99.75%	95.47%
1000-1000-1000-2000-31	98.90%	94.13%

2.4.2.3 Feature representations

We analyze the type of visual features learned by units of the first hidden layer by plotting weights of some unit samples for the three architectures mentioned above

(Fig. 2.3). Weights of each 1st hidden layer unit are converted from a 900 raw vector into a 30×30 matrix. From the plots of all three architectures, we notice that some unit weights show low-level features (with simple Gaussian filters) while other show more complex features such as developed edge detectors and Gabor filters with different phases and orientations.

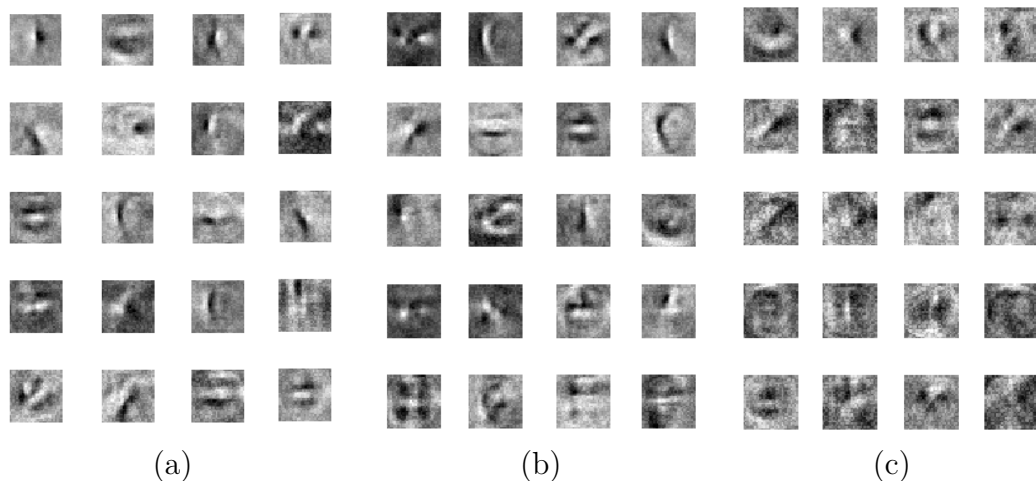


Figure 2.3: 20 sample weights taken from the units of the 1st hidden layer for three DBN architectures. Each sample is a 900 raw vector of a single 1st hidden unit sample that is reshaped into a 30×30 matrix. (a) weights for the 500-500-2000-31 architecture, (b) weights for the 500-1000-2000-31 architecture, (c) weights for the 1000-1000-2000-31 architecture.

2.4.3 ConvNet vs DBN results

In order to explain the difference in accuracy between ConvNets and DBNs, let's first discuss properties and advantages of each of these two approaches. DBN is a generative unsupervised model that has the following advantages: (i) it acts as some special kind of regularizer and thus reduces overfitting (lower performance in training data, but better generalization power), (ii) it gives a better initialization of the weights in the deep neural network, which can then be adjusted or refined in a supervised manner, leading then to better local optima, (iii) it is useful when we have many unlabeled data and few labeled data. However, DBN is not so powerful when the number of labeled training samples is large enough, which is the case for our AMHCD database. In ConvNet, even though the weights are randomly generated and the first few layers in deep networks change very slowly due to diffusion of gradients, they will eventually change with enough training samples and a long enough training time. That explains why ConvNet performs better than DBN when using the relatively large dataset AMHCD. Another reason why DBN performs lower than ConvNet is that the all the neurons of the DBN will try to learn not only the Tifinagh characters but also the place of those characters in the

images because it will not have the concept of 'local image patch' inside weights. Nevertheless, ConvNet has the weight sharing property and the subsampling concept that make the network translation-invariant and rotation-invariant. And, we know that the AMHCD is composed of hand-drawn characters that are not located in the same place and at the same angle.

2.4.4 Confusion errors and character similarity

Based on both architectures mentioned above, we compute the individual accuracy for each class of the Tifinagh characters in the test data. The results obtained are shown in Table 2.4. For the ConvNet and DBN models, we notice that some characters such as \odot (s), \circ (r), \mathcal{Q} (rr) have a relatively lower classification rate than others. This may be due to the bad writing of some characters in the database whose classification is difficult even for a human operator. Figure 2.4 illustrates some badly written letters in the database. Furthermore, due to the structural similarity of characters such as characters \mathcal{K} (z) with \mathcal{K} (zz) and \mathbf{E} (dd) with \mathbf{E} (tt), these characters are misclassified by the DBN while being correctly classified by the ConvNet.

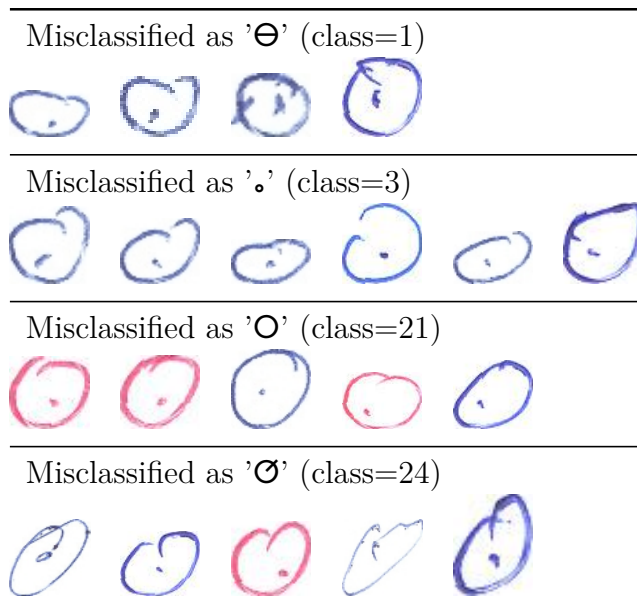


Figure 2.4: Some samples of the character \odot (class=23) badly written in the database.

2.4.5 Comparative results

Table 2.5 displays results of previous Tifinagh handwritten recognition works as well as our results. We notice that our ConvNet model outperforms all traditional works except the work of (Aharrane et al., 2017) which consists of a combination of

Table 2.4: Individual recognition rate for each character for both DBN (architecture 1000-1000-2000) and ConvNet models.

Character	Images correctly classified by the DBN		Images correctly classified by the ConvNet		
	# images	Accuracy (%)	# images	Accuracy (%)	
A	◦	260	100.00	259	99.62
Aa	⋈	250	96.15	257	98.85
B	⊖	258	99.23	259	99.62
Ch	Ⓒ	239	91.92	249	95.77
D	∧	258	99.23	257	98.85
Dd	Ⓔ	245	94.23	258	99.23
E	⊘	248	95.38	260	100.00
F	Ⓗ	243	93.46	255	98.08
G	⌘	243	93.46	253	97.31
gh	⋈	248	95.38	255	98.08
h	⓪	251	96.54	258	99.23
hh	∧	256	98.46	260	100.00
i	⋈	251	96.54	250	96.15
j	Ⓘ	246	94.62	257	98.85
k	Ⓕ	244	93.85	258	99.23
kh	⌘	249	95.77	260	100.00
l	Ⓛ	260	100.00	260	100.00
m	Ⓖ	256	98.46	259	99.62
n	Ⓛ	259	99.62	260	100.00
q	Ⓔ	260	100.00	259	99.62
r	⓪	249	95.77	245	94.23
rr	⓪	241	92.69	248	95.38
s	⓪	207	79.62	225	86.54
ss	⓪	255	98.08	256	98.46
t	⋈	254	97.69	258	99.23
tt	Ⓔ	245	94.23	259	99.62
u	⊘	251	96.54	258	99.23
w	Ⓛ	260	100.00	260	100.00
y	⋈	249	95.77	254	97.69
z	⌘	226	86.92	258	99.23
zz	⌘	234	90.00	255	98.08

classifiers. Meanwhile, our DBN model outperforms several state-of-the-art works except (Es Saady et al., 2011b; Amrouch et al., 2012b; Aharrane et al., 2017). This suggests that deep learning approaches are able to automatically capture the low-, mid-, and high-level features from Tifinagh characters. This suggests that automatically learned feature representations via deep learning methods are globally more efficient than engineered ones in terms of recognizing handwritten Tifinagh characters.

Table 2.5: Comparison between our method and other existing approaches. Images used are taken from AMHCD. The abbreviation % stands for the percentage with respect to the total size of images used.

Method used	# images	# Training images and %	# Test images	Accuracy
(Djematene et al., 1997)	1700 (06.60%)	1000 (59%)	700	92.30
(Es Saady et al., 2011b)	20,150 (78.28%)	18,135 (90%)	2015	96.32
(Es Saady et al., 2014)	24,180 (93.93%)	21,762 (90%)	2418	94.96
(Amrouch et al., 2012b)	20,180 (93.93%)	16120 (80%)	8060	97.89
(Aharrane et al., 2017)	24,180 (93.93%)	16,926 (70%)	7254	99.03
Our DBN (Sadouk et al., 2017)	24,180 (93.93%)	16,120 (67%)	8060	95.47
Our ConvNet (Sadouk et al., 2017)	24,180 (93.93%)	16,120 (67%)	8060	98.25

2.5 Conclusion

In this study, we proposed two deep learning frameworks for Tifinagh handwritten character recognition. We presented Convolutional Neural Network (ConvNet) and Deep Belief Network (DBN) models as powerful automatic approaches for classifying Tifinagh handwritten characters. By using the AMHCD database with 31 class labels of character patterns, we showed that the ConvNet surpassed existent works with an accuracy of 98.25%, whereas the DBN achieved satisfying results with 95.47% accuracy. Even with a lower performance than ConvNet, DBN is yet faster and its architecture structure is less complex. As a perspective, we could apply Convolutional Deep Belief Network approach as a generative model for learning hierarchical representations of Tifinagh handwritten characters. We believe that replacing RBMs by CRBMs by sharing the weights between the hidden and visible layers among all locations in an image will give higher recognition accuracy. Another perspective would be to build an ensemble of ConvNet and traditional methods at test time to provide the best classification. A different research direction would be to develop a framework robust to misclassified characters.

Bibliography

- Pooja Agrawal, M Hanmandlu, and Brejesh Lall. Coarse Classification of Handwritten Hindi Characters. *Science And Technology*, 10:43–54, 2009.
- N. Aharrane, A. Dahmouni, K. El Moutaouakil, and K. Satori. A robust statistical set of features for Amazigh handwritten characters. *Pattern Recognition and Image Analysis*, 27(1):41–52, jan 2017. ISSN 1054-6618. doi: 10.1134/S1054661817010011.
- Y. Ait ouguengay and M. Taalabi. Elaboration d’un réseau de neurones artificiels pour la reconnaissance optique de la graphie amazighe: Phase d’apprentissage. *Systèmes intelligents-Théories et applications*, 2009.
- Adnan Amin. Off-line Arabic character recognition: The state of the art. *Pattern Recognition*, 31(5):517–530, 1998. ISSN 00313203. doi: 10.1016/S0031-3203(97)00084-8.
- M Amrouch, Y. Es Saady, A Rachidi, M. El Yassa, and D. Mammass. Printed amazigh character recognition by a hybrid approach based on hidden markov models and the hough transform. In *International Conference on Multimedia Computing and Systems -Proceedings*, pages 356–360, 2009. ISBN 9781424437573. doi: 10.1109/MMCS.2009.5256672.
- M Amrouch, A Rachidi, M El Yassa, and D Mammass. Handwritten amazigh character recognition based on hidden Markov models. *ICGST-GVIP Journal*, 10(5):11—18, 2010.
- M. Amrouch, Y. Es-Saady, A. Rachidi, M. El Yassa, and Mammass D. A New Approach Based On Strokes for Printed Tifnagh Characters Recognition Using the Discriminating Path-HMM. *International Review on Computers and Software (IRECOS)*, 7(2), 2012a.
- M Amrouch, A. Rachidi, M. El Yassa, and D. Mammass. Handwritten Amazigh Character Recognition System Based on Continuous HMMs and Directional Features. *International Journal of Modern Engineering Research*, 2(2):436–441, 2012b.
- Issam Bazzi, Richard Schwartz, and John Makhoul. An omnifont open-vocabulary OCR system for English and Arabic. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(6):495–504, 1999. ISSN 01628828. doi: 10.1109/34.771314.

- O Bencharef, M Fakir, B. Minaoui, and B. Bouikhalene. Tifnagh Character Recognition Using Geodesic Distances, Decision Trees & Neural Networks. *International Journal of Advanced Computer Science and Applications*, (Special Issue on Artificial Intelligence):51–55, 2011.
- A Djematene, B Taconet, and A Zahour. A geometrical method for printed and handwritten Berber character recognition. In *Proceedings of the Fourth International Conference on Document Analysis and Recognition*, volume 2, pages 564–567, 1997. ISBN 0-8186-7898-4. doi: 10.1109/ICDAR.1997.620564.
- R Ebrahimpour, R.D. Vahid, and B.M. Nezhad. Decision Templates with Gradient based Features for Farsi Handwritten Word Recognition. *International Journal of Hybrid Information Technology*, 4(1):1–12, 2011.
- R. EL Ayachi, M. Fakir, and B. Bouikhalene. Recognition of Tifnaghe Characters Using a Multilayer Neural Network. *International Journal Of Image Processing*, 5(2):109–118, 2011. doi: 10.1109/MMCS.2009.5256681.
- R. El Yachi, K. Moro, M. Fakir, and Bouikhalene B. On the Recognition of Tifnagh Scripts. *Journal of Theoretical and Applied Information Technology*, 20(2):61–66, 2010.
- Y. Es Saady, A. Rachidi, M. El Yassa, and D. Mammass. Printed Amazigh Character Recognition by a Syntactic Approach using Finite Automata. *ICGST-GVIP Journal*, 10(2):1–8, 2011a.
- Y. Es Saady, A Rachidi, M. El Yassa, and D. Mammass. Amazigh Handwritten Character Recognition based on Horizontal and Vertical Centerline of Character. *International Journal of Advanced Science and Technology*, 33:33–50, 2011b.
- Y. Es Saady, A Rachidi, M. El Yassa, and D. Mammass. AMHCD : A Database for Amazigh Handwritten Character Recognition Research. *International Journal of Computer Applications*, 27(4):44–48, 2011c.
- Y. Es Saady, M Amrouch, A. Rachidi, M. El Yassa, and D. Mammass. Handwritten Tifnagh Character Recognition Using Baselines Detection Features. *International Journal of Scientific & Engineering Research*, 5(4):1177–1182, 2014.
- S Gounane, M Fakir, and B Bouikhalene. Recognition of Tifnagh Characters Using Self Organizing Map And Fuzzy K-Nearest Neighbor. *Global Journal of Computer Science and Technology*, (15):29–33, 2011. ISSN 0975-4350.

- Geoffrey E. Hinton, Simon Osindero, and Yee-Whye Teh. A Fast Learning Algorithm for Deep Belief Nets. *Neural Computation*, 18(7):1527–1554, jul 2006. ISSN 0899-7667. doi: 10.1162/neco.2006.18.7.1527.
- El Kessab, C Daoui, K Moro, B Bouikhalene, M Fakir, and B El Kessab. Recognition of Handwritten Tifinagh Characters Using a Multilayer Neural Networks and Hidden Markov Model. *Global Journal of Computer Science and Technology*, 11, 2011. ISSN 0975-4350.
- Nobuyuki Otsu. A Threshold Selection Method from Gray-Level Histograms. *IEEE Transactions on Systems, Man, and Cybernetics*, 9(1):62–66, 1979. ISSN 0018-9472. doi: 10.1109/TSMC.1979.4310076.
- M Oujaoura, R El Ayachi, B Minaoui, M Fakir, B Bouikhalene, and O Bencharef. Invariant descriptors and classifiers combination for recognition of isolated printed Tifinagh characters. *International Journal of Advanced Computer Science and Applications*, (Special Issue on Selected Papers from Third international symposium on Automatic Amazigh processing):22–28, 2013.
- A Oulamara and J. Duvernoy. An application of the Hough transform to automatic recognition of Berber characters. *Signal Processing*, 14(1):79–90, 1988. ISSN 01651684. doi: 10.1016/0165-1684(88)90045-X.
- L. Sadouk, T. Gadi, and E.H. Essoufi. Handwritten tifinagh character recognition using deep learning architectures. In *Proceedings of the 1st International Conference on Internet of Things and Machine Learning*, page 59, 2017. ISBN 9781450352437. doi: 10.1145/3109761.3109788.
- Tal Steinherz, Ehud Rivlin, and Nathan Intrator. Offline cursive script word recognition - a survey. *International Journal on Document Analysis and Recognition*, 2(2-3):90–110, 1999. ISSN 1433-2833. doi: 10.1007/s100320050040.
- L. Zenkour. L’écriture Amazighe Tifinaghe et Unicode. *Etudes et documents berbères*, 22:175–192, 2004.
- Zhiyi Zhang, Lianwen Jin, Kai Ding, and Xue Gao. Character-SIFT: A novel feature for offline handwritten Chinese character recognition. In *Proceedings of the International Conference on Document Analysis and Recognition, ICDAR*, pages 763–767, 2009. ISBN 9780769537252. doi: 10.1109/ICDAR.2009.27.

Part IV

Learning from few data (with lack of data)

As we know, deep learning models in general and Convolutional Neural Networks (ConvNets) in particular require large datasets during training. Indeed, they require excessive amount of labeled (e.g., annotated) data to avoid overfitting and obtain full convergence of parameters. Although in recent years several labeled datasets have become available, some fields such as medicine experience a lack of annotated data as manually annotating a large set requires human expertise and is time consuming. For instance, labeling acceleration signals of autistic children as SMMs or non-SMMs requires knowledge of a specialist. The conventional approach to deal with this kind of problem is to act on the data level via data augmentation by applying transformations to the existing data, as shown in section 1.4.1.1. Data augmentation achieves slightly better time-series classification rates, but the ConvNet is still prone to overfitting. In this part, we show how to solve the issue of lack of data when training ConvNets. In this matter, we attempt to present solutions to this problem by proposing the following approaches: two multi-modal learning approaches (an ensemble learning approach and a transfer learning one) as well as an approach based on a ConvNet voting technique at test time.

- i a ConvNet Transfer learning approach with a source domain that is more global and more general than the target. This approach is applied on an Optical Character Recognition (OCR) task where the target domain is any Handwritten Alphabet with lack of data and the source domain is the Phoenician handwritten alphabet,
- ii another algorithm-level approach adapted to time-series classification tasks with limited annotated data, which is a global, fast and light-weight framework based on a transfer learning technique with a source learning task similar or different but related to the target learning task. This approach is conducted on a medical application: the recognition of Stereotypical Motor Movements of autistic subjects,
- iii an ensemble learning approach which combines between different ConvNets, some of which employing knowledge transfer. The proposed ensemble learning framework embeds between features learned by training different ConvNet models including randomly initialized ConvNets and ConvNets already pre-trained on tasks different from the target domain task. This framework is implemented on a computer vision task, namely the sketch recognition task,
- iv a novel augmentation technique at the training phase and a voting approach at the testing phase. Implementation was also conducted on the sketch recognition task.

Each of these approaches is presented in the chapters below.

Chapter 1

Multimodal learning: Transfer learning approach with fine-tuning

Many cognitive tasks require reusing knowledge acquired in one domain to perform one or more tasks in a somewhat related domain, without needing to learn the new task completely from scratch. In particular, given a source domain with a source learning task and a target domain with a target learning task, transfer learning aims to improve learning of the target predictive function using the knowledge in the source domain and source task, with the assumption that source and target domains share a common feature space. This functionality comes in handy especially when the target domain has a limited number of training data. When dealing with Convolutional Neural Networks (ConvNet), transfer learning (TL) consists of using a ConvNet model already pre-trained on a source domain dataset and updating/fine-tuning part of or all of its weights by training them on the target dataset.

In this chapter, we show how feature learning with lack of training data using ConvNet TL can be improved by selecting the appropriate source domain. Indeed, we demonstrate that picking a source domain that is more global and more general than the target domain results in a better feature learning. To this matter, we choose an application on Optical Character Recognition (OCR) task where the target domain is any Handwritten Alphabet with lack of data and the source domain is the Phoenician Handwritten Alphabet. Let's note that the Phoenician alphabet is the ancestor of almost all current alphabets.

The outline is as follows: an overview about the Phoenician alphabet is introduced (chapter 1.1). Then we describe how our new Phoenician Handwritten Characters' Database has been implemented (chapter 1.2). In chapter 1.3, we present the ConvNet model used for the source domain task (e.g., the Phoenician Handwritten Alphabet recognition). Afterward, we introduce the light-weight transfer learning system for the recognition of alphabets which lack of annotated data (1.4). Finally,

chapter 1.5 concludes our work.

1.1 Overview of the Phoenician alphabet

The Phoenician alphabet is the oldest verified alphabet in the wider sense of the term "alphabet". The earliest Phoenician inscription found dates from the 11th century BC (Fig.1.1). It was used to write Phoenician, a Northern Semitic language, used by the ancient civilization of Phoenicia in modern-day Syria, Lebanon, and northern Israel. The Phoenician alphabet was widely-used since Phoenicians traded around the Mediterranean world and built some cities North Africa and Southern Europe. The origins of most alphabetic writing systems are derived from the Phoenician alphabet, including Greek, Etruscan, Latin, Arabic and Hebrew, as well as the scripts of India and East Asia.

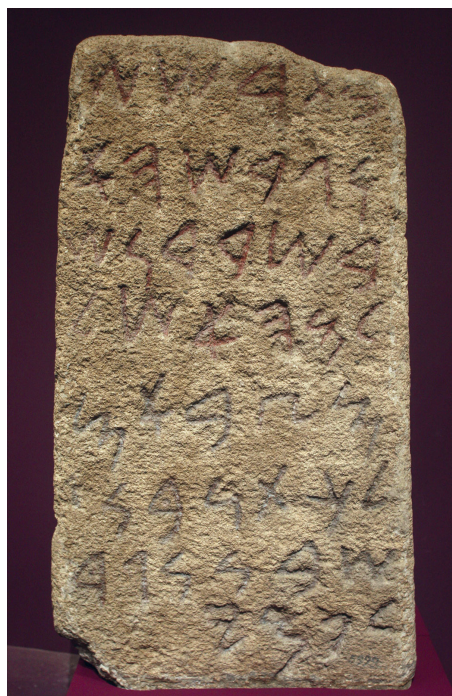


Figure 1.1: Phoenician text inscribed on stone, 1st millennium BCE.

The type of writing of the Phoenician alphabet is Abjad (Fischer, 2003) with 22 consonants and no vowel indication, leaving readers to infer vowel sounds (Table 1.1).

Lately, some interest has been given to Phoenician scripts. According to generally accepted estimates, the corpus of Phoenician-Punic inscriptions comprises about 12,000 inscriptions from all the countries of the Mediterranean (Xella and Zamora, 2019). And, as noted by (Cunchillos et al., 2005), the sheer quantity and scattered nature of the documents, spread over a very wide span of time, have severely affected research and caused considerable difficulties in the knowledge,

Table 1.1: Phoenician alphabet with their correspondents (names and values) in Latin Characters.

Name {value}	Letter(s)	Name {value}	Letter(s)
Aleph {,}	𐤀	Lameth {L}	𐤁
Beth {B}	𐤂	Mem {M}	𐤃
Gimmel {G}	𐤄	Nun {N}	𐤅
Daleth {D}	𐤆	Samekh {S}	𐤇
He {H}	𐤈	Ayin {Ap}	𐤉
Waw {W}	𐤊	Pe {P}	𐤋
Zayin {Z}	𐤌	Tsadi ts {So}	𐤍
Heth { χ - Ho}	𐤎	Qoph {Q}	𐤏
Teth { θ - To}	𐤐	Resh {R}	𐤑
yodh{Y}	𐤒	Sin {Shat}	𐤓
Kaph {K}	𐤔	Taw {T}	𐤕

availability and use of these sources. To alleviate this problem, a collection of all Phoenician and Punic epigraphic documents was transferred into a database by the CIP project (Xella and Zamora, 2019), thus making Phoenician texts accessible to everyone. To this matter, and to further digitize and preserve the heritage of the Phoenician script, we propose a system for the recognition of Phoenician Handwritten characters, which can be used in applications such as processing of records of land conservation and the editing of old documents. To do so, we first need a Phoenician reference database, which up until now is not available. As such, we propose a new database for Phoenician Handwritten Characters (PHCDB).

1.2 Phoenician Handwritten Character Dataset

To create our Phoenician Handwritten Character Dataset, dataset of Phoenician handwritten characters is collected first, then pre-processing of characters is conducted. First, data was collected by 500 writers (authors) who were given either 'form A' or 'form B' to reproduce, as shown in Figure 1.2. Let's note that, form A has Phoenician characters slightly different from form B (Fig.1.2.a). As a result, by making each author write 22 characters, a total of 11,000 characters are collected.

Phoenician Alphabet: FORM A				Phoenician Alphabet: FORM B			
✚	𐤁	𐤂	𐤃	✚	𐤁	𐤂	𐤃
𐤄	𐤅	𐤆	𐤇	𐤄	𐤅	𐤆	𐤇
𐤈	𐤉	𐤊	𐤋	𐤈	𐤉	𐤊	𐤋
𐤌	𐤍	𐤎	𐤏	𐤌	𐤍	𐤎	𐤏
𐤐	𐤑	𐤒	𐤓	𐤐	𐤑	𐤒	𐤓
𐤔	𐤕			𐤔	𐤕		

(a)

✚	𐤁	𐤂	𐤃	✚	𐤁	𐤂	𐤃
𐤄	𐤅	𐤆	𐤇	𐤄	𐤅	𐤆	𐤇
𐤈	𐤉	𐤊	𐤋	𐤈	𐤉	𐤊	𐤋
𐤌	𐤍	𐤎	𐤏	𐤌	𐤍	𐤎	𐤏
𐤐	𐤑	𐤒	𐤓	𐤐	𐤑	𐤒	𐤓
𐤔	𐤕			𐤔	𐤕		

(b)

Figure 1.2: (a) consists of the two forms (“A” and “B”) given to writers, where each form has a different style of alphabet writing. (b) is an example of two sample forms filled by two writers.

The forms were scanned with a quality of 300 dpi. Also, a program was written in Matlab to extract each character image separately from each filled form and saving it into the character folder (i.e., all similar character image will be grouped into the same folder). This program helps in reducing processing time. Next, pre-processing including noise filtering and image binarization is performed. Also, character images are re-sized to remove extra white space. Also, badly written


or unclear characters (extracted from the collected forms) were deleted manually, ending up with 10,714 images i.e., 487 images per character. Figure 1.3 shows some image samples of character 'Kaph'  stored in our database.



Figure 1.3: Example of images of character 'Kaph' K stored in our database.

The HPCDB database is publicly available for the purpose of research at: <https://osf.io/4j9b6/>

1.3 Handwritten Phoenician Character recognition

In this chapter, we propose our Deep Learning framework for the recognition of Handwritten Phoenician Characters based on a Convolutional Neural Network (ConvNet) and the 'PHCDB'. First, the ConvNet architecture and simulation settings are described, then the experiments are laid out and results are discussed.

1.3.1 Phoenician ConvNet.

Data pre-processing. We choose to feed the ConvNet with 60 by 60 images. As such, Phoenician images are re-sized proportionally to a 60×60 frame. Also, to prevent overfitting, data augmentation is performed by randomly rotating, translating/shifting and jittering half of the training images.

Architecture. The architecture of the ConvNet was selected after a trial and error process to maximize the performance rate of recognition of Phoenician characters. As such, the final architecture, illustrated in Figure 1.4, has seven layers in total: (i) four convolutional layers composed of either a stack of Convolution, ReLU and Maxpooling units or a stack of Convolution and ReLU units, and (ii) three fully connected layers with a Fully Connected layer followed by a ReLU and a dropout. A summary of the architecture parameters is given in Table 1.4 of the Annex.

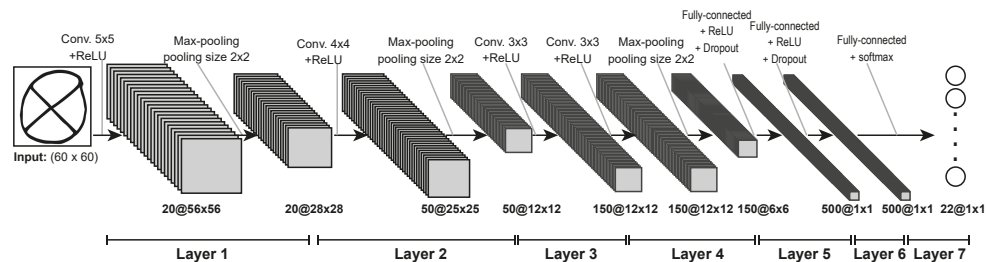


Figure 1.4: Architecture of the proposed ConvNet.

Training. Training is performed using the adaptive learning rate optimization algorithm ADAM with the following parameters: 0.001 as the learning rate, 0.9 and 0.99 as the exponential decay rates for the first and second moment estimates respectively. A high dropout rate of 0.5 is set to avoid overfitting. The selected batch size is 100 and the number of training epochs for full convergence is 70 epochs. We report results with 3-fold cross validation and we take the average accuracy of the validation set as the final evaluation.

1.3.2 Experiments and Results.

ConvNet trained on PHCDB. After training the ConvNet on PHCDB, results are reported in Tables 1.2 and 1.5 (Annex). The first table displays the recognition rate of each Phoenician character while the second table shows the confusion matrix corresponding to the best recognition performance recorded by the system. From these results, a high classification accuracy of 0.9851 is recorded. Thus our ConvNet is able to capture features from handwritten Phoenician characters.

Furthermore, some characters achieve a 100% accuracy while some others achieve a recognition rate between 96.3% and 99.6%. In order to detect where misclassification occurs, let's observe some of the misclassified characters. From these characters, we notice that misrecognition occurs because of either badly written characters within the database such as in (Figure 1.5) or structural similarities between some







Phoenician characters namely  and ,  and , and  and  as illustrated in the confusion matrix 1.5 (Annex).

Table 1.2: Classification rate of each character in the PHCDB database during the validation phase.

Character	Accuracy	Character	Accuracy
𐤀	1	𐤛	0.9753
𐤁	0.9815	𐤜	0.9938
𐤂	0.9630	𐤝	0.9753
𐤃	0.9753	𐤞	1
𐤄	0.9815	𐤟	0.9877
𐤅	1	𐤠	0.9877
𐤆	1	𐤡	0.9815
𐤇	1	𐤢	0.9321
𐤈	0.9877	𐤣	0.9938
𐤉	0.9691	𐤤	1
𐤊	0.9938	Average	0.9851
𐤋	0.9938		

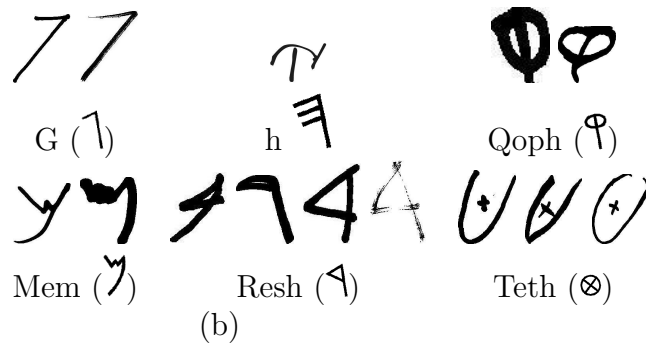


Figure 1.5: Examples of badly written characters in our database.

Comparison with other techniques. Our ConvNet is further compared to two other recognition approaches. The first one involves HOG (Histogram of Oriented Gradient) features (Dalal and Triggs, 2005) (by setting the HOG cell size to 4×4) and a multi-class SVM (Support Vector Machine) classifier. The second one consists of training a Deep Belief Network (DBN) as follows: (i) given the same preprocessed and augmented training images as in chapter 1.2, images are resized from 60×60 to 30×30 then turned into a 900 raw vector; (ii) we set the DBN architecture to

500-500-2000-23, meaning there are three RBMs with 500 hidden neurons in the first and second layers, and 2000 hidden neurons in the third layer (see Figure 1.2 of Annex); (iii) training parameters are set according to Table 1.6 (Annex).

After running simulations, the best performance for each of these approaches is retained and reported in Table 1.3. From these results, we observe that the proposed ConvNet surpasses both the traditional approach (HOG-SVM) and the deep learning one (DBN).

Table 1.3: Comparison of different used classifiers.

	HOG-SVM	DBN	Phoenician ConvNet
Accuracy	0.9052	0.8566	0.9851

1.4 Transfer Learning for recognizing Handwritten Alphabets with lack of annotated data

1.4.1 Methodology

Since the Phoenician alphabet is the ancestor of most existing alphabets, these latter must probably share common features with the Phoenician alphabet. As such, the main idea is to use the Phoenician alphabet features to improve recognition of existing alphabets, especially alphabets which have only few annotated data.

As we know, in order for a ConvNet to be efficient in feature extraction and classification, it needs to be trained by a relatively large number of training instances for full convergence. And, one way to solve the issue of few training labeled data is to first train the ConvNet on data whose domain is close to the target data domain (in hand) and then fine-tune the resulting ConvNet on the target data. Hence, we propose a light-weight and global transfer learning (TL) system based on the following steps:

1. A ConvNet is trained on Phoenician characters according to steps of chapter 1.3 ;
2. The resulting ConvNet weights are frozen except those within the last layer which are replaced by new randomly initialized weights (the number of weights being set according to the number of characters within the target alphabet) ;
3. These last layer weights are then trained on the target alphabet characters.

By doing so, the number of parameters to be learned within the system is limited and we ensure that the training process is much faster.

1.4.2 Target datasets

To determine how well the proposed TL system works, we conduct a series of experiments on a number of different target alphabet datasets for character classification, namely Tifinagh, Latin, Arabic, Russian and Bengali handwriting character datasets.

Latin dataset. The Latin dataset is taken from the Emnist database (Cohen et al., 2017). This repository contains 12,051 images (of size 28×28) of the 31 Latin characters.

Arabic dataset. The Arabic script is written from right to left and is composed of 28 characters (Figure 1.6.a). Arabic characters data is taken from the Handwritten Arabic Characters Database (HACDB) (Lawgali et al., 2013) which has two versions: one with 66 shapes or labels (58 shape characters and 8 shapes of overlapping characters) and a basic one with 24 shapes (representing 24 basic characters). The latter version is the one used in our study and is composed of a total of 2400 jpg images (of size 128×128) representing all 24 characters.

Russian dataset. The Russian alphabet uses letters from the Cyrillic script to write the Russian language (Figure 1.6.c). Russian characters data is taken from Cyrillic-oriented MNIST dataset (CoMNIST) (Comnist, 2019) which is a free, crowd-sourced version of MNIST that contains digitized letters from the Cyrillic and Latin alphabet. The Cyrillic repository currently contains 15,233 png handwritten images of size 278×278 representing all 33 letters of the Russian alphabet.

Bengali dataset. Bengali is the native language of Bangladesh (Figure 1.6.d). The Bengali character dataset is obtained from the Handwritten Indian script character database CMATERdb 3 (Das et al., 2012). The dataset contains 15,000 images representing the 50 basic Bengali characters.



Figure 1.6: Some existing alphabets.

Training hyper-parameters are the same as in the chapter 1.3, except that the number of epochs needed for full convergence is reduced to 35 epochs.

1.4.3 Experiments and results

Experiment 1. In this experiment, we train our Phoenician TL system as well as other TL systems pre-trained on different source domains (e.g., different datasets). Below is the list of conducted simulations:

- i TL using Phoenician ConvNet: this transfer learning consists of fine-tuning one Phoenician ConvNet per target dataset (target datasets are the ones mentioned in section 1.4.2) according to steps of the methodology (section 1.4.1). However, fine-tuning involves training the Phoenician ConvNet on only a subset of the target data training set (e.g., n instances from the total training instances);
- ii TL using Latin ConvNet: we first train a ConvNet on the whole Latin dataset using the same network architecture, simulation settings and cross validation as the Phoenician ConvNet’s. The obtained ConvNet, referred to as the “Latin ConvNet”, is further fine-tuned (according to methodology of section 1.4.1) on n instances only of each and every target dataset (except the Latin dataset);
- iii TL using Digits ConvNet: we first train a ConvNet on digit images contained within the training set of the MNIST dataset (Li Deng, 2012), based on the same network architecture and simulation settings as the Phoenician ConvNet’s. Digit images are resized from 28×28 to 60×60 before being fed to the network. Then, the obtained ConvNet, denoted as “Digits ConvNet”, is further fine-tuned (according to methodology of section 1.4.1) on n instances only of each and every target dataset;
- iv TL using Cifar ConvNet: we train a ConvNet on natural images contained within the training set of the Cifar dataset (Krizhevsky and Hinton, 2009), based on the same network architecture and simulation settings as the Phoenician ConvNet’s. Digit images are resized from 28×28 to 60×60 before being fed to the network. the obtained ConvNet, denoted as “Cifar ConvNet”, is further fine-tuned (according to methodology of section 1.4.1) on n instances only of each and every target dataset;
- v RI ConvNet: we train one randomly initialized (RI) ConvNet on n instances of each of the mentioned target datasets, using the same network architecture, simulation settings and cross validation as the Phoenician ConvNet’s.

For this experiment, the number of training instances n is set to 34 instances.

Corresponding results are illustrated in Table 1.4. From these results, we observe that all TL systems are able to recognize target datasets but each at different rates.

For instance, TL using Phoenician ConvNet approach achieves a higher score than TL using Latin ConvNet, TL using Digits ConvNet and TL using Cifar ConvNet approaches over all target alphabets. Hence, a prior pre-training of the ConvNet on Phoenician characters gives a better recognition than a prior pre-training on Latin letters, digits or natural Cifar images. This suggests that features learned from Phoenician characters generalize better on existing alphabets than features learned from Latin characters, digits or natural images, implying that the former features are more global than the latter ones. Furthermore, the TL using Phoenician ConvNet approach can be considered as an efficient technique to recognize any alphabet with few labeled data. Also, the Phoenician ConvNet can be seen as a baseline ConvNet to train any alphabet.

Comparing the TL using Phoenician ConvNet approach with RI ConvNet approach shows that the former performs better than the latter when dealing with Tifnagh and Latin as target datasets and less than the latter for Arabic, Russian and Bengali target datasets. However, let's note that, according to Table ?? (Annex), RI ConvNet requires training a total of 3,248,892 parameters for 70 epochs as opposed to TL using Phoenician ConvNet system which requires training 11,022 parameters only for 35 epochs. Thus, TL using Phoenician ConvNet can be regarded as a light-weight system for the recognition of any alphabet with few labeled data.

Table 1.4: Comparative results using different Transfer Learning ConvNets.

Target dataset	RI ConvNet	TL using Phoenician ConvNet	TL using Latin ConvNet	TL using Digits ConvNet	TL using Cifar ConvNet
Tifnagh	0.9277	0.9327	0.8266	0.8970	0.7184
Latin	0.9594	0.9786	—	0.7868	0.6471
Arabic	0.8657	0.8333	0.7361	0.7980	0.6765
Russian	0.9158	0.8771	0.8451	0.7610	0.6114
Bengali	0.8867	0.6056	0.5333	0.5533	0.4612

Experiment 2. We run the same simulations as Experiment 1 but this time with $n = 66$ and $n = 134$. Given the set of training instances $n = \{34, 66, 134\}$, classification results of these techniques on the Tifnagh, Latin, Arabic, Russian, and Bengali target datasets are displayed in Figures 1.7.a, .b, .c, .d and .e respectively.

From these results, the following observations can be made:

- i The larger the number of training instances n , the higher the classification score for all techniques on almost all target datasets; Moreover, increasing n makes the recognition performance of TL using Phoenician ConvNet comparable to that of TL using Latin ConvNet, meaning that the ConvNet pre-trained

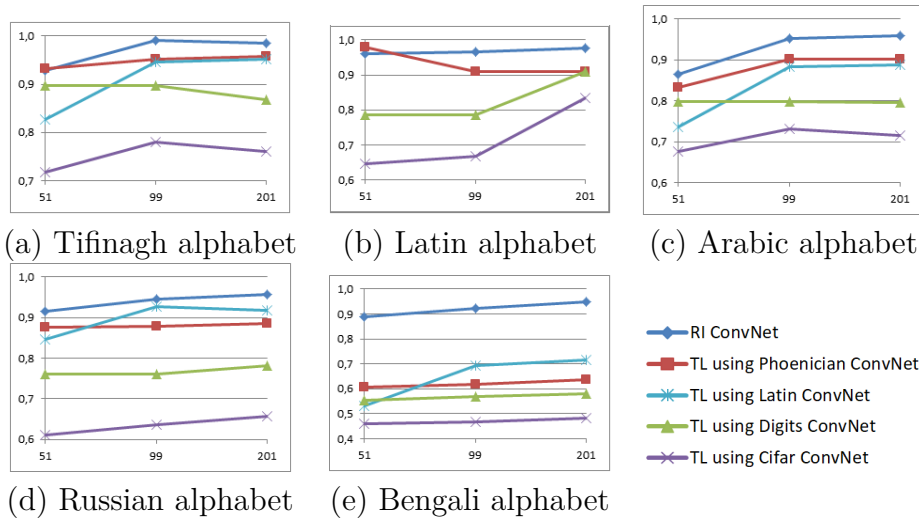


Figure 1.7: Classification results of the RI ConvNet, TL using Phoenician ConvNet, TL using Latin ConvNet, TL using Digits ConvNet, and TL using Cifar ConvNet techniques as a function of the number of training instances n per target dataset.

on Latin characters may hold features as global as the Phoenician ConvNet’s but that the last layer (representing the classifier) of the Latin ConvNet needs more training instances to converge.

- ii Even after increasing the number of training instances, we still have the RI ConvNet the best at recognizing alphabets, followed by TL using Phoenician ConvNet and TL using Latin ConvNet, then by TL using Digits ConvNet, then by TL using Cifar ConvNet. However, as mentioned in the previous experiment, RI ConvNet remains a computationally heavy technique compared with the rest of the techniques, making the TL using Phoenician ConvNet the most optimal technique for recognizing alphabets with few labeled data.

1.5 Conclusion

In this chapter, we first proposed a Phoenician Handwritten dataset (PHCDB) which provides a repository of handwritten Phoenician characters containing 10,714 character shapes. Then, we further used this database to train and test a deep learning system (a Convolutional Neural Network) for the recognition of handwritten Phoenician characters. Finally, we developed a fast, global and light-weight transfer learning network (with limited fine-tuning) based on Phoenician character shapes which can be used for the recognition of any alphabet which experiences a lack of annotated data.

Bibliography

- Yang Cao, Hai Wang, Changhu Wang, Zhiwei Li, Liqing Zhang, and Lei Zhang. MindFinder: Interactive Sketch-based Image Search on Millions of Images. In Proceedings of the International Conference on Multimedia (MM), pages 1605–1608, 2010. ISBN 9781605589336. doi: 10.1145/1873951.1874299.
- Ken Chatfield, Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Return of the Devil in the Details: Delving Deep into Convolutional Nets. In British Machine Vision Conference, 2014. ISBN 1-901725-52-9. doi: 10.5244/C.28.6.
- Gregory Cohen, Saeed Afshar, Jonathan Tapson, and Andre Van Schaik. EMNIST: Extending MNIST to handwritten letters. In Proceedings of the International Joint Conference on Neural Networks, volume 2017-May, pages 2921–2926, 2017. ISBN 9781509061815. doi: 10.1109/IJCNN.2017.7966217.
- Comnist. Cyrillic oriented MNIST: A dataset of Latin and Cyrillic letter images, 2019.
- Gabriella Csurka, Christopher R. Dance, Lixin Fan, Jutta Willamowski, and Cédric Bray. Visual categorization with bags of keypoints. In European Conference on Computer Vision (ECCV) International Workshop on Statistical Learning in Computer Vision, pages 59–74, 2004. ISBN 9780335226375. doi: 10.1234/12345678.
- JL Cunchillos, P Xella, and JÁ Zamora. Il corpus informatizzato delle iscrizioni fenicie e puniche: un progetto italo-spagnolo. Università di Palermo, 2005.
- Navneet Dalal and William Triggs. Histograms of Oriented Gradients for Human Detection. 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition CVPR05, 1(4):886–893, 2005. ISSN 1063-6919. doi: 10.1109/CVPR.2005.177.
- Nibaran Das, Ram Sarkar, Subhadip Basu, Mahantapas Kundu, Mita Nasipuri, and Dipak Kumar Basu. A genetic algorithm based region sampling for selection of local features in handwritten digit recognition application. Applied Soft Computing Journal, 12(5):1592–1606, 2012. ISSN 15684946. doi: 10.1016/j.asoc.2011.11.030.
- Mathias Eitz, Kristian Hildebrand, Tamy Boubekeur, and Marc Alexa. Sketch-Based Image Retrieval: Benchmark and Bag-of-Features Descriptors. {IEEE} Trans. Vis. Comput. Graph., 17(11):1624–1636, 2011.

Mathias Eitz, James Hays, and Marc Alexa. How do humans sketch objects? In *ACM Transactions on Graphics*, volume 31, pages 44–1, 2012. ISBN 0730-0301. doi: 10.1145/2185520.2335395.

SR Fischer. *History of Writing*. Reaktion b edition, 2003.

Xavier Frazão and LA A. Alexandre. Weighted Convolutional Neural Network Ensemble. In *Iberoamerican Congress on Pattern Recognition*, pages 674–681, 2014. ISBN 978-3-319-12568-8; 978-3-319-12567-1. doi: 10.1007/978-3-319-12568-8_82.

Rui Hu and John Collorosse. A performance evaluation of gradient field HOG descriptor for sketch based image retrieval. *Computer Vision and Image Understanding*, 117(7):790–806, 2013. ISSN 1090235X. doi: 10.1016/j.cviu.2013.02.005.

Hervé Jégou, Matthijs Douze, Cordelia Schmid, and Patrick Pérez. Aggregating local descriptors into a compact image representation. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 3304–3311, 2010. ISBN 9781424469840. doi: 10.1109/CVPR.2010.5540039.

Jia Deng, Wei Dong, R Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A large-scale hierarchical image database. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009. ISBN 978-1-4244-3992-8. doi: 10.1109/CVPRW.2009.5206848.

B Klare, Li Zhifeng, and A K Jain. Matching Forensic Sketches to Mug Shot Photos. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(3):639–646, 2011. ISSN 0162-8828. doi: 10.1109/TPAMI.2010.180.

A Krizhevsky and G Hinton. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012. ISBN 9780415468442. doi: 10.1061/(ASCE)GT.1943-5606.0001284.

A Lawgali, M Angelova, and A Bouridane. HACDB: Handwritten Arabic Characters Database for Automatic Character Recognition. In *Visual Information Processing (EUVIP), 2013 4th European Workshop on*, pages 255–259, 2013. ISBN 9788293269137.

- Thomas Leung and Jitendra Malik. Representing and recognizing the visual appearance of materials using three-dimensional textons. *International Journal of Computer Vision*, 43(1):29–44, 2001. ISSN 09205691. doi: 10.1023/A:1011126920638.
- Yi Li, Yi-Zhe Song, and Shaogang Gong. Sketch Recognition by Ensemble Matching of Structured Features. In *Proceedings of the British Machine Vision Conference 2013*, pages 35.1–35.11, 2013. ISBN 1-901725-49-9. doi: 10.5244/C.27.35.
- Yi Li, Timothy M. Hospedales, Yi Zhe Song, and Shaogang Gong. Free-hand sketch recognition by multi-kernel feature learning. *Computer Vision and Image Understanding*, 137:1–11, 2015. ISSN 1090235X. doi: 10.1016/j.cviu.2015.02.003.
- Li Deng. The MNIST Database of Handwritten Digit Images for Machine Learning Research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012. ISSN 1053-5888. doi: 10.1109/msp.2012.2211477.
- David G. Lowe. Distinctive Image Features from Scale-Invariant Keypoints. *International Journal of Computer Vision*, 60(2):91–110, nov 2004. ISSN 0920-5691. doi: 10.1023/B:VISI.0000029664.99615.94.
- Shuxin Ouyang, Timothy Hospedales, Yi Zhe Song, and Xueming Li. Cross-Modal face matching: Beyond viewed sketches. In *Lecture Notes in Computer Science*, volume 9004, pages 210–225. 2015. ISBN 9783319168074. doi: 10.1007/978-3-319-16808-1_15.
- F Perronnin and C Dance. Fisher Kenrels on Visual Vocabularies for Image Categorization. In *Proc. {CVPR}*, pages 1–8, 2007. ISBN 1424411807.
- Ravi Kiran Sarvadevabhatla and R. Venkatesh Babu. Freehand Sketch Recognition Using Deep Features. arXiv preprint arXiv:1502.00254, feb 2015.
- Rosália G. Schneider and Tinne Tuytelaars. Sketch classification and classification-driven analysis using Fisher vectors. *ACM Transactions on Graphics*, 33(6):1–9, 2014. ISSN 07300301. doi: 10.1145/2661229.2661231.
- Omar Seddati. DeepSketch : Deep Convolutional Neural Networks for Sketch Recognition and Similarity Search. In *13th International Workshop on Content-Based Multimedia Indexing (CBMI)*, pages 1—6, 2015. ISBN 9781467368704.
- Pierre Sermanet, David Eigen, Xiang Zhang, Michael Mathieu, Rob Fergus, and Yann LeCun. OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks. arXiv preprint arXiv:1312.6229, 2013. ISSN 10636919. doi: 10.1109/CVPR.2015.7299176.

- Karen Simonyan and Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. In Proceedings of the 3rd International Conference on Learning Representations, sep 2015. ISBN 1097-0142 (Electronic)\n0008-543X (Linking). doi: 10.2146/ajhp170251.
- Sivic and Zisserman. Video Google: a text retrieval approach to object matching in videos. In Proceedings Ninth IEEE International Conference on Computer Vision, pages 1470–1477 vol.2, 2003. ISBN 0-7695-1950-4. doi: 10.1109/ICCV.2003.1238663.
- Andrea Vedaldi and Karel Lenc. MatConvNet - Convolutional Neural Networks for MATLAB. In Proceedings of the 23rd ACM international conference on Multimedia, pages 689–692, 2015. ISBN 978-1-4503-3459-4. doi: 10.1145/2733373.2807412.
- F Wang, L Kang, and Y Li On. Sketch-based 3d shape retrieval using convolutional neural networks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 1875–1883, 2015.
- Xinggong Wang, Xiong Duan, and Xiang Bai. Deep sketch feature for cross-domain image retrieval. *Neurocomputing*, 207:387–397, 2016. ISSN 18728286. doi: 10.1016/j.neucom.2016.04.046.
- Paolo Xella and José Á. Zamora. 7 Phoenician Digital Epigraphy: CIP Project, the State of the Art. In *Crossing Experiences in Digital Epigraphy*, pages 93–101. 2019. doi: 10.1515/9783110607208-008.
- J Yang, K Yu, and T Huang. Supervised translation-invariant sparse coding. 2010.
- Qian Yu, Yongxin Yang, Feng Liu, Yi-Zhe Song, Tao Xiang, and Timothy M Hospedales. Sketch-a-net: A deep neural network that beats humans. *International journal of computer vision*, 122(3):411–425, 2017.
- Matthew D. Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833, 2014. ISBN 9783319105895. doi: 10.1007/978-3-319-10590-1_53.
- Xi Zhou, Kai Yu, Tong Zhang, and Thomas S. Huang. Image classification using super-vector coding of local image descriptors. In *European conference on computer vision*, volume 6315 LNCS, pages 141–154, 2010. ISBN 3642155545. doi: 10.1007/978-3-642-15555-0_11.

Chapter 2

Multimodal learning: Novel transfer learning approach

In this chapter, we present a solution to the lack of data by introducing a “knowledge transfer” framework which is a global, fast and light-weight framework that combines between a transfer learning technique and a SVM classifier. We conduct this study again on the SMM recognition task.

As a reminder, let’s note that, due to the inter-subject variability of SMMs, a ConvNet trained on movements of an atypical subject i performs badly on detecting SMMs of another atypical subject and therefore cannot be applied on SMMs other than subject i ’s SMMs. For instance, testing one of the trained ConvNets of experiments within Section 1.6 (let’s say the trained ConvNet of subject i study j) on SMMs of a subject other than subject i produces a very low F1-score with less than 30%. As a result, ConvNet features learned from SMMs of one subject differ from the ones learned from another subject and are not general enough to detect SMMs of another subject. One way to resolve this problem is to train a ConvNet for each and every atypical subject individually, as was done in experiments of Section 1.6). However, we do not always have a large number of annotated SMMs in hand per atypical subject. As such, the idea is to develop a framework capable of detecting SMMs across atypical subjects.

The rest of this chapter is organized as follows. Section 2.1 describes the basic component of our technique. Afterward, Section 2.2 implements this technique on a SMM recognition task which is different from the SMM recognition task mentioned in chapter 1. This task consists of recognizing SMMs across different atypical subjects rather than recognizing SMMs across multiple sessions within one subject (as in experiments of section 1.6). Results are reported and discussed in Section 2.3.

2.1 Methodology: Feature learning via knowledge transfer

In this section, the target domain is the SMM dataset of an atypical (e.g., autistic) subject i while the source domain is either similar to the target domain (such as SMM datasets of multiple atypical subjects other than i) or different but related to it (such as datasets of human activities performed by typical subjects). So, the purpose of this section is to prove that some of the complex features emerging from discriminative learning of our ConvNet models (time-domain and frequency-domain ConvNets) in one of the 2 source domains can be successively used via transfer learning to classify SMM patterns of any atypical subject.

Source domains. To detect SMMs of an atypical subject i , feature learning is performed according to Step1 of Figure 2.1, whereby features are learned by training a ConvNet using data from 2 source domains. We consider:

- i features learned from SMMs of some atypical subjects other than subject i as the 1st source domain. These features are further embedded within a Transfer Learning system to perform recognition on the target domain which is “the recognition of SMMs of subject i ”. Thus, the source and target domains of this knowledge transfer process are the same;
- ii features learned from movements of typical subjects as the 2nd source domain. Indeed, this domain is about standard movements (e.g., simple human activities) of normal individuals taken from everyday life basis, such as walking, sitting, standing, jumping, running, etc.. Features learned are further employed to recognize SMMs via knowledge transfer. Therefore, this knowledge transfer process relies on two different but related domains.

Knowledge transfer. Features learned from either one of the two source domains are then embedded within a knowledge transfer process to identify instances of the target-domain. In particular, the ConvNet low and mid-level features are used (kept unchanged) while high-level features are removed and replaced by new randomly initialized ones. So, the transfer learning framework is composed of low and mid-level features of the source domain, followed by a “readout module” mapping the representation of these low and mid-level features into 2 labels: SMM or non-SMM. This read-out module consists of a simple classifier, the Support Vector Machine with the RBF kernel (a popular kernel function used in various kernelized learning), whose goal is to train high-level weights using instances of the target-domain, as shown in Step 2 of Figure 2.1. Supplying the transfer learning framework with SMM instances consists into feeding these instances into its low and mid-level features, resulting in output features which serve as input to train the SVM classifier (knowing that all SMM instances are labeled).

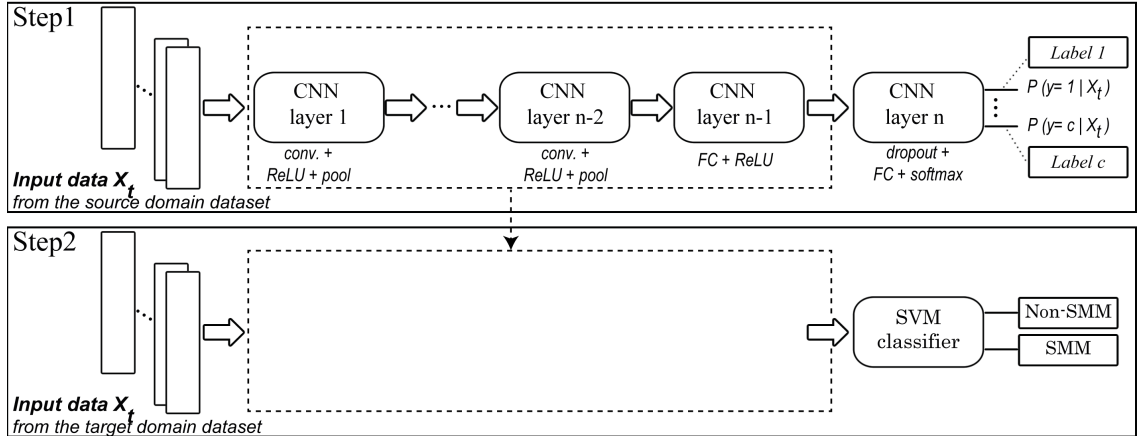


Figure 2.1: The overall transfer learning framework. Abbreviations “conv.” and “FC” stand for convolutional and fully connected layers respectively.

2.2 Experiments

2.2.1 Experiment 1

. Through this experiment, we want to show that the “transfer learning with SVM read-out” framework (also referred to as “TL SVM framework”) is a global, fast and light-weight approach that deals with time-series classification tasks with the lack of annotated data. The target learning task involves recognizing SMMs of an atypical subject i . On the other hand, the source learning task is one of the two following tasks: (i) a task similar to the target learning task such as recognizing SMMs of multiple subjects other than i , (ii) a task different but related to the target task such as the recognition of basic human movements. Running the “TL SVM framework” with the former and the latter target tasks is referred to as TL SVM similar domains and TL SVM across domains respectively. Through this experiment, we also prove that these source learning tasks produce features that are global enough to recognize SMMs of any new atypical subject.

Datasets. The dataset used for the target learning task is the same SMM dataset used in section 1.5.1. The dataset used for the source domain of the “TL SVM similar domains” experiment is also the SMM dataset, whereas the one used for the source domain of the “TL SVM across domains” experiment is the PUC dataset which is described in section 1.5.2.

When using the SMM dataset in the target and source learning tasks, signals of right and left wrist accelerometers are not needed while signals of the torso sensor are the ones employed in this experiments. The resulting inputs have then 3 channels instead of 9. So, with torso measurements only, the only SMMs that could be identified by our frameworks are the rock and flap-rock SMMs (and no flap SMMs), which implies that inputs used for this experiment are rock and flap-rock

SMM instances only.

When applying the PUC dataset as the source learning task, signals recorded from accelerometers located at the thigh, ankle and arm are not needed while the waist accelerometer (waist being next to torso) is the only one relevant to the SMM recognition task during transfer learning. Indeed, the waist location is quite similar to the torso location so that the ConvNet pre-trained on the source learning dataset can further be transferred to the target learning task (SMM recognition). Accordingly, input instances will have 3 channels instead of 12.

Pre-processing. The pre-processing phase is the same as in section 1.5.

Experimental setup. In this experiment, the time and frequency domain ConvNet architectures and hyper-parameters are the same as to the ones used in section 1.5. The target learning task is the SMM recognition of a target subject i of study j where $i \in [1, 6]$ and $j \in [1, 2]$. As such, one “TL SVM framework” is to be trained per domain (time or frequency domain), per subject and per study. The training and testing sets of subject i (study j) are selected using the same k-fold cross-validation used in section 1.5. Nonetheless, since the purpose of this experiment is to test our framework under the constraint of lack of training labeled data, we pick only a subset of the whole training set (knowing that the training data instances vary from 10,000 to 30,000), i.e., 2000 randomly selected training instances.

In order to run our experiment, we need to follow steps below for each domain (time and frequency) for each study i within each study j :

1. For each and every subject i of every study j , we start by training a ConvNet in both time and frequency domains for 5 to 15 epochs, using: (i) signal instances of all 6 atypical subjects within study j except subject i for the TL SVM similar domains framework, and (ii) basic human activities’ instances for the TL SVM across domains framework. This process results in a pre-trained ConvNet model for each subject i .
2. For each resulting ConvNet model, we employ all of its layers except the last layer which is replaced by a SVM classifier. Then, 2000 samples taken from subject i ’s training data are fed into the new framework (e.g., the ConvNet layers followed by the SVM classifier) for training the classifier, resulting in learned high-level features. Afterwards, the remaining instances of subject i are used for testing the framework. Since only a subset of subject i ’s training dataset is used, this experiment is run for 5 times, with 2000 randomly selected samples in each run. Then, we average out the F1-scores relative to the 5 runs in order to obtain more realistic results.

2.2.2 Experiment 2: Comparison with other techniques

. By way of comparison, several other techniques are laid out:

- i . The “ConvNet with few data” approach trains a randomly initialized ConvNet in time and frequency domains using the same number of target training instances as in experiment 1, i.e., 2000 samples. This approach is further denoted as ConvNet few data. Let’s note that ConvNet few data is different from the ConvNet of section 1.6 in several aspects: (i) Only 2000 input instances are used for training the former as opposed to 10,000-30,000 training instances for the latter, (ii) inputs in the former has less channels than the latter since the former takes into account only torso sensor measurements (versus torso, right and left wrist sensor measurements for the latter) and, (3) as opposed to the latter approach, the former rocking SMM samples and keeps rock and flap-rock SMM instances only.
- ii . The “transfer learning with full fine-tuning” approach (referred to as TL full fine-tuning) trains a ConvNet in time and frequency domains for every subject i within study j for 5 to 15 epochs, using instances of all 6 atypical subjects within study j except subject i (as in Step 1 of Experiment 1). Then a fine-tuning process is run on the each resulting ConvNet (e.g., all ConvNet layers’ weights) using the same reduced target training data.
- iii . The “transfer learning with limited fine-tuning” approach (denoted as TL limited fine-tuning) is the same as “transfer learning with full fine-tuning” except that the fine-tuning process is effective only on weights of the last ConvNet layer L while weights of other layers $1, \dots, L - 1$ are kept fixed.

2.3 Results

We report results and characteristics of experiment 1 and 2 in Table 2.1 and 2.2 respectively. From these results and characteristics, the following observations can be made:

- As depicted in Table 2.1, the TL SVM similar domains framework is successful at recognizing SMMs at a mean F1-score of 74.50 % and 91.81 % for time and frequency domains respectively. Therefore, we can infer the following: (i) good SMM features can be extracted thanks to TL SVM similar domains, (ii) since freezing pre-trained ConvNet layers $1, \dots, L - 1$ during training on target data generate good classification scores, this suggests that low and mid-level SMM features share the same information from one subject to another,

Table 2.1: Results of our transfer learning approach as well as other ConvNet approaches per domain (time or frequency) per subject and per study.

Approaches	Study 1						Study 2					Mean
	S1	S2	S3	S4	S5	S6	S1	S2	S3	S4	S5	
Time domain												
ConvNet few data	72.02	62.31	52.98	88.47	60.61	88.86	80.90	53.28	16.00	82.66	75.82	66.72
TL full fine-tuning	75.73	71.31	59.04	91.67	59.47	91.89	85.66	63.84	38.57	92.24	82.31	73.79
TL limited fine-tuning	75.44	56.50	50.86	91.74	63.86	93.11	85.88	62.62	27.14	93.63	81.16	71.09
TL SVM similar domains	75.37	76.44	56.53	91.74	63.37	92.76	84.86	62.97	41.60	93.32	80.55	74.50
TL SVM across domains	71.66	74.40	66.80	90.69	61.87	92.19	81.35	58.13	35.66	88.44	73.98	72.29
Frequency domain												
ConvNet few data	76.64	96.55	63.44	93.13	82.58	94.61	84.94	76.42	29.51	93.66	83.41	79.54
TL full fine-tuning	88.51	97.22	88.15	97.53	91.29	98.26	92.17	88.19	52.17	96.59	91.98	89.28
TL limited fine-tuning	87.98	94.59	62.70	97.57	87.94	98.36	91.62	87.08	40.00	97.82	90.82	85.14
TL SVM similar domain	90.54	97.22	83.86	95.24	86.19	98.45	92.71	90.49	84.99	97.99	92.22	91.81
TL SVM across domains	74.50	91.56	43.77	93.11	76.03	94.2	85.16	74.67	66.98	93.66	83.99	79.78

Table 2.2: Characteristics and resources used for the different techniques implemented in experiments.

Approaches	# parameters updated for one pass (1 batch)	# batches per iteration	# iterations (epochs)
ConvNet few data	1.2e+06 (time-domain) 7.1e+05 (frequency-domain)	14 (2000/150)	20-35 (time-domain) 55-85 (frequency-domain)
TL full fine-tuning	1.2e+06 (time-domain) 7.1e+05 (frequency-domain)	14 (2000/150)	5-15
TL limited fine-tuning	1000 (500*2)	14 (2000/150)	5-15
TL SVM	500	1	1

(iii) the last ConvNet layer weights are the ones that vary from one atypical subject to another and are the ones that characterize each atypical subject, (iv) TL SVM similar domains can be considered as a general and light-weight framework to identify SMMs of any new atypical subject. Furthermore, low- and mid-level features captured from a source learning task can be further applied as low- and mid-level features of a target learning task close to the source task.

- The TL SVM across domains framework yields an average F1-score of 72.29% and 79.78% in time- and frequency-domain respectively (Table 2.1). So, fixing low and mid-level weights learned from basic movements and modulating only high-level features via an SVM seem to produce satisfying results on recognizing SMMs. From time-domain results, the following conclusions can be drawn: (i) low- and mid-level features of basic human movements hold features general enough to adapt to SMMs of any new atypical subject i , (ii) normal and stereotypical movements must have common low and mid-level features, (iii) low- and mid-level details learned by a ConvNet from a source

target task different but related to the source learning task, can be further used as features for learning target tasks, especially when only few labeled data are present within that task.

- The TL SVM similar domains framework yields higher classification scores than the three frameworks ConvNet few data, TL full fine-tuning and TL limited fine-tuning in both time- and frequency-domain. This is due to overfitting of the three frameworks during training since weights are fine-tuned using few training data. As we know, training neural network weights by back-propagation necessitates a large amount of labelled training data to achieve full convergence.
- TL SVM similar domains and TL SVM across domains architectures produce better rates than ConvNet few data by 7.78% and 5.57% respectively in time-domain and by 12.27% and 0.24% respectively in frequency-domain. Thus, more general features are learned within our two frameworks. Moreover, these two frameworks offer a better convergence speed than ConvNet few data by a total of 5-15 epochs (in both time and frequency domains) for full convergence compared to 20-35 epochs and 55 -85 epochs in time- and frequency-domain respectively for full convergence, as shown in Table 2.2. Furthermore, in terms of memory consumption, the number of parameters to be learned by our frameworks is, which is 500 (in both time and frequency domains) compared to $1.2e+06$ and $7.1e+05$ for the time- and frequency- domain ConvNet few data respectively (Table 2.2). Thus, in contrast to ConvNet few data, the proposed frameworks can be regarded as a global, fast and light-weight framework for SMM recognition across subjects.
- TL full fine-tuning performs slightly better than TL limited fine-tuning by an average of 2.71% and 4.14% in time- and frequency-domain respectively, suggesting that fine-tuning weights of layers $1, \dots, L - 1$ (where L is the number of ConvNet layers) is not necessary since it does not improve SMM recognition significantly. This confirms the earliest assumption that atypical subjects have similar low- and mid-level features but different high-level features.
- TL SVM across domains framework has a lower performance than TL SVM similar domains by 2.21% and 12.02% in time- and frequency-domain respectively. In time-domain, the small rate difference (2.21%) between TL SVM across domains and TL SVM similar domains implies that atypical and typical subjects share common low- and mid-level information within their movements. Nonetheless, this similarity in features is not reflected in frequency-domain because of the large score difference between TL SVM across domains

and TL SVM similar domains. Indeed, the relatively low frequency-domain low performance could be explained by the imperfect choice of the frequency bandwidth (set to the range 0-3Hz) when turning time-series input signals into frequency signals. In fact the 0-3Hz range was the perfect bandwidth for SMM signals but not for typical movements which, according to (?), have 98% of their FFT amplitude contained below 10Hz.

- TL SVM similar domains and TL SVM across domains could be further implemented in Android portable devices, as illustrated in Table 2.2. To do so, only two steps are required: first, an expert (a doctor for instance) would receive continuous acceleration signals from the torso accelerometer of a subject, and label them on the fly (as SMM/non-SMM) as the subject performs his activities/movements. A one-minute recording of these signals is enough for our experiment. Then, our framework preprocesses the resulting annotated time-series and use them for training either TL SVM similar domains or TL SVM across domains. Afterwards, this framework can be operational for recognizing further SMMs on that same subject.

2.4 Conclusion

In this chapter we introduced a deep learning approach to address time-series classification with limited annotated data. Our proposed within-domain and across-domain transfer learning frameworks (“TL SVM similar domains” and “TL SVM across domains”) are two ConvNet frameworks whose goal is to generate features general and global enough to recognize time-series of the target learning task, given time-series of a source learning task that is similar or different but related to the target learning task. All these approaches were implemented on the recognition of Stereotypical Motor Movements (SMMs) across atypical subjects (e.g., subjects on the Autism Spectrum). Experimental results showed the superiority of our frameworks and their ability to extract relevant features from time-series inputs. We illustrated how our within-, and across-domain transfer learning frameworks are scalable, fast and light-weight solutions which: (i) adjust to stereotypical behavior patterns of any new atypical subject requiring only few labeled SMMs, (ii) alleviate the problem of the lack of annotated SMMs per subject. Furthermore, we showed that, as opposed to the within-domain transfer learning framework, the cross-domain transfer learning framework does require SMMs for training since training is implemented using a source domain dataset different from the target domain. Thus, the lack of training annotated data in any medical target domain no longer limits the size and learning ability of ConvNet models. As a perspective,

the time-, and frequency-domain cross-domain transfer learning frameworks can be used as a baseline to the recognition of movement disorders with any frequency and with a frequency range $[0, 10Hz]$ respectively.

Bibliography

Yang Cao, Hai Wang, Changhu Wang, Zhiwei Li, Liqing Zhang, and Lei Zhang. MindFinder: Interactive Sketch-based Image Search on Millions of Images. In Proceedings of the International Conference on Multimedia (MM), pages 1605–1608, 2010. ISBN 9781605589336. doi: 10.1145/1873951.1874299.

Ken Chatfield, Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Return of the Devil in the Details: Delving Deep into Convolutional Nets. In British Machine Vision Conference, 2014. ISBN 1-901725-52-9. doi: 10.5244/C.28.6.

Gregory Cohen, Saeed Afshar, Jonathan Tapson, and Andre Van Schaik. EMNIST: Extending MNIST to handwritten letters. In Proceedings of the International Joint Conference on Neural Networks, volume 2017-May, pages 2921–2926, 2017. ISBN 9781509061815. doi: 10.1109/IJCNN.2017.7966217.

Comnist. Cyrillic oriented MNIST: A dataset of Latin and Cyrillic letter images, 2019.

Gabriella Csurka, Christopher R. Dance, Lixin Fan, Jutta Willamowski, and Cédric Bray. Visual categorization with bags of keypoints. In European Conference on Computer Vision (ECCV) International Workshop on Statistical Learning in Computer Vision, pages 59–74, 2004. ISBN 9780335226375. doi: 10.1234/12345678.

JL Cunchillos, P Xella, and JÁ Zamora. Il corpus informatizzato delle iscrizioni fenicie e puniche: un progetto italo-spagnolo. Università di Palermo, 2005.

Navneet Dalal and William Triggs. Histograms of Oriented Gradients for Human Detection. 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition CVPR05, 1(4):886–893, 2005. ISSN 1063-6919. doi: 10.1109/CVPR.2005.177.

Nibaran Das, Ram Sarkar, Subhadip Basu, Mahantapas Kundu, Mita Nasipuri, and Dipak Kumar Basu. A genetic algorithm based region sampling for selection of local features in handwritten digit recognition application. Applied Soft Computing Journal, 12(5):1592–1606, 2012. ISSN 15684946. doi: 10.1016/j.asoc.2011.11.030.

- Mathias Eitz, Kristian Hildebrand, Tamy Boubekeur, and Marc Alexa. Sketch-Based Image Retrieval: Benchmark and Bag-of-Features Descriptors. {IEEE} Trans. Vis. Comput. Graph., 17(11):1624–1636, 2011.
- Mathias Eitz, James Hays, and Marc Alexa. How do humans sketch objects? In ACM Transactions on Graphics, volume 31, pages 44–1, 2012. ISBN 0730-0301. doi: 10.1145/2185520.2335395.
- SR Fischer. History of Writing. Reaktion b edition, 2003.
- Xavier Frazão and LA A. Alexandre. Weighted Convolutional Neural Network Ensemble. In Iberoamerican Congress on Pattern Recognition, pages 674–681, 2014. ISBN 978-3-319-12568-8; 978-3-319-12567-1. doi: 10.1007/978-3-319-12568-8_82.
- Rui Hu and John Collorosse. A performance evaluation of gradient field HOG descriptor for sketch based image retrieval. Computer Vision and Image Understanding, 117(7):790–806, 2013. ISSN 1090235X. doi: 10.1016/j.cviu.2013.02.005.
- Hervé Jégou, Matthijs Douze, Cordelia Schmid, and Patrick Pérez. Aggregating local descriptors into a compact image representation. In Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, pages 3304–3311, 2010. ISBN 9781424469840. doi: 10.1109/CVPR.2010.5540039.
- Jia Deng, Wei Dong, R Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A large-scale hierarchical image database. In IEEE Conference on Computer Vision and Pattern Recognition, pages 248–255, 2009. ISBN 978-1-4244-3992-8. doi: 10.1109/CVPRW.2009.5206848.
- B Klare, Li Zhifeng, and A K Jain. Matching Forensic Sketches to Mug Shot Photos. IEEE Transactions on Pattern Analysis and Machine Intelligence, 33(3):639–646, 2011. ISSN 0162-8828. doi: 10.1109/TPAMI.2010.180.
- A Krizhevsky and G Hinton. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.
- Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In Advances in neural information processing systems, pages 1097–1105, 2012. ISBN 9780415468442. doi: 10.1061/(ASCE)GT.1943-5606.0001284.
- A Lawgali, M Angelova, and A Bouridane. HACDB: Handwritten Arabic Characters Database for Automatic Character Recognition. In Visual Information

Processing (EUVIP), 2013 4th European Workshop on, pages 255–259, 2013. ISBN 9788293269137.

Thomas Leung and Jitendra Malik. Representing and recognizing the visual appearance of materials using three-dimensional textures. *International Journal of Computer Vision*, 43(1):29–44, 2001. ISSN 09205691. doi: 10.1023/A:1011126920638.

Yi Li, Yi-Zhe Song, and Shaogang Gong. Sketch Recognition by Ensemble Matching of Structured Features. In *Proceedings of the British Machine Vision Conference 2013*, pages 35.1–35.11, 2013. ISBN 1-901725-49-9. doi: 10.5244/C.27.35.

Yi Li, Timothy M. Hospedales, Yi Zhe Song, and Shaogang Gong. Free-hand sketch recognition by multi-kernel feature learning. *Computer Vision and Image Understanding*, 137:1–11, 2015. ISSN 1090235X. doi: 10.1016/j.cviu.2015.02.003.

Li Deng. The MNIST Database of Handwritten Digit Images for Machine Learning Research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012. ISSN 1053-5888. doi: 10.1109/msp.2012.2211477.

David G. Lowe. Distinctive Image Features from Scale-Invariant Keypoints. *International Journal of Computer Vision*, 60(2):91–110, nov 2004. ISSN 0920-5691. doi: 10.1023/B:VISI.0000029664.99615.94.

Shuxin Ouyang, Timothy Hospedales, Yi Zhe Song, and Xueming Li. Cross-Modal face matching: Beyond viewed sketches. In *Lecture Notes in Computer Science*, volume 9004, pages 210–225. 2015. ISBN 9783319168074. doi: 10.1007/978-3-319-16808-1_15.

F Perronnin and C Dance. Fisher Kernels on Visual Vocabularies for Image Categorization. In *Proc. {CVPR}*, pages 1–8, 2007. ISBN 1424411807.

Ravi Kiran Sarvadevabhatla and R. Venkatesh Babu. Freehand Sketch Recognition Using Deep Features. *arXiv preprint arXiv:1502.00254*, feb 2015.

Rosália G. Schneider and Tinne Tuytelaars. Sketch classification and classification-driven analysis using Fisher vectors. *ACM Transactions on Graphics*, 33(6):1–9, 2014. ISSN 07300301. doi: 10.1145/2661229.2661231.

Omar Seddati. DeepSketch : Deep Convolutional Neural Networks for Sketch Recognition and Similarity Search. In *13th International Workshop on Content-Based Multimedia Indexing (CBMI)*, pages 1–6, 2015. ISBN 9781467368704.

Pierre Sermanet, David Eigen, Xiang Zhang, Michael Mathieu, Rob Fergus, and Yann LeCun. OverFeat: Integrated Recognition, Localization and Detection

using Convolutional Networks. arXiv preprint arXiv:1312.6229, 2013. ISSN 10636919. doi: 10.1109/CVPR.2015.7299176.

Karen Simonyan and Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. In Proceedings of the 3rd International Conference on Learning Representations, sep 2015. ISBN 1097-0142 (Electronic)\n0008-543X (Linking). doi: 10.2146/ajhp170251.

Sivic and Zisserman. Video Google: a text retrieval approach to object matching in videos. In Proceedings Ninth IEEE International Conference on Computer Vision, pages 1470–1477 vol.2, 2003. ISBN 0-7695-1950-4. doi: 10.1109/ICCV.2003.1238663.

Andrea Vedaldi and Karel Lenc. MatConvNet - Convolutional Neural Networks for MATLAB. In Proceedings of the 23rd ACM international conference on Multimedia, pages 689–692, 2015. ISBN 978-1-4503-3459-4. doi: 10.1145/2733373.2807412.

F Wang, L Kang, and Y Li On. Sketch-based 3d shape retrieval using convolutional neural networks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 1875–1883, 2015.

Xinggong Wang, Xiong Duan, and Xiang Bai. Deep sketch feature for cross-domain image retrieval. *Neurocomputing*, 207:387–397, 2016. ISSN 18728286. doi: 10.1016/j.neucom.2016.04.046.

Paolo Xella and José Á. Zamora. 7 Phoenician Digital Epigraphy: CIP Project, the State of the Art. In *Crossing Experiences in Digital Epigraphy*, pages 93–101. 2019. doi: 10.1515/9783110607208-008.

J Yang, K Yu, and T Huang. Supervised translation-invariant sparse coding. 2010.

Qian Yu, Yongxin Yang, Feng Liu, Yi-Zhe Song, Tao Xiang, and Timothy M Hospedales. Sketch-a-net: A deep neural network that beats humans. *International journal of computer vision*, 122(3):411–425, 2017.

Matthew D. Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833, 2014. ISBN 9783319105895. doi: 10.1007/978-3-319-10590-1_53.

Xi Zhou, Kai Yu, Tong Zhang, and Thomas S. Huang. Image classification using super-vector coding of local image descriptors. In *European conference on computer vision*, volume 6315 LNCS, pages 141–154, 2010. ISBN 3642155545. doi: 10.1007/978-3-642-15555-0_11.

Chapter 3

MultiModal Learning: Novel ensemble learning approach

In the previous chapter, we have seen how to overcome the lack of data while training supervised Deep learning models via knowledge transfer, i.e., by transferring part of features learned from similar or different domain data to our transfer domain task. In this chapter, we propose to use fuse different deep learning models, some of which employing knowledge transfer. Indeed, we propose an ensemble learning framework which embeds between features learned by training different ConvNet models including randomly initialized ConvNets and ConvNets already pre-trained on tasks different from the target domain task. We choose to implement this study on a computer vision task, namely the “sketch recognition” task.

This chapter is organized as follows. First, a definition of sketch and its applications are given in Section 3.1. Afterward, a review on sketch recognition techniques is provided (Section 3.2). Next, Section 3.3 describes components of our ensemble learning framework. In Section 3.4, the experimental setup is described. Then, experiments related to sketch classification are carried out (Sec.3.5. Then results are reported and analyzed 3.6). Finally, a conclusion is presented in Section 3.7.

3.1 Introduction

Sketches have been used by humans to describe objects of the world around us. Sketches are known to be simple and hasty drawings, giving the essential features without the details (Fig.3.1.a). In other words, a sketch is defined as a quick, rough drawing that shows the main features of an object or scene. Even though they are considered to be a rough design of the reality, they are an effective communicative tool for humans since they are considered to be the simplest way for people to show ideas intuitively. And, with the growing field of touchscreens, sketching became

more and more popular. Since then, more research has been conducted on sketching, giving rise to many applications such as sketch recognition (Eitz et al., 2012; Schneider and Tuytelaars, 2014), sketch-based image retrieval (Eitz et al., 2011; Hu and Collorosso, 2013), sketch-based 3D model retrieval (Wang et al., 2015), and forensic sketch analysis (Klare et al., 2011; Ouyang et al., 2015). One of the popular applications is sketch-to-image retrieval which uses a sketch as query to find the images with the similar semantic content in a large image set. One example of this application is “MindFinder.1”, a popular product of Microsoft that compares a query sketch with the edge-maps of millions images collected from web (Cao et al., 2010). However, this product is only able to detect simple sketch features and cannot extract all edges in sketch images, making it difficult to recognize complex sketches.

Sketches and natural photo-based images are very different in appearance even though they appear to have a lot of things in common and convey the same meaning. Compared to natural images, sketches are very abstract and don’t have color or texture. Consequently, existing methods for natural images cannot be directly applied to our sketch recognition problem. Moreover, the same object can be drawn with different levels of detail (abstraction), e.g., every person has his or her own way to sketch an object, drawing a stickman for example with either a rough structure or a portrait with fine details, as illustrated in Figure 3.1.b. Therefore, internal structures of sketches are very complex and details of the drawing are very sparse. All these reasons make the sketch classification a more challenging task. In fact, even humans can only achieve a recognition accuracy of 73.1% (Eitz et al., 2012).

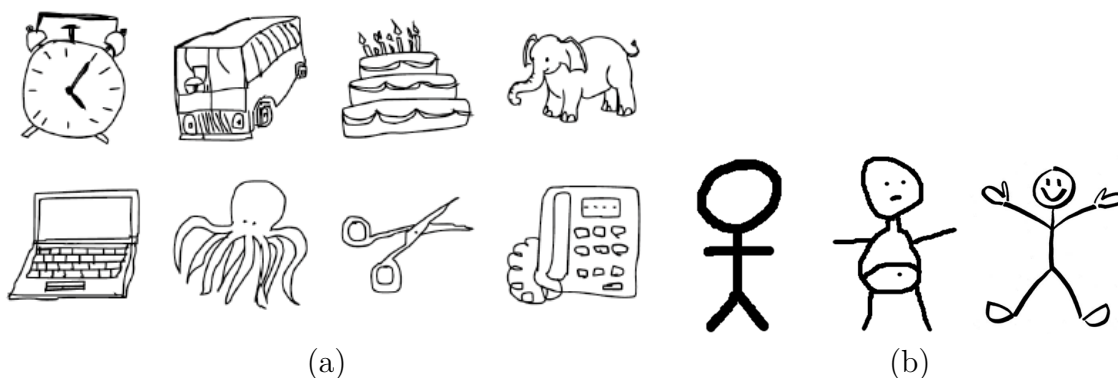


Figure 3.1: (a) Examples of hand-sketched objects, and (b) three samples of a stickman object drawn with different levels of abstraction.

3.2 Related work

Natural photo-based image recognition. Most image understanding and computer vision methods build on image representations such as textons (Leung and Malik, 2001), histogram of oriented gradients (SIFT (Lowe, 2004) and HOG (Dalal and Triggs, 2005)), bag of visual words (Csurka et al., 2004; Sivic and Zisserman, 2003), sparse and local coding (Yang et al., 2010), super vector coding (Zhou et al., 2010), VLAD (Jégou et al., 2010), Fisher Vectors (Perronnin and Dance, 2007), and, lately, deep neural networks, particularly of the convolutional variety (Krizhevsky et al., 2012; Sermanet et al., 2013; Zeiler and Fergus, 2014). The progress in the development of visual representations made us go from shallow hand-crafted features to deep representations, where millions of parameters are learned from data.

Sketch recognition. Sketch recognition did not get much attention until 2012 when a large crowd-sourced dataset was published in the article “How do Humans Sketch Objects” (Eitz et al., 2012). Using this dataset, works have been conducted on sketch recognition (Eitz et al., 2012; Li et al., 2015; Schneider and Tuytelaars, 2014) applying hand-crafted features representation borrowed from photos and using the SVM classifier. Authors in (Eitz et al., 2012) collected a dataset of 20000 unique sketches evenly distributed over 250 object categories, which are totally completed by no-expert free hands, and they investigated the sketch recognition issue in both the humans and the proposed method using SIFT-variant descriptor. Later, authors in (Leung and Malik, 2001) demonstrated that fusing different local features using multiple kernel learning helps improve the recognition performance. They also examined the performance of many features individually and found that HOG generally outperformed others. Very recently, Schneider and Tuytelaars (Schneider and Tuytelaars, 2014) demonstrated that Fisher Vectors, an advanced feature representation scheme successfully applied to image recognition, can be adapted to sketch recognition and achieve near-human accuracy (68.9% vs. 73.1% for humans on the TU-Berlin sketch dataset). Overall, those works directly apply the hand-designed local natural image features for sketch recognition. Indeed, most of the sketch recognition works do not make use of sketch shape features to design low-level descriptors that are specific to sketches, but rather take into consideration the low-level features used on natural images since they outperform sketch shape features. Their methods are simple and undergo three easy steps: selecting the most informational areas or points, computing the best local features, and building a structure model as well as an evaluation scheme by spatial cues (structural information) or local feature similarity (matching score). These frameworks has the following disadvantages: (i) they seriously depend on parameters of the chosen local features and models selected, meaning that it is necessary to play randomly with combinations of features

and models until the best recognition score is achieved; (ii) they treat sketches as low level concepts such as shape, edge map, contour or strokes. However, a total sketch is associated with a real object or scene which makes it a high-level concept with an certain degree of abstraction. Therefore, sketches can be hardly classified with such frameworks (with specific local features), which explains why none of the above methods was able to exceed the humans' performance in sketch recognition.

Recently, with the emergence of deep learning models including Convolutional Neural Networks (ConvNets) which have dominated top benchmark results on visual recognition challenges (such as ILSVRC (Jia Deng et al., 2009)), some works introduced ConvNet based models for sketch recognition (Sarvadevabhatla and Babu, 2015; Seddati, 2015; Wang et al., 2016; Yu et al., 2017). All of these works run there experiments on the TU Berlin dataset (Eitz et al., 2012). The work of (Yu et al., 2017) builds a ConvNet from scratch by using sequential ordering information to model multiple channels per image as well as using a multi-scale network ensemble why Bayesian fusion. Similarly, authors of (Seddati, 2015) choose to train a ConvNet from scratch too by adjusting the architecture for a better performance. (Sarvadevabhatla and Babu, 2015) retrains two popular pre-trained ConvNets (Imagenet ConvNet and a modified version of LeNet ConvNet) but uses only a subset of the whole dataset by eliminating drawn sketches whose identity is difficult to discern for human beings. Another attempt was made by (Wang et al., 2016) who introduce an augmentation technique via edge-preserving sketch resizing as well as a multi-angle voting technique. Following the path of learning sketch feature using deep networks, we introduce a novel deep learning strategy for sketch classification that copes with the lack of data. Indeed, the challenge is to perform a good sketch classification even with a limited number of training data (13500 training instances with 56 instances per category). As we know, deep neural networks require a very large number of data to be able to learn the correct value of all parameters of network (millions of parameters). To do so, we propose a multi-modal learning approach for sketch recognition based on an ensemble learning framework which is composed of two deep learning models: (i) a randomly initialized ConvNet model, and (ii) an existing model pre-trained on images (photos of natural objects), knowing that images and sketches are two different domains having different semantic description. We further show that our framework achieves better or comparable recognition performance to existent sketch recognition techniques.

3.3 Methodology

One reliable approach to improving the performance of neural networks by a few percent is to train multiple independent and different models and, at test time,

combine individual predictions of each model as an ensemble in order to obtain final predictions. Moreover, the improvements are more dramatic with higher model variety in the ensemble. In our study, we suggest/propose to use the ensemble approach as an alternative to solve the issue of lack of labeled data. In other words, we suggest that combining models properly will boost the overall performance even if we have few training data in hand.

One common approach to forming an ensemble is to use the top n models having the highest recognition score during cross-validation. In practice, this process is easily done since it does not require additional retraining of models after cross-validation. Indeed, fusing the predictions of top models at test time is most helpful since these models are different from each other and have different parameters. There are different ways to combine models. The work of (Frazão and Alexandre,

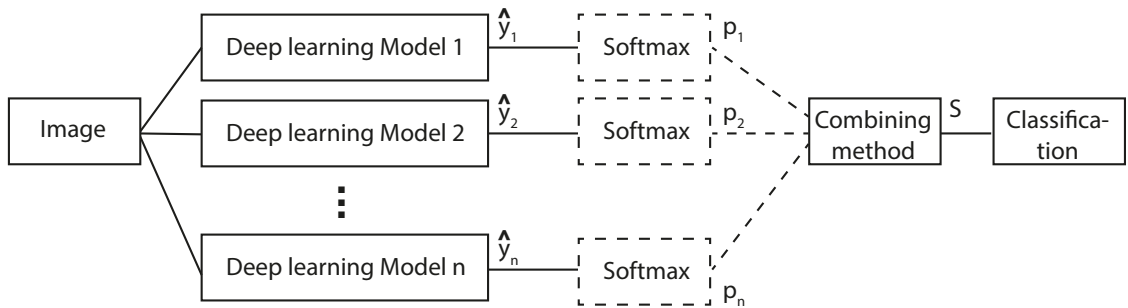


Figure 3.2: Ensemble learning framework at test time using either direct output vectors $\hat{\mathbf{y}}_1, \dots, \hat{\mathbf{y}}_n$ or output probabilities $\mathbf{p}_1, \dots, \mathbf{p}_n$ coming out the softmax layer (the softmax being represented by dashed boxes).

2014) propose to combine output probability vectors of different ConvNet models $\mathbf{p}_1, \dots, \mathbf{p}_n$, as shown in Fig. 3.4 where the softmax layer (dashed boxes) is included. Within this work, different combination ways are presented including: simple average, weighted average based on model accuracies, and weighted average based on model rankings.

For an index i of the output probability vector (i.e., the class index), the simple average over output probabilities s^i is given by,

$$s^i = \frac{1}{n} \sum_{j=1}^n p_j^i \quad (3.1)$$

where p_j^i is the output of the model j of a given input for the class label i , and n the number of models in our network ensemble.

As for the method Weighted average based on model accuracies, the purpose is to give more importance to a model by applying a different weight for each network. In the validation set, models that have a higher recognition rate will have a larger weight when combining the predictions. Given some input image, the

output probabilities of each model j are multiplied by a weight λ_j before the final prediction as explained in the equation below:

$$s^i = \sum_{j=1}^n \lambda_j p_j^i, \quad (3.2)$$

$$\lambda_j = \frac{A_j}{\sum_i A_i}$$

where λ_j is the weight of model j , and A_j the accuracy at test time of model j . In this model architecture of combining networks, weights are proportional to the accuracy in validation set.

As for the Weighted average based on model rankings method, more importance is given to the model which has the higher accuracy score. In other words, the weight λ_j of each model j is based on the order of accuracy in the validation set as follows,

$$\lambda_j = \frac{\text{Ranking}(A_j)}{\sum_i \text{Ranking}(A_i)} \quad (3.3)$$

In our study, we introduce new combination architectures. First, instead of applying these three methods (mentioned below) on output probabilities $\mathbf{p}_1, \dots, \mathbf{p}_n$, we propose to apply these methods on $\hat{\mathbf{y}}_1, \dots, \hat{\mathbf{y}}_n$ which correspond to output feature vectors coming out of the last fully connected layer, as shown when omitting the softmax layer (dashed boxes) within Fig.3.4. Those output feature vectors will be further denoted as “direct outputs”. As such, we propose the four methods below,

(i) Simple average over direct outputs. For an index i of the direct outputs vector (i.e., the class index), this combination method is defined as,

$$s^i = \frac{1}{n} \sum_{j=1}^n \hat{y}_j^i \quad (3.4)$$

(ii) Weighted average based on model accuracies over direct outputs.

$$s^i = \sum_{j=1}^n \lambda_j \hat{y}_j^i, \lambda_j = \frac{A_j}{\sum_i A_i} \quad (3.5)$$

(iii) Weighted average based on model rankings over direct outputs.

$$s^i = \sum_{j=1}^n \lambda_j \hat{y}_j^i, \lambda_j = \frac{\text{Ranking}(A_j)}{\sum_i \text{Ranking}(A_i)} \quad (3.6)$$

(iv) Maximum. This ensemble architecture combines the predictions of our models (either the direct outputs or output probabilities) by keeping, for each class label, the maximum class label prediction of the n models. The Maximum over direct

outputs is expressed as,

$$s^i = \max(\hat{y}_i^1, \dots, \hat{y}_i^n) \quad (3.7)$$

Furthermore, we propose a new combination approach based on maximum predictions on output probabilities (which will be referred to as Maximum over output probabilities), which is defined as,

$$s^i = \max(p_i^1, \dots, p_i^n) \quad (3.8)$$

After applying one of the previously mentioned ensemble methods, the index within the vector S which has the maximum element/value represents the predicted class label or prediction as follows,

$$prediction = \operatorname{argmax}_i s^i \quad (3.9)$$

3.4 Experimental setup

In order to demonstrate the effectiveness of our ensemble learning approaches over existent ones, the following steps are used within our experiment. First, we train a randomly initialized ConvNet on sketch images. Second, we select two known pre-trained ConvNet models (already pre-trained on natural images e.g., photos), namely the VGG-F and VGG-deep-16 models, then fine-tune their parameters by training them on sketch images. Finally, we perform our ensemble learning approach at test time on validation set instances. In order to complete these steps, a description of the dataset as well as the training process of our ConvNet models are given below.

3.4.1 Dataset.

We perform our sketch recognition evaluation on the TU Berlin dataset (Eitz et al., 2012). This latter consists of 20000 non-expert sketches, which are drawn by humans, divided into 250 categories. Each category has 80 sketches and each sketch size is 1111 by 1111. In order to implement these sketch images into our ConvNet models, they need to be resized and shrunk to a fixed size before training. Accordingly, they are resized to [224x224] in our experiment. Moreover, since our training dataset is very small (56 images per category), data augmentation is performed to reduce overfitting by replicating training instances (sketches) with a number of transformations. For each epoch, we randomly select half of the images within the batch and perform the following transformations upon them: horizontal reflection

e.g., mirroring (across vertical axis), random rotation between -30 and 30 degrees, as well as rescaling the width and height of each image independently using a random resizing range between 0.4 and 1.

3.4.2 Deep learning Model architecture and training.

For training all ConvNet models, we use the stochastic (mini-batch) gradient descent as our gradient-based optimization method. Hyper-parameters of the ConvNet models are set according to Table 3.1.

Table 3.1: Training hyper-parameters of ConvNet models. The abbreviations 'LR' and 'Mom.' stand for the learning rate and the momentum respectively.

ConvNet models	LR	Batch size	Mom.	#epochs	#layers	#parameters
Randomly initialized ConvNet	0.01	135	0.9	300	20 layers:5conv.+3FC.	6.8e+06(26MB)
VGG-F	0.001	135	0.9	40	22 layers:5conv.+3FC.	5.8e+07(220MB)
VGG-deep-16	0.001	135	0.9	40	44 layers:16conv.+3FC.	1.4e+08(548MB)

Randomly initialized ConvNet. As mentioned in the methodology 3.3, due to the lack of data, we need to have a high dropout rate. As such, the dropout rate is set to 0.5 and applied on the first two fully connected layers. Moreover, setting a high dropout requires having a large/deep neural network (with more hidden layers). Accordingly, our ConvNet has eight layers, each having a different configuration as described in Table 3.2. Unlike the existing pre-trained models, the filter size of the first convolutional layer are chosen to be relatively large 15×15 – compared to 11×11 filter in VGG-F model (Chatfield et al., 2014) and 3×3 filter in VGG-deep-16 model (Simonyan and Zisserman, 2015), since larger filters are more appropriate for sketch modeling. Indeed, as mentioned in (Yu et al., 2017), sketches lack texture information, i.e., a small round-shaped patch can be recognized as eye or button in a photo based on texture, but this is infeasible for sketches. So, small filters which are useful for detecting textured information are not necessary in our case; and rather employ larger filters which are good at capturing more structured context than textured one.

The final layer has 250 output units corresponding to 250 categories (that is the number of unique classes in the TU-Berlin sketch dataset), upon which we place a Softmax loss.

Pre-trained models (VGG-f and VGG-verydeep-16). The goal is to build a

Table 3.2: Architecture of the ConvNet model trained from scratch.

Id	Layer	Type	Filter size	Filter Num	Stride	Pad	Output Size
0		Input	-				224×224
1		Conv	15×15	64	3	0	70×70
2	L1	ReLU	-	-	-	-	70×70
3		Maxpool	3×3	-	2	0	34×34
4		Conv	5×5	128	1	0	30×30
5	L2	ReLU	-	-	-	-	30×30
6		Maxpool	3×3	-	2	0	14×14
7		Conv	3×3	256	1	1	14×14
8	L3	ReLU	-	-	-	-	14×14
9		Conv	3×3	256	1	1	14×14
10	L4	ReLU	-	-	-	-	14×14
11		Conv	3×3	256	1	1	14×14
12	L5	ReLU	-	-	-	-	14×14
13		Maxpool	3×3	-	2	0	6×6
14		Conv (FC)	6×6	512	1	0	1×1
15	L6	ReLU	-	-	-	-	1×1
16		Dropout	-	-	-	-	1×1
17		Conv (FC)	1×1	512	1	0	1×1
18	L7	ReLU	-	-	-	-	1×1
19		Dropout	-	-	-	-	1×1
20	L8	Conv (FC)	1×1	250	1	0	1×1

rough global sketch model by fine-tuning the two famous ConvNet models (Chatfield et al., 2014; Simonyan and Zisserman, 2015) using the sketch dataset (Eitz et al., 2012), with the help of transfer learning. But, before doing so, we need to make appropriate changes within the training set instances and the models’ architectures, which are summarized below:

- i Input images replicated. Since the two models were pre-trained on color images, they accept only three channel images (corresponding to RGB channels). Therefore, to fine-tune these models, we replicate each training instance three times to obtain three-channel image instances.
- ii Local Response Normalization for VGG-f model removed. Local Response Normalization (LRN) implements a form of lateral inhibition, which is found in real neurons. This is used pervasively in contemporary ConvNet recognition architectures (Krizhevsky et al., 2012; Chatfield et al., 2014; Simonyan and Zisserman, 2015). However, in practice LRN’s benefit is due to providing “brightness normalization”. This is not necessary in sketches since brightness

is not an issue in line-drawings. Thus removing LRN layers makes learning faster without sacrificing performance.

- iii Last Fully Connected Layer changed. The two ConvNet models are previously pre-trained using the ImageNet image classification data, which is designed for 1000 categories. However, the sketch classification dataset contains 250 categories only. So, we need to change the 1000-classes way into 2-classes by replacing the last fully connected layer that has 1000 nodes with one that has 250 nodes.
- iv Dropout layers added. Two dropout layers are added before the two fully connected layers FC6 and FC7.
- v Learning rate adjusted. Since VGG models have already been trained, a small learning rate 0.001 is used.

Let's note that the VGG-f and VGG-verydeep-16 models are deeper than the randomly initialized ConvNet with larger filters banks (more neurons) and more layers. Indeed, VGG-f and VGG-verydeep-16 have $5.8e+07$ and $1.4e+08$ parameters respectively compared to $6.8e+06$ parameters for the randomly initialized ConvNet.

3.5 Experiments

The first step is to train ConvNets, namely the randomly initialized ConvNet as well as the pre-trained VGG-f and VGG-verydeep-16 models, according to the experimental setup mentioned above. Next, we form our ensemble learning framework by selecting the top two models which achieved the highest accuracy/performance and applying a combination method upon them. The overall process is summarized in Figure 3.3. The ensemble learning framework is tested using the proposed combination methods: (i) simple average over direct outputs, (ii) weighted average based on model accuracies over direct outputs, (iii) weighted average based on model rankings over direct outputs, (iv) maximum of direct outputs, (v) maximum of output probabilities. And, in order to show the superiority of our combination methods with respect to existing ones, we run the same ensemble learning framework using combination methods of (Frazão and Alexandre, 2014) including the simple average over output probabilities and weighted average based on model rankings over output probabilities.

Afterward, in order to prove the effectiveness of our ensemble learning framework, we compare it with the most famous state-of-the art sketch recognition techniques.

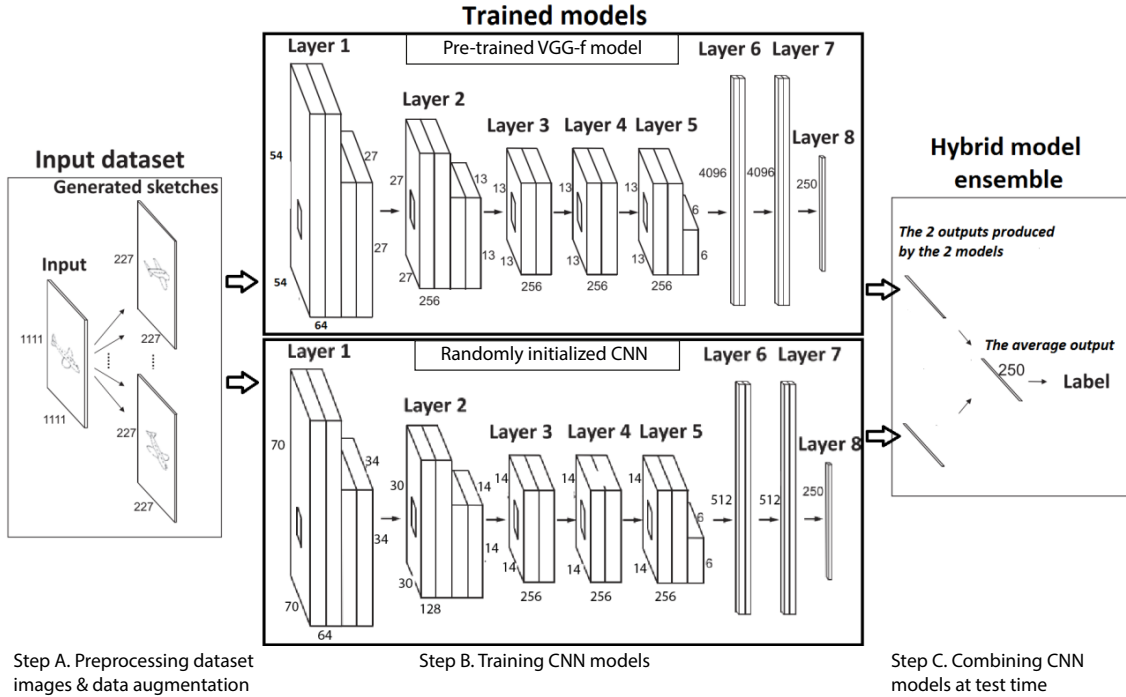


Figure 3.3: Overall steps of our experiment.

3.6 Results

3.6.1 Results of training ConvNet models.

After running the experiments on ConvNet models, results are reported in Table 3.4. We observe that training the VGG-f and VGG-verydeep-16 models yields an accuracy of 0.691 and 0.686 respectively 3.4. Even though VGG-verydeep-16 model has a deeper architecture than VGG-f model, its performance is less than this latter. We deduce that adding more layers helps with photo image classification but does not improve sketch classification results. This is due to the small filter sizes used in VGG-verydeep-16 model. Its small 3 by 3 first layer filters are here to capture texture and low-level information of photo images that are not present in sketch images. So, as illustrated in Fig. 3.4.b, VGG-verydeep-16 first layer filters (right) fail to display obvious edges. Nonetheless, half of VGG-f first layer filters (middle) resemble Gabor filters (as shown in Fig. 3.4.c), conveying that the large first layer filters (of size 11×11) of VGG-f pre-trained model were able to capture the details of sketches.

As to VGG-deep-16 model, even with small first layer filters (3×3), it was able to do a good job on sketch recognition with an accuracy of 0.686. Therefore, we can conclude that any model previously pre-trained on photo-based images can perform quite well on the sketch classification task.

On the other hand, training the randomly initialized ConvNet model achieves

a recognition rate of 0.717 (Table 3.4). This implies that training a deep network from scratch yields better classification results than training a pre-trained model. Getting better accuracy results using our model was expected since its architecture was designed specifically for sketch images. Indeed, larger filters (15 by 15) used in the first layer of our model catches more details than the 11 by 11 filters of the VGG-f model and 3 by 3 filters of the VGG-deep-19 model. Larger filters are used here to capture mid-level and high-level representations of sketch images. As illustrated in Fig.3.4.a, most of the first layer filters of our model from scratch (left) have strong edges and look like Gabor filters. On the other hand, these edges happen to be stronger and more complex than VGG-f first layer filter edges (middle).

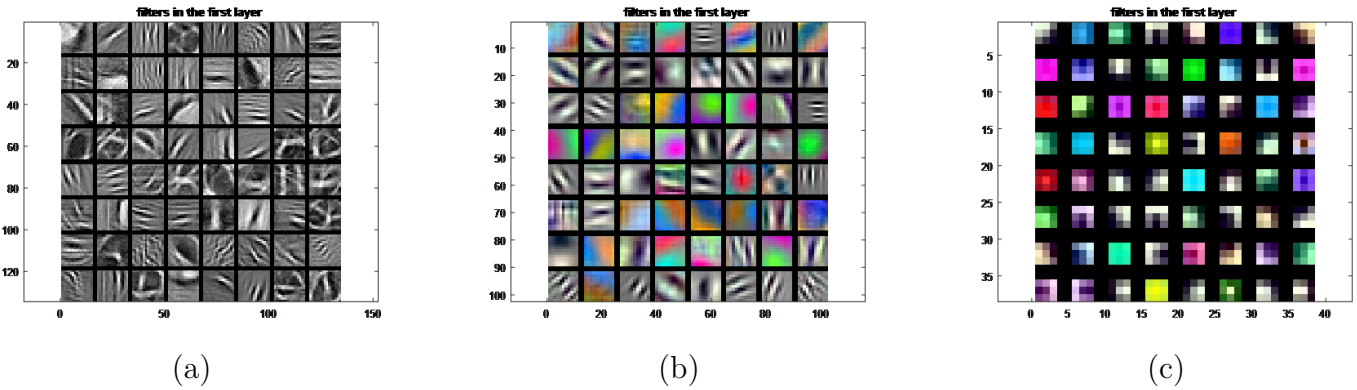


Figure 3.4: Visualization of the learned filters in the first layer of each model. Left: first layer filters of our ConvNet model from scratch (64 filters in total, each having a size of 15 by 15 pixels). Middle: first layer filters of our fine-tuned model of VGG-f (64 filters in total, each having a size of 11 by 11 pixels). Right: first layer filters of our fine-tuned model of VGG-deep (64 filters in total, each having a size of 3 by 3 pixels). We can see the similarities and differences between the three models.

3.6.2 Results of the ensemble learning framework using different combination approaches.

Upon applying the hybrid model ensemble at test time using our proposed combinations methods as well as existing ones (Frazão and Alexandre, 2014), results are obtained and displayed in Table 3.3. From this latter, the following observations can be made:

- i The ensemble learning framework based on any combination method performs higher than the trained ConvNets (the randomly initialized ConvNet, VGG-f and VGG-deep-16).
- ii Running the ensemble learning framework based on the combination method of simple average over direct outputs yields the best recognition rate (74.7%) and thus is superior to existing combination methods (simple average over

Table 3.3: Results of the hybrid model using different model ensemble architectures, i.e. different methods of combining models at test time.

Output type -> Combination method	Accuracy
Output probabilities -> Simple average (Eq. 3.1)	0.7417
Output probabilities -> Weighted average based on model rankings (Eq. 3.3)	0.7186
Output probabilities -> Maximum (Eq. 3.8)	0.7463
Direct outputs -> Simple average (Eq. 3.4)	0.7470
Direct outputs -> Weighted average based on model accuracies - (Eq.3.5)	0.7470
Direct outputs -> Weighted average based on model rankings - (Eq.3.6)	0.7412
Direct outputs -> Maximum (Eq. 3.7)	0.7057

output probabilities and weighted average based on model rankings over output probabilities. Therefore, the simple average over direct outputs method can be regarded as a proper combination approach for ensemble learning frameworks.

- iii Also, results show that the ensemble framework with the weighted average based on model accuracies over direct outputs achieves a little less same performance than the method of simple average over direct outputs. This is because the weight λ_1 corresponding to the randomly initialized ConvNet and the weight λ_2 attributed to the pre-trained VGG-f model are close to 0.5 with $\lambda_1 = 0.717/(0.717+0.691) = 0.51$ and $\lambda_2 = 0.691/(0.717+0.691) = 0.49$.
- iv On the other hand, employing the method of weighted average based on model rankings over direct outputs achieves less accuracy than the method of simple average over direct outputs. By looking at the corresponding weights $\lambda_1 = 2/3$ and $\lambda_2 = 1/3$, we can see that giving a greater importance to the network achieving a better accuracy in the validation set doesn't improve the final prediction of the model ensemble because the two model accuracies are not far apart and the difference between them is only 0.026.
- v Running the ensemble learning framework using the proposed maximum over output probabilities method achieves better performance than existing combination methods (simple average over output probabilities and weighted average based on model rankings over output probabilities). Thus, the maximum over output probabilities method can also be considered as a good combination method for ensemble learning frameworks.
- vi Applying the maximum over output probabilities method yields 4.13% less accuracy than the maximum over direct outputs method. Indeed, unlike the latter method which treats the outputs as uncalibrated (i.e., direct outputs

are difficult to interpret), the former method uses the softmax classifier which gives a slightly more intuitive output and also has normalized class probabilities. That is why employing the maximum method to the normalized class probabilities gives a better result than employing the maximum method on direct outputs.

3.6.3 Comparative results.

Our results as well as results of previous sketch recognition works are presented in Table 3.4. By observing these results, we infer the following remarks:

Table 3.4: Comparison of different sketch classification methods as well as the human recognition performance.

Method	Accuracy
Humans’ recognition (Eitz et al., 2012)	0.732
HOG-SVM (Eitz et al., 2012)	0.56
Stargraph + KNN (Li et al., 2013)	0.615
MKL-SVM (Li et al., 2015)	0.658
FV-SP-SVM (Schneider and Tuytelaars, 2014)	0.689
Sketch-a-Net M-Cha+M-Sca (Yu et al., 2017)	0.749
DeepSketch (Seddati, 2015)	0.7542
Deep sketch feature for cross domain image retrieval (Wang et al., 2016)	0.773
Ours	
* Randomly initialized ConvNet	0.717
* Pre-trained VGG-f	0.691
* Pre-trained VGG-deep-16	0.686
* Ensemble learning framework (Eq. 3.4) (?)	0.747

- i Comparison between our ConvNet models and hand-crafted feature based techniques: The randomly initialized ConvNet yields a better performance than hand-crafted feature techniques (Eitz et al., 2012; Li et al., 2013, 2015; Schneider and Tuytelaars, 2014) architecture, which implies that our model is able to compete with the recent sketch recognition methods. Similarly, fine-tuning the two pre-trained models (VGG-f (Chatfield et al., 2014) and VGG-deep-16 (Simonyan and Zisserman, 2015)) achieves a higher recognition rate than hand-crafted feature techniques, conveying that fine-tuned models fill the semantic gap better than low-level representations of hand-crafted feature works. Finally, our hybrid model surpasses by far all engineered feature studies with an accuracy of 0.747.
- ii Comparison between our ensemble learning framework and automated feature based techniques: Our hybrid model yields almost the same accuracy than “Sketch-a-net” work (Yu et al., 2017), with the former having the following

advantages over the latter: (i) the former model is based on two network ensemble only with a simple average fusion while the latter is based on five network ensemble with a Bayesian fusion, which is more complex and computationally more expensive; (ii) our hybrid model uses only one channel per input image whereas the model in (Yu et al., 2017) has to first discretize the image strokes into sequential groups for each input image, then use these groups as three channels per input image, which takes longer processing time and larger memory. On the other hand, our hybrid model achieves comparable results to (Seddati, 2015; Wang et al., 2016) works with 0.72% and 2.6% less accuracy than (Seddati, 2015) and (Wang et al., 2016) respectively.

3.6.4 Running cost.

Our models were implemented on a 2.60GHz CPU (without explicit parallelization) using the MatConvNet (Vedaldi and Lenc, 2015) toolbox of Matlab. We trained our model from scratch for 500 epochs, with each epoch having different input data thanks to random data augmentation. Training the randomly initialized ConvNet took 68 hours (500 epochs) compared to 72 hours for VGG-f model (40 epochs) and 240 hours for VGG-deep-16 (40 epochs). As for combining/fusing ConvNet models at test time, it does not require more than 10 minutes.

3.7 Conclusion

In this chapter, we investigated an ensemble learning framework to alleviate the issue of lack of data by combining deep neural networks at test time. The process consists of : (i) training multiple ConvNet models, each having a different architecture and each being trained differently on different training sets, (ii) selecting the top two ConvNet models having the highest scores and combining their outputs at test time to get a hybrid model. Experiments were conducted on the sketch classification task using a sketch dataset with 250 object categories/classes. Comparing our hybrid model with recent works showed that our hybrid method outperforms most of the up-to-date works. We also showed that fusing the two models based on averaging over their direct outputs is better than averaging or computing the maximum over their output probabilities. Finally, regarding sketch recognition, we believe that the input image should be treated as a whole and do not have to be discretized into sequential parts as done in (Yu et al., 2017). Indeed, a well-trained ConvNet should behave like human neural networks (e.g., our neurons) and has to be able to recognize the whole image without having to do additional processing by splitting it into segments.

Bibliography

- Yang Cao, Hai Wang, Changhu Wang, Zhiwei Li, Liqing Zhang, and Lei Zhang. MindFinder: Interactive Sketch-based Image Search on Millions of Images. In Proceedings of the International Conference on Multimedia (MM), pages 1605–1608, 2010. ISBN 9781605589336. doi: 10.1145/1873951.1874299.
- Ken Chatfield, Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Return of the Devil in the Details: Delving Deep into Convolutional Nets. In British Machine Vision Conference, 2014. ISBN 1-901725-52-9. doi: 10.5244/C.28.6.
- Gregory Cohen, Saeed Afshar, Jonathan Tapson, and Andre Van Schaik. EMNIST: Extending MNIST to handwritten letters. In Proceedings of the International Joint Conference on Neural Networks, volume 2017-May, pages 2921–2926, 2017. ISBN 9781509061815. doi: 10.1109/IJCNN.2017.7966217.
- Comnist. Cyrillic oriented MNIST: A dataset of Latin and Cyrillic letter images, 2019.
- Gabriella Csurka, Christopher R. Dance, Lixin Fan, Jutta Willamowski, and Cédric Bray. Visual categorization with bags of keypoints. In European Conference on Computer Vision (ECCV) International Workshop on Statistical Learning in Computer Vision, pages 59–74, 2004. ISBN 9780335226375. doi: 10.1234/12345678.
- JL Cunchillos, P Xella, and JÁ Zamora. Il corpus informatizzato delle iscrizioni fenicie e puniche: un progetto italo-spagnolo. Università di Palermo, 2005.
- Navneet Dalal and William Triggs. Histograms of Oriented Gradients for Human Detection. 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition CVPR05, 1(4):886–893, 2005. ISSN 1063-6919. doi: 10.1109/CVPR.2005.177.
- Nibaran Das, Ram Sarkar, Subhadip Basu, Mahantapas Kundu, Mita Nasipuri, and Dipak Kumar Basu. A genetic algorithm based region sampling for selection of local features in handwritten digit recognition application. Applied Soft Computing Journal, 12(5):1592–1606, 2012. ISSN 15684946. doi: 10.1016/j.asoc.2011.11.030.
- Mathias Eitz, Kristian Hildebrand, Tamy Boubekeur, and Marc Alexa. Sketch-Based Image Retrieval: Benchmark and Bag-of-Features Descriptors. {IEEE} Trans. Vis. Comput. Graph., 17(11):1624–1636, 2011.

- Mathias Eitz, James Hays, and Marc Alexa. How do humans sketch objects? In *ACM Transactions on Graphics*, volume 31, pages 44–1, 2012. ISBN 0730-0301. doi: 10.1145/2185520.2335395.
- SR Fischer. *History of Writing*. Reaktion b edition, 2003.
- Xavier Frazão and LA A. Alexandre. Weighted Convolutional Neural Network Ensemble. In *Iberoamerican Congress on Pattern Recognition*, pages 674–681, 2014. ISBN 978-3-319-12568-8; 978-3-319-12567-1. doi: 10.1007/978-3-319-12568-8_82.
- Rui Hu and John Collorosso. A performance evaluation of gradient field HOG descriptor for sketch based image retrieval. *Computer Vision and Image Understanding*, 117(7):790–806, 2013. ISSN 1090235X. doi: 10.1016/j.cviu.2013.02.005.
- Hervé Jégou, Matthijs Douze, Cordelia Schmid, and Patrick Pérez. Aggregating local descriptors into a compact image representation. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 3304–3311, 2010. ISBN 9781424469840. doi: 10.1109/CVPR.2010.5540039.
- Jia Deng, Wei Dong, R Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A large-scale hierarchical image database. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009. ISBN 978-1-4244-3992-8. doi: 10.1109/CVPRW.2009.5206848.
- B Klare, Li Zhifeng, and A K Jain. Matching Forensic Sketches to Mug Shot Photos. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(3):639–646, 2011. ISSN 0162-8828. doi: 10.1109/TPAMI.2010.180.
- A Krizhevsky and G Hinton. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.
- Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012. ISBN 9780415468442. doi: 10.1061/(ASCE)GT.1943-5606.0001284.
- A Lawgali, M Angelova, and A Bouridane. HACDB: Handwritten Arabic Characters Database for Automatic Character Recognition. In *Visual Information Processing (EUVIP)*, 2013 4th European Workshop on, pages 255–259, 2013. ISBN 9788293269137.

- Thomas Leung and Jitendra Malik. Representing and recognizing the visual appearance of materials using three-dimensional textons. *International Journal of Computer Vision*, 43(1):29–44, 2001. ISSN 09205691. doi: 10.1023/A:1011126920638.
- Yi Li, Yi-Zhe Song, and Shaogang Gong. Sketch Recognition by Ensemble Matching of Structured Features. In *Proceedings of the British Machine Vision Conference 2013*, pages 35.1–35.11, 2013. ISBN 1-901725-49-9. doi: 10.5244/C.27.35.
- Yi Li, Timothy M. Hospedales, Yi Zhe Song, and Shaogang Gong. Free-hand sketch recognition by multi-kernel feature learning. *Computer Vision and Image Understanding*, 137:1–11, 2015. ISSN 1090235X. doi: 10.1016/j.cviu.2015.02.003.
- Li Deng. The MNIST Database of Handwritten Digit Images for Machine Learning Research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012. ISSN 1053-5888. doi: 10.1109/msp.2012.2211477.
- David G. Lowe. Distinctive Image Features from Scale-Invariant Keypoints. *International Journal of Computer Vision*, 60(2):91–110, nov 2004. ISSN 0920-5691. doi: 10.1023/B:VISI.0000029664.99615.94.
- Shuxin Ouyang, Timothy Hospedales, Yi Zhe Song, and Xueming Li. Cross-Modal face matching: Beyond viewed sketches. In *Lecture Notes in Computer Science*, volume 9004, pages 210–225. 2015. ISBN 9783319168074. doi: 10.1007/978-3-319-16808-1_15.
- F Perronnin and C Dance. Fisher Kenrels on Visual Vocabularies for Image Categorization. In *Proc. {CVPR}*, pages 1–8, 2007. ISBN 1424411807.
- Ravi Kiran Sarvadevabhatla and R. Venkatesh Babu. Freehand Sketch Recognition Using Deep Features. arXiv preprint arXiv:1502.00254, feb 2015.
- Rosália G. Schneider and Tinne Tuytelaars. Sketch classification and classification-driven analysis using Fisher vectors. *ACM Transactions on Graphics*, 33(6):1–9, 2014. ISSN 07300301. doi: 10.1145/2661229.2661231.
- Omar Seddati. DeepSketch : Deep Convolutional Neural Networks for Sketch Recognition and Similarity Search. In *13th International Workshop on Content-Based Multimedia Indexing (CBMI)*, pages 1–6, 2015. ISBN 9781467368704.
- Pierre Sermanet, David Eigen, Xiang Zhang, Michael Mathieu, Rob Fergus, and Yann LeCun. OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks. arXiv preprint arXiv:1312.6229, 2013. ISSN 10636919. doi: 10.1109/CVPR.2015.7299176.

- Karen Simonyan and Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. In Proceedings of the 3rd International Conference on Learning Representations, sep 2015. ISBN 1097-0142 (Electronic)\n0008-543X (Linking). doi: 10.2146/ajhp170251.
- Sivic and Zisserman. Video Google: a text retrieval approach to object matching in videos. In Proceedings Ninth IEEE International Conference on Computer Vision, pages 1470–1477 vol.2, 2003. ISBN 0-7695-1950-4. doi: 10.1109/ICCV.2003.1238663.
- Andrea Vedaldi and Karel Lenc. MatConvNet - Convolutional Neural Networks for MATLAB. In Proceedings of the 23rd ACM international conference on Multimedia, pages 689–692, 2015. ISBN 978-1-4503-3459-4. doi: 10.1145/2733373.2807412.
- F Wang, L Kang, and Y Li On. Sketch-based 3d shape retrieval using convolutional neural networks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 1875–1883, 2015.
- Xinggang Wang, Xiong Duan, and Xiang Bai. Deep sketch feature for cross-domain image retrieval. *Neurocomputing*, 207:387–397, 2016. ISSN 18728286. doi: 10.1016/j.neucom.2016.04.046.
- Paolo Xella and José Á. Zamora. 7 Phoenician Digital Epigraphy: CIP Project, the State of the Art. In *Crossing Experiences in Digital Epigraphy*, pages 93–101. 2019. doi: 10.1515/9783110607208-008.
- J Yang, K Yu, and T Huang. Supervised translation-invariant sparse coding. 2010.
- Qian Yu, Yongxin Yang, Feng Liu, Yi-Zhe Song, Tao Xiang, and Timothy M Hospedales. Sketch-a-net: A deep neural network that beats humans. *International journal of computer vision*, 122(3):411–425, 2017.
- Matthew D. Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833, 2014. ISBN 9783319105895. doi: 10.1007/978-3-319-10590-1_53.
- Xi Zhou, Kai Yu, Tong Zhang, and Thomas S. Huang. Image classification using super-vector coding of local image descriptors. In *European conference on computer vision*, volume 6315 LNCS, pages 141–154, 2010. ISBN 3642155545. doi: 10.1007/978-3-642-15555-0_11.

Chapter 4

Novel augmentation and voting approach at test time

In previous sections, we have seen how transfer learning and ensemble learning could improve deep neural networks performance with a limited number of training data. Further in this chapter, we propose other techniques to alleviate the issue of the lack of data, including a novel augmentation technique at the training phase and a voting approach at testing phase. By running experiments on the sketch recognition task too (using the same dataset as the previous chapter - chapter 4), we show that these techniques increase the accuracy score.

In the study below, we define our augmentation and voting approach (Section 4.1). In the next section (4.2), a description of our experiments is given and results of our approach are reported and compared to state-of-art methods. Finally, Section 4.3 summarizes our work.

4.1 Methodology

In this section, we introduce the key components of our approaches. First, we discuss our scaling augmentation technique at the training stage and its advantage in increasing the training data size for the neural network to be properly trained (4.1.1). Next, we explore our multi-angle and multi-scale voting strategy at test time based on the augmentation technique used (4.1.2). Fig.4.1 illustrates steps of our overall framework.

4.1.1 Data augmentation technique.

While the work of (Wang et al., 2016) suggests to resize sketches proportionally via an edge-preserving technique, we propose a skewing method that preserves the overall meaning of the original sketch. Unlike natural images which lose information

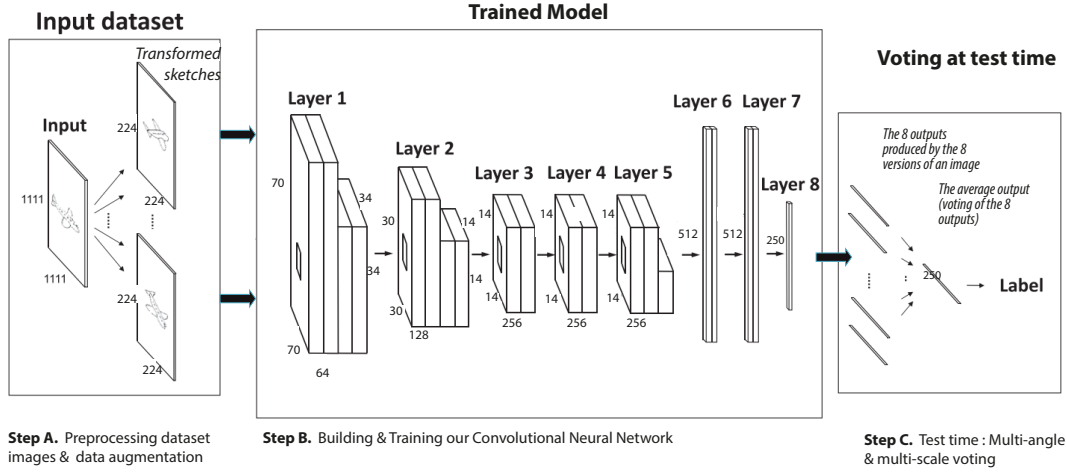


Figure 4.1: Illustration of the overall steps within our deep learning framework.

after undergoing skewing transformations due to their nature (having texture and colors, and being complex with a lot of details), sketches keep the most meaningful information/representations after being skewed. This can be seen through the example of an airplane sketch object (Fig.4.2) whose details are preserved even after the skewing transformation.

The proposed skewing strategy is based on rescaling of the width and the height of the sketch independently (e.g., non-uniformly). During the training phase, for each epoch and within each batch, we select half of the batch instances then perform the following transformations for each instance:

- i we randomly apply a horizontal reflection i.e. mirroring (across vertical axis) as well a rotation between -30 and 30 degrees,
- ii We randomly rescale the width and the height of the sketch independently using random values w and h respectively which range from 0.4 to 1 to obtain a skewed sample of size $round(224 * w) \times round(224 * h)$; then we place the sketch at the middle of a 224×224 frame ;
- iii we add horizontal and vertical translations which range randomly from 0 to $112 - 224 * w$ and 0 to $224 - 224 * h$ respectively.

4.1.2 Multi-angle and multi-scale voting during test time

After training our model, it is time to go to the testing phase. In the conventional way, we simply feed each testing image into the trained network then see whether the computed output corresponds to the true label of that image. However, in our study, for each and every testing image, we make multiple transformed versions of it (applying the same transformations of augmentation techniques performed

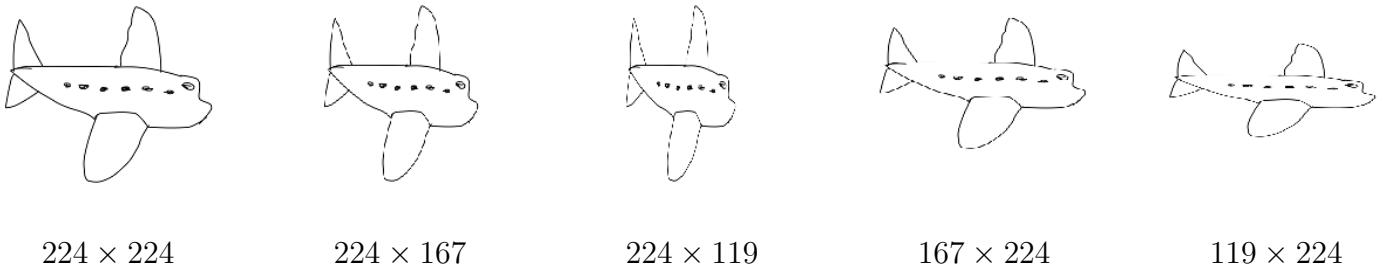


Figure 4.2: (a) Original sample of an airplane object, (b) Transformed samples of the airplane generated via our skewing augmentation technique.

during training), and their corresponding network outputs are fed into a voting system that votes for the best output, which is then compared to the true label of that image. The idea is to take into consideration properties of the training process during evaluation (at test time). Indeed, the goal is to take advantage of the features diversity learned by data augmentation at test time. As we know, our deep learning model was trained on an augmented data composed of original images as well as multiple rescaled, shifted, mirrored and rotated variants of the original images. This implies that our neural network will have no trouble recognizing these variants during testing. Accordingly, we propose to test each testing instance (testing sketch image) several times in different ways:

- i We perform multiple transformations on that instance which are similar to the transformations done to augment the data at the training phase. Indeed, we rotate and rescale the instance non-uniformly (by resizing the width and height independently at random) in order to obtain n variants ;
- ii Each variant j within the n variants is fed into the deep learning model to produce the output vector $\hat{y}_j = \hat{y}(angle_j, width_j, height_j)$;
- iii The n output vectors $\{\hat{y}_j\}_{j=1}^n$ are averaged before making a prediction. Indeed, according to (Wang et al., 2016), using the voting scheme with different angles to recognize a sketch is like that humans recognize object for several times. The same thing applies for our multi-angle multi-scaling voting scheme. For an index i of the output vector (i.e., the class index), the average output at that index s^i is given by Eq.4.1,

$$s^i = \frac{1}{n} \sum_{j=1}^n \hat{y}_j^i \quad (4.1)$$

where \hat{y}_j^i is the output of the variant j (rotated or rescaled variant of the original image) of the class label i . Thus, the final prediction or output vector \mathbf{s} is a result of the voting of all transformed versions of the testing instance;

- iv We pick the index i within s^i having the maximum value and set it as the predicted class label, as illustrated in Eq.4.2. Then, we compare it with the true label of the instance.

$$C = \operatorname{argmax}_i s^i \quad (4.2)$$

We follow these steps for every image in the testing dataset to get the final accuracy.

Our approach is similar to the learning ensemble approach (chapter 3). While this latter combines between predictions produced by different models for each input instance at test time, our voting method combines predictions produced by feeding multiple transformed variants of the instance into one trained model.

Unlike the work of (Wang et al., 2016) which performs multi-angle voting on computed Softmax output probabilities vectors (where each vector is produced by feeding the network outputs vector to the softmax layer), our work uses multi-angle and multi-scale voting directly on the network outputs vector rather than Softmax output probabilities because it gives a better performance, as shown in chapter 3.

4.2 Experiments and results

4.2.1 Voting technique based on different variants.

To conduct this experiment, we first train the ConvNet model on augmented data using the augmentation techniques mentioned above. Then we apply the voting technique at test time by following steps mentioned in the methodology (Section 4.1). Indeed, for each testing sketch instance, we choose to perform 1 to 3 angle rotations at random where the angle is chosen among the following values $\{-25^\circ, 0^\circ, 25^\circ\}$ and 1 to 4 skewing (rescaling) operations where sketch dimensions (width and height) are chosen randomly among one of the following values $\{224, 197, 167, 141, 119\}$. The resultant variants are listed in Table 4.1. We finally obtain 2 to 7 variants of the testing instance which are then fed into our network to produce 2 to 7 output vectors. These output vectors are then averaged out to make the final prediction.

We run multiple simulations at test time with different configurations (or combinations) of variants, as illustrated in Table 4.1. And their corresponding results are reported in the same table. After feeding the trained ConvNet model with original testing instances, we achieve a classification accuracy of 0.717. Meanwhile, running the voting technique on the original testing instance and 2 of its rotated variants $\{-25^\circ, +25^\circ\}$ improves the accuracy by 1.4%. Additionally, adding an extra rescaled variant (of size 168×168) to these variants makes the recognition rate

even higher by 1.5%. Adding again an extra rescaled variant (of size 127×127) increases accuracy by 0.7%. Nonetheless, adding 2 extra rescaled variants helps improve the accuracy by only 0.13%. From these results, we deduce that: (i) more predictions (coming from transformed versions of the image) tend to contribute to a higher accuracy, (ii) not only the multi-voting technique improves accuracy, but the multi-scaling technique too, which demonstrates the efficiency of our voting technique, (iii) as the number of transformed versions of the image in the ensemble increases, the performance typically monotonically improves though with diminishing returns.

Table 4.1: Results of our trained model at test time using different characteristics. For each image, we perform rotation and rescaling to get n variants of the image. Then, these images are fed into our network, producing n outputs. After that, we see which output has the highest number of votes. Finally, this output is compared to the true label of the image.

#variants	Variant(s)	Accuracy
1	$(0^\circ, 224 \times 224)$	0.717
3	$(0^\circ, 224 \times 224); (-25^\circ, 224 \times 224); (25^\circ, 224 \times 224)$	0.731
4	$(0^\circ, 224 \times 224); (-25^\circ, 224 \times 224); (25^\circ, 224 \times 224); (0^\circ, 168 \times 168)$	0.746
5	$(0^\circ, 224 \times 224); (-25^\circ, 224 \times 224); (25^\circ, 224 \times 224); (0^\circ, 168 \times 168); (0^\circ, 127 \times 127)$	0.753
7	$(0^\circ, 224 \times 224); (-25^\circ, 224 \times 224); (25^\circ, 224 \times 224); (0^\circ, 197 \times 197); (0^\circ, 167 \times 167); (0^\circ, 141 \times 141); (0^\circ, 119 \times 119)$	0.7543

4.2.2 Comparative study.

In order to prove the effectiveness of our augmentation and voting technique, we evaluate/compare our results against models previously mentioned in section 3.6.1. Results are reported in Table 4.2. From these results, the following remarks can be made:

1. Comparison between our model and hand-crafted feature based techniques: Our model produces an accuracy of 0.7543, which is higher than all engineered feature works including HOG-SVM (Eitz et al., 2012), Stargraph-KNN (Li et al., 2013), MKL-SVM (Li et al., 2015) and FV-SP-SVM (Schneider and Tuytelaars, 2014).
2. Comparison between our model and automated feature based techniques: Our model performs better than the Ensemble learning framework (?) as well as Sketch-a-net (Yu et al., 2017), and slightly better than DeepSketch work (Seddati, 2015), implying that our model can easily compete with state-of-the-art

Table 4.2: Comparison of different sketch classification methods as well as the human recognition performance.

Method	Accuracy
Humans' recognition (Eitz et al., 2012)	0.732
HOG-SVM (Eitz et al., 2012)	0.56
Stargraph + KNN (Li et al., 2013)	0.615
MKL-SVM (Li et al., 2015)	0.658
FV-SP-SVM (Schneider and Tuytelaars, 2014)	0.689
Ensemble learning framework (?)	0.747
Sketch-a-Net M-Cha+M-Sca (Yu et al., 2017)	0.749
DeepSketch (Seddati, 2015)	0.7542
Ours (with 7 variants at test time) (?)	0.7543
Deep sketch feature for cross domain image retrieval (Wang et al., 2016)	0.773

deep learning recognition works. On the other hand, our model has 0.0187 less accuracy than the work (Wang et al., 2016). However, Even our model happens to be simpler than (Wang et al., 2016) because: (i) at the preprocessing stage, the former model is faster than the latter with the rescaling augmentation technique being easily computed compared to the edge-preserving resizing technique. Indeed, this edge-preserving resizing technique requires that we locate coordinates of all the sketch points whose pixel value is 1 for each sketch instance, before rotating and scaling the list of coordinates; (ii) Furthermore, during training, the former model is less complex, computationally less expensive and less time consuming than the latter with $6.8e+06$ vs $5.93e+07$ parameters having to be learned; (iii) finally, at test time, the former model is faster the latter since only 7 variants of each sketch instance are used for voting versus 18 variants.

4.3 Conclusion

In this chapter, we proposed a novel augmentation and voting technique for ConvNet classification of sketch images which helps resolve the issue of lack of data. The augmentation approach is based on skewing sketches as well as applying other transformations (rotations, shifting, mirroring) on them before feeding them to the ConvNet for training. Indeed, as opposed to natural images, sketch images have most of their details preserved even after being skewed due to the free-hand nature. As for the voting technique, we take advantage of the rich ConvNet feature representations learned via our augmentation approach to vote, for each sketch instance at test time, among multiple augmented variants of that instance by averaging over ConvNet outputs of these variants. Running experiments over the sketch database

shows the superiority of our approach over all hand-crafted feature based techniques and most automated feature based techniques.

Bibliography

Yang Cao, Hai Wang, Changhu Wang, Zhiwei Li, Liqing Zhang, and Lei Zhang. MindFinder: Interactive Sketch-based Image Search on Millions of Images. In Proceedings of the International Conference on Multimedia (MM), pages 1605–1608, 2010. ISBN 9781605589336. doi: 10.1145/1873951.1874299.

Ken Chatfield, Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Return of the Devil in the Details: Delving Deep into Convolutional Nets. In British Machine Vision Conference, 2014. ISBN 1-901725-52-9. doi: 10.5244/C.28.6.

Gregory Cohen, Saeed Afshar, Jonathan Tapson, and Andre Van Schaik. EMNIST: Extending MNIST to handwritten letters. In Proceedings of the International Joint Conference on Neural Networks, volume 2017-May, pages 2921–2926, 2017. ISBN 9781509061815. doi: 10.1109/IJCNN.2017.7966217.

Comnist. Cyrillic oriented MNIST: A dataset of Latin and Cyrillic letter images, 2019.

Gabriella Csurka, Christopher R. Dance, Lixin Fan, Jutta Willamowski, and Cédric Bray. Visual categorization with bags of keypoints. In European Conference on Computer Vision (ECCV) International Workshop on Statistical Learning in Computer Vision, pages 59–74, 2004. ISBN 9780335226375. doi: 10.1234/12345678.

JL Cunchillos, P Xella, and JÁ Zamora. Il corpus informatizzato delle iscrizioni fenicie e puniche: un progetto italo-spagnolo. Università di Palermo, 2005.

Navneet Dalal and William Triggs. Histograms of Oriented Gradients for Human Detection. 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition CVPR05, 1(4):886–893, 2005. ISSN 1063-6919. doi: 10.1109/CVPR.2005.177.

Nibaran Das, Ram Sarkar, Subhadip Basu, Mahantapas Kundu, Mita Nasipuri, and Dipak Kumar Basu. A genetic algorithm based region sampling for selection of local features in handwritten digit recognition application. Applied Soft Computing Journal, 12(5):1592–1606, 2012. ISSN 15684946. doi: 10.1016/j.asoc.2011.11.030.

- Mathias Eitz, Kristian Hildebrand, Tamy Boubekeur, and Marc Alexa. Sketch-Based Image Retrieval: Benchmark and Bag-of-Features Descriptors. {IEEE} Trans. Vis. Comput. Graph., 17(11):1624–1636, 2011.
- Mathias Eitz, James Hays, and Marc Alexa. How do humans sketch objects? In ACM Transactions on Graphics, volume 31, pages 44–1, 2012. ISBN 0730-0301. doi: 10.1145/2185520.2335395.
- SR Fischer. History of Writing. Reaktion b edition, 2003.
- Xavier Frazão and LA A. Alexandre. Weighted Convolutional Neural Network Ensemble. In Iberoamerican Congress on Pattern Recognition, pages 674–681, 2014. ISBN 978-3-319-12568-8; 978-3-319-12567-1. doi: 10.1007/978-3-319-12568-8_82.
- Rui Hu and John Collorosse. A performance evaluation of gradient field HOG descriptor for sketch based image retrieval. Computer Vision and Image Understanding, 117(7):790–806, 2013. ISSN 1090235X. doi: 10.1016/j.cviu.2013.02.005.
- Hervé Jégou, Matthijs Douze, Cordelia Schmid, and Patrick Pérez. Aggregating local descriptors into a compact image representation. In Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, pages 3304–3311, 2010. ISBN 9781424469840. doi: 10.1109/CVPR.2010.5540039.
- Jia Deng, Wei Dong, R Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A large-scale hierarchical image database. In IEEE Conference on Computer Vision and Pattern Recognition, pages 248–255, 2009. ISBN 978-1-4244-3992-8. doi: 10.1109/CVPRW.2009.5206848.
- B Klare, Li Zhifeng, and A K Jain. Matching Forensic Sketches to Mug Shot Photos. IEEE Transactions on Pattern Analysis and Machine Intelligence, 33(3):639–646, 2011. ISSN 0162-8828. doi: 10.1109/TPAMI.2010.180.
- A Krizhevsky and G Hinton. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.
- Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In Advances in neural information processing systems, pages 1097–1105, 2012. ISBN 9780415468442. doi: 10.1061/(ASCE)GT.1943-5606.0001284.
- A Lawgali, M Angelova, and A Bouridane. HACDB: Handwritten Arabic Characters Database for Automatic Character Recognition. In Visual Information

Processing (EUVIP), 2013 4th European Workshop on, pages 255–259, 2013. ISBN 9788293269137.

Thomas Leung and Jitendra Malik. Representing and recognizing the visual appearance of materials using three-dimensional textures. *International Journal of Computer Vision*, 43(1):29–44, 2001. ISSN 09205691. doi: 10.1023/A:1011126920638.

Yi Li, Yi-Zhe Song, and Shaogang Gong. Sketch Recognition by Ensemble Matching of Structured Features. In *Proceedings of the British Machine Vision Conference 2013*, pages 35.1–35.11, 2013. ISBN 1-901725-49-9. doi: 10.5244/C.27.35.

Yi Li, Timothy M. Hospedales, Yi Zhe Song, and Shaogang Gong. Free-hand sketch recognition by multi-kernel feature learning. *Computer Vision and Image Understanding*, 137:1–11, 2015. ISSN 1090235X. doi: 10.1016/j.cviu.2015.02.003.

Li Deng. The MNIST Database of Handwritten Digit Images for Machine Learning Research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012. ISSN 1053-5888. doi: 10.1109/msp.2012.2211477.

David G. Lowe. Distinctive Image Features from Scale-Invariant Keypoints. *International Journal of Computer Vision*, 60(2):91–110, nov 2004. ISSN 0920-5691. doi: 10.1023/B:VISI.0000029664.99615.94.

Shuxin Ouyang, Timothy Hospedales, Yi Zhe Song, and Xueming Li. Cross-Modal face matching: Beyond viewed sketches. In *Lecture Notes in Computer Science*, volume 9004, pages 210–225. 2015. ISBN 9783319168074. doi: 10.1007/978-3-319-16808-1_15.

F Perronnin and C Dance. Fisher Kernels on Visual Vocabularies for Image Categorization. In *Proc. {CVPR}*, pages 1–8, 2007. ISBN 1424411807.

Ravi Kiran Sarvadevabhatla and R. Venkatesh Babu. Freehand Sketch Recognition Using Deep Features. *arXiv preprint arXiv:1502.00254*, feb 2015.

Rosália G. Schneider and Tinne Tuytelaars. Sketch classification and classification-driven analysis using Fisher vectors. *ACM Transactions on Graphics*, 33(6):1–9, 2014. ISSN 07300301. doi: 10.1145/2661229.2661231.

Omar Seddati. DeepSketch : Deep Convolutional Neural Networks for Sketch Recognition and Similarity Search. In *13th International Workshop on Content-Based Multimedia Indexing (CBMI)*, pages 1–6, 2015. ISBN 9781467368704.

Pierre Sermanet, David Eigen, Xiang Zhang, Michael Mathieu, Rob Fergus, and Yann LeCun. OverFeat: Integrated Recognition, Localization and Detection

- using Convolutional Networks. arXiv preprint arXiv:1312.6229, 2013. ISSN 10636919. doi: 10.1109/CVPR.2015.7299176.
- Karen Simonyan and Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. In Proceedings of the 3rd International Conference on Learning Representations, sep 2015. ISBN 1097-0142 (Electronic)\n0008-543X (Linking). doi: 10.2146/ajhp170251.
- Sivic and Zisserman. Video Google: a text retrieval approach to object matching in videos. In Proceedings Ninth IEEE International Conference on Computer Vision, pages 1470–1477 vol.2, 2003. ISBN 0-7695-1950-4. doi: 10.1109/ICCV.2003.1238663.
- Andrea Vedaldi and Karel Lenc. MatConvNet - Convolutional Neural Networks for MATLAB. In Proceedings of the 23rd ACM international conference on Multimedia, pages 689–692, 2015. ISBN 978-1-4503-3459-4. doi: 10.1145/2733373.2807412.
- F Wang, L Kang, and Y Li On. Sketch-based 3d shape retrieval using convolutional neural networks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 1875–1883, 2015.
- Xinggong Wang, Xiong Duan, and Xiang Bai. Deep sketch feature for cross-domain image retrieval. *Neurocomputing*, 207:387–397, 2016. ISSN 18728286. doi: 10.1016/j.neucom.2016.04.046.
- Paolo Xella and José Á. Zamora. 7 Phoenician Digital Epigraphy: CIP Project, the State of the Art. In *Crossing Experiences in Digital Epigraphy*, pages 93–101. 2019. doi: 10.1515/9783110607208-008.
- J Yang, K Yu, and T Huang. Supervised translation-invariant sparse coding. 2010.
- Qian Yu, Yongxin Yang, Feng Liu, Yi-Zhe Song, Tao Xiang, and Timothy M Hospedales. Sketch-a-net: A deep neural network that beats humans. *International journal of computer vision*, 122(3):411–425, 2017.
- Matthew D. Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833, 2014. ISBN 9783319105895. doi: 10.1007/978-3-319-10590-1_53.
- Xi Zhou, Kai Yu, Tong Zhang, and Thomas S. Huang. Image classification using super-vector coding of local image descriptors. In *European conference on computer vision*, volume 6315 LNCS, pages 141–154, 2010. ISBN 3642155545. doi: 10.1007/978-3-642-15555-0_11.

Part V

Learning from imbalanced data

Data imbalance is one of the problems that we face when applying machine learning to real-world problems, especially in image classification. With all the improvements in machine learning, especially deep learning, research in this area is drawing more attention from academics and even industry. This part addresses the data imbalance problem. Indeed, we propose cost-sensitive learning approaches applied on neural networks to deal with both classification and regression under imbalanced domains.

In the first chapter, we address classification under imbalanced domains by introducing an approach able to automatically learn robust features for both frequent and rare classes. Our approach automatically assigns misclassification penalties to each class based on the frequency of occurrence of that class. This approach is investigated in the context of shallow networks (Multi-Layer Perceptrons) and deep networks (Convolutional Neural Networks). Moreover, it offers not only a better convergence but also a faster convergence boosting optimization. Furthermore, we show that the efficiency of our cost-sensitive learning approach on classifying imbalanced data relies on the loss function used and that the loss function should satisfy certain properties in order for our approach to be efficient. Running experiments with any loss function satisfying these properties along with our approach shows the superiority of this latter over the baseline algorithm as well as existent techniques.

In the second chapter, we propose a cost-sensitive algorithm as well as evaluation techniques for regression tasks under imbalanced domains. As opposed to the previous chapter in which the data is nominal, meaning that the number of data target variables (e.g., the number of classes or labels) the classifier needs to learn to recognize is fixed, data in this chapter is continuous. Classification algorithms for imbalanced data have been thoroughly studied within machine learning; however few predictive models for regression tasks with imbalanced continuous target variables exist. As such, a cost-sensitive learning algorithm based on a neural network trained on the minimization of a biased loss function is introduced. Results show a higher or comparable performance and convergence speed to existent techniques. Moreover, there is also a lack of evaluation techniques dealing with regression tasks within imbalanced domains. To this matter, new approaches for performance assessment of regression tasks with imbalanced domains are proposed, including new scalar measures, namely Geometric Mean Error (*GME*) and Class-Weighted Error (*CWE*), and graphical-based ones, namely REC_{TPR} , REC_{TNR} , REC_{G-Mean} and REC_{CWA} curves. Unlike standard measures, our evaluation strategies are shown to be more robust to data imbalance as they reflect the performance of both rare and frequent events.

Chapter 1

Approaches for Handling Classification under Imbalanced Domains

1.1 Introduction

The development in information science has enabled an explosive growth of data, which attracts more and more researchers to engage in the field of big data analytics. This brings a great opportunity for data mining and knowledge discovery and many challenges as well. A noteworthy challenge is data imbalance. Indeed, in many real-world classification applications, the collected data manifests a ‘skewed’ or ‘imbalanced’ distribution where data for some object classes is abundant while data for others is scarce. With the presence of minority (positive or rare) classes with heavily under-represented data compared to majority (frequent or negative) classes, during the training process, classification algorithms tend to learn more from data belonging to the majority classes, thus being more biased towards these latter and resulting in showing very poor classification accuracy on the minority classes. Therefore, features or representations present within minority classes’ data tend to be ignored and not adequately learned. Within many applications, the minority instances actually represent the concept of interest (such as fraud in banking operations, abnormal cell in medical data, dangerous activity in a a continuous surveillance task, object classification etc.), which makes the detection of these rare events even more important. By feeding such imbalanced data at training, standard classifiers such as k-NN, SVM, decision tree and neural network tend to learn more from more observed instances (e.g., belonging to the majority classes) than from under-represented ones (e.g. belonging to the minority classes), therefore resulting in erroneous performance on test data, especially rare ones. Indeed, the imbalanced distribution of data forces classification algorithms to be biased towards the majority classes. To this end, we need to find means to (it is necessary to) improve

the overall accuracy of these algorithms without unduly sacrificing the precision of any of the majority or minority class. These challenges motivate us to propose a new neural network model to tackle the class imbalance problem in real-world data (Sadouk et al., 2020). In this chapter, we introduce a cost-sensitive learning approach using either shallow or deep neural networks (Multi-Layer Perceptrons or Convolutional Neural Networks) to achieve promising performance in classifying largely imbalanced datasets.

The key contributions of this study are as follows.

- i Our approach achieves a better classification performance than the baseline algorithm and existent techniques.
- ii It can be regarded as a boosting technique in optimization which makes a faster NN learning and thus a faster convergence.
- iii It generalizes to shallow neural networks such as Multi-Layer Perceptrons (MLPs) and to deep neural networks such as Convolutional Neural Networks (ConvNets), therefore handling classification of 1-, 2-, and 3-dimensional input datasets.
- iv We also propose a demonstration to show which properties have to be present within a loss function for the cost-sensitive strategy to be efficient. Then, we further show which of the common loss functions is suitable for this strategy.

The remainder of this chapter is organized as follows. We briefly discuss the related work in the next section (Section 1.2). Then, we introduce and describe our proposed cost-sensitive learning algorithm (Section 1.3.1), analyze properties of the loss function that are necessary for the cost-sensitive strategy to hold/be efficient (Section 1.3.2), and discuss the convergence speed once the cost-sensitive technique is applied (Section 1.3.3). Implementation details including the description of datasets as well as NN hyper-parameters are provided in Section 1.4. Experiments and results are summarized in Sec. 1.5 and the paper concludes in Sec. 1.6.

1.2 Related work

To solve the class imbalance problem, research is ongoing with variety of techniques. Approaches to solve this imbalanced data are broadly divided into two categories: Algorithm- and Data-level approaches.

1.2.1 Data-level methods

The data-level techniques do not affect the learning algorithm itself but rather modify the data distribution to solve the imbalanced classification problem. Also known as data-preprocessing techniques, they employ a pre-processing step to re-balance the label distribution. Preprocessing is often performed before building learning model so as to attain better input data. This means that instead of applying a learning algorithm directly to the provided training sample, we first pre-process this data according to the goals of the user. Any standard learning algorithm can then be applied to the pre-processed dataset. The most frequently used data-level approaches are re-sampling techniques whose goal is to re-balance the sample space for an imbalanced dataset in order to alleviate the effect of the skewed distribution in the learning process. Attempts of hybrid sampling which combines between over-sampling and under-sampling techniques were also proposed (Batista et al., 2004; Jeatrakul et al., 2010; Ramentol et al., 2012). Another direction toward balancing data is dynamic sampling. One example of such sampling is the work of (Pouyanfar et al., 2018) where a deep learning model (Convolutional Neural Network) is trained such that the performance metric on the reference dataset is utilized to adjust the class distribution of training samples of the next iteration. In other words, more samples of classes with low F1-scores are to be selected for the next iteration. Nonetheless, all these data-level approaches present a drawback: the computational cost required for data pre-processing and for the learning of a classification model.

1.2.2 Algorithm-level methods

The main purpose of algorithm-based approaches is to optimize the performance of learning algorithms on unseen data to address the class/data imbalance problem. These approaches modify the learning procedure to improve the sensitivity of the classifier towards minority classes, based on the consideration of the cost associated with misclassifying instances (samples). To this matter, several learning algorithms have been proposed such as cost-sensitive SVM learning (Zhang, Yong and Wang 2013), neuro-fuzzy modeling Gao et al. (2014), an extreme learning machine (ELM) with a weighting based on Adaboost Li et al. (2014), and an ensemble of soft-margin SVMs formed using boosting (Wang and Japkowicz, 2010). In the scope of shallow neural networks also known as Multi-Layer Perceptions (MLP), several cost-sensitive approaches applied to imbalanced problems have been proposed in (Alejo et al., 2007; Castro and de Padua Braga, 2009; Oh, 2011; Castro and Braga, 2013). Such approaches are similar with respect to the loss function which is the L_2 loss (Euclidean loss), with respect to the cost function formulation which is based on the dissociation of the class objectives, and also with respect to the learning rule used

namely the extension of the backpropagation algorithm (Rumelhart et al., 1986). The peculiarities of these methods are in the strategies used to incorporate costs into the classes. In the study of (Alejo et al., 2007), authors proposed to weight the L_2 loss function of a RBF neural network with the parameter $\lambda_k = \frac{\max(N_j)}{N_k}$, where N_k is the number of examples of the k th class, and $\max(N_j)$ is the number of examples of the majority class. Authors also argued that the values of λ_k should be gradually diminished along the training. Following the same direction as (Alejo et al., 2007), the work of (Castro and de Padua Braga, 2009) consists of weighting the same L_2 loss function for a NN defining but this time with $\lambda_k = \frac{1}{N_k}$. Another study (Sangyoon Oh et al., 2011) which assumes a two-class MLPs also proposes to intensify the error signal resulting from the target output neuron of the minority class by setting the parameters that control the order of the modified cost function unequal and equal to 2 and 4 for the dominant and rare classes respectively. Another attempt to deal with binary imbalanced classification was made by (Castro and Braga, 2013) who incorporate into the loss function parameters $\lambda_1 = \frac{N_2}{N_1+N_2}$ and $\lambda_2 = \frac{N_1}{N_1+N_2}$ which weight the positive and negative classes respectively. In the scope of deep neural networks, few studies recently integrated existing class imbalance solutions into deep learning models. The study of (Wang et al., 2016) is an extension of the work of ConvNet binary classification (Castro and de Padua Braga, 2009) to deal with multi-class ConvNet classification. The corresponding authors first propose $\lambda_k = \frac{1}{N_k}$ then propose $\lambda_k = \frac{FE_k}{N_k}$ where $FE_k = \sum_{i=1}^{N_k} \sum_{m=1}^M \hat{y}_m^i - y_m^i$, where \hat{y}_m^i and y_m^i represent the network output and real/desired output for the i th sample at the m th neuron respectively. On the other hand, a novel approach based on a learned embedding with ConvNet is addressed by (Huang et al., 2016) whereby a ConvNet is trained with instances selected through a new quintuplet sampling scheme and the associated triple-header hinge loss. The learned embedding produces features that preserve not only locality across the same-class clusters but also discrimination between classes. Another attempt to solve class imbalance with ConvNets is made by (Yue, 2017) using a weighted softmax loss function where $\lambda_k = 1 + \frac{\max(N_j) - N_k}{\beta * \max(N_j)}$, with β a parameter that controls the scaling of the weighted loss (β chosen to be 20). Another study of (Khan et al., 2018) introduces a cost-sensitive Convolutional Neural Network (ConvNet) whose class-dependent cost matrix E (or weight) is optimized along with neural network parameters during the training phase. With $E(p, q)$ denoting the misclassification cost of classifying an instance belonging to a class p into a different class q , the goal is to turn $E(p, q) = 0$ as $p = q$ during the backpropagation process, using different loss functions (L_2 , SVM hinge, and cross entropy). Another way to address class imbalance is to adopt a hybrid (algorithm and data) approach as in (Harliman and Uchida, 2018) where a novel over-sampling method called Ripple-SMOTE is combined with the weighted

loss function suggested in (Yue, 2017). Our cost-sensitive learning approach comes as an extension of the previously mentioned algorithm-level studies for multi-class classification tasks by providing an asymmetrical learning of NN via a modified (backpropagation) weight update rule.

1.3 Methodology

In this section, we introduce the proposed cost-sensitive loss function for classification under imbalanced domains.

1.3.1 Cost-sensitive learning approach

Empirical studies performed with the backpropagation algorithm show that the imbalance problem is due to the contribution to the loss function from the positive classes in relation to negative classes, where the most contribution to the loss function is produced by negative classes. Therefore, the training process is dominated by these latter. In order to achieve solutions that are sensitive to the importance of each class (i.e., that give equal importance to each class), we propose a cost-sensitive cost function defined as a sum of functionals $\chi(\mathbf{y}^i, \hat{\mathbf{y}}^i)$, $i = 1, \dots, N$ which are weighted functionals $\ell(\mathbf{y}^i, \hat{\mathbf{y}}^i)$ such that $\chi(\mathbf{y}^i, \hat{\mathbf{y}}^i) = \lambda(m)\ell(\mathbf{y}^i, \hat{\mathbf{y}}^i)$, where $\lambda(m)$ is the weighting parameter corresponding to the class m of instance i . In other words, given M classes and N training samples $N = \sum_{m=1}^M n_m$ with n_m the number of samples of the m^{th} class, the expression of the cost-sensitive cost function is given by,

$$X(\boldsymbol{\theta}) = \frac{1}{N} \sum_{i=1}^N \chi_i(\boldsymbol{\theta}) = L_i(\boldsymbol{\theta}) = \sum_{m=1}^M \lambda(m) \sum_{i=1}^{n_m} \ell(\mathbf{y}^i, \hat{\mathbf{y}}^i) \quad (1.1)$$

The question is how to find the proper parameter $\lambda(m)$ that maximizes classification performance given imbalanced datasets. The solution proposed by (Alejo et al., 2007) was to set $\lambda(m) = \max(n_k)/n_m$, where $k = 1, \dots, M$ and $\max(n_k)$ is the number of examples of the most frequent class. However, with such a solution, $1 \leq \lambda(m) \leq \max(n_k)/\min(n_k)$ implying that $\lambda(m)$ does not have a fixed bound and can get very high as $n_m \ll \max(n_k)$ (i.e., as the number of instances of the most frequent class is much higher than the one of the rare class). Thus, this method can yield to an aggressive weighting of the loss function. Another solution is to come up with a fixed bounded $\lambda(m)$. As such, we first define the relevance of a class m as,

$$\phi(m) = 1 - \frac{n_m}{\max(n_k)} \quad (1.2)$$

where the relevance $\phi(m)$ is a probability ranging from 0 to 1 which increases as the

class m is more positive (more rare) and decreases as the class m is more negative (more frequent). For instance, $\phi(m) \approx 1$ for $n_m \ll \max(n_k)$ and $\phi(m) = 0$ for $m = \max(n_k)$.

Afterward, the weighting parameter $\lambda(m)$ is obtained,

$$\lambda(m) = 1 + \tau\phi(m) \quad (1.3)$$

where τ is the parameter that regulates the weighting of the relevance $\phi(m)$, defined as $\tau \geq 1$. Accordingly, $\lambda(m)$ ranges from 1 to $1 + \tau$. For example, given an instance i whose target is the most frequent class m , the weighting parameter $\lambda(m)$ at instance i (also denoted as λ^i) is equal to 1, meaning that no extra weight is attributed to the functional $\ell(\mathbf{y}^i, \hat{\mathbf{y}}^i)$. On the other hand, with an instance i whose target is the most rare class, $\lambda^i = 1 + \tau$; so $\ell(\mathbf{y}^i, \hat{\mathbf{y}}^i)$ is multiplied by $1 + \tau$.

In order to show the impact of the cost-sensitive approach on the neural network (NN) learning process, let's analyze the cost-sensitive gradient of the cost function. Given equation 1.1 and based on the standard backpropagation algorithm (Rumelhart et al., 1986), the cost-sensitive gradient (e.g., the gradient of the cost-sensitive objective function) for the weight vector $\boldsymbol{\theta}_k$ of the network (which connects the k output node located at the last NN layer l to input nodes at layer $l - 1$) can be written as,

$$\begin{aligned} \nabla X(\boldsymbol{\theta}_k) &= \frac{1}{N} \sum_{i=1}^N \lambda^i \frac{\partial \ell(\mathbf{y}^i, \hat{\mathbf{y}}^i)}{\partial \hat{\mathbf{y}}_k^i} \frac{\partial \hat{\mathbf{y}}_k^i}{\partial \boldsymbol{\theta}_k} \\ &= \frac{1}{N} \sum_{i=1}^N \lambda^i \frac{\partial \ell(y_k^i, \hat{y}_k^i)}{\partial \hat{y}_k^i} \mathbf{a}^i \end{aligned} \quad (1.4)$$

where y_k^i and \hat{y}_k^i are the true label and the network output respectively at node k for instance i , $\frac{\partial \ell(y_k^i, \hat{y}_k^i)}{\partial \hat{y}_k^i}$ is the partial derivative of the loss with respect to \hat{y}_k^i , and \mathbf{a}^i is the activation vector at layer $l - 1$ for that same instance. Let's note that, in our case, no activation function is present between the sum of weighted input nodes (e.g. the linear combination of inputs) and the loss function. We will further show in section 3.2 that omitting this activation function is responsible for improving the efficiency of our cost-sensitive approach. And knowing that the standard cost function (eq. 2.3) has the gradient with respect to $\boldsymbol{\theta}_k$ expressed as,

$$\begin{aligned} \nabla \ell(\boldsymbol{\theta}_k) &= \frac{1}{N} \sum_{i=1}^N \frac{\partial \ell(\mathbf{y}^i, \hat{\mathbf{y}}^i)}{\partial \hat{\mathbf{y}}_k^i} \frac{\partial \hat{\mathbf{y}}_k^i}{\partial \boldsymbol{\theta}_k} \\ &= \frac{1}{N} \sum_{i=1}^N \frac{\partial \ell(y_k^i, \hat{y}_k^i)}{\partial \hat{y}_k^i} \mathbf{a}^i \end{aligned} \quad (1.5)$$

, the cost-sensitive gradient $\nabla X(\boldsymbol{\theta}_k)$ of eq. 1.4 (along with the expression in eq. 1.3) becomes,

$$\nabla X(\boldsymbol{\theta}_k) = \nabla \ell(\boldsymbol{\theta}_k) + \frac{1}{N} \sum_{i=1}^N \tau \phi^i \frac{\partial \ell(y_k^i, \hat{y}_k^i)}{\partial \hat{y}_k^i} \mathbf{a}^i \quad (1.6)$$

where $\phi^i = \phi(m)$ with m being the target class (label) of instance i . So $\nabla X(\boldsymbol{\theta}_k)$ is simply $\nabla \ell(\boldsymbol{\theta}_k)$ plus an extra cost (second term of eq. 1.6) whose value depends on the relevance of instances within the batch. We notice that this extra cost gets higher as the relevance of the instances $i \in \{1, \dots, n\}$ is higher i.e., as the classes of instances are less frequent (more rare). And conversely, this extra cost goes to 0 as the classes of instances are more frequent. Consequently, $|\nabla X(\boldsymbol{\theta}_k)| \gg |\nabla \ell(\boldsymbol{\theta}_k)|$ for more rare instances and $\nabla X(\boldsymbol{\theta}_k) \approx \nabla \ell(\boldsymbol{\theta}_k)$ for more frequent instances. Hence, more learning (more weight update) occurs at positive instances.

Moreover, during the training process, the impact of the extra cost of the gradient (second term of eq. 1.6) diminishes since $\frac{\partial \ell(y_k^i, \hat{y}_k^i)}{\partial \hat{y}_k^i}$ (the gradient of the loss with respect to \hat{y}_k^i) gets smaller over epochs.

1.3.2 Efficiency of the cost-sensitive learning approach based on the nature of the loss function used

Now that the cost-sensitive cost function is defined, it is necessary to define the proper loss function. For this, we first lay out a background on different loss functions for classification. Then, we discuss properties which should be present within loss functions in general to guarantee the efficiency of our cost-sensitive approach. Finally, we apply these properties on some commonly used loss functions to see which ones are applicable to our approach.

1.3.2.1 Background on loss functions for classification

Given loss functions within Table 2.1 (II), graphical representations of these loss functions are displayed in Fig.1.1.a for $y_k^i = 1$ and in Fig.1.1.b for $y_k^i = 0$. Note that in these figures, (i) *hinge*, *hinge₂*, and *hinge₃* are variants of *Mshinge*, *Mshinge₂*, and *Mshinge₃* respectively where $\max_{q \neq t} \hat{y}_q^i = 0$, and (ii) in order to visualize the cross-entropy loss, its activation function $\sigma(\cdot)$ is chosen to be the sigmoid function instead of the softmax function. In our study, we call ‘‘probability estimate loss functions’’ loss functions which are applied to probability estimates $\sigma(\cdot)$ such as softmax or sigmoid functions. In other words, these loss functions are connected to final layer activations (output neurons) which are based on probability estimates. In Table 2.1, examples of such loss functions are $\log \circ \sigma$ and $L_2 \circ \sigma$, while the rest of

the loss functions are applied to final layer activations based on a linear output. As

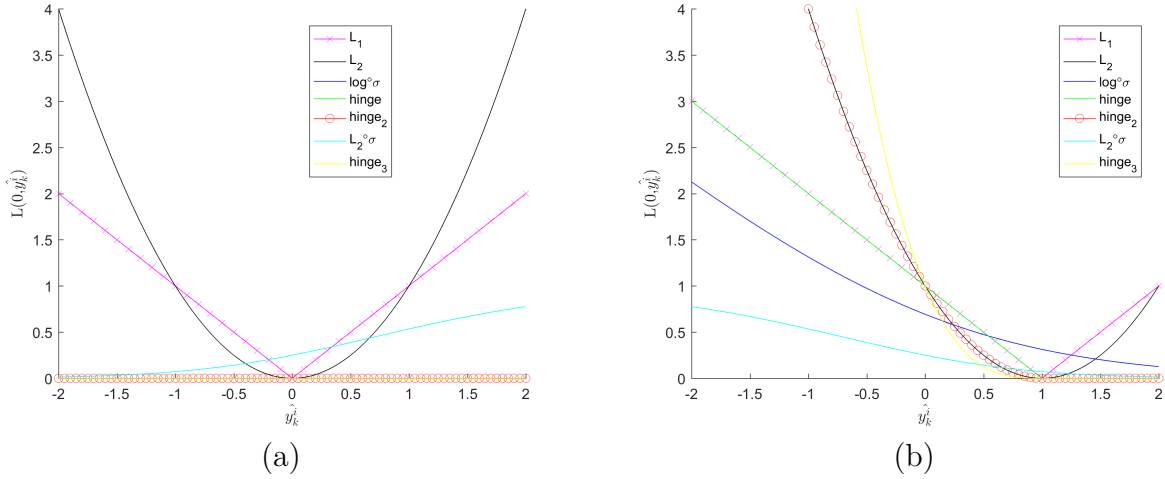


Figure 1.1: Plots (a) and (b) represent different loss functions at an instance i with respect to the network output \hat{y}_k^i when $y_k^i = 1$ and $y_k^i = 0$ respectively. $\sigma(\cdot)$ denotes the sigmoid function. Curves that may not appear in plots are equal to 0.

illustrated in Fig.1.1, each loss function has its own characteristics. For instance, $Mshinge$ loss function is affine whereas $Mshinge_2$, $Mshinge_3$, L_2 , $\log \cdot \sigma$ and $L_2 \circ \sigma$ loss functions are non-affine. Furthermore, as discussed in the study of (Khan et al., 2018), a faster and better convergence is achieved when architecture together with loss function produce a piecewise linear partial derivatives (but not constant) with respect to network outputs, as it is the case for $Mshinge_2$ and L_2 . Also, the same study states that $L_2 \circ \sigma$ loss function has the advantage of being more robust to high noise. So this implies that each loss function has its own properties with advantages and disadvantages. To this matter, the question is how each one of these loss functions behaves when merged with our cost-sensitive approach.

1.3.2.2 Cost-sensitive learning approach applied on loss functions

In this section, we discuss properties that should be present within a loss function $\ell(\cdot)$ for our cost-sensitive learning approach to be efficient and to have a greater impact on NN learning at rare instances i than the standard approach. Given a target value y_k^i defined as $y_k^i \in \{0, 1\}$ (i.e., $y_k^i = 1$ if the target class of instance i is k) and a network output \hat{y}_k^i at node (neuron) k of instance i , and having a misclassification at this latter such that $\hat{y}_k^i < y_k^i$, $y_k^i = 1$, Algorithm 3 below summarizes these properties. Indeed, the goal of Algorithm 3 is to check whether a loss function $\ell(\cdot)$ is suitable for our cost-sensitive learning algorithm. Algorithm 3 is further explained in subsection a. Next, Algorithm 3 is applied on loss functions mentioned in Table 2.1 (subsection b). Then, a graphical interpretation of properties of Algorithm 3 is given (subsection

Algorithm 3: Decision of the efficiency of the cost-sensitive approach on classifying imbalanced data based on the type of loss function used.

Data: network output of instance i at node k of the last layer of a neural network: \hat{y}_k^i , target variable of instance i at node k : y_k^i , loss function: $\ell(\cdot)$, a boolean that is set to true if the objective function $X(\theta)$, the cost-sensitive version of the standard objective function $\ell(\theta)$ is efficient at classifying imbalanced data and to false otherwise: b .

Result: b

initialization;

instructions;

c). if $\frac{\partial \ell(y_k^i, \hat{y}_k^i)}{\partial \hat{y}_k^i}$ is constant with $\frac{\partial \ell(y_k^i, \hat{y}_k^i)}{\partial \hat{y}_k^i} = \beta$ then

- if $\beta \leq -1$ then
 - $b = true$;
- else
 - $b = false$;
- end

else

- if $\frac{\partial^2 \ell(y_k^i, \hat{y}_k^i)}{\partial (\hat{y}_k^i)^2}$ is constant with $\frac{\partial^2 \ell(y_k^i, \hat{y}_k^i)}{\partial (\hat{y}_k^i)^2} = c'$ then
 - if $c' \geq 1$ then
 - $b = true$;
 - else
 - $b = false$;
 - end
- else
 - if $\frac{\partial^2 \ell(y_k^i, \hat{y}_k^i)}{\partial (\hat{y}_k^i)^2} > 0$ is constant with $\frac{\partial^3 \ell(y_k^i, \hat{y}_k^i)}{\partial (\hat{y}_k^i)^3} < 0$ then
 - $b = true$;
 - else
 - $b = false$;
 - end
- end

end

1.3.2.2.1 Explanation of Algorithm 3 Given that $\chi(y_k^i, \hat{y}_k^i) = \lambda^i \ell(y_k^i, \hat{y}_k^i)$ (eq. 1.1), the partial derivative of the cost-sensitive loss function with respect to \hat{y}_k^i is given by,

$$\frac{\partial \chi(\mathbf{y}^i, \hat{\mathbf{y}}^i)}{\partial \hat{y}_k^i} = \lambda^i \frac{\partial \ell(y_k^i, \hat{y}_k^i)}{\partial \hat{y}_k^i} = \frac{\partial \ell(y_k^i, \hat{y}_k^i)}{\partial \hat{y}_k^i} + \phi^i \frac{\partial \ell(y_k^i, \hat{y}_k^i)}{\partial \hat{y}_k^i} \quad (1.7)$$

where $\frac{\partial \ell(y_k^i, \hat{y}_k^i)}{\partial \hat{y}_k^i}$ is the standard (pure) partial derivative of that loss function with respect to \hat{y}_k^i . The question is how large $\left| \frac{\partial \ell(y_k^i, \hat{y}_k^i)}{\partial \hat{y}_k^i} \right|$ should be when misclassification occurs (i.e., when \hat{y}_k^i is far from y_k^i) to make the extra term $\phi^i \frac{\partial \ell(y_k^i, \hat{y}_k^i)}{\partial \hat{y}_k^i}$ (eq. 1.7) at a rare instance i large enough to influence gradient NN learning for that instance. First, let's discuss when misclassification occurs. Usually, it occurs at:

- $\hat{y}_k^i < y_k^i, y_k^i = 1$ since we wish to assign a larger cost for predictions that are further from the true score 1 and that tend to approach the false label score 0. As depicted in Fig.1.1.a, all loss functions share a common feature: they are all increasing as \hat{y}_k^i gets smaller and further from y_k^i . Meanwhile, any prediction $\hat{y}_k^i \geq 1$ shall not be regarded as a misclassification. Indeed, most loss functions (in Fig.1.1.a) assign for $\hat{y}_k^i \geq 1$ either a zero loss as it is the case for *hinge*, *hinge*², and *hinge*³ or a very small loss as for $\log \circ \sigma$ and $L_2 \circ \sigma$,
- $\hat{y}_k^i > y_k^i, y_k^i = 0$, since larger costs are assigned for predictions that are further from 0 and that are approaching the true label score 1. However, as illustrated in Fig.1.1.a.b, not all functions assign a cost when $\hat{y}_k^i > y_k^i$ (at $y_k^i = 0$). Examples of no cost (0 loss) at $y_k^i = 0$ include *hinge* _{n} loss functions with $n \in \{1, 2, 3\}$.

To this matter, we focus on misclassifications occurring for $\hat{y}_k^i < y_k^i$ at $y_k^i = 1$. So, assuming that most NN learning occurs for such misclassification, let's discuss each of the cases within the algorithm.

Case 1: Affine loss function For $\hat{y}_k^i < y_k^i$ such that $y_k^i = 1$, if the loss function is affine in the form of $l(y_k^i, \hat{y}_k^i) = -c\hat{y}_k^i + y_k^i = -c\hat{y}_k^i + 1$, then $\frac{\partial \ell(y_k^i, \hat{y}_k^i)}{\partial \hat{y}_k^i} = -c$. Accordingly,

$$\frac{\partial \chi(y_k^i, \hat{y}_k^i)}{\partial \hat{y}_k^i} = -\lambda^i c \quad (1.8)$$

Having $1\lambda^i + \tau$ ($\tau \geq 1$), we obtain,

$$-(1 + \tau)c \leq \frac{\partial \chi(y_k^i, \hat{y}_k^i)}{\partial \hat{y}_k^i} \leq -c \quad (1.9)$$

So, in order to increase the learning process for positives, we need a large $|\partial \chi(y_k^i, \hat{y}_k^i) / (\partial \hat{y}_k^i)|$, and accordingly a large c . If we set c to a very small value such that $c \approx 0$, then $-(1 + \tau)c \approx 0$. Hence, not only $\frac{\partial \ell(y_k^i, \hat{y}_k^i)}{\partial \hat{y}_k^i} \approx 0$ but also $(\partial \chi(y_k^i, \hat{y}_k^i)) / (\partial \hat{y}_k^i) \approx 0$ for any τ . And, let's note that a partial derivative close to 0 results in the vanishing gradient problem (i.e., a very small partial derivative at the last layer produces very small gradients of last layer weights, which makes gradients of the first and middle

layer weights die and become 0). As such, we propose to set c to a value far from 0 such as $c \geq 1$.

Case 2 and 3: Non-affine loss functions If the loss function is not affine for $\hat{y}_k^i < y_k^i$ at $y_k^i = 1$, it means that its partial derivative with respect to \hat{y}_k^i is not a constant function and can be either an affine or a non-affine function. In other words, $\frac{\partial^2 \ell(y_k^i, \hat{y}_k^i)}{\partial \hat{y}_k^i{}^2}$ can be defined either constant with $\frac{\partial^2 \ell(y_k^i, \hat{y}_k^i)}{\partial \hat{y}_k^i{}^2} = c'$ (referred to as Case 2) or non-constant (referred to as Case 3).

Given a loss function with a non-constant $\frac{\partial \ell(y_k^i, \hat{y}_k^i)}{\partial \hat{y}_k^i}$ and a dataset with rare instances, then, for $y_k^i = 1$, ideally $\frac{\partial \ell(y_k^i, \hat{y}_k^i)}{\partial \hat{y}_k^i} = 0$ at $\hat{y}_k^i = y_k^i$ and $\frac{\partial \ell(y_k^i, \hat{y}_k^i)}{\partial \hat{y}_k^i}$ should be decreasing as $\hat{y}_k^i \rightarrow -\infty$, implying that $\frac{\partial \ell(y_k^i, \hat{y}_k^i)}{\partial \hat{y}_k^i}$ needs to be an increasing function and $\frac{\partial^2 \ell(y_k^i, \hat{y}_k^i)}{\partial \hat{y}_k^i{}^2} > 0$ for $\hat{y}_k^i < y_k^i$. Indeed, if we were to use a loss function with a non-increasing $\frac{\partial \ell(y_k^i, \hat{y}_k^i)}{\partial \hat{y}_k^i}$, then $\frac{\partial \ell(y_k^i, \hat{y}_k^i)}{\partial \hat{y}_k^i} \approx 0$ as \hat{y}_k^i decreases; and as a result this loss function will tend to regard rare instances as outliers and attribute a very small learning to such instances. In fact, knowing that the NN tends to be influenced by frequent instances (i.e., its weights will learn more from these instances) and results in wrong predictions \hat{y}_k^i for rare instances i with $\hat{y}_k^i \ll y_k^i$ at $y_k^i = 1$, this loss function considers those predictions as outliers since they are very far from y_k^i . Therefore, even applying the cost-sensitive approach to such loss function will result in a small $\frac{\partial \chi(y_k^i, \hat{y}_k^i)}{\partial \hat{y}_k^i}$ as \hat{y}_k^i decreases, suggesting that this approach has no impact on NN learning given rare instances. However, applying the cost-sensitive approach on loss functions with $\frac{\partial^2 \ell(y_k^i, \hat{y}_k^i)}{\partial \hat{y}_k^i{}^2} > 0$ makes the slope of $\frac{\partial \ell(y_k^i, \hat{y}_k^i)}{\partial \hat{y}_k^i}$ larger (more positive) and thus $\left| \frac{\partial \chi(y_k^i, \hat{y}_k^i)}{\partial \hat{y}_k^i} \right|$ even larger as \hat{y}_k^i decreases. Having said that $\frac{\partial^2 \ell(y_k^i, \hat{y}_k^i)}{\partial \hat{y}_k^i{}^2} > 0$, the question that arises is how the slope of $\frac{\partial \ell(y_k^i, \hat{y}_k^i)}{\partial \hat{y}_k^i}$ should be (or what the value of $\frac{\partial^2 \ell(y_k^i, \hat{y}_k^i)}{\partial \hat{y}_k^i{}^2}$ should be) in order for the cost-sensitive approach to improve the NN learning process for positives and thus enhance the overall classification performance.

Case 2: Non-affine loss functions with affine partial derivatives w.r.t network outputs Given a constant $\frac{\partial^2 \ell(y_k^i, \hat{y}_k^i)}{\partial \hat{y}_k^i{}^2}$ with $\frac{\partial^2 \ell(y_k^i, \hat{y}_k^i)}{\partial \hat{y}_k^i{}^2} = c'$ ($c' > 0$), then the second-order partial derivative of the cost-sensitive loss function with respect to network output \hat{y}_k^i is defined as,

$$\frac{\partial^2 \chi(y_k^i, \hat{y}_k^i)}{\partial \hat{y}_k^i{}^2} = \lambda^i c' \quad (1.10)$$

If we set c' to be small and close to 0, then $\frac{\partial \ell(y_k^i, \hat{y}_k^i)}{\partial \hat{y}_k^i}$ will decrease at a slow rate as \hat{y}_k^i gets far from y_k^i ($y_k^i = 1$). Knowing that $\frac{\partial \ell(y_k^i, \hat{y}_k^i)}{\partial \hat{y}_k^i} \approx 0$ at $\hat{y}_k^i = y_k^i$ for almost all

loss functions, then as $c' \approx 0$ we get $\frac{\partial \ell(y_k^i, \hat{y}_k^i)}{\partial \hat{y}_k^i} \approx 0$ for $\hat{y}_k^i < y_k^i$, meaning that there will be a very slow learning within the NN. Similarly, the cost-sensitive approach will not influence learning for the positive instance i since the cost-sensitive weighting parameter λ^i does not increase (sharpen) the slope of $\frac{\partial \ell(y_k^i, \hat{y}_k^i)}{\partial \hat{y}_k^i}$ as $\lambda^i c' \approx c' \approx 0$. Hence, c' needs to be sufficiently large for λ^i to have a large impact on $\frac{\partial^2 \chi(y_k^i, \hat{y}_k^i)}{\partial \hat{y}_k^i{}^2}$.

Accordingly, we propose that $c' \geq 1$ so that $(\partial \chi(y_k^i, \hat{y}_k^i)) / (\partial \hat{y}_k^i)$ has a larger slope than $\frac{\partial \ell(y_k^i, \hat{y}_k^i)}{\partial \hat{y}_k^i}$ and so that higher gradient is attributed to positive instances.

Case 3: Non-affine Loss functions with non-affine partial derivatives w.r.t network outputs As mentioned earlier, when dealing with rare instances, a proper non-affine loss function shall have $\frac{\partial^2 \ell(y_k^i, \hat{y}_k^i)}{\partial \hat{y}_k^i{}^2} > 0$ for $\hat{y}_k^i < y_k^i$, $y_k^i = 1$. If this loss function has a non-affine $\frac{\partial \ell(y_k^i, \hat{y}_k^i)}{\partial \hat{y}_k^i}$ with a non-constant and positive $\frac{\partial^2 \ell(y_k^i, \hat{y}_k^i)}{\partial \hat{y}_k^i{}^2}$, this implies that $\frac{\partial \ell(y_k^i, \hat{y}_k^i)}{\partial \hat{y}_k^i}$ is either an increasing and concave or an increasing and convex function. An increasing and concave $\frac{\partial \ell(y_k^i, \hat{y}_k^i)}{\partial \hat{y}_k^i}$ for $\hat{y}_k^i < y_k^i$ at $y_k^i = 1$ would produce a very small gradient close to 0 (i.e., a small NN learning) as the network output \hat{y}_k^i approaches the true label y_k^i and a very large gradient as \hat{y}_k^i gets further away from y_k^i . Consequently, applying the cost-sensitive approach on such partial derivative i.e., multiplying $\frac{\partial \ell(y_k^i, \hat{y}_k^i)}{\partial \hat{y}_k^i}$ by λ_i will still give rise to a small gradient for \hat{y}_k^i close to y_k^i but will result in a higher gradient for \hat{y}_k^i far from y_k^i , which is exactly what we wish for. On the other hand, a convex and increasing partial derivative with respect to \hat{y}_k^i would produce: (i) a quite large gradient for \hat{y}_k^i close to y_k^i resulting in a large NN learning, which is not a desirable property for such \hat{y}_k^i , and (ii) a stagnating and almost constant gradient for \hat{y}_k^i far from y_k^i that could result in a small learning if $\lim_{\hat{y}_k^i} \frac{\partial \ell(y_k^i, \hat{y}_k^i)}{\partial \hat{y}_k^i} = -1$. Accordingly the cost-sensitive version of such partial derivative produces even larger gradients for \hat{y}_k^i close to y_k^i (which is not desirable either) and still a small gradient for \hat{y}_k^i far from y_k^i if $\lim_{\hat{y}_k^i \rightarrow -\infty} \frac{\partial \ell(y_k^i, \hat{y}_k^i)}{\partial \hat{y}_k^i} = -1$.

Therefore, when dealing with non-affine loss functions with non-affine partial derivatives w.r.t network outputs, it is necessary to have an increasing and concave $\frac{\partial \ell(y_k^i, \hat{y}_k^i)}{\partial \hat{y}_k^i}$ (i.e., $\frac{\partial^2 \ell(y_k^i, \hat{y}_k^i)}{\partial \hat{y}_k^i{}^2} > 0$ and $\frac{\partial^3 \ell(y_k^i, \hat{y}_k^i)}{\partial \hat{y}_k^i{}^3} < 0$) for $\hat{y}_k^i < y_k^i$ at $y_k^i = 1$ in order for the cost-sensitive learning strategy to have a large impact on NN learning.

1.3.2.2.2 (b) Application of Algorithm 3 on loss functions L_2 loss function. Among loss functions discussed in Table 2.1, L_2 belongs to Case 2 since its partial derivative with respect to \hat{y}_k^i ,

$$\frac{\partial \ell(y_k^i, \hat{y}_k^i)}{\partial \hat{y}_k^i} = 2(\hat{y}_k^i - y_k^i) = \begin{cases} 2(\hat{y}_k^i - 1), & y_k^i = 1 \\ 2\hat{y}_k^i, & y_k^i = 0 \end{cases} \quad (1.11)$$

is not constant and its second-order partial derivative with respect to \hat{y}_k^i is constant with $\frac{\partial^2 \ell(y_k^i, \hat{y}_k^i)}{\partial \hat{y}_k^i{}^2} = 2$. And, knowing that $\frac{\partial^2 \ell(y_k^i, \hat{y}_k^i)}{\partial \hat{y}_k^i{}^2} \geq 1$, then the L_2 loss function satisfies $b = true$ in Algorithm 3, implying that L_2 along with the proposed cost-sensitive approach could enhance classification under imbalanced domains.

$L_2 \circ \sigma$ loss function. We know that $L_2 \circ \sigma$ is not affine since its partial derivative with respect to \hat{y}_k^i ,

$$\begin{aligned} \frac{\partial \ell(y_k^i, \hat{y}_k^i)}{\partial \hat{y}_k^i} &= 2(\sigma(\hat{y}_k^i) - y_k^i) \sigma'(\hat{y}_k^i) \\ &= 2(\sigma(\hat{y}_k^i) - y_k^i) \sigma(\hat{y}_k^i) (1 - \sigma(\hat{y}_k^i)) \end{aligned} \quad (1.12)$$

is not constant. Moreover, $\frac{\partial^2 \ell(y_k^i, \hat{y}_k^i)}{\partial \hat{y}_k^i{}^2}$ which can be expressed as, and which can also be written as,

$$\frac{\partial^2 \ell(y_k^i, \hat{y}_k^i)}{\partial \hat{y}_k^i{}^2} = 2\sigma(\hat{y}_k^i)(1 - \sigma(\hat{y}_k^i))^2(3\sigma(\hat{y}_k^i) - 1), y_k^i = 1$$

is not constant either, which implies that the partial derivative with respect to \hat{y}_k^i is not affine. Having shown that $L_2 \circ \sigma$ belongs to Case 3 by being non-affine and having a non-affine partial derivative with respect to \hat{y}_k^i , and knowing that $0 \leq \sigma(\hat{y}_k^i) \leq 1$, $0 \leq (1 - \sigma(\hat{y}_k^i))^2 \leq 1$ and $-1 \leq (3\sigma(\hat{y}_k^i) - 1) \leq 2$, we obtain $\frac{\partial^2 \ell(y_k^i, \hat{y}_k^i)}{\partial \hat{y}_k^i{}^2} < 0$ for $]-\infty, -\log(2)[$, which, according to Algorithm 3, suggests that $b = false$ and that applying the cost-sensitive approach on $L_2 \circ \sigma$ (where $\sigma(\hat{y}_k^i) = 1/(1 + e^{(-\hat{y}_k^i)})$) will not improve classification under imbalanced domains.

$Mshinge$ loss function. $\frac{\partial \ell(y_k^i, \hat{y}_k^i)}{\partial \hat{y}_k^i}$ of $Mshinge$ is constant and falls within Case 1 since it is defined as,

$$\frac{\partial \ell(y_k^i, \hat{y}_k^i)}{\partial \hat{y}_k^i} = \begin{cases} -1, & \hat{y}_k^i - \max_{q \neq t} \hat{y}_q^i < 1, y_k^i = 1 \\ +1, & \hat{y}_t^i - \hat{y}_k^i < 1, \hat{y}_k^i = \max_{q \neq t} \hat{y}_q^i, y_t^i = 1 \\ 0, & \text{otherwise} \end{cases} \quad (1.13)$$

And knowing that $\frac{\partial \ell(y_k^i, \hat{y}_k^i)}{\partial \hat{y}_k^i} = -1 : \hat{y}_k^i < 1 + \max_{q \neq t} \hat{y}_q^i, y_k^i = 1$, thus the $Mshinge$ loss function satisfies the condition $b = true$ in Algorithm 3 provided that $\max_{q \neq t} \hat{y}_q^i \geq 0$, inferring that the cost-sensitive approach can be efficient at classifying imbalanced data based on $Mshinge$ only when $\max_{q \neq t} \hat{y}_q^i \geq 0$.

$Mshinge_2$ loss function. $\frac{\partial \ell(y_k^i, \hat{y}_k^i)}{\partial \hat{y}_k^i}$ of $Mshinge_2$ which is computed as follows,

$$\frac{\partial \ell(y_k^i, \hat{y}_k^i)}{\partial \hat{y}_k^i} = \begin{cases} -2(1 + \max_{q \neq t} \hat{y}_q^i - \hat{y}_k^i), & y_k^i = 1 \\ +2(1 + \hat{y}_k^i - \hat{y}_t^i), & \hat{y}_k^i = \max_{q \neq t} \hat{y}_q^i, y_t^i = 1 \end{cases} \quad (1.14)$$

, is not constant, which means that $Mshinge_2$ is not affine at $\hat{y}_k^i < y_k^i, y_k^i = 1$. Also, $\frac{\partial^2 \ell(y_k^i, \hat{y}_k^i)}{\partial \hat{y}_k^i{}^2}$ which can be written as,

$$\frac{\partial^2 \ell(y_k^i, \hat{y}_k^i)}{\partial \hat{y}_k^i{}^2} = \begin{cases} +2, & y_k^i = 1 \\ -2, & \hat{y}_k^i = \max_{q \neq t} \hat{y}_q^i, y_t^i = 1 \end{cases}$$

is constant with $\frac{\partial^2 \ell(y_k^i, \hat{y}_k^i)}{\partial \hat{y}_k^i{}^2} \geq 1$ for $\hat{y}_k^i < y_k^i, y_k^i = 1$, implying that the cost-sensitive approach can be efficient at classifying imbalanced data based on $Mshinge_2$ loss function.

$Mshinge_3$ loss function. $\frac{\partial \ell(y_k^i, \hat{y}_k^i)}{\partial \hat{y}_k^i}$ of $Mshinge_3$ which is expressed as, (18)

$$\frac{\partial \ell(y_k^i, \hat{y}_k^i)}{\partial \hat{y}_k^i} = \begin{cases} -3(1 + m \max_{q \neq t} \hat{y}_q^i - \hat{y}_k^i)^2, & \hat{y}_k^i - \max_{q \neq t} \hat{y}_q^i < 1, y_k^i = 1 \\ +3(1 + \max_{q \neq t} \hat{y}_q^i - \hat{y}_k^i)^2, & \hat{y}_t^i - \hat{y}_k^i < 1, \hat{y}_k^i = \max_{q \neq t} \hat{y}_q^i, y_t^i = 1 \\ 0, & \text{otherwise} \end{cases} \quad (1.15)$$

is not constant at $\hat{y}_k^i < 1, y_k^i = 1$, implying that $Mshinge_3$ is not affine. Furthermore, its second-order partial derivative with respect to \hat{y}_k^i is not constant either at $\hat{y}_k^i < y_k^i, y_k^i = 1$ since it is expressed as,

$$\frac{\partial^2 \ell(y_k^i, \hat{y}_k^i)}{\partial \hat{y}_k^i{}^2} = \begin{cases} 6(1 + \max_{q \neq t} \hat{y}_q^i - \hat{y}_k^i), & \hat{y}_k^i < 1 + \max_{q \neq t} \hat{y}_q^i, y_k^i = 1 \\ -6(1 + \hat{y}_k^i - \hat{y}_t^i), & \hat{y}_t^i < 1 + \hat{y}_k^i, \hat{y}_k^i = \max_{q \neq t} \hat{y}_q^i, y_t^i = 1 \\ 0, & \text{otherwise} \end{cases}$$

For $\hat{y}_k^i < y_k^i, y_k^i = 1$, we obtain $1 + \max_{q \neq t} \hat{y}_q^i - \hat{y}_k^i > \max_{q \neq t} \hat{y}_q^i$. So, only when $\max_{q \neq t} \hat{y}_q^i \geq 0$, we obtain $\frac{\partial^2 \ell(y_k^i, \hat{y}_k^i)}{\partial \hat{y}_k^i{}^2} > 0$ and $\frac{\partial^3 \ell(y_k^i, \hat{y}_k^i)}{\partial \hat{y}_k^i{}^3} = -6 < 0$, which satisfies the condition $b=true$ of Algorithm 3. Hence, applying the cost-sensitive approach on $Mshinge_3$ loss function is efficient at classifying imbalanced data when $\max_{q \neq t} \hat{y}_q^i \geq 0$.

$\log \circ \sigma$ loss function. $\frac{\partial \ell(y_k^i, \hat{y}_k^i)}{\partial \hat{y}_k^i}$ of $\log \circ \sigma$ with respect to \hat{y}_k^i is given by,

$$\begin{aligned} \frac{\partial \ell(y_k^i, \hat{y}_k^i)}{\partial \hat{y}_k^i} &= \sum_{m=1}^M y_m^i \frac{\partial \log(p_m^i)}{\partial \hat{y}_k^i} \\ &= - \sum_{m=1}^M y_m^i \frac{\partial \log(p_m^i)}{\partial p_m^i} / (\partial p_m^i / \partial \hat{y}_k^i) \\ &= - \sum_{m=1}^M y_m^i \frac{1}{p_m^i} \frac{\partial p_m^i}{\partial \hat{y}_k^i} \end{aligned} \quad (1.16)$$

where $p_k^i = \sigma(\hat{y}_k^i) = e^{\hat{y}_k^i} / (\sum_{j=1}^M e^{\hat{y}_j^i})$, $\sigma(\cdot)$ being the softmax function. Since $(\partial p_m^i) / (\partial \hat{y}_k^i)$ can be expressed as,

$$\frac{\partial p_m^i}{\partial \hat{y}_k^i} = \begin{cases} p_m^i(1 - p_k^i) & \text{if } m = k \\ -p_m^i p_k^i & \text{if } m \neq k \end{cases} \quad (1.17)$$

, then equation 1.16 becomes,

$$\begin{aligned} \frac{\partial \ell(y_k^i, \hat{y}_k^i)}{\partial \hat{y}_k^i} &= -y_k^i(1 - p_k^i) - \sum_{m \neq k}^M y_m^i \frac{1}{p_m^i} (-p_m^i p_k^i) \\ &= p_k^i(y_k^i + \sum_{m \neq k}^M y_m^i) - y_k^i \end{aligned}$$

And, knowing that $\sum_{m=1}^M y_m^i = 1$, then,

$$\begin{aligned} \frac{\partial \ell(y_k^i, \hat{y}_k^i)}{\partial \hat{y}_k^i} &= p_k^i - y_k^i \\ &= \frac{e^{\hat{y}_k^i}}{\sum_{j=1}^M e^{\hat{y}_j^i}} - 1, y_k^i = 1 \end{aligned}$$

, suggesting that $\log \circ \sigma$ is not affine for $\hat{y}_k^i < y_k^i, y_k^i = 1$.

The second order partial derivative of $\log \circ \sigma$ with respect to \hat{y}_k^i can be expressed as,

$$\frac{\partial^2 \ell(y_k^i, \hat{y}_k^i)}{\partial \hat{y}_k^i{}^2} = \frac{\partial p_k^i}{\partial \hat{y}_k^i} = p_k^i(1 - p_k^i) \quad (21) \quad (1.18)$$

, meaning that $\frac{\partial^2 \ell(y_k^i, \hat{y}_k^i)}{\partial \hat{y}_k^i{}^2}$ is not constant and that $\frac{\partial \ell(y_k^i, \hat{y}_k^i)}{\partial \hat{y}_k^i}$ of $\log \circ \sigma$ is not affine for $\hat{y}_k^i < y_k^i, y_k^i = 1$, which indicates that $\log \circ \sigma$ belongs to Case 3.

The softmax function p_k^i has the property of outputting a probability distribution since it transforms \hat{y}_k^i into a real number in the range $]0, 1[$ such that $\sum_{m=1}^M p_m^i = 1$. So, given that $0 < p_k^i < 1$ and $0 < 1 - p_k^i < 1$, then $0 < \frac{\partial^2 \ell(y_k^i, \hat{y}_k^i)}{\partial \hat{y}_k^i{}^2} < 1$, which satisfies the expression $0 < \frac{\partial^2 \ell(y_k^i, \hat{y}_k^i)}{\partial \hat{y}_k^i{}^2}$.

Furthermore, the third-order partial derivative of $\log \circ \sigma$ with respect to \hat{y}_k^i is given by,

$$\begin{aligned} \frac{\partial^3 \ell(y_k^i, \hat{y}_k^i)}{\partial \hat{y}_k^i{}^3} &= p_k^i(1 - p_k^i) - 2p_k^i{}^2(1 - p_k^i) \\ &= p_k^i(1 - p_k^i)(1 - 2p_k^i) \end{aligned} \quad (1.19)$$

Having $0 < p_k^i < 1$, $0 < 1 - p_k^i < 1$, and $-1 < 1 - 2p_k^i < 1$, then $\frac{\partial^3 \ell(y_k^i, \hat{y}_k^i)}{\partial \hat{y}_k^i{}^3}$ is

not always positive. Thus, with $0 < \frac{\partial^2 \ell(y_k^i, \hat{y}_k^i)}{\partial \hat{y}_k^i{}^2}$ and $\frac{\partial^3 \ell(y_k^i, \hat{y}_k^i)}{\partial \hat{y}_k^i{}^3} < 0$, then the $\log \circ \sigma$ loss function satisfies $b = false$ of Algorithm 3, suggesting that the cost-sensitive approach applied on $\log \circ \sigma$ is not efficient at classifying imbalanced data.

1.3.2.2.3 (c) Graphical interpretation In order to visualize how the cost-sensitive approach impacts loss functions, we choose to plot in Fig.2.a and 2.b respectively the standard and cost-sensitive partial derivatives of loss functions (displayed in Fig. 1.1) with respect to to the network output \hat{y}_k^i . For the cost-sensitive version of loss functions, we pick an instance i whose class m^i is very rare such that $\phi^i = 0.9$, and we set $\tau = 1$ to obtain $\lambda^i = 1.9$. Fig.2.a and 2.b depict $\frac{\partial \ell(y_k^i, \hat{y}_k^i)}{\partial \hat{y}_k^i}$ and $\frac{\partial \chi(y_k^i, \hat{y}_k^i)}{\partial \hat{y}_k^i}$ at $y_k^i = 1$, respectively. By analyzing plots of Fig.2.a and 2.b, we observe that loss

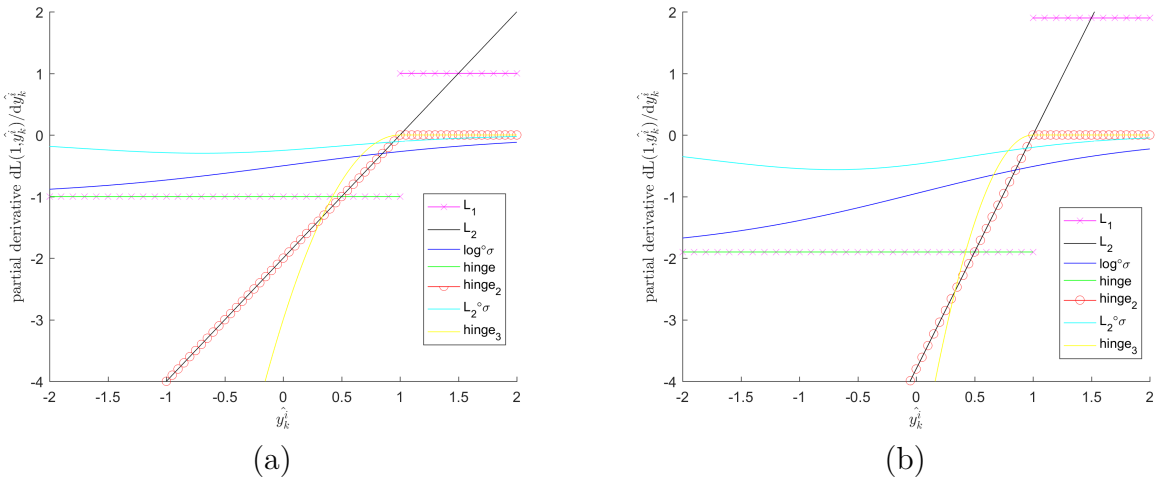


Figure 1.2: Plots in (a) and (b) represent respectively standard and cost-sensitive partial derivatives of different loss functions with respect to to \hat{y}_k^i at instance i for $y_k^i = 1$. Instance i is chosen to have a high relevance $\phi(m^i) = 0.9$, and τ is set to 1.

functions which satisfy $b=true$ of Algorithm 3 are the ones which have the highest absolute value of partial derivatives with respect to \hat{y}_k^i ($\left| \frac{\partial \ell(y_k^i, \hat{y}_k^i)}{\partial \hat{y}_k^i} \right|$) and consequently the highest absolute value of cost-sensitive partial derivatives with respect to \hat{y}_k^i ($\left| \frac{\partial \chi(y_k^i, \hat{y}_k^i)}{\partial \hat{y}_k^i} \right|$) for $\hat{y}_k^i < y_k^i$ at $y_k^i = 1$. These loss functions include L_2 as well as *hinge*, *hinge₂* and *hinge₃* which are special versions of *Mshinge*, *Mshinge₂* and *Mshinge₃* respectively where $\max_{q \neq t} \hat{y}_q^i = 0$. On the other hand, loss functions for which $b=false$ of Algorithm 3 holds have relatively low $\left| \frac{\partial \chi(y_k^i, \hat{y}_k^i)}{\partial \hat{y}_k^i} \right|$. $\log \circ \sigma$ and $L_2 \circ \sigma$ are examples of such loss functions. Indeed, as our prediction \hat{y}_k^i decreases and goes further from the true label $y_k^i = 1$, $\left| \frac{\partial \ell(y_k^i, \hat{y}_k^i)}{\partial \hat{y}_k^i} \right|$ of probability estimate loss functions ($\log \circ \sigma$ and $L_2 \circ \sigma$) increases very slowly (at a slow rate), as opposed to $\left| \frac{\partial \ell(y_k^i, \hat{y}_k^i)}{\partial \hat{y}_k^i} \right|$ of *Mshinge*, *Mshinge₂*, *Mshinge₃*, and L_2 loss functions. As a result, multiplying partial derivatives by $\lambda(m^i)$ still produces a small $\left| \frac{\partial \chi(y_k^i, \hat{y}_k^i)}{\partial \hat{y}_k^i} \right|$ for probability estimate

loss functions (i.e., $\left| \frac{\partial \chi(y_k^i, \hat{y}_k^i)}{\partial \hat{y}_k^i} \right| < 1$ for $\hat{y}_k^i \leq 0$) and a relatively high $\left| \frac{\partial \chi(y_k^i, \hat{y}_k^i)}{\partial \hat{y}_k^i} \right|$ for other loss functions (i.e., $\left| \frac{\partial \chi(y_k^i, \hat{y}_k^i)}{\partial \hat{y}_k^i} \right| \geq 1$ for $\hat{y}_k^i \leq 0$), as shown in Fig.2.b.

For instance, for a network output $\hat{y}_k^i = 0$ far from the true label $y_k^i = 1$, $\frac{\partial \ell(y_k^i, \hat{y}_k^i)}{\partial \hat{y}_k^i}$ is equal to -1, -2, -12 and -2 for *hinge*, *hinge₂*, *hinge₃*, and L_2 respectively, compared to -1/2 and -1/4 for $\log \circ \sigma$ and $L_2 \circ \sigma$ respectively; and $\frac{\partial \chi(y_k^i, \hat{y}_k^i)}{\partial \hat{y}_k^i}$ is equal to -1.9, -3.8, -22.8 and -3.8 for *hinge*, *hinge₂*, *hinge₃*, and L_2 respectively, compared to -0.95 and -0.475 for $\log \circ \sigma$ and $L_2 \circ \sigma$ respectively. Hence, as the network output gets further from the true label, the impact of the cost-sensitive approach is very sparse for the probability estimate loss functions and relatively high for *Mshinge*, *Mshinge₂*, *Mshinge₃*, and L_2 .

1.3.3 Faster convergence with cost-sensitive learning approach

One of the factors which contribute to the fast NN convergence is the use of an adaptable learning rate α which is not too large for the network to diverge and not too small for it to converge too slowly. Below we show that our cost-sensitive approach is able to improve the network's convergence via a dynamic learning rate per instance α_i which depends on the rarity or relevance of that instance.

Using the cost-sensitive gradient $\nabla \chi(\boldsymbol{\theta}_k)$ (eq.1.5), the stochastic gradient descent learning rule is obtained as,

$$\begin{aligned} \boldsymbol{\theta}_k &:= \boldsymbol{\theta}_k - \alpha \frac{1}{n} \sum_{i=1}^N \lambda^i \frac{\partial \ell(y_k^i, \hat{y}_k^i)}{\partial \hat{y}_k^i} \mathbf{a}^i \\ &:= \boldsymbol{\theta}_k - \frac{1}{n} \sum_{i=1}^N \alpha^i \frac{\partial \ell(y_k^i, \hat{y}_k^i)}{\partial \hat{y}_k^i} \mathbf{a}^i \end{aligned} \tag{1.20}$$

where $\alpha^i = \alpha \lambda^i$ and $\alpha \leq \alpha^i \leq (1 + \tau)\alpha$. So, we can consider α^i as a dynamic learning rate which changes with respect to the rarity or relevance of the instance, by increasing up to $(1 + \tau)\alpha$ for the rarest instances and decreasing up to the original learning rate α for the most frequent instances. As such, our cost-sensitive gradient $\nabla \chi(\boldsymbol{\theta}_k)$ has the property of boosting learning as it helps the network converge faster. Indeed, our gradient has the property of increasing small weight updates, which is one of the two objectives of the Adam optimizer Kingma and Ba (2015) (whole goal is to make relatively small weight updates bigger and relatively high weights updates smaller).

In order to show this property, let's consider training a two-class neuron network (NN) under imbalanced domains using stochastic gradient descent with a batch of size 10. By applying the standard cost function (eq. 2.3), this NN tends to classify frequent instances as negatives and rare instances as positives. For example, given

a run composed of 8 negative instances and 2 positive ones, the NN is more likely to correctly classify the 8 negatives as negatives and to wrongly classify the 2 positives as negatives, resulting in a relatively small average gradient $\nabla\chi(\boldsymbol{\theta})$ and thus a small learning. So, more runs (e.g., backpropagations) are needed in order to learn from positive instances. However, employing the cost-sensitive learning approach produces a higher average error (since the loss of every positive instance is increased), and a higher $\nabla\chi(\boldsymbol{\theta})$, which makes learning from positives greater and faster. Hence, the cost-sensitive learning algorithm boosts learning by making small gradients bigger.

1.4 Experimental study

1.4.1 Datasets

1.4.1.1 1D datasets

The experiments were carried out on eight real one-dimensional datasets (i.e., whose input data is a vector) extracted from the UCI Database Repository <http://www.ics.uci.edu/~mllearn>. The datasets and some of their characteristics are summarized in Table 1.1. All these datasets have passed through the following preprocessing steps: categorical attributes were expanded into the corresponding binary vectors, and then each attribute (metric or binary) was normalized to the interval $[0, 1]$. Furthermore, the “Yeast_8l” dataset is simply the “yeast” dataset with class 9 and 10 removed (since these latter contain very few instances, which makes the NN hard to train).

1.4.1.2 2D dataset MNIST

MNIST is considered a simple and solved problem that involves digits’ images classification. The dataset consists of grayscale images of size 28×28 . There are ten classes corresponding to digits from 0 to 9. The number of examples per class in the original training dataset ranges from 5421 in class 5 to 6742 in class 1. In our study, we sub-sample uniformly at random each class to obtain no more than 600 examples par class. This dataset will be referred to as “Mnist”. Afterward, in order to show the performance of our cost-sensitive algorithm, the “Mnist” dataset needs to be imbalanced. To do so, we first set the classes 1 and 3 (e.g., number “1” and “3”) to be our minority classes. Next, we define the ratio r between the number of examples in majority classes and the number of examples in minority classes as follows, $r = \frac{\max_k n_k}{\min_k n_k}$. The ratio is set to one of the values $\{10, 30, 40, 50\}$, meaning that the number of instances within class 1 and 3 will be either 60, 30, 15 or 12.

These imbalanced datasets will be denoted as “Mnist10”, “Mnist30”, “Mnist40” and “Mnist50” respectively.

Table 1.1: Characteristics of the 1D datasets.

Dataset	No. of Attributes	No. of classes	No. of instances	Class distribution
Ionosphere	34	2	181	126/55
Pima Indians Diabetes	8	2	768	268/500
WP Breast Cancer	30	2	198	47/151
SPECTF Heart	43	2	267	55/212
Yeast_8l	8	8	1645	464/430/424/163/51/47/35/31
Car	6	4	1728	1210/384/69/65
Satimage	36	6	6435	1533/703/1358/626/707/ 1508
Thyroid	21	3	7200	166/368/6666
Mnist	28×28	10	6000	600/600/600/600/600/600/600/600/600/600
Mnist10	28×28	10	4920	60/600/60/600/600/600/600/600/600/600
Mnist30	28×28	10	4860	20/600/20/600/600/600/600/600/600/600
Mnist40	28×28	10	4850	15/600/15/600/600/600/600/600/600/600

1.4.2 Training and Experimental setup

In our study, a Multi-Layer Perceptron is used to train the 1D datasets, whereas the 2D datasets are trained using a Convolutional Neural Network (ConvNet). The optimization algorithm used for experiments within section 5.1 and 5.2 is stochastic gradient descent (SGD) with a momentum of 0.9 and a weight decay of 0.0005. As for experiments of section 5.3, SGD with the same momentum and weight decay is used for training ConvNets, whereas the Adam optimizer (Kingma and Ba, 2015) is used for training MLPs with the exponential decay rate for 1st and 2nd moment estimates β_1 and β_2 set to 0.9 and 0.999 respectively, and the offset ϵ set to 10^{-8} . Adam is used for training MLPs only because, as stated in (Proefschrift, 2017), Adam shows large performance improvement over SGD with momentum for MLPs versus marginal improvement for ConvNets. Hyper-parameters such as the learning rate, batch size and network architecture vary from one dataset to another and are set according to Table 1.2. Note that learning rates displayed in Table 1.2.a are used when the optimizer is SGD with momentum. On the other hand, when dealing with the Adam optimizer, the learning rate is set to 0.001 for all datasets within Table 1.2.a. The dropout rate for ConvNet is set to 0.5. Training is performed for 15 to 100 epochs, depending on the dataset used and the experimental method employed. We report results with 3-fold cross validation.

As for parameters of the cost-sensitive approach, τ is set to 2 and 50 for the MLP and ConvNet models respectively. Indeed, as the proposed ConvNet architecture

has more hidden layers than the proposed MLP architectures, the magnitude of the gradients with each subsequent layer of the ConvNet gets exponentially smaller in the backpropagation process, which results in very slow learning of weights in the ConvNet lower layers. So, in order for ConvNet lower layers' weights to be affected by the cost-sensitive technique, the weighting parameter λ needs to be relatively high compared to λ of a MLP. That's why the parameter τ used for ConvNet training is chosen to be higher than the one used for MLP training.

Table 1.2: Training hyper-parameters for the 1D datasets in (a) and the 2D datasets in (b). $[n1, n2, n3]$ defines the architecture of the MLP for the 1D datasets, with $n1$, $n2$ and $n3$ denoting the number of neurons in the first, second and third layer respectively. In (b), the architecture of the ConvNet is defined by the layer type, its kernel size, stride, and depth.

(a) MLP training hyper-parameters

Dataset	Learning rate	Batch size	$[n1, n2, n3]$
Ionosphere	0.010	5	500, 50, 2
Pima Indians Diabetes	0.010	5	200, 20, 2
WP Breast Cancer	0.005	5	500, 50, 2
SPECTF Heart	0.010	5	650, 65, 2
Yeast_8l	0.001	10	200, 100, 8
Car	0.010	10	150, 75, 4
Satimage	0.010	50	600, 100, 6
Thyroid	0.010	50	350, 70, 3

(b) ConvNet training hyper-parameters

Dataset	Learning rate	Batch size	Architecture			
			Layer	Depth	Kernel size	Stride
Mnist10/	0.0001	30	Convolution	20	5x5	1
Mnist30/			ReLU	20	-	-
Mnist40/			Max-pooling	20	2x2	2
Mnist50			Convolution	50	5x5	1
			ReLU	50	-	-
			Max-pooling	50	2x2	2
			Fully connected	500	4x4	1
			ReLU	500	-	-
			Dropout	500	-	-
	Fully connected	10	1x1	1		

As for the performance metric, the most widely used one for evaluating performance in the context of multi-class classification within neural networks (MLPs or ConvNets) is overall accuracy which is the proportion of correctly classified test examples. However, this metric has some significant and long acknowledged limitations, particularly in the context of imbalanced datasets. Specifically, when the test

set is imbalanced, accuracy favors classes that are overrepresented in some cases leading to highly misleading assessment. In order to make the classification performance of each class equally represented in the evaluation measure, (Kubat et al., 1998) suggested the G-mean as the geometric means of recall values for the bi-class scenario. Expanding this measure to the multiple class scenario was introduced by (Sun, Yanmin and Kamel, Mohamed S and Wang, 2006) whereby the G-mean is the geometric means of recall values of every classes as follows,

$$G - mean = \prod_{i=1}^m TP(i)^{1/m} \quad (1.21)$$

where m is the number of classes, and TP is the recall or true positive rate. As each recall value representing the classification performance of a specific class is equally accounted, G-mean is chosen to be the metric for our study.

1.5 Experiments and results

In this section, we first want to show the effect that each loss function (mentioned in Table 2.1) has on our cost-sensitive approach by training shallow and deep neural networks (MLPs and ConvNets) under imbalanced datasets using the standard version of that loss function and comparing them with networks trained on the cost-sensitive version. Next, convergence speeds of MLPs using the standard and cost-sensitive versions of each of the loss functions are visualized by displaying, in Figure 3, learning curves relative to one of the 1D datasets. Finally, in order to evaluate the effectiveness of the cost-sensitive algorithm in improving the performance of both shallow and deep neural networks under imbalanced domains, an empirical study is conducted by comparing classification rates of the cost-sensitive approach to classification rates of well-known methods used for addressing class imbalance (Table 1.4).

1.5.1 Effect of our cost-sensitive approach on classification performance

Training MLPs on the 1D datasets and ConvNets on the 2D datasets is performed using the standard version of loss functions of Table 2.1 as well as the cost-sensitive version of the following loss functions: L_2 , $Mshinge$, $Mshinge_2$, $Mshinge_3$, $\log \circ \sigma$ ($\sigma(\cdot)$ being the softmax function), and $L_2 \circ \sigma$ ($\sigma(\cdot)$ being the sigmoid function). Each experiment is repeated three times and its mean performance across all three runs is depicted in Table 1.3. From results in Table 1.3, we observe that applying the cost-sensitive approach on L_2 , $Mshinge$, $Mshinge_2$, $Mshinge_3$ loss functions

improves the G-mean performance in overall. Indeed G-mean results for the cost-sensitive version of these loss functions are higher than results for the standard version for all 1D and 2D datasets (except “WB breast cancer” dataset for $Mshinge_3$ loss function). For instance, the cost-sensitive approach boosts performance from 0% to 60.69% for the “thyroid” dataset when applied on $Mshinge$, from 0% to 60.66% for the “Yeast_8l” dataset when applied on $Mshinge_2$, and from 0% to 98.41% for the “Mnist50” dataset when applied on L_2 . However, we can see that applying the same cost-sensitive approach on $\log \circ \sigma$ loss function seems to decrease performance for several datasets (such as “Ionosphere”, “Yeast_8l”, “Mnist30”, “Mnist40” and “Mnist50”) and to increase performance for the rest of the datasets. The same behavior is observed for $L_2 \circ \sigma$ with a decrease in performance for datasets “Ionosphere”, “Pid”, “WB breast cancer”, and “Satimage”, versus an increase in performance for the other datasets.

Table 1.3: Mean classification results of neural networks over 3 runs using the standard version (denoted as “Std.”) and the cost-sensitive version of different loss functions in terms of average values of g-mean (in %). Best rates per loss function and per dataset are in bold.

	$\log \circ \sigma$		$Mshinge$		$Mshinge_2$		$Mshinge_3$		$L_2 \circ \sigma$		L_2	
	Std.	Ours	Std.	Ours	Std.	Ours	Std.	Ours	Std.	Ours	Std.	Ours
Ionosphere	90.60	88.29	86.25	89.17	86.25	87.72	89.17	90.93	85.29	82.02	70.26	81.58
Pid	75.52	76.38	75.22	75.27	75.50	76.54	74.65	75.77	76.08	32.12	75.77	75.97
WB breast cancer	92.83	93.36	92.40	93.19	92.17	92.62	94.13	93.98	93.27	89.38	82.30	89.88
SPECTF_Heart	79.66	81.18	83.68	83.68	80.52	82.12	81.32	83.68	73.49	82.02	80.05	82.90
Yeast_8l	58.72	58.02	56.36	59.84	0.00	60.66	53.11	59.43	0.00	0.00	57.32	63.15
Car	98.01	100.00	98.75	99.63	98.50	99.94	98.67	99.82	87.57	96.52	96.56	99.57
Satimage	87.83	87.94	86.86	88.73	87.60	88.94	87.58	88.58	88.34	82.74	85.55	88.12
Thyroid	51.03	54.93	0.00	60.69	57.97	72.26	50.51	72.65	0.00	41.08	46.75	73.14
Mnist10	94.88	95.15	94.78	96.55	94.96	95.18	95.18	95.92	90.74	96.21	91.76	96.27
Mnist30	92.00	90.94	91.95	93.45	91.89	92.50	90.31	90.75	90.43	92.36	90.43	92.64
Mnist40	97.77	98.50	95.81	98.41	87.81	97.13	96.80	98.52	0.00	97.81	0.00	98.41
Mnist50	98.70	98.11	97.89	98.70	98.66	98.29	98.61	98.67	0.00	97.05	0.00	98.43

Given these observations, the following reflections can be made:

- i When dealing with L_2 , $Mshinge$ and $Mshinge_2$ and $Mshinge_3$ loss functions, our cost-sensitive approach is able to capture more relevant features i.e., features that are shared between positive and negative instances. Indeed, at every instance i , multiplying such loss functions by λ^i contributes in balancing weights θ between positives and negatives, thus generating better weights. Furthermore, the positive impact of this approach with such loss functions is drawn for both the shallow NN models (MLPs) and deep learning ones (ConvNets).
- ii The cost-sensitive approach can be regarded as a reliable technique when applied on loss functions which satisfy the property $b = true$ of Algorithm

3 (namely, L_2 , $Mshinge$ and $Mshinge_2$ and $Mshinge_3$), which verifies that Algorithm 3 is correct and can be further used to test whether a loss function is suitable for the cost-sensitive approach to improve classification performance within imbalanced data.

- iii We believe that our cost-sensitive strategy does not generalize to probability estimate loss functions but rather to loss functions with no probability estimate (or activation function) $\sigma(\cdot)$ applied at the last layer. Indeed, as $\sigma(\cdot)$ turns predicted outputs into probabilities ranging from 0 to 1, partial derivatives of the loss function with respect to this output tend to be small within the range $[0, 1]$. As a result, applying the cost-sensitive technique on such small partial derivatives has a little impact on NN learning.
- iv The performance decrease of NNs with probability estimate loss functions when applying the cost-sensitive approach instead of the standard one can be explained by the fact that $\partial\ell(y_k^i, \hat{y}_k^i)/(\partial\hat{y}_k^i) \neq 0$ for a correct classification $\hat{y}_k^i = y_k^i, y_k^i = 1$ for these loss functions (knowing that, ideally, $\partial\ell(y_k^i, \hat{y}_k^i)/(\partial\hat{y}_k^i) = 0$ for a correct classification). So, as the cost-sensitive weighting parameter λ^i is applied on $\frac{\partial\ell(y_k^i, \hat{y}_k^i)}{\partial\hat{y}_k^i}$, the resultant $\partial\chi(y_k^i, \hat{y}_k^i)/(\partial\hat{y}_k^i)$ gets further from 0 which leads to an even larger gradient update even for a correct classification, resulting in a wrong NN learning.
- v Also, let's note that the non-linearity present within partial derivatives of loss functions $\log \circ \sigma$ and $L_2 \circ \sigma$ with respect to \hat{y}_k^i is not the property responsible for the inefficiency of the cost-sensitive approach. Indeed, even with the non-linearity of $Mshinge_3$ partial derivative, $Mshinge_3$ along with the cost-sensitive method improves classification performance.

1.5.2 Effect of our cost-sensitive approach on convergence speed

Plots in Fig.3.a, 3.b, 3.c, 3.d, 3.e and 3.f illustrate learning curves resulting from training NNs using $Mshinge$, $Mshinge_2$, $Mshinge_3$, L_2 , $L_2 \circ \sigma$ and $\log \circ \sigma$ loss functions respectively in the standard and cost-sensitive form (based on the ‘‘Ionosphere’’ dataset for $Mshinge$, $Mshinge_2$, $Mshinge_3$, the ‘‘Pima Indians Diabetes’’ dataset for L_2 , the ‘‘WP Breast Cancer’’ dataset for L_2 , and the ‘‘Satimage’’ dataset for $L_2 \circ \sigma$). From these plots, we observe that the cost-sensitive strategy increases the convergence speed for $Mshinge$, $Mshinge_2$, $Mshinge_3$ and L_2 loss functions. As explained in the methodology (section 3.3), this improvement over the NN convergence speed is due to the coefficient λ^i within the gradient which acts like a learning rate magnifier for positive instances i and which boosts the learning process for those instances. Nonetheless, we notice a different behavior for both $\log \circ \sigma$

and $L_2 \circ \sigma$ where the cost-sensitive approach either does not affect convergence speed as it is the case for $\log \circ \sigma$ (Fig.3.f) or reduces convergence speed as observed for $L_2 \circ \sigma$ (Fig.3.e). These observations confirm that the cost-sensitive technique applied on loss functions with probability estimates such as $\log \circ \sigma$ and $L_2 \circ \sigma$ has a sparse impact on the network learning speed compared to the large impact of that technique when applied on loss functions whose partial derivative with respect to network output is relatively large. Thus, proving the non-efficiency of the cost-sensitive approach on improving classification with data imbalance when either $\log \circ \sigma$ or $L_2 \circ \sigma$ is applied verifies Algorithm 3.

1.5.3 Comparison with other techniques

Our cost-sensitive approach is compared with known methods in the literature for imbalanced learning such as under-sampling, over-sampling, and the cost-sensitive method ST1 (Alejo et al., 2007). A pure neural network, i.e., without any strategy to deal with imbalanced data, was also tested within exactly the same conditions of the other algorithms. In these experiments, the chosen loss function to conduct all these methods is L_2 . Moreover, each experiment is performed three times and its mean G-mean classification result is reported in Table 1.4.

Table 1.4: Comparative results between different methods in terms of the G-mean performance metric (in %).

	L_2	ST1	Ours	Over-sampling	Under-sampling
Ionosphere	87.72	87.72	88.81	84.19	88.29
Pid	75.81	77.36	77.05	78.21	76.79
WB breast cancer	92.83	93.06	94.27	92.62	92.62
SPECTF_Heart	80.43	83.68	84.50	81.18	82.86
yeast_8l	59.01	62.46	62.99	62.04	55.87
Car	99.40	99.26	99.56	99.56	93.99
Satimage	88.01	89.04	88.86	88.38	88.48
Thyroid	78.06	93.09	86.35	96.69	77.64
mnist10	91.76	95.15	96.27	94.37	90.18
mnist30	90.43	92.34	92.64	91.08	86.53
mnist40	0.00	97.68	98.41	97.85	82.44
mnist50	0.00	97.29	98.43	93.56	80.47

From these results, we deduce that, in general, our method surpasses other methods. Furthermore, several observations can be drawn:

- Our cost-sensitive method surpasses by far the under-sampling technique. Indeed, as we under-sample, we are able to balance the data for proper neural network training, but we remove training instances which could hold valuable characteristics, thus losing relevant information.

- For almost all datasets (except “Pid” and “Thyroid”), the performance of our approach is higher than the over-sampling technique. This is because over-sampling generates redundant instances which might cause overfitting of the NN especially if dealing with a complex NN such as ConvNet, thus preventing this latter from correctly classifying new unseen instances (e.g. test instances).
- Our approach performs slightly better than the ST1 technique. As opposed to ST1 which up-weights positive instances with an unbounded weight that could get very high as $n_m \ll \max(n_k)$ (n_m being the number of instances of a minority class and $\max(n_k)$ being the number of instances of the most frequent class), our method up-weights these instances with a bounded weight that varies from 0 to $\tau + 1$, where τ is chosen to be small or large depending on whether we are training MLPs or ConvNets.

1.6 Conclusion

Our cost-sensitive learning algorithm addressed the class imbalance problem which is commonly encountered when dealing with real-world datasets, by introducing a cost-sensitive strategy applied on neural networks at the training phase. Based on a cost-sensitive error function, its objective was to correctly classify the minority classes and favor them as much as the frequent ones by assigning a weighted misclassification cost based on the distribution of classes. By properly weighting the loss function weight-updating is intensified for the minority class based on the probability of their occurrence. Throughout this chapter, results on several popular datasets showed that:

- i. our approach has a better convergence than the baseline algorithm and existent approaches,
- ii. it also offers a faster convergence by boosting the optimizer when positive instances are present,
- iii. it can handle classification of imbalanced datasets whose inputs are one-, two-, or three-dimensional since it can be applied on shallow and deep neural networks (MLPs and ConvNets).

We also showed that the cost-sensitive approach is efficient only when applied on loss functions which ensure, via certain conditions, that the gradient of the loss is relatively large when misclassification occurs i.e., for $\hat{y}_k^i < y_k^i$ at $y_k^i = 1$. Finally, we conclude that adding a probability estimate activation function at the last NN

layer (which is done in all classification research papers) in general and using the cross entropy loss (which is the most commonly used loss function in the majority of papers) in particular is not a good choice as it comes to classifying imbalanced datasets with a cost-sensitive learning approach based on a weighted cost such as ours.

Bibliography

R Alejo, V Garcia, J.M. Sotoca, R.A. Mollineda, and J.S. Sanchez. Improving the performance of the RBF neural networks trained with imbalanced samples. *Computational and Ambient Intelligence*, 4507:162–169, 2007. ISSN 03029743. doi: 10.1007/978-3-540-73007-1.

Gustavo E. A. P. A. Batista, Ronaldo C. Prati, and Maria Carolina Monard. A study of the behavior of several methods for balancing machine learning training data. *ACM SIGKDD Explorations Newsletter*, 6(1):20, 2004. ISSN 19310145. doi: 10.1145/1007730.1007735.

Cristiano L Castro and Antonio P Braga. Novel cost-sensitive approach to improve the multilayer perceptron performance on imbalanced data. *IEEE Transactions on Neural Networks and Learning Systems*, 24(6):888–899, 2013. ISSN 2162237X. doi: 10.1109/TNNLS.2013.2246188.

Cristiano Leite Castro and Antonio de Padua Braga. ARTIFICIAL NEURAL NETWORKS LEARNING IN ROC SPACE. In *IJCCI*, pages 484–489, 2009. ISBN 9789896740146.

Ming Gao, Xia Hong, and Chris J. Harris. Construction of neurofuzzy models for imbalanced data classification. *IEEE Transactions on Fuzzy Systems*, 22(6):1472–1488, 2014. ISSN 10636706. doi: 10.1109/TFUZZ.2013.2296091.

R. Harliman and K. Uchida. Data- and algorithm-hybrid approach for imbalanced data problems in deep neural network. *International Journal of Machine Learning and Computing*, 8(3):208–213, 2018. ISSN 20103700. doi: 10.18178/ijmlc.2018.8.3.689.

Chen Huang, Yining Li, Chen Change Loy, and Xiaoou Tang. Learning Deep Representation for Imbalanced Classification. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5375–5384, 2016. ISBN 978-1-4673-8851-1. doi: 10.1109/CVPR.2016.580.

- Piyasak Jeatrakul, Kok Wai Wong, and Chun Che Fung. Classification of imbalanced data by combining the complementary neural network and SMOTE algorithm. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 6444 LNCS, pages 152–159, 2010. ISBN 3642175333. doi: 10.1007/978-3-642-17534-3_19.
- S H Khan, M Hayat, M Bennamoun, F Sohel, R Togneri, and C V Mar. Cost-Sensitive Learning of Deep Feature Representations from Imbalanced Data. *IEEE Transactions on Neural Networks and Learning Systems*, 29(8):3573–3587, 2018.
- Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. In *Proceedings of the 3rd International Conference on Learning Representations (ICLR)*, dec 2015. ISBN 9781450300728. doi: 10.1063/1.4902458.
- Miroslav Kubat, Robert C. Holte, and Stan Matwin. Machine Learning for the Detection of Oil Spills in Satellite Radar Images. *Machine Learning*, 30:195–215, 1998. ISSN 08856125. doi: 10.1023/A:1007452223027.
- Kuan Li, Xiangfei Kong, Zhi Lu, Liu Wenyin, and Jianping Yin. Boosting weighted ELM for imbalanced learning. *Neurocomputing*, 128:15–21, 2014. ISSN 09252312. doi: 10.1016/j.neucom.2013.05.051.
- Sang Hoon Oh. Error back-propagation algorithm for classification of imbalanced data. *Neurocomputing*, 74(6):1058–1061, 2011. ISSN 09252312. doi: 10.1016/j.neucom.2010.11.024.
- Samira Pouyanfar, Yudong Tao, Anup Mohan, Haiman Tian, Ahmed S Kaseb, Kent Gauen, Ryan Dailey, Sarah Aghajanzadeh, Yung Hsiang Lu, Shu Ching Chen, and Mei Ling Shyu. Dynamic Sampling in Convolutional Neural Networks for Imbalanced Data Classification. In *Proceedings - IEEE 1st Conference on Multimedia Information Processing and Retrieval, MIPR 2018*, number June, pages 112–117, 2018. ISBN 9781538618578. doi: 10.1109/MIPR.2018.00027.
- Academisch Proefschrift. Variational inference & deep learning: A new synthesis. PhD thesis, 2017.
- Enislay Ramentol, Yailé Caballero, Rafael Bello, and Francisco Herrera. SMOTE-RSB *: a hybrid preprocessing approach based on oversampling and undersampling for high imbalanced data-sets using SMOTE and rough sets theory. *Knowledge and Information Systems*, 33(2):245–265, nov 2012. ISSN 0219-1377. doi: 10.1007/s10115-011-0465-6.

David E. Rumelhart, James L. McClelland, Chisato Asanuma, Francis H. C. Crick, Jeffrey L. Elman, Geoffrey E. Hinton, Michael I. Jordan, Alan H. Kawamoto, Paul W. Munro, Donald A. Norman, Daniel E. Rabin, Terrence J. Sejnowski, Paul Smolensky, Gregory O. Stone, Ronald J. Williams, and David Zipser. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*. In *Computational Models of Cognition and Perception*, pages 318–362. 1986. ISBN 0262181207. doi: 10.1017/CBO9781107415324.004.

Lamyaa Sadouk, Taoufiq Gadi, El Hassan Essoufi, and Abdelhak Bassir. A cost-sensitive learning approach applied on shallow and deep neural networks for classification of imbalanced data. Unpublished paper submitted to *Computational Intelligence and Neuroscience*, 2020.

Sangyoon Oh, Min Su Lee, Byoung-Tak Zhang, Sangyoon Oh, Min Su Lee, and Byoung Tak Zhang. Ensemble learning with active example selection for imbalanced biomedical data classification. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 8(2):316–325, 2011. ISSN 15455963. doi: 10.1109/TCBB.2010.96.

Yang Sun, Yanmin and Kamel, Mohamed S and Wang. Boosting for Learning Multiple Classes with Imbalanced Class Distribution. In *Proceedings of the Sixth International Conference on Data Mining*, pages 592—602, 2006. ISBN 0769527019.

Benjamin X. Wang and Nathalie Japkowicz. Boosting support vector machines for imbalanced data sets. *Knowledge and Information Systems*, 25(1):1–20, 2010. ISSN 02191377. doi: 10.1007/s10115-009-0198-y.

Shoujin Wang, Wei Liu, Jia Wu, Longbing Cao, Qinxue Meng, and Paul J Kennedy. Training Deep Neural Networks on Imbalanced Data Sets. In *Neural Networks (IJCNN), 2016 International Joint Conference on*, pages 4368–4374, 2016. ISBN 9781509006205.

Songqing Yue. Imbalanced Malware Images Classification: a CNN based Approach. arXiv preprint arXiv:1708.08042, 2017.

Chapter 2

Techniques for Handling Regression under Imbalanced Domains: algorithm and evaluation techniques

2.1 Introduction

As discussed in the previous chapter, data imbalance is caused by imbalanced data distributions where some events (e.g., cases) are abundant (denoted as negative, majority or frequent) while others only have limited representations (denoted as positive, minority, extreme or rare events), which leads to unexpected mistakes and even serious consequences in data analysis especially in classification and regression tasks. Indeed the skewed distribution of instances forces predictive models to be biased to labels of frequent events and thus to ignore or not learn properly from rare ones.

One example of regression tasks under imbalanced domains involves forecasting rare extreme returns in financial markets. Unfortunately, in such domains, cases that are more important to the user are the rare events which are very few on the available training set, which gives rise to the following challenges: (i) we need means for making a learning regression algorithm focus on these rare cases, and (ii) we need a special purpose evaluation approach that is biased towards the performance of the models on these rare cases while taking into consideration frequent cases. To this end, this chapter, which is further discussed in (Sadouk et al., 2021), attempts to address these two questions through the proposal of:

- i a cost-sensitive learning approach with neural networks for tackling the issue of rare/extreme values prediction,
- ii new evaluation strategies for regression models that handle imbalanced datasets, including scalar and graphical-based measures.

The rest of the chapter is outlined as follows. Section 2 reviews related works on techniques and evaluation approaches for handling regression tasks under imbalanced domains. Section 3 describes our cost-sensitive approach as well as our evaluation measures designed to tackle regression with imbalanced data distribution. Section 4 develops the experimental setup and explores fundamental characteristics of datasets employed. Section 5 highlights experiments and results, and then compares them to existing methods. Finally, Section 6 concludes our work.

2.2 Related work

2.2.1 Regression Strategies for Handling Imbalanced Domains

Few strategies have been developed to address this problem in regression setting and are divided into two categories: data-level and algorithm-level methods.

2.2.1.1 Data-level methods.

As mentioned in the previous chapter, the most common data-level methods are re-sampling techniques namely, under-sampling whose goal is to discard the intrinsic samples in the frequent events, over-sampling which duplicates minority events or create/synthesize new ones, and hybrid method which combines over-sampling with under-sampling. So far, in regression, under-sampling was performed by Torgo et al. (Torgo et al., 2015) as a strategy for addressing the imbalance problem. This method uses a relevance function and a user defined threshold to determine the common and uninteresting values which need to be under-sampled. Works (Torgo et al., 2015; Branco et al., 2017) proposed a hybrid method combining under-sampling and over-sampling, where over-sampling in (Torgo et al., 2015) consists of generating new synthetic data through the SMOTER algorithm whereas over-sampling in (Branco et al., 2017) incorporates two over-sampling strategies: SMOTER and introduction of Gaussian Noise. .

2.2.1.2 Algorithm-level methods.

Generally, we can distinguish between two algorithm-level types of methods. The first one involves special-purpose learning methods which comprise solutions that change the existing algorithms to be able to learn from imbalanced data. The work of Ribeiro (Torgo and Ribeiro, 2007; Ribeiro, 2011) is the only work that makes use of this type of method via a utility-based regression rule ensemble system designed for obtaining models biased according to a specific utility function which is based on the assumption that the user assigns a non-uniform relevance to the values of

the target variable domain. The system main goal is to first generate different regression trees which are then converted to rule ensembles, and then select the best rules to include in the final ensemble. The utility function is used as criterion at several stages of the algorithm. The second type of algorithm-level methods deals with cost-sensitive post-processing approaches do not modify the algorithm/model itself but rather imposes an expensive cost when an error happens. For example, a model assigns larger cost to rare observations compared to frequent ones thus emphasizing the learning from rare observations. In regression, introducing costs at a post-processing level still remains an under-explored issue with few limited solutions. Recent studies on cost sensitive learning for imbalance regression distribution has been proposed (Bansal et al., 2008; Zhao et al., 2011). In the area of finance, several works have been made to take into account differentiated prediction costs through different asymmetric loss functions such as quad-quad (Zellner, 1986), lin-lin (Christoffersen and Diebold, 1997) and others (Cain and Janssen, 1995; Christoffersen and Diebold, 1996; Granger, 1999; Crone et al., 2005; Lee, 2008). However, these solutions have the same disadvantage of being capable of only distinguishing between over and under-predictions. Thus, they remain unsuitable for tackling the problem of imbalanced domains with a user preference bias towards some specific ranges of values. Another alternative is the concept of a reframing framework by J. Hernandez-Orallo (Hernandez-Orallo, 2012; Hernández-Orallo, 2014) which was not developed specifically for imbalanced domains but still works for these latter by adjusting the predictions of a previously built model to different deployment contexts. The notion of reframing was established as the process of applying a previously built model to a new operating context by the proper transformation of inputs, outputs and patterns. This framework changes the obtained predictions by adapting them to a different distribution.

Although efficient, some of these data- and algorithm-level methods present some limitations. Indeed, data-level approaches require mapping the given data distribution into an optimal new distribution which is not an easy task, and alter the data distribution such that over-sampling aggravates the burden of computation and cause overfitting while under-sampling loses some useful information (Khan et al., 2018). As to special-purpose learning methods, they require a deep knowledge of the learning algorithms implementation and restrict the user to utilize learning algorithms that have been modified to be able to optimize his goals. However, post-processing approaches keep the original dataset and the standard learning algorithm unchanged, and only manipulate predictions and learning of the given model. Moreover, it is a straight forward learning mechanism with no user intervention required before or at learning time (no hard threshold being imposed) and its obtained model can be further applied to different deployment scenarios. To

this matter, our study focuses on such prediction post-processing approaches and introduces a novel cost-sensitive error function for neural networks.

2.2.2 Evaluation approaches for imbalanced regression domains

Very few efforts have been made regarding evaluation approaches for regression tasks in imbalanced domains (Branco et al., 2016). Performance measures can be either scalar or graphical-based. Graphical-based metrics. Following the efforts made within classification, some attempts were made to adapt the existing notion of ROC curves to regression tasks. One of these attempts is the ROC space for regression (RROC space) (Hernández-Orallo, 2013) which is motivated by the asymmetric loss often present on regression applications where both over-estimations and under-estimations entail different costs. RROC space is defined by plotting the total over-estimation and under-estimation on the x-axis and y-axis, respectively. Other evaluation metrics were explored, such as the Area Over the RROC curve (AOC) which was shown to be equivalent to the error variance. Another relevant effort towards the adaptation of the concept of ROC curves to regression tasks was made by (Bi, J.; Bennett, 2003) with the proposal of Regression Error Characteristic (REC) curves that provide a graphical representation of the cumulative distribution function (CDF) of the error of a model. These curves plot the error tolerance and the accuracy of a regression function which is defined as the percentage of points predicted within a given tolerance. REC curves illustrate the predictive performance of a model across the range of possible errors. The Area Over the Curve (AOC) can also be evaluated and is a biased estimate of the expected error of the model. RROC and REC curves, although interesting, are still not sensitive to the error location across the target variable domain. To address this problem, (Torgo, 2005) proposed Regression Error Characteristic Surfaces (RECS) whose goal is to add an extra dimension into REC curves representing the cumulative distribution of the target variable. RECS shows how the errors corresponding to a certain point of the REC curve are distributed across the range of the target variable (e.g., label range). Another contribution of (Torgo, 2005) is the partial REC curve, which is a particular case of a REC curve where the analysis of the error distribution is limited to a certain label range such as rare events, which permits the analysis of errors within that range. Accordingly, with a unimodal data distribution (one peak within the distribution), we will obtain two different ranges of rare labels. And with a multimodal data distribution (with n peaks), we will end up with $n + 1$ ranges of rare labels and with $n+1$ partial REC curves, which will be difficult to analyze (all at once). Therefore, the partial REC curve for rare events is suitable only when having one range of rare labels i.e., when dealing with data whose distribution has no peak

such as J shaped data distributions (exponential or log normal data distributions). To solve this problem and to further improve graphical-based metrics for imbalanced domains, we propose a REC curve which incorporates all $n + 1$ ranges of rare labels, which will be denoted as True Positive Rate REC REC_{TPR} curve. We also define another REC curve which pools all ranges of frequent labels, referred to as the True Negative Rate REC (REC_{TNR}) curve. The goal of REC_{TPR} and REC_{TNR} curves will be to evaluate the error magnitude of positive (rare) and negative (frequent) instances respectively, based on the non-uniform relevance of the target variable domain. Then, inspired by scalar metrics for classification such as Geometric Mean (G-Mean) and Class-Weighted Accuracy (CWA), these two curves can be further fused into one single curve which can be either REC_{G-Mean} curve or REC_{CWA} curve. Scalar metrics. Performance measures commonly used in regression such as Mean Squared Error (MSE) and Mean Absolute Error (MAE) are not adequate to these specific problems. These measures assume a uniform relevance of the target variable domain and evaluate only the magnitude of the error. Although the sum of the magnitude of numeric errors holds valuable information, for tasks with imbalanced domains of the target variable, this sum should also be sensitive to the errors location within the target variable domain, because it is frequently biased to the performance on poorly represented values of the target. This means that the error magnitude must have a differentiated impact depending on the values of the target domain where the error occurs. Attempts to apply this differentiated impact within measures were made by (Torgo and Ribeiro, 2009; Torgo et al., 2015) through the notions of precision/recall and F1-measure derived from the utility function (Torgo and Ribeiro, 2007) (which is a special-purpose learning method as mentioned in the previous subsection). However, these measures can only be applied to this special-purpose learning methods and cannot generalize to all algorithm-level methods. As such, we suggest evaluation strategies which can be applied to all regression models and which take into account the imbalance distribution of datasets, namely the Geometric Mean Error (GME) and Class-Weighted Error (CWE).

2.3 Methodology

In this section, we introduce the key components for our cost-sensitive approach handling regression tasks given imbalanced data (Sec. 3.1). Next, define our graphical-based and scalar evaluation approaches suitable for measuring the performance of regression tasks under imbalanced-domains (Sec. 3.2).

2.3.1 Cost-sensitive learning approach

In this subsection, we introduce the cost-sensitive learning technique for regression with highly imbalanced data based on training a neural network using a proposed probabilistic loss function. Given a highly imbalanced training dataset $\{(\mathbf{x}^s, y^s)\}_{s=1}^S$ where S is the total number of instances (e.g., samples) in the training set, our goal is to train a neural network represented by a function $f(\cdot)$ which maps between the input vector \mathbf{x}^s and the output or target variable y^s via parameters $\boldsymbol{\theta}$ as follows,

$$\hat{y}^s = f(\mathbf{x}^s, \boldsymbol{\theta}) \quad (2.1)$$

where \hat{y}^s is the network output (e.g., estimated output value).

Optimal tuned parameters $\boldsymbol{\theta}$ are obtained after the training process of the neural network which is accomplished through the minimization of an objective function that measures the error between the target variable y^s and the network output \hat{y}^s as follows,

$$L(\boldsymbol{\theta}) = \frac{1}{N} \sum_{i=1}^N \ell(r^i), \quad (2.2)$$

where N is the number of instances per batch, $\ell(\cdot)$ is the loss function, $r^i = y^i - \hat{y}^i$, r^i denoted/referred to as the residual, y^i and \hat{y}^i being the target and predicted values respectively for an instance i within the batch. In regression problems, the typical loss function used is the quadratic loss function (called $L2$ loss) defined as,

$$\ell_2(r^i) = (r^i)^2 \quad (2.3)$$

However, training a neural network with $L2$ using imbalanced data yields a poor performance especially on scarce/rare events, since it tends to learn more from the normal examples (most frequent ones), neglecting the rare ones. In other words, with frequent data being the majority data, the network tends to overfit, generating parameters $\boldsymbol{\theta}$ that tend to fit the most frequent data much more than the rare ones. Thus, $L2$ deteriorates the performance of the network on predicting infrequent events and needs further adjustment to take into account this imbalance in the data.

To this end, we propose a biased loss function which favors rare events (e.g., rare data) over frequent ones by adding more loss on instances whose target variable is rare and adding no loss on instances whose target variable is frequent. The main idea is to add an additional term to $L2$ that takes into account the importance of the target variable itself, the rare ones being more important than the frequent

ones. To do, we first define the importance function, also denoted as the “relevance” function, which gives the degree of importance to each target variable based on a probabilistic approach. The next step is to incorporate this relevance function into our biased loss function as well as describe the behavior of this latter given a rare or frequent instance and its derivative with respect to the predicted value of that instance.

2.3.1.1 Relevance function

The concept of relevance was first introduced by Torgo and Ribeiro (Torgo and Ribeiro, 2007) and its goal is to express the domain-specific biases concerning the different importance of the target values. It is expressed as a continuous function $\phi(y) : y \rightarrow [0, 1]$ that maps the target variable domain y into a $[0, 1]$ scale of relevance, where 0 and 1 represent the minimum and maximum relevance respectively. In (Torgo and Ribeiro, 2007), the relevance function ϕ was obtained by applying piecewise cubic Hermite interpolation to the box-plot (Cleveland, 1993) statistics of the target variable. In our study, we define two relevance functions: (i) Normal Density Relevance (NDR) for any uni-modal symmetric distribution of the target variable domain, and (ii) Kernel Density Relevance (KDR) for any asymmetric target distribution (uni-modal and multimodal).

2.3.1.1.1 Normal Density Relevance (NDR). Let’s consider the most common type of probability density functions: the probability density of the normal (Gaussian) distribution. Given y , a target variable or label of a an instance (e.g., of an observation or event) i within the batch, its normal density function (NDF) can be written as,

$$f(y|\mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{y-\mu}{2\sigma^2}} \quad (2.4)$$

where μ is the average over events y^s within the training set $\{(\mathbf{x}^s, y^s)\}_{s=1}^S$, i.e., $\mu = \frac{1}{S} \sum_{s=1}^S y^s$, and σ is its standard deviation. In order to illustrate this, as an example, let’s consider labels (e.g. target variables) of a dataset called “cpuSM” where the label corresponds to the relative performance of a CPU based on its attributes. Note that labels have been normalized to the interval $[0, 1]$. Figure 2.1.a consists of a histogram representing labels of this dataset. On the other hand, the NDF of the same dataset is shown in Figure 2.1.b with an average μ and standard deviation σ of 0.817 and 0.1864 respectively. Note that the NDF approaches zero at rare events (low label values) and attains its peak $\frac{1}{\sigma\sqrt{2}}$ at most frequent events (i.e. at high label values). Multiplying the probability density function $f(y|\mu, \sigma)$ by its peak value gives rise to $P(y|\mu, \sigma)$, a distribution function in the range $[0, 1]$ which

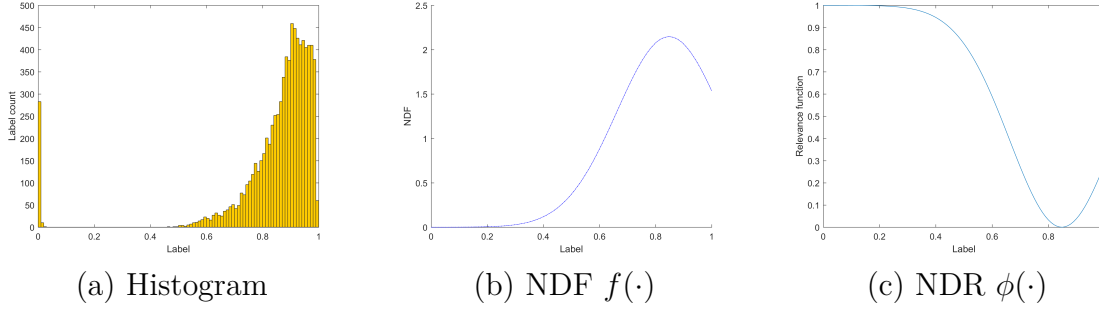


Figure 2.1: Plot (a) is a histogram that represents the “cpuSM” target distribution, grouped into bins of interval 0.01. Plots (b) and (c) are the NDF $f(\cdot)$ and the NDR function $\phi(\cdot)$ of the same distribution.

can be regarded as a probability of having a target variable y^i given an average value μ and standard deviation σ ,

$$P(y|\mu, \sigma) = e^{\frac{-y-\mu^2}{2\sigma^2}} \quad (2.5)$$

Given $P(y|\mu, \sigma)$, the relevance function of the normal density (also denoted as Normal Density Relevance NDR) can be thought of as the probability of the complement of the event y as follows,

$$\phi(y) = 1 - P(y|\mu, \sigma) \quad (2.6)$$

This implies that, when y is a label of a rare event, the relevance term $\phi(y)$ attains its peak 1; and conversely, the relevance term $\phi(y)$ approaches 0 on frequent events, as illustrated by the NDR in Figure 2.1.c. For example, Figure 2.1.c illustrates the relevance for the “cpuSM” dataset. Nonetheless, comparing the histogram of Figure 2.1.a and the NDF of Figure 2.1.b shows that the NDF doesn’t really suit the histogram bins as the *cpuSM* target distribution is asymmetrically distributed and multimodal. This conveys the need of a more robust relevance function whose density function fits multimodal and asymmetric data.

2.3.1.1.2 Kernel Density Relevance (KDR). The proposed Kernel Density Relevance (KDR) is computed based on the Kernel Density Estimation (KDE). Also known as the Parzen’s window (Parzen, 1962), KDE is one of the popular approaches to estimate the underlying probability density function of a dataset. Compared to the normal probability density distribution stated above which requires parameters such as the mean and standard deviation of the distribution, KDE is a nonparametric density estimator which automatically learns the shape of the density from the data. Let y^s be an independent, identically distributed random sample from the unknown distribution taken from the training set $\{(\mathbf{x}^s, y^s)\}_{s=1}^S$. Formally, given

y , a target variable of a an event i within the batch, the density function f of KDE can be expressed as,

$$f(y) = \frac{1}{Sh} \sum_{j=1}^S K\left(\frac{y - y^j}{h}\right) \quad (2.7)$$

where K is a smooth function called the kernel function and $h > 0$ is the smoothing bandwidth that controls the amount of smoothing. Intuitively, KDE has the effect of smoothing out each data point into a smooth bump, whose shape is determined by the kernel function K . Then, KDE sums over all these bumps to obtain a density estimator. At regions with many observations, because there will be many bumps around, KDE will yield a large value. On the other hand, for regions with only a few observations, the density value from summing over the bumps is low, because only a few bumps contribute to the density estimate. Figure 2.2.a illustrates the KDE applied to the same dataset as above, where the selected parameters K (kernel) and h (bandwidth) are chosen to be respectively the normal kernel and a bandwidth computed using least square cross-validation (as explained below). Compared to the NDF of Figure 2.1.b, the KDE fits better the histogram of “cpuSM” target distribution (Figure 2.2.a).

Kernel selection. A range of kernel functions K are commonly used: uniform, triangular, biweight, triweight, Epanechnikov, normal, and others. The normal kernel (also known as the Gaussian kernel) is often used and is defined as $K(x) = \varphi(x)$, where $\varphi(x)$ is the standard normal density function: $\varphi(x) = \frac{1}{\sqrt{2}}e^{-\frac{1}{2}x^2}$.

Bandwidth selection. Choosing the most appropriate smoothing bandwidth h for KDE is crucial for computing the best density estimate. Indeed, when h is too small, there are many wiggles in the density estimate. Conversely, when h is too large, we smooth out important features. Common approaches to bandwidth selection include the rule of thumb (Dehnad, 1987), least square cross-validation (Rudemo, 1982), biased cross-validation (Scott and Terrell, 1987), and plug-in method (Woodroffe, 1970). In our study, the applied bandwidth selection method is the least square cross-validation. After selecting the optimal parameters K and h for a particular dataset, the probability of having a target variable can be derived by turning $f(y)$ (eq. 2.7) into a distribution function in the range $[0, 1]$, as follows

$$P(y) = \frac{1}{MSh} \sum_{j=1}^S K\left(\frac{y - y^j}{h}\right) \quad (2.8)$$

where $M = \max(\{P(y^1), \dots, P(y^s)\})$, M being the peak value of $f(y)$. Thus, given $P(y)$, the Kernel Density Relevance KDR can be obtained using Equation 2.10. Computing KDR of the “cpuSM” dataset gives us the plot in Figure 2.2.b. Let’s

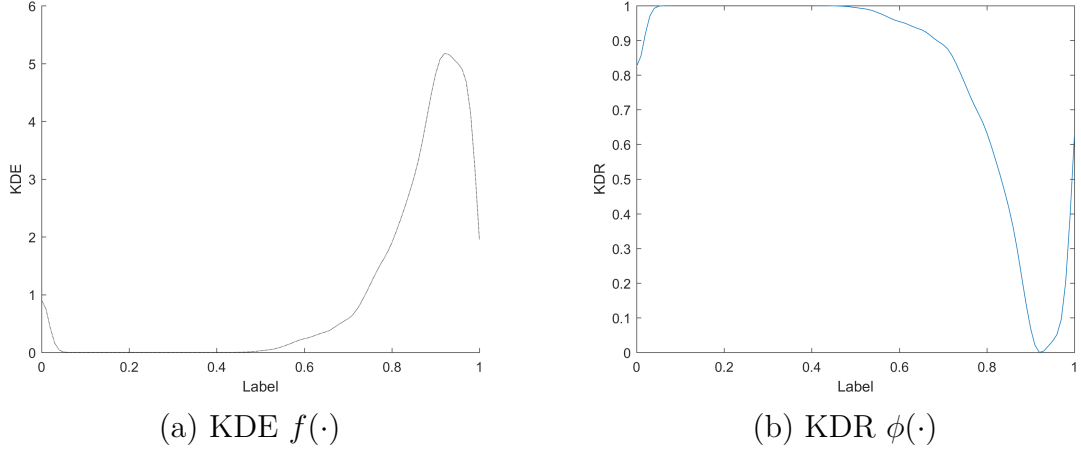


Figure 2.2: Plot (a) and (b) are the KDE $f(\cdot)$ and the KDR function $\phi(\cdot)$ of the same target distribution of the “cpuSm” dataset.

note that, as the number of instances S within the training set becomes large, computing the KDR $\phi(y)$ for each and every element y within the batch of the neural network becomes heavy as we have to compute a sum of S exponentials for every element y . Computation becomes even heavier when this process is repeated for each batch and for each epoch. To alleviate this problem, we apply sampling by computing a sum of U exponentials rather than S exponentials, where $U \ll S$. As a reminder, sampling is a statistical procedure that is concerned with the selection of some observations (e.g., samples) to help us make statistical inferences about the whole population. As for the choice of the sample size U , the work of (Cohen et al., 2017) states that, in survey research, 100 samples should be identified for each major sub-group in the population and between 20 to 50 samples for each minor sub-group. And according to (DelÍce, 2001), a sample size between 30 and 500 at 5% confidence level is generally sufficient for many researchers. Accordingly, we choose a relatively large sample size $U = 1000$. As such, we under-sample the training set by randomly selecting U instances from it and making these instances represent our final training set. So, the corresponding probability becomes,

$$P_U(y) = \frac{1}{MUh} \sum_{j=1}^U K\left(\frac{y - y^j}{h}\right) \quad (2.9)$$

Finally, given $P_U(y)$, the relevance function (or Kernel Density Relevance – KDR) is given by,

$$\phi(y) = 1 - P_U(y) \quad (2.10)$$

As seen through the KDR of the “cpuSm” dataset (Figure 2.2), $\phi(y)$ attains its peak 1 for rare target variables y and approaches 0 on frequent target variables.

2.3.1.2 Probabilistic loss function

The goal of this section is to integrate one of the relevance functions mentioned above into the original ℓ_2 -loss function (eq. 2.3). The idea is that, for every instance i within the batch, a cost term $C(y^i)$ is added to ℓ_2 , which is proportional to the relevance function ϕ and to the absolute value of the residual r^i , as follows,

$$C(y^i) = |y^i - \hat{y}^i| * \phi(y^i) \quad (2.11)$$

where y^i and \hat{y}^i are the target value and prediction of the model at an instance i . The resultant probabilistic loss function ℓ_p for an instance i can be written as,

$$\ell_p(y^i, \hat{y}^i) = (y^i - \hat{y}^i)^2 + C(y^i) \quad (2.12)$$

The ℓ_p loss function is a regularized version $L2$ whose goal is to control the trade-off between fitting the data well and giving more importance to infrequent (rarely occurring) events. The role of the first term $(y^i - \hat{y}^i)^2$ is to fit the training data well, whereas the role of the second $C(y^i)$ is to avoid overfitting most frequent events and to give more weight/importance to residuals r^i generated/coming from scarce events. Furthermore, by taking a closer look at eq. 2.11, the second term of the equation $C(y^i)$ will have an impact on the loss function $\ell_p(y^i, \hat{y}^i)$ only when the first term $(r^i)^2$ is kept small. Indeed, if the absolute value of the residual is bigger than 1 (> 1), then $(y^i, \hat{y}^i)^2 \gg C(y^i)$, thus ending up with a loss function similar to the original $L2$ with $\ell_p(y^i, \hat{y}^i) = \ell_2(y^i, \hat{y}^i)$. So, the absolute value of the residual needs to be small enough (in the range $[0, 1]$) for the cost term $C(y^i)$ to have an effect on the loss function $\ell_p(i)$. To do so, target values need to be normalized into the interval $[0, 1]$. Accordingly, equation 2.11 becomes,

$$\begin{aligned} \ell_p((y^i)', \hat{y}^i) &= (y^i - \hat{y}^i)^2 + C((y^i)') \\ (y^i)' &= \frac{y^i - \min_{j \in \{1, \dots, S\}} y^j}{\max_{j \in \{1, \dots, S\}} y^j - \min_{j \in \{1, \dots, S\}} y^j} \end{aligned} \quad (2.13)$$

where y^i and $(y^i)'$ are the original and normalized target variables for instance i respectively.

For convenience, the normalized target variable $(y^i)'$ will be further denoted as y^i and the normalized loss function $\ell_p((y^i)', \hat{y}^i)$ as $\ell_p(y^i, \hat{y}^i)$ in the rest of the paper.

Furthermore, it is worth mentioning that the loss function ℓ_p is not differentiable at $\hat{y}^i = y^i$ because of the absolute value $|y^i - \hat{y}^i|$ present within the cost $C(y^i)$ (eq.

2.8). So, to resolve this issue, $C(y^i)$ is updated as follows,

$$C(y^i) = \sqrt{(y^i - \hat{y}^i)^2 + \varepsilon} * \phi(y^i) \quad (2.14)$$

where ε is a very small term ($\varepsilon = 10^{-9}$).

Using Equation 2.13 and 2.14, the final probabilistic loss function for an instance i of the batch can be written as,

$$\ell_p(y^i, \hat{y}^i) = (y^i - \hat{y}^i)^2 + \sqrt{(y^i - \hat{y}^i)^2 + \varepsilon} * \phi(y^i) \quad (2.15)$$

In practice, when dealing with a frequent event y^i , the probability of having y^i approaches 1; so the cost term $C(y^i)$ approaches 0, resulting in $\ell_p(y^i, \hat{y}^i) \approx \ell(y^i, \hat{y}^i)$. Contrariwise, when having a rare event y^i , $P(y^i) \approx 0$ and $C(y^i) \approx 1$, resulting in $\ell_p(y^i, \hat{y}^i) \approx (y^i - \hat{y}^i)^2 + |y^i - \hat{y}^i|$. To illustrate this, two labels of the “cpuSM” dataset (e.g. performance of CPUs) are used: a rare label $y^i = 0.3$ (e.g., a low CPU performance value) and a frequent one $y^i = 0.9$ (e.g., a high CPU performance value). Then, both ℓ_p and ℓ are plotted as a function of the residual r where $r = \hat{y}^i - y^i$, with y^i being set to 0.3 (Figure 2.3.a) or 0.9 (Figure 2.3.b).

prediction \hat{y}^i , with y^i being set to 0.3 (as shown in Figure 2.3.a) or 0.9 (as depicted in Figure 2.3.b). In the case of $\hat{y}^i = 0.3$, the probabilistic loss ℓ_p gets higher than the quadratic loss ℓ as the residual gets further from 0 i.e., as the prediction gets further from the target variable. For instance, for a large residual $r^i = 0.6$, $\ell_p(0.6) = 0.96$ while $\ell(0.6) = 0.36$. Hence, it is clear that an extra cost is added to the probabilistic loss function ℓ_p for rare events. On the other hand, for the frequent label $y^i = 0.9$ (Figure 2.3.b), $\ell_p(r^i)$ and $\ell(r^i)$ are almost equal, meaning that no extra cost is applied when dealing with frequent events. During backpropagation, the probabilistic loss function ℓ_p produces a gradient whose magnitude is linearly proportional to the residual plus a constant term (bias). The gradient of the loss of an instance i with respect to the network output \hat{y}^i is given by,

$$\frac{\partial \ell_p(y^i, \hat{y}^i)}{\partial \hat{y}^i} = 2(y^i - \hat{y}^i) + \frac{y^i - \hat{y}^i}{\sqrt{(y^i - \hat{y}^i)^2 + \varepsilon}} * \phi(y^i) \quad (2.16)$$

This means that, for a frequent event y^i , the second term of the derivative is close to 0, whereas for a rare event y^i , this second term approaches 1, producing a higher magnitude of the gradient which (in turn) biases the whole training process, enabling the neural network to adapt to rare events, thereby improving the performance of the network. Plots in Figure 2.4 display derivatives of both ℓ_p and ℓ with respect to the residual r^i for $y^i = 0.3$ (Fig. 2.4.a) and $y^i = 0.9$ (Fig. 2.4.b). We observe that the magnitude of the gradient of ℓ_p is higher than the gradient of ℓ by a constant

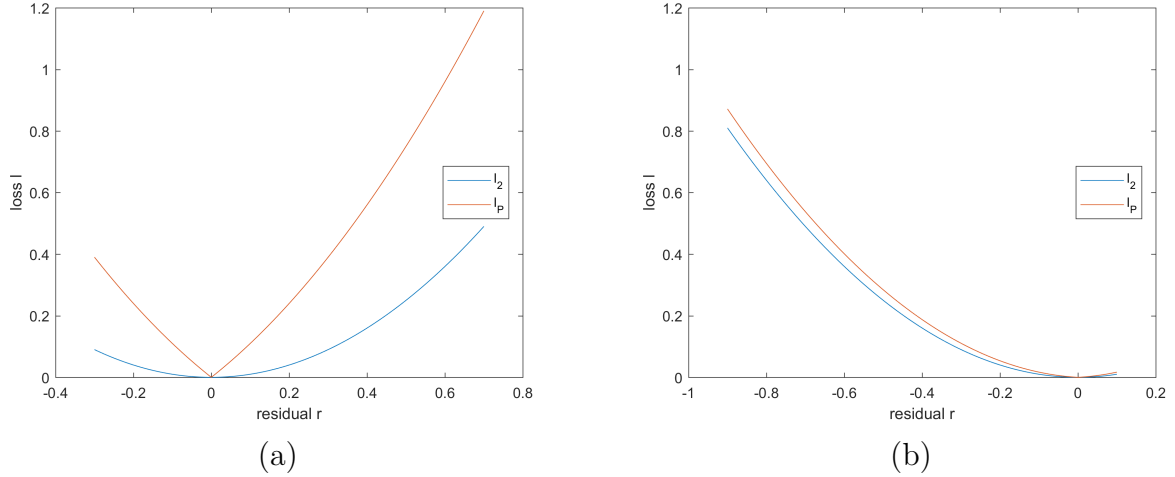


Figure 2.3: Red and blue plots represent the value of ℓ and ℓ_p respectively with respect to the residual r^i (x-axis) for two scenarios: a rare event i having a low CPU performance label $y^i = 0.3$ (a), and a frequency event having a high CPU performance label $y^i = 0.9$ (b).

$\phi(y^i)$ for $\hat{y}^i > y^i$ and lower by the same constant $\phi(y^i)$ for $\hat{y}^i < y^i$. So, for $y^i = 0.9$, $\phi(y^i) = 0$. And for $y^i = 0.3$, the constant term is almost 1 with $\phi(y^i) = 0.9786$. Using the backpropagation rule and gradient descent, the cost-sensitive gradient at an instance i can be written as,

$$\frac{\partial \ell_p(y^i, \hat{y}^i)}{\partial \theta} = \frac{\hat{y}^i}{\theta} (2(y^i - \hat{y}^i) + \frac{r^i}{\sqrt{(y^i - \hat{y}^i)^2 + \epsilon}} * \phi(y^i)) \quad (2.17)$$

Therefore, for any rare event i , the probabilistic loss function has the property of increasing the magnitude of the partial derivative with respect to the network output during backpropagation, thereby increasing the magnitude of the gradient by an amount that is inversely proportional to the frequency of occurrence of that event. In other words, each training sample has a different contribution to the minimization depending on the frequency of its occurrence. Other advantages of the use of our probabilistic loss function is that (i) the minimization and weighting are integrated into a single function and (ii) only soft constraints are imposed without the need of setting a hard threshold on the loss function. Algorithm 4 summarizes steps for optimizing network parameters θ using our cost-sensitive learning approach based on one of the relevance functions (of section 2.3.1.1) and the probabilistic loss function (2.3.1.2). Let's note that the term U was set to 1000 when the number of training instances becomes larger than 1000.

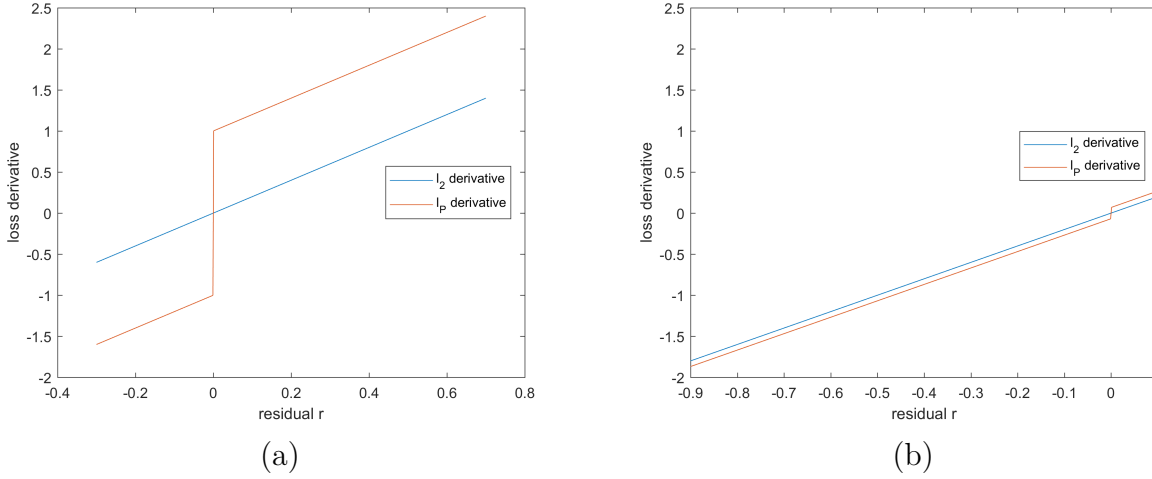


Figure 2.4: Blue and red plots represent the derivatives of ℓ and ℓ_p functions respectively with respect to residual r^i for two scenarios: an instance i with a rare label $y^i = 0.3$ (a), and another instance i with a frequent label $y^i = 0.9$ (b).

Algorithm 4: optimization for parameter θ of cost-sensitive learning approach.

Data: Imbalanced training dataset $[x, y]$ (x the inputs and y the outputs),
Maximum epoch: M , Number of batches per epoch: B , learning
rate: α , relevance type: NDR or KDR, vector of labels of mini-batch
instances: y_{batch} , Number of training instances: S .

Result: θ^*

initialization;

instructions;

```

for ( epoch = 1; epoch < M; epoch = epoch + 1 ) {
  for ( batch = 1; batch < B; batch = batch + 1 ) {
    forward passing ;
    switch relevance  $\phi$  do
      case NDR do
        Compute  $P(y_{batch}|\mu, \sigma)$  (eq. 2.5) ;
      end
      case NDR do
        if  $S < 1000$  then
          Compute  $P(y_{batch})$  (eq. 2.8) ;
        end
        else
          Compute  $P_{1000}(y_{batch})$  (eq. 2.9) ;
        end
      end
    end
  }
}

```

2.3.2 Evaluation methods for regression under imbalanced domains

2.3.2.1 Graphical-based evaluation approaches

Since standard evaluation criteria are not suitable for describing regression performance for imbalanced domains as they tend to focus their evaluation of the model on the most frequent events only, in this section, we propose to update the graphical-based evaluation technique REC (Bi, J.; Bennett, 2003) such as to take into consideration the imbalanced distribution of datasets. We first start with an overview or background on classification metrics, including the True Positive Rate (TPR) and the True Negative Rate (TNR) in classification tasks. Next, we show how to derive TPR and TNR for regression. Then, based on these measures (TPR and TNR), we attempt to define their corresponding REC curves (REC_{TPR} and REC_{TNR} curves) as well as their corresponding Area Over the Curve (AOC). Finally, we propose to merge these 2 curves using interpolation in order to obtain a G-mean REC (REC_{G-Mean}) curve as well as a CWA REC (REC_{CWA}) curve.

2.3.2.1.1 Background on classification metrics. In this section, we present an overview of the confusion matrix typically used in classification and its elements as well as the classification measures TPR and TNR . In the field of statistical classification and specifically the binary-class problem, a confusion matrix (Stehman, 1997), also known as an error matrix, is a specific table layout that allows visualization of the performance of an algorithm or model, typically a supervised learning one. For the two class classifier, the confusion matrix consists of information about actual and predicted classification return by a classifier. The entries in the confusion matrix are summarized in Table 2.1 and are denoted as: (i) True Positive (TP) referring to the number of positive examples which are correctly predicted as positives by the model, (ii) True Negative (TN) denoting the number of negative examples correctly classified as negatives by the model, (iii) False Positive (FP), often referred to as false alarm, which is defined by the number of negative examples incorrectly classified as positives by the model, and (iv) False Negative (FN), sometimes known as miss, which is determined as the number of positive examples incorrectly assigned as negatives by a classifier. However, analyzing the four entries

Table 2.1: Confusion matrix.

		Predicted (Classified as)	
		Positive	Negative
Actual (Really is)	Positive (p)	True Positive (TP)	False Negative (FN)
	Negative (n)	False Positive (FP)	True Negative (TN)

in the confusion matrix is not enough in determining the performance of a classifier.

Therefore, several derivatives based on the previously discussed confusion matrix are used in evaluating classification models:

- True Positive Rate or Recall or Sensitivity, referring to the ability of a model in correctly identifying a positive class as such (ranging from 0 to 1), is denoted as;

$$TPR = \frac{TP}{TP + FN} = \frac{TP}{p} \quad (2.18)$$

where p is the number of positive examples (instances) which corresponds to the sum of true positives (TP) and false negatives (FN).

- True Negative Rate or Specificity, denoting the ability a model in correctly identifying negative class as such, is determined as,

$$TNR = \frac{TN}{TN + FP} = \frac{TN}{n} \quad (2.19)$$

where n is the number of negative target examples (instances) which corresponds to the sum of true negatives (TN) and false positives (FP).

2.3.2.1.2 Definition of TPR and TNR for regression. As explained, the confusion matrix comprises of four results from classification outputs that report the number of true positives (TP), true negatives (TN), false positives (FP) and false negatives (FN). In our study, we propose a mapping of these confusion matrix elements used for classification into the regression field.

Positive and negative classes for target variables.

As such, we choose ‘positive’ to refer to rare events (e.g., events whose target variable is considered as rare) and ‘negative’ to stand for frequent events (e.g., events whose target variable is considered as frequent). In other words, positive events represent the minority ‘class’ while negative events constitute the majority ‘class’. Given m instances in the test set $\{(\mathbf{x}^i, y^i)\}_{i=1}^m$ with \mathbf{x}^i and y^i being respectively the input vector and the continuous target variable at instance i , the idea/goal is to change the target variable domain as follows: $R \rightarrow \{0, 1\}$ where 0 and 1 stand for the negative target class (negative label) and the positive target class (positive label) respectively. To do so, we propose a function τ , that maps the original domain of continuous target variables into these two discrete classes,

$$\tau(\cdot) =] - \infty, +\infty[\rightarrow \{0, 1\} \quad (2.20)$$

The role of the function τ is to convert/change target variable values within the test set whose relevance is greater than a user-defined threshold t_E into “1” (the

positive class), and to turn target variable values whose relevance is lower than t_E into “0” (the negative class). The function τ of a target variable y^i at the instance i can be expressed as:

$$\tau(y^i) = \begin{cases} 1 & \text{if } \phi(y^i) \geq t_E \\ 0 & \text{otherwise} \end{cases} \quad (2.21)$$

In other words, any target variable whose relevance is greater than the threshold t_E is considered to be a positive instance and to belong to the positive class ($\tau(y^i) = 1$); and any target variable whose relevance is lower than t_E is regarded as a negative instance with $\tau(y^i) = 0$.

Positive and negative classes for network output. Now that we have defined the class of each target variable, we need to define the rule/formula for determining whether the network output (e.g., prediction) of the model at a given instance is positive or negative. In other words, each network output which is a continuous value has to be mapped also into a positive or a negative class. To do so, for each instance i , we make use of its residual r^i (e.g., the difference between the network output and the target variable of that instance). The basic idea is that the loss of the residual at that instance - e.g., $\ell(r^i)$ or $\ell(\hat{y}^i, y^i)$ such that $\hat{y}^i = f(\mathbf{x}^i, \theta)$, must be less than a tolerance ϵ before it is considered as a good/correct prediction. So, if $\ell(\hat{y}^i, y^i)$ is less than ϵ , then the model has made a good prediction about i and the predicted value \hat{y}^i is turned into the same class as the class of the target/response variable y^i via a function τ' . And, conversely, having $\ell(\hat{y}^i, y^i)$ greater than ϵ turns \hat{y}^i into the opposite class of that of y^i via the same function τ' . The following algorithm is used to describe τ' .

Algorithm 5: function τ' .

Data: $\epsilon_i = \ell(\hat{y}^i, y^i)$, $i = 1, \dots, m$. (m being the number of instances within the test set).

Result: \hat{c}_i , $i = 1, \dots, m$. (\hat{c}_i being the predicted class of instance i)

initialization;

instructions;

for ($i = 1$; $batch < M$; $i = i + 1$) {

if $\epsilon_i \leq \epsilon$ then		$\hat{c}_i = \tau(y^i)$;
else		$\hat{c}_i = 1 - \tau(y^i)$;
end		

}

TPR and **TNR** for multiple values of ϵ . In the previous subsection, we discussed how target variables $\{y^i\}_{i=1}^m$ and network outputs $\{\hat{y}^i\}_{i=1}^m$ were labeled into positive and negative classes via function τ and τ' respectively. After defining positive and negative classes for each of the target values and predicted values, we can build our regression confusion matrix, and thereby compute **TPR** and **TNR**. However, using function τ' is conditioned by the parameter ϵ , which is hard to define. In that sense, we propose to compute τ' (of all test set instances) for multiple values of ϵ . Thus, for each and every value of ϵ , we will obtain **TPs** and **TNs** and then compute the associated **TPR** and **TNR**. The first step is to split the m instances of the test set into two categories: one that contains the p instances whose target variable is positive with $\tau(y^i) = 1$, and another one composed of the n instances whose target variable is negative with $\tau(y^i) = 0$. As shown in the confusion matrix (Table 2.1), $p = TP + FN$ and $n = FP + TN$. Meanwhile, having already obtained the predicted class (e.g., label) of all m instances (including the p and n instances) for each error/tolerance ϵ , we can compute True Positives for each ϵ ($TP(\epsilon)$) which corresponds to the number of positive instances correctly predicted as positives i.e., the number of correctly predicted instances within the following p instances: $\{(\mathbf{x}^{i_p}, y^{i_p})\}_{i_p=1}^p$,

$$TP(\epsilon) := \left| \left\{ (\mathbf{x}, y) : \hat{c}_{i_p} = \tau(y^{i_p}), i_p = 1, \dots, p \right\} \right| \quad (2.22)$$

which, according to the algorithm of function τ' , can also be written as

$$TP(\epsilon) := \left| \left\{ (\mathbf{x}, y) : \ell(\hat{y}^{i_p}, y^{i_p}) \leq \epsilon, i_p = 1, \dots, p \right\} \right| \quad (2.23)$$

Hence, based on equation 2.18 and 2.22, we can compute $TPR(\epsilon)$, the true positive rate with respect to ϵ ,

$$TPR(\epsilon) = TP(\epsilon)/p \quad (2.24)$$

Similarly, we can compute $TN(\epsilon)$, the number of negative instances which are correctly predicted as negative i.e., the number of correctly predicted instances within the following n instances $\{(\mathbf{x}^{i_n}, y^{i_n})\}_{i_n=1}^n$,

$$TN(\epsilon) := \left| \left\{ (\mathbf{x}, y) : \hat{c}_{i_n} = \tau(y^{i_n}), i_n = 1, \dots, n \right\} \right| \quad (2.25)$$

which, according to the algorithm of function τ' , can also be written as,

$$TN(\epsilon) := \left| \left\{ (\mathbf{x}, y) : \ell(\hat{y}^{i_n}, y^{i_n}) \leq \epsilon, i_n = 1, \dots, n \right\} \right| \quad (2.26)$$

Thus, based on equation 2.19 and 2.23, the true negative rate with respect to ϵ becomes, estimate of the CDF of the error. And, given a set of m instances, $(\mathbf{x}^1, y^1), (\mathbf{x}^2, y^2), \dots, (\mathbf{x}^m, y^m)$, the accuracy at tolerance ϵ is,

$$TNR(\epsilon) = \frac{TN(\epsilon)}{n} \quad (2.27)$$

2.3.2.1.3 Definition of REC curves for TPR and TNR. Given $TPR(\epsilon)$ and $TNR(\epsilon)$ for each ϵ , the next step is to build a REC curve which plots TPR as a function of ϵ and another REC curve which plots TNR as a function of ϵ . To do so, an overview of the REC curve is laid out. Then, the process of developing REC curves for TPR and TNR is described.

Overview of REC curve. As stated in the work of (Bi, J.; Bennett, 2003), the REC curve considers an e-insensitive loss for all possible values of a tolerance ϵ , where residuals must be greater than ϵ before they are considered as errors. In the same study, the REC curve is also an estimate of the CDF of the error. And, given a set of m instances, $(\mathbf{x}^1, y^1), (\mathbf{x}^2, y^2), \dots, (\mathbf{x}^m, y^m)$, the accuracy at tolerance ϵ is,

$$acc(\epsilon) := \frac{|\{(\mathbf{x}, y) : \ell(\hat{y}^i, y^i) \leq \epsilon, i = 1, \dots, m\}|}{m} \quad (2.28)$$

The set of points (coordinates) used to plot the REC curve is $\{(\epsilon_j, acc(\epsilon_j))\}_{j=1}^m$, $j = 1, \dots, m$ where $\epsilon_j = l(\hat{y}^j, y^j)$. In other words, given the m instances of the set, their associated tolerance/error is computed. Then the errors ϵ_i are sorted in ascending order and the “plot” command interpolates between the plotted points with a line (Bi, J.; Bennett, 2003). The REC curve plots the error tolerance ϵ on the x-axis and the accuracy at ϵ which is the percentage of points predicted correctly within that tolerance on the y-axis. The resulting curve estimates the cumulative distribution function (CDF) of the error. According to (Bi, J.; Bennett, 2003), the expectation of ϵ defined as

$$E(\epsilon) \approx \epsilon_m \hat{P}(\epsilon_m) - \left(\sum_{j=1}^{m-1} (\epsilon_{j+1} - \epsilon_j) \hat{P}(\epsilon_j) + \epsilon_1 \hat{P}(\epsilon_0) \right) \quad (2.29)$$

where $\epsilon_0 = 0$, ϵ_m is the maximum observed error, and $P(\epsilon)$ is the empirical distribution estimated on the sample data to approximate the probability distribution P at ϵ . The first terms $\epsilon_m P(\epsilon_m)$ corresponds to the area of the box corresponding to the REC curve, assuming $P(\epsilon_m) = 1$, while the second term (in parenthesis) stands for the area under the curve. Thus, $E(\epsilon)$ can be approximated by the area over the curve (AOC) within that box, and AOC can be regarded as is a biased estimate of the expected error.

REC_{TPR} curve. Suppose we want to build a REC curve for a certain range of

the target variable, the positive target variables only which are represented by the positive instances (p instances). We define $acc_p(\epsilon)$, the accuracy within the positive instances $i_p = 1, \dots, p$ at tolerance ϵ as:

$$acc_p(\epsilon) := \frac{|\{(\mathbf{x}, y) : \ell(\hat{y}^{i_p}, y^{i_p}) \leq \epsilon, i_p = 1, \dots, p\}|}{p} \quad (2.30)$$

which is nothing but the true positive rate at ϵ (eq. 2.24). Thus, $TPR(\epsilon) = acc_p(\epsilon)$. The resultant *REC* curve, denoted as REC_{TPR} , plots the error tolerance ϵ on the x-axis and the accuracy of the model for the positive instances (e.g., the range of positive target variables) which represents the TPR (the percentage of correctly classified instances as positives over the total number of positive instances p) in the y-axis.

*REC*_{TNR} curve. Similarly, we build a *REC* curve for another range of the target variable, namely the negative instances (n instances). Accordingly, the accuracy within the negative instances at tolerance ϵ can be expressed as,

$$acc_n(\epsilon) := \frac{|\{(\mathbf{x}, y) : \ell(\hat{y}^{i_n}, y^{i_n}) \leq \epsilon, i_n = 1, \dots, n\}|}{n} \quad (2.31)$$

which is equal to the true negative rate at ϵ (eq. 2.27). Thus, $TNR(\epsilon) = acc_n(\epsilon)$. The resultant *REC* curve, denoted as REC_{TNR} , plots the accuracy of the model within the range of negative instances i.e., the *TNR* (the percentage of correctly classified instances as negatives over the total number of negative instances) in the y-axis.

2.3.2.1.4 Definition of *REC*(G-Mean) and *REC*(CWA) curves. Just like *TPR* and *TNR* metrics, REC_{TPR} and REC_{TNR} curves often exhibit a trade-off and it is impractical to simultaneously monitor both of them. The goal of this subsection is to come up with a single *REC* curve that describes the behavior of both *TPR* and *TNR* *REC* curves. To this matter, we propose novel graphical models that combine information held by both *TPR* and *TNR* *REC* curves, namely REC_{G-Mean} and REC_{CWA} curves. First, an overview of G-mean and CWA measures for classification is laid out. Then, the fusion of both REC_{TPR} and REC_{TNR} curves into REC_{G-Mean} and REC_{CWA} curves is described.

Background on scalar metric for imbalanced domains. When dealing with classification in imbalanced domains, the most frequently used scalar measures are: the F_β score (?), the Geometric Mean (G-Mean) (Kubat et al., 1998) and the Class-Weighted Accuracy (CWA) (Cohen et al., 2006). The F_β score is defined as, $F_\beta = ((1 + \beta^2) \times TPR \times precision) / (\beta^2 \times TPR + precision)$ where $precision = TP / (TP + FP)$ and where β is a coefficient set by the user to ad-

just the relative importance of TPR with respect to precision. One disadvantage of using the F_β measure is that this latter does not take into account the performance of the negative class and only tests the effectiveness of a classifier on predicting correctly the positive class. The geometric mean ($G - Mean$) which is defined as,

$$G - Mean = \sqrt{TPR \times TNR} \quad (2.32)$$

was developed specifically for assessing the performance under imbalanced domains. It computes the geometric mean of the accuracies of the two classes, attempting to maximize them while obtaining good balance. However, with this formulation, equal importance is given to both classes (the positive and the negative class), which is sometimes not desired by the user. The Class-Weighted Accuracy (CWA) was introduced to deal with the issue of F_β which does not consider the performance of the negative class and that of G-Mean which gives equal importance to the negative (majority) and positive (minority) classes and does not let the user assign more weight to the minority class. CWA is defined as,

$$CWA = w \times TPR + (1 - w) \times TNR \quad (2.33)$$

where $0 \leq w \leq 1$, w being the user-defined weight of the positive class. REC_{G-Mean} and REC_{CWA} curves. Given equation 2.31 and 2.32, the goal is to combine both REC_{TPR} and REC_{TNR} curves to obtain either REC_{G-Mean} or REC_{CWA} curves. However, as mentioned in section 3.2.2.b, the REC curve is a plot of a set of points interpolated with a line, where the points are $\{(\epsilon_j, P(\epsilon_j))\}_{j=1}^m$ with m the number of samples within the set and ϵ_i being sorted in ascending order ($\epsilon_j = \ell(\hat{y}^j, y^j)$). Therefore, the set of points of the REC_{TPR} curve $\{(\epsilon_{j_p}, P(\epsilon_{j_p}))\}_{j_p=1}^p$ (where $\epsilon_{j_p} = \ell(\hat{y}^{j_p}, y_{j_p})$) is different from the set of points of the REC_{TNR} curve $\{(\epsilon_{j_n}, P(\epsilon_{j_n}))\}_{j_n=1}^n$ (where $\epsilon_{j_n} = \ell(\hat{y}^{j_n}, y_{j_n})$), the former having a set of errors $\{\epsilon_{j_p}\}_{j_p=1}^p$ different from the latter set of errors $\{\epsilon_{j_n}\}_{j_n=1}^n$. Moreover, let's note that the number of points p within REC_{TPR} curve is less than the number of points n within the REC_{TNR} curve, the positive instances being less than the negative ones due to the imbalanced distribution of the dataset. As a result, we cannot simply perform a scalar multiplication between points of REC_{TPR} curve and REC_{TNR} curve in order to plot REC_{G-Mean} and REC_{CWA} curves. An alternative is to apply interpolation to the set of points of the REC_{TPR} curve as well as the set of points of the REC_{TNR} curve in order to get interpolated values at k specific points (for instance, $k = 0 : 0.001 : \max(\epsilon_{j_p})$) using the "nearest" interpolation. The algorithm below shows how interpolation is conducted/performed. Furthermore, extrapolation is used for evaluating points that lie outside the domain of the p error values. With interpolation, we no longer have p

points in the REC_{TPR} curve and n points in the REC_{TNR} curve but rather k points in both curves (see algorithm 6). This suggests that we can compute the k points of the REC_{G-Mean} curve by doing a point-wise (scalar) operation/multiplication. Similarly, the k points of the REC_{CWA} curve can also be computed.

Algorithm 6: plot either REC_{G-mean} or REC_{CWA} curve.

Data: selected measure (“*Gmean*” or “*CWA*”): $measure$, selected weight w if $measure = CWA$, Coordinates of points of the REC_{TPR} curve $[\epsilon_p, \hat{P}_p]$.

Result: $\epsilon_{measure} = \epsilon_{i=1}^k$, where $k = \frac{\max(\epsilon_p)}{step} + 1$ (k the number of error points of $REC_{measure}$ curve), and $\hat{P}_{measure} = P(\epsilon_i)_{i=1}^k$

$step = 0.001$;

$\epsilon_{measure} = 0 : step : \max(\epsilon_p)$;

$\hat{P}_p = interpolate(\epsilon_p, \hat{P}_p, \epsilon_{measure}, 'nearest', 'extrapolate')$;

$\hat{P}_n = interpolate(\epsilon_n, \hat{P}_n, \epsilon_{measure}, 'nearest', 'extrapolate')$;

if $measure = “Gmean”$ then

 | $\hat{P}_{measure} = (\hat{P}_p * \hat{P}_n)^{1/2}$;

else if $measure = “CWA”$ then

 | $\hat{P}_{measure} = \hat{P}_p * w + \hat{P}_n * (1 - w)$;

2.3.2.2 Scalar evaluation approaches

2.3.2.2.1 Definition of MAE for negatives and MAE for positives. According to (Bi, J.; Bennett, 2003), instead of computing the expectation $E(\epsilon)$ via the *AOC* of the *REC* curve, one alternative to finding $E(\epsilon)$ is the use of the sample mean *MAE* or *MSE* as an estimation of the expected error, which are expressed as,

$$MSE = \frac{1}{N} \sum_{i=1}^N (\hat{y}^i - y^i)^2 \quad (2.34)$$

and,

$$MAE = \frac{1}{N} \sum_{i=1}^N |\hat{y}^i - y^i| \quad (2.35)$$

However, as mentioned in section 2, these metrics are not suitable for regression in imbalanced domains since they assume a uniform relevance of the target variable domain by evaluating the magnitude of the numeric error only. The scalar evaluation techniques needed when dealing with imbalanced datasets should be sensitive to the errors location within the target variable domain. Instead of applying *MAE* by computing the mean error of all instances (i.e., by averaging over all instances),

the idea is to compute the mean error for specific ranges of the target variable, just as done with our graphical-based evaluation approach (section 3.2.1). In that sense, as performed in section 3.2.1, target variables are split into 2 categories: positive and negative ones; then the mean absolute error for positive instances MAE_p and the mean absolute error for negative instances MAE_n are computed. To do so, we first need to specify what the positive and negative instances are. According to equation 2.19 (section 3.2.1.b), any instance whose target variable has a relevance greater than the threshold t_E is considered positive, whereas any instance whose target variable has a relevance lower than t_E is a negative. So, aggregating all errors coming from positive instances (whose target variable is positive) produces MAE_p while averaging over negative instances gives MAE_n ,

$$MAE_p = \frac{1}{\sum_{i:c=1}^N} \sum_{i:c=1} |\hat{y}^i - y^i| \quad (2.36)$$

and,

$$MAE_n = \frac{1}{\sum_{i:c=0}^N} \sum_{i:c=0} |\hat{y}^i - y^i| \quad (2.37)$$

where

$$c = I(\phi(y^i) \geq t_E), \quad (2.38)$$

with \hat{y}^i and y^i being the network output and target variable respectively (defined in the range $[0, 1]$).

2.3.2.2.2 Definition of GME and CWE. As discussed in section 2.3.2.1, G-Mean (eq. 2.32) is a scalar metric for classification tasks which computes the geometric mean of the accuracies of two classes (positive and negative classes), attempting to maximize them while obtaining good balance. Similarly, in the case of regression, given the mean errors of the positive and negative classes, we can compute the Geometric Mean Error (GME) which is the geometric mean of these two errors as follows,

$$GME = \sqrt{MAE_p \times MAE_n} \quad (2.39)$$

Also, inspired by *CWA* (eq. 2.33) which gives more importance to the positive class than the negative class (by allowing the user to add more weight to the minority class), we introduce the Class-Weighted Error (*CWE*) which sums the mean errors of the positive and negative classes by giving more weight to the mean error of the

positive instances,

$$CWE = w \times MAE_p + (1 - w) \times MAE_n \quad (2.40)$$

where $0 \leq w \leq 1$, w being the user-defined weight of the positive (rare) instances.

2.3.2.3 Comparison between our scalar and graphical-based measures

Given that the two estimates of $E(\epsilon)$, the AOC of a REC curve and the mean absolute error MAD are close to each other (Bi, J.; Bennett, 2003), let's assume that MAE is an approximation of the AOC , implying $AOC \approx MAE$. So, working with only positives as our target data, MAE_p will be an approximation of the AOC of the REC_{TPR} curve, with $AOC_{REC_{TPR}} \approx MAE_p$. And similarly, $AOC_{REC_{TNR}} \approx MAE_n$.

2.3.2.3.1 AOC of REC(CWA) versus CWE. Based on these approximations and given equation 2.40 we obtain,

$$CWE \approx w * AOC_{REC_{TPR}} + (1 - w) * AOC_{REC_{TNR}} \quad (2.41)$$

And, knowing that REC_{CWA} is plotted using $\epsilon_{measure}$ (see algorithm 6) which are interpolated values of positive errors ϵ_p and negative errors ϵ_n , we replace positive and negative errors of equation 2.41 by these interpolated values. Then, substituting the $AOCs$ by their expectations (eq. 2.29) gives,

$$\begin{aligned} CWE \approx & w * (\epsilon_m \hat{P}_p(\epsilon_m) - (\sum_{i=1}^{m-1} (\epsilon_{i+1} - \epsilon_i) \hat{P}_p(\epsilon_i) + \epsilon_1 \hat{P}_p(\epsilon_0))) + \\ & (1 - w) * (\epsilon_m \hat{P}_n(\epsilon_m) - (\sum_{i=1}^{m-1} (\epsilon_{i+1} - \epsilon_i) \hat{P}_n(\epsilon_i) + \epsilon_1 \hat{P}_n(\epsilon_0))) \end{aligned} \quad (2.42)$$

which is equivalent to,

$$\begin{aligned} CWE \approx & \epsilon_m (w * \hat{P}_p(\epsilon_m) + (1 - w) * \hat{P}_n(\epsilon_m)) + \\ & \sum_{i=1}^{m-1} (\epsilon_{i+1} - \epsilon_i) (w * \hat{P}_p(\epsilon_i) + (1 - w) * \hat{P}_n(\epsilon_i)) + \\ & \epsilon_1 (w * \hat{P}_p(\epsilon_0) + (1 - w) * \hat{P}_n(\epsilon_0)) \end{aligned} \quad (2.43)$$

where $\hat{P}_p(\epsilon)$ and $\hat{P}_n(\epsilon)$ are the empirical distributions estimated on the sample data to approximate the probability distribution of positives P_p and the probability distribution of negatives P_n at ϵ respectively. In the former equation, we assume that ϵ_m , the maximum observed error, is the same for both REC_{TPR} and REC_{TNR} curves. Finally, knowing that $\hat{P}_{CWA} = \hat{P}_p(\epsilon) * w + \hat{P}_n(\epsilon) * (1 - w)$ (see line 18 of

algorithm 6), equation 2.40 becomes,

$$CWE \approx \epsilon_m \hat{P}_{CWA}(\epsilon_m) + \sum_{i=1}^{m-1} (\epsilon_{i+1} - \epsilon_i) \hat{P}_{CWA}(\epsilon_i) + \epsilon_1 \hat{P}_{CWA}(\epsilon_0) \quad (2.44)$$

or simply,

$$CWE \approx AOC_{REC_{CWA}} \quad (2.45)$$

Thus, the scalar measure CWE can be regarded as an approximation of the AOC of the REC_{CWA} curve.

2.3.2.3.2 AOC of REC(G-Mean) versus GME. The scalar measure GME cannot be an approximation of the AOC of REC_{G-Mean} curve since. Indeed, as opposed to the CWE which involves a weighted sum of the MAE_p and the MAE_n , the GME involves a multiplication of these two terms. And, knowing that an AOC consists of a fixed rectangle area (first term of eq. 2.27) minus an integral of the REC curve (a sum in discrete terms) (second term of eq. 2.27), then, multiplying two AOC s ($AOC_{REC_{TPR}} * AOC_{REC_{NR}}$) implies multiplying two integrals. And, unlike the fact that the integral of a sum of two functions is equal to the sum of their integrals, the integral of a product of two functions is not equal to the product of their integrals, meaning that $GME \neq AOC_{REC_{G-Mean}}$.

2.4 Experimental study

2.4.1 Datasets

The proposed cost-sensitive learning approach is tested on 11 imbalanced datasets to: (i) predict continuous labels for 7 datasets drawn from multiple sources: UCI (Dheeru and Karra Taniskidou, 2017), Delve (Akujuobi and Zhang, 2017), etc., and (ii) forecast traffic flow of 4 freeway datasets collected from California State. Since none of these datasets has target variables whose distribution is normal, the Kernel Density Relevance (KDR) is applied rather than the Normal Density Relevance (NDR) in our experiments. Furthermore, we consider a threshold of 0.7 on the relevance values in all datasets ($t_E = 0.7$) to obtain P , the set of positive/rare cases, and N , the set of negative/frequent cases.

2.4.1.1 The 7 datasets

Seven regression datasets are selected from different imbalanced domains. These datasets as well as their characteristics are summarized in Table 2.2. The data

distribution (in the form of histogram bins), NDF , KDE and KDR of each dataset can be visualized in Table 1.7 (Annex). Based on the KDR and the selected threshold t_E , a set of rare and frequent cases is formed for each dataset, as shown in Table 2.2. We can observe that different percentages of rarity are obtained by these datasets, with values ranging between 11.73% and 21.52%.

Table 2.2: A description of the 7 datasets (N : number of instances; $p.total$: number of features in the input; $p.nom$: number of nominal features; $p.num$: number of numeric features; $nRare$: number of rare cases using the Kernel Density Relevance (KDR) with $\phi(y) > 0.7$; Rare: $100 \times nRare/N$).

Dataset	Source	N	p.total	p.nom	p.num	nRare	% Rare
abalone	UCI	4177	8	1	7	679	16.25
accel	—	1732	14	3	11	232	13.39
heat	—	7400	12	4	8	1255	16.96
cpuSm	Delve	8192	12	0	12	1273	15.54
bank8FM	Delve	4499	8	0	8	968	21.52
parkinson telem.	UCI	5875	16	0	16	689	11.73
dAiler	Experiments of Rui Camacho	7129	5	0	5	1363	19.12

As discussed in the methodology (section 2.3), all dataset labels (e.g., target variables) are further normalized to the $[0, 1]$ range using eq. 2.13.

2.4.1.2 The 4 Traffic flow datasets

As for these datasets, our objective is to make precise short-term forecasts of the data traffic flow for different locations of a freeway at any time, based on measurement-based observations. To do so, we first need to feed the proper and relevant observations to our learning system, which is explained in this section.

Traffic network and its spatio-temporal matrix. A traffic network \aleph is defined as a freeway with multiple detectors which are non-uniformly distributed along the freeway and are usually located about junctions such as exits and entrances. These detectors record speeds at every time interval δ along the day for multiple days. In our study, the traffic network \aleph is represented by network points that are uniformly distributed along \aleph by averaging over traffic speed data of different detectors within \aleph per postmile (e.g., per one mile). In the space dimension, traffic speeds of these network points are ordered spatially with reference to a predefined starting point of \aleph , and then fitted into the y-axis. In the time dimension, traffic speeds at successive time intervals are laid out in the x-axis. As a result, a spatio-temporal matrix S_{\aleph}

is formed,

$$S_{\mathbb{N}} = \begin{pmatrix} x_1^1 & \cdots & x_{t-1}^1 & x_T^1 \\ x_1^2 & \cdots & x_{t-1}^2 & x_T^2 \\ \vdots & \ddots & \vdots & \vdots \\ x_1^M & \cdots & x_{t-1}^M & x_T^M \end{pmatrix}$$

where T is the length of time intervals, M is the length of network points; and x_t^i is the average traffic speed on the network point i at time t .

Interpolation. To ensure a more reliable result, missing and erroneous records were properly remedied using temporally adjacent records via interpolation on space, i.e. the missing speed measurement x_t^i for the network point i at time t becomes $(x_{t-1}^i + x_{t+1}^i)/2$.

Spatio-temporal frames. The next step is to turn our spatio-temporal matrix $S_{\mathbb{N}}$ into multiple frames. $S_{\mathbb{N}}$ is segmented using a fixed length sliding window of dimension $m \times \tau$ (chosen to be $m = 10$ and $\tau = 10$) with a high overlap rate of 0.9 between consecutive frames, which means that the sliding window moves with 1 step along either the temporal or spacial dimension. We formally express the traffic speed frame X_t^i of a network point i at time t as a $m \times \tau$ grid denoted by,

$$X_t^i = \begin{pmatrix} ((x')_{t-\tau+1}^{i-m+1}) & \cdots & (x')_{t-1}^{i-m+1} & (x')_t^{i-m+1} \\ \vdots & \ddots & \vdots & \vdots \\ (x')_{t-\tau+1}^{i-1} & \cdots & (x')_{t-1}^{i-1} & (x')_t^{i-1} \\ (x')_{t-\tau+1}^i & \cdots & (x')_{t-1}^i & (x')_t^i \end{pmatrix}$$

where x_k^j indicates traffic speed of network point j at time k . Let's note that the traffic frame X_t^i (of point i at time t) contains speeds of the previous m adjacent network points $\{i - m + 1, \dots, i - 1, i\}$ at times series $\{t - \tau + 1, \dots, t - 1, t\}$ which correspond to previous time points. So, the proposed frame accounts for spatial and temporal correlations of traffic flow. Then, the traffic frame X_t^i is vectorized to form a $1 \times m\tau$ vector \mathbf{x}_t^i as follows,

$$\mathbf{x}_t^i = [(x')_{t-\tau+1}^{i-m+1}, \dots, (x')_{t-1}^{i-m+1}, (x')_t^{i-m+1}, \dots, (x')_{t-\tau+1}^i, \dots, (x')_{t-1}^i, (x')_t^i]$$

, and each traffic speed frame vector \mathbf{x}_t^i will then be used as an input sample to predict the output $y = x_{t+k}^i$ which stands for traffic speed of network point i at time $t + k$ where k denotes the number of time intervals (i.e., forecasting time divided by δ). In our study, $k = 2$ so that the traffic speed of network point i to be predicted/forecasted is at time $t + 2$ (e.g., $t + 10$ -min).

Normalization. Data is normalized by turning speeds above the speed limit to

the speed limit (where this latter is 70 mph). Accordingly, parameters of eq. 2.13 are set as follows: $\max_{j \in \{1, \dots, S\}} y^j = 70$, $\min_{j \in \{1, \dots, S\}} y^j = 0$, and $r = 70$.

Dataset description. Data was taken from the Caltrans Performance Measurement System (PeMS) database found at the PeMS website <http://pems.dot.ca.gov>, where traffic data is collected over 15000 individual detectors which are deployed in freeway systems across California [27]. The collected data are aggregated into 5-min interval each for each detector station (e.g., $\delta = 5$). In our study, the traffic flow data is collected from District 7 from September 1 to September 30 of 2017. Networks used in our experiments as well as their properties are listed in Table 2.3. As shown in Table 2.3, each network consists of a freeway (with multiple sensors) with “starting postmile” denoting the starting point, “ending postmile” the ending point, and “postmile length” the difference between the starting and ending postmile. Using the KDR and $t_E = 0.7$, the number and percentage of rare instances (with respect to the total number of instances) per freeway is computed (Table 2.3).

Table 2.3: Description of networks used in our study. (N : number of instances; $nRare$: number of rare cases using the *KDR* with $t_E = 0.7$; $\%Rare$: $100 \times nRare/N$). Starting postmile, ending postmile and postmile length are in miles.

Network	Starting postmile	Ending postmile	Postmile length	# Sensors	N	nRare	% Rare
US101-North	69.75	2.05	67.70	82	203150	69276	34.10
I5-South	116.77	182.05	65.28	99	244300	84147	34.44
I5-North	194.62	116.83	77.78	90	243766	80514	33.03
I270-West	0.32	52.23	51.91	91	216920	48084	22.16

2.4.2 Experimental setup

In our study, we train neural networks, also referred to as Multi-Layer Perceptrons (MLP). To do so, the MLP hyper-parameters need to be defined. We use the stochastic (mini-batch) gradient descent as our gradient-based optimization method. The dropout rate is set to 0.5, the weight decay to 0.0005 and the momentum to 0.9. Training is performed for 10 to 40 epochs, depending on the dataset used and the experimental method employed (which will be further discussed in the next section). We report results with 10-fold and 4-fold cross validation for the 7 datasets and the traffic flow datasets respectively. Other learning parameters such as the network architecture (e.g, the size of each hidden layer), the learning rate and batch size vary from one dataset to another and are set according to Table 2.4. Indeed, the size of hidden layers (the number of weights per layer) depends on the input size (e.g., the number of attributes within a dataset) and the batch size varies based on

the number of training instances per dataset. Using 10th cross validation for each Table 2.4: Training hyperparameters for each dataset. $[n1; n2; n3]$ defines the architecture of the MLP for each dataset, where $n1$, $n2$ and $n3$ denote the number of neurons in the first, second and third layer respectively.

Dataset	Learning rate	$[n1;n2;n3]$	Batch size
abalone	0.010	[32;64;1]	
accel	0.010	[56;112;1]	
heat	0.005	[72;144;1]	
cpuSM	0.010	[72;144;1]	50
bank8FM	0.005	[32;64;1]	
parkinson	0.001	[64;128;1]	
dAiler	0.010	[20;40;1]	
H101_North_D7			
I5_South_D7	0.005	[500;1000;1]	150
I5_North_D7			
I210_West_D7			

dataset, we randomly choose one tenth of the data to be our testing/validation data and the remaining part as our training data. Performance measures used are the graphical-based evaluation techniques of section 3.2.1 (REC_{G-Mean} and REC_{CWA} curves) as well the scalar evaluation techniques of section 3.2.2 (GME and CWE). The parameter w for the REC_{CWA} curve is set to $2/3$.

2.5 Experiments and results

2.5.1 Experiments

To ensure the diversity of the algorithms and to illustrate the effectiveness of the proposed graphical and scalar evaluation strategies, we compare our cost-sensitive learning approach against different re-sampling strategies whose goal is to balance the number of positive (rare) and negative (normal) instances. As such, the following experiments are conducted on the 7 datasets as well as the traffic flow datasets:

- i Carrying out no sampling i.e., training a basic or classic MLP on each of the original imbalanced datasets using ℓ_2 as the loss function, which will be referred to as “ $\ell_2Unb.$ ”,
- ii Performing random under-sampling on the original imbalanced datasets before training the MLP. In other words, an MLP is trained on every under-sampled version of the original dataset using ℓ_2 , which will be denoted as “ ℓ_2Bal_u ”,

- iii Applying random over-sampling on the original datasets before training the MLP, which means that an MLP is trained on every over-resampled version of the original dataset using ℓ_2 , which will be called “ ℓ_2Bal_o ”,
- iv Training an MLP on each of the original imbalanced datasets using ℓ_p as the loss function (referred to as “ $\ell_pUnb.$ ”). As discussed in previous sections, in imbalanced regression problems it is necessary to use suitable evaluation measures. Accordingly, all experiments are conducted using graphical-based and scalar evaluation methods defined in section 3.2 (e.g., REC_{G-Mean} , REC_{CWA} , GME and CWE).

2.5.2 Results

Conducting the four experiments “ $\ell_2Unb.$ ”, “ ℓ_2Bal_u ”, “ ℓ_2Bal_o ”, “ $\ell_pUnb.$ ” (on the 7 datasets and the traffic flow datasets) and applying the proposed graphical-based evaluation approach gives rise to REC_{TPR} , REC_{TNR} , REC_{G-Mean} and REC_{CWA} curves for each dataset, which are displayed in Table 1.8 of the Supplementary material. Also, the AOC of REC_{G-Mean} and REC_{CWA} curves are computed for each dataset and displayed in Table 2.5.

Table 2.5: Results of the 11 datasets, in terms of the AOC of the REC_{G-Mean} and REC_{CWA} curves. The best results per AOC and per dataset are marked in bold.

Datasets	$AOC_{REC_{G-Mean}}$				$AOC_{REC_{CWA}}$			
	$\ell_2Unb.$	ℓ_2Bal_u	ℓ_2Bal_o	$\ell_pUnb.$	$\ell_2Unb.$	ℓ_2Bal_u	ℓ_2Bal_o	$\ell_pUnb.$
abalone	2.037	2.069	1.851	1.829	2.210	2.048	1.918	1.897
accel	3.689	2.806	2.655	2.724	3.613	2.962	2.781	2.876
heat	24.144	21.782	22.031	21.731	25.105	21.968	22.345	22.421
cpuSM	7.661	6.463	6.413	6.472	8.313	6.532	6.359	6.721
bank8FM	0.027	0.027	0.025	0.025	0.029	0.028	0.026	0.026
parkinson	15.612	12.778	12.301	14.020	14.685	13.151	12.819	13.010
dAiler	1.317	1.101	1.076	1.085	1.384	1.123	1.111	1.100
H101_North_D7	2.356	2.336	2.327	2.174	2.583	2.562	2.543	2.388
I5_South_D7	2.421	2.401	2.397	2.305	2.636	2.612	2.607	2.490
I5_North_D7	2.760	2.740	2.720	2.660	2.980	2.968	2.939	2.880
I210_West_D7	2.644	2.629	2.578	2.440	2.898	2.855	2.806	2.777

After running the four experiments using the proposed scalar evaluation techniques GME and CWE , results are computed and reported in Table 2.6.

Moreover, another way of showing the efficiency of each method is to look at its time consumption i.e., how fast it takes for each MLP method to converge. So, the goal is to compare between their convergence speed. Note that, usually, the measure employed to quantify the convergence speed of an MLP is the number of epochs. Nonetheless, ℓ_2Bal_u and ℓ_2Bal_o techniques use a different training set than the one used for $\ell_2Unb.$ and $\ell_pUnb.$ techniques, with the latter techniques having

Table 2.6: Results for the four experiments applied on the 7 datasets as well as the traffic flow datasets, in terms of scalar measures GME and CWE . The best results per measure and per dataset are in bold.

Datasets	GME				CWE			
	$\ell_2Unb.$	ℓ_2Bal_u	ℓ_2Bal_o	$l_PUnb.$	$\ell_2Unb.$	ℓ_2Bal_u	ℓ_2Bal_o	$l_PUnb.$
abalone	1.929	1.993	1.925	1.891	2.389	2.059	2.050	1.998
accel	3.013	2.655	2.507	2.537	3.750	3.052	2.854	2.953
heat	22.050	21.350	21.628	21.557	25.423	21.832	21.993	22.279
cpuSm	6.939	6.598	6.543	6.503	8.592	6.515	6.536	6.878
bank8FM	0.026	0.027	0.025	0.025	0.029	0.029	0.026	0.026
parkinson	11.284	11.354	11.191	11.083	14.847	13.294	12.851	12.719
dAiler	1.140	1.102	1.076	1.089	1.397	1.135	1.109	1.107
H101_North_D7	2.113	2.095	2.116	1.949	2.712	2.686	2.658	2.477
I5_South_D7	2.275	2.249	2.255	2.120	2.763	2.737	2.735	2.591
I5_North_D7	2.292	2.288	2.255	2.117	3.036	3.026	2.995	2.788
I210_West_D7	2.352	2.331	2.284	2.217	3.112	3.011	2.951	2.886

less training instances than ℓ_2Bal_o (due to over-sampling) and more instances than ℓ_2Bal_u (due to under-sampling). So, for instance, having epochs of ℓ_2Bal_o lasting longer (having to go through more training samples) than epochs of $\ell_2Unb.$ suggests that the number of epochs is not an appropriate convergence speed measure for our study. An alternative is the use of the total number of backpropagation runs performed by the MLP before convergence as the convergence speed measure, which is equal to $\frac{nb_{trainingSamples} \times nb_{epochs}}{size_{batch}}$ where $size_{batch}$ is the batch size (e.g., the number of instances per batch). Convergence speed of each technique for each dataset is displayed in Table 2.7.

Table 2.7: Convergence speeds (in terms of the number of backpropagations required before convergence) of the different approaches for the 7 datasets and the traffic flow datasets. The least number of backpropagations recorded per dataset is reported in bold.

	# training instances				# epochs				# backpropagations			
	$\ell_2Unb.$	ℓ_2Bal_u	ℓ_2Bal_o	$l_PUnb.$	$\ell_2Unb.$	ℓ_2Bal_u	ℓ_2Bal_o	$l_PUnb.$	$\ell_2Unb.$	ℓ_2Bal_u	ℓ_2Bal_o	$l_PUnb.$
abalone	3753	1204	6274	3753	100	140	30	30	7506	3371	3764	2252
accel	1557	420	2694	1557	85	260	75	100	2647	2184	4041	3114
heat	6660	2254	11066	6660	90	44	24	24	11988	1984	5312	3197
cpuSM	7371	2272	12470	7371	61	54	39	29	8993	2454	9727	4275
bank8FM	4041	1756	6326	4041	40	45	21	17	3233	1580	2657	1374
parkinson	5283	2254	11066	5283	98	100	135	150	10355	4508	29878	15849
dAiler	7120	3152	11088	7120	32	29	11	9	4557	1828	2439	1282
H101_North_D7	203150	144931	261369	203150	20	25	15	15	81260	72466	78411	60945
I5_South_D7	244300	175866	312734	244300	20	25	15	15	97720	87933	93820	73290
I5_North_D7	243766	202339	285193	243766	20	25	15	15	97506	101170	85558	73130
I210_West_D7	216920	108160	325680	216920	20	25	15	15	86768	54080	97704	65076

Graphical-based evaluation measures. From REC_{TPR} , REC_{TNR} , REC_{G-Mean} and REC_{CWA} curves (Table 1.8) as well as the $AOC_{REC_{G-Mean}}$ and the $AOC_{REC_{CWA}}$ (Table 2.5), the following remarks can be drawn: From curves within Table 1.8,

REC_{TPR} curves of all datasets successfully show higher accuracies for ℓ_2Bal_u and ℓ_2Bal_o techniques compared to the $\ell_2Unb.$ technique. Indeed, training an MLP with $\ell_2Unb.$ on the original imbalanced dataset makes the network learn more from negatives (e.g., negative instances) than from positives (e.g., positive instances) as weights tend to fit to what it sees the most (the most frequent instances being the negatives). On the other hand, if we train MLPs with as many positives as negatives using either ℓ_2Bal_u or ℓ_2Bal_o , the corresponding MLPs learn as much from positives as from negatives. These correct findings/results convey that the REC_{TPR} curve acts as a good tool for visualizing accuracies of positives. On the other hand, REC_{TNR} curves of all datasets successfully show higher accuracies for the $\ell_2Unb.$ technique compared to ℓ_2Bal_u and ℓ_2Bal_o techniques. Indeed, ℓ_2Bal_u and ℓ_2Bal_o are less prone to fitting negatives than $\ell_2Unb.$ since they are more influenced by positives than $\ell_2Unb.$ weights. As we know, training MLPs on balanced datasets (such as in ℓ_2Bal_u and ℓ_2Bal_o) produce higher G-Mean and CWA rates than those obtained after training MLPs on imbalanced datasets (as in $\ell_2Unb.$), as both G-Mean and CWA metrics give as much importance to the accuracy of positives as the accuracy of negatives. In parallel, REC_{G-Mean} and REC_{CWA} curves show the same behavior and display a higher performance using ℓ_2Bal_u and ℓ_2Bal_o techniques versus using the $\ell_2Unb.$ technique (Table 1.8). Thus, both REC_{G-Mean} and REC_{CWA} curves can be regarded as a generalization of REC curves with the main goal of providing means to facilitate the analysis of the predictive performance of regression models on imbalanced datasets.

Scalar evaluation measures. As graphical-based evaluation measures, obtained GME and CWE errors are lower when training MLPs on balanced datasets (as in ℓ_2Bal_u and ℓ_2Bal_o) than when training MLPs on imbalanced datasets (as in $\ell_2Unb.$), as conveyed by results of Table 2.6. Thus GME and CWE can be seen as proper measure for evaluating the performance of regression with imbalanced datasets.

Comparing graphical-based and scalar evaluation measures. From Tables 2.5 and 2.6, we notice that $AOC_{REC_{CWA}}$ values are close to CWE values for all datasets, with 0.11 as the average of the absolute difference between the former value and the latter value over all 11 datasets and all 4 techniques. This confirms our previous statement (of section 2.3.2.3) that the scalar evaluation measure CWE can be regarded as an approximation of the AOC of REC_{CWA} curve and can be considered as an estimate of the expected error. On the other hand, we obtain 0.56 as an average of the absolute difference between GME and $AOC_{REC_{G-Mean}}$ values over all datasets and all techniques, implying that GME and $AOC_{REC_{G-Mean}}$ are different evaluation strategies.

Comparing our cost-sensitive learning approach to existing techniques. From

REC_{G-Mean} and REC_{CWA} curves and their AOCs (Table 2.5 and Table 1.8), GME and CWE results (Table 2.6) and convergence speed results (Table 2.7), the following observations can be made:

- i Training MLPs on imbalanced datasets with the ℓ_p loss function (as in $\ell_pUnb.$) yields better performance than training them with ℓ_2 ($\ell_2Unb.$) by having less error as shown in GME , CWE , AOC of REC_{G-Mean} and AOC of REC_{CWA} . This suggests that $\ell_pUnb.$ successfully updates network parameters to take into account the scarcity of rare instances/observations (whose target variables are rare). Moreover, the $\ell_pUnb.$ technique succeeds in learning proper weights for recognizing not only positive instances but also negative ones, as seen through REC_{TPR} and REC_{TNR} curves (in Fig. B of Supplementary material). Indeed, $\ell_pUnb.$ pushes/converges weights toward recognizing more positive instances than $\ell_2Unb.$, making the accuracy on the REC_{TPR} curve better than $\ell_2Unb.$ and the accuracy on the REC_{TNR} curve lower than $\ell_2Unb.$. Furthermore, $\ell_pUnb.$ converges faster than $\ell_2Unb.$, the former needing 30% less backpropagation runs than the latter for full convergence, as illustrated in Table 2.7. This means that our proposed method ($\ell_pUnb.$) presents clear advantages when compared with the classic method ($\ell_2Unb.$) including a faster convergence and better generalization, suggesting that our method is a suitable technique for dealing with imbalanced datasets.
- ii In general, our algorithm achieves results slightly higher or close to the ones obtained via ℓ_2Bal_u and ℓ_2Bal_o techniques (in terms of scalar and graphical-based evaluation measures) except for the “accel”, “cpuSm” and “Parkinson” datasets where results of $\ell_pUnb.$ is less favorable. Let’s also note that, as opposed to other dataset distributions, the “Parkinson” and “accel” target distributions have multiple fluctuations and the “cpuSm” target distribution has 2 sharp peaks (at lowest and highest label values), as shown in the distributions within Table 1.7 (see the supplementary material). So, $\ell_pUnb.$ technique is good at regression tasks (i.e., at predicting continuous labels) of imbalanced datasets when these latter have a data target distribution with no sharp peaks. This can be explained by that fact that, when a data target distribution is multimodal and exhibits sharp and narrow peaks close to each other (as the ones mentioned above), the KDR has a hard time approximating such a distribution, missing these peaks, as illustrated through KDEs of “accel”, “cpuSm” and “Parkinson” datasets (table 1.7). So, in such cases, the $\ell_pUnb.$ technique, whose loss function gives more or less weight/importance to instances based on the KDE distribution, fails in giving the proper weight/importance to instances whose labels have an up or down fluctuation within the data distri-

bution. A solution to this issue is to decrease the bandwidth h of the KDE to make the KDR more sensitive to peaks and fluctuations.

- iii In addition to that, let's note that $\ell_pUnb.$ is a straight forward method which simply trains the MLP on the original dataset whereas re-sampling approaches require an extra preprocessing step to balance the original dataset, risk over-fitting and having a burden in computational time when over-sampling, and cause the loss of useful information when under-sampling.
- iv Comparing ℓ_2Bal_o technique to $\ell_pUnb.$ technique in terms of speed shows that the latter converges much faster than the former, requiring 36% less backpropagation runs, as depicted by Table 2.7. Therefore, $\ell_pUnb.$ can be regarded as a fast technique for handling imbalanced datasets.
- v Comparing ℓ_2Bal_u technique to $\ell_pUnb.$ technique shows the latter converges faster than the former for “abalone”, “bank8FM”, “dAiler” and traffic flow datasets, and slower than the former for the rest of the datasets. However, it is worth mentioning that the MLP of ℓ_2Bal_u is less performant than $\ell_pUnb.$ since the former does not always lead to global minima due to the removal of valuable negative instances by ℓ_2Bal_u .

2.6 Conclusion

In this chapter, we first introduced a robust cost-sensitive learning algorithm for regression on datasets with imbalanced target distribution, using a neural network model whose optimization is based on a probabilistic loss function. The main idea is to up-weight the gradient at positive instances such that these latter maximally influence the training process. Other interesting properties of this loss function are the integration of weighting and minimization in a single function as well as the soft constraints that it imposes without any hard threshold. Comparative results with existent methods suggest that our algorithm has a faster convergence and better generalization than the classical method, and has comparable or slightly higher results than sampling techniques, with a faster convergence than the over-sampling approach and a convergence speed comparable to that of the under-sampling method. Also, our second contribution is the proposal of scalar and graphical-based evaluation strategies suitable for regression tasks under imbalanced domains. Inspired by common classification concepts such as positives (p) and negatives (n), and based on the relevance function induced from one of the two estimates of the probability density function of a dataset (NDF or KDE), we derived scalar measures such as the Geometric Mean Error (GME) and Class-Weighted Error (CWE), as well as

graphical-based measures REC_{TPR} , REC_{TNR} , REC_{G-Mean} and REC_{CWA} . Running experiments showed that these scalar and graphical-based metrics could be regarded as proper measures for evaluating regression models under imbalanced domains. We further showed that REC_{G-Mean} and REC_{CWA} could be considered as a generalization of REC curves regression algorithms with imbalanced datasets. As a perspective, our cost-sensitive learning technique can be further applied to any deep learning feed-forward model to handle such regression tasks. It can even be used as an estimator for any linear regression model with imbalanced datasets. Also, our proposed scalar and graphical-based evaluation approaches could be further used to evaluate the performance of any model under imbalanced domains.

Bibliography

- Uchenna Akujuobi and Xiangliang Zhang. Delve: a dataset-driven scholarly search and analysis system. *ACM SIGKDD Explorations Newsletter*, 19(2):36–46, 2017. ISSN 1931-0145.
- Gaurav Bansal, Atish P. Sinha, and Huimin Zhao. Tuning Data Mining Methods for Cost-Sensitive Regression: A Study in Loan Charge-Off Forecasting. *Journal of Management Information Systems*, 25(3):315–336, 2008. ISSN 0742-1222. doi: 10.2753/MIS0742-1222250309.
- K.P. Bi, J.; Bennett. Regression Error Characteristic Curves. In *Proceedings of the Twentieth International Conference on Machine Learning (ICML-2003)*, pages 43–50, 2003. ISBN 1577351894.
- Paula Branco, Luís Torgo, and Rita P Ribeiro. A survey of predictive modeling on imbalanced domains. *ACM Computing Surveys*, 49(2):1–50, 2016. ISSN 03600300. doi: 10.1145/2907070.
- Paula Branco, Luís Torgo, Rita P Ribeiro, Bartosz Krawczyk, and Nuno Moniz. SMOGN: a Pre-processing Approach for Imbalanced Regression. *Proceedings of Machine Learning Research*, 74:36–50, 2017.
- Michael Cain and Christian Janssen. Real estate price prediction under asymmetric loss. *Annals of the Institute of Statistical Mathematics*, 47(3):401–414, 1995. ISSN 00203157. doi: 10.1007/BF00773391.
- Peter F. Christoffersen and Francis X. Diebold. Further results on forecasting and model selection under asymmetric loss. *Journal of Applied Econometrics*, 11(5): 561–571, sep 1996. ISSN 08837252. doi: 10.1002/(SICI)1099-1255(199609)11:5<561::AID-JAE406>3.0.CO;2-S.

- Peter F. Christoffersen and Francis X. Diebold. Optimal Prediction Under Asymmetric Loss. *Econometric Theory*, 13(06):808, 1997. ISSN 0266-4666. doi: 10.1017/S0266466600006277.
- W S Cleveland. *Visualizing Data*. 1993. ISBN 0963488406. doi: 10.2307/1269376.
- Gilles Cohen, Mélanie Hilario, Hugo Sax, Stéphane Hugonnet, and Antoine Geissbuhler. Learning from imbalanced data in surveillance of nosocomial infection. *Artificial Intelligence in Medicine*, 37(1):7–18, 2006. ISSN 09333657. doi: 10.1016/j.artmed.2005.03.002.
- Louis Cohen, Lawrence Manion, and Keith Morrison. *Research Methods in Education*. routledge, 2017. ISBN 978-0-415-36878-0. doi: 10.4324/9781315456539.
- Sven F. Crone, Stefan Lessmann, and Robert Stahlbock. Utility based data mining for time series analysis: cost-sensitive learning for neural network predictors. In *Proceedings of the 1st international workshop on Utility-based data mining - UBDM '05*, pages 59–68, 2005. ISBN 1595932089. doi: 10.1145/1089827.1089835.
- Khosrow Dehnad. Density Estimation for Statistics and Data Analysis. *Technometrics*, 29(4):495–495, nov 1987. ISSN 0040-1706. doi: 10.1080/00401706.1987.10488295.
- Ali Delİce. The sampling issues in quantitative research. *Educational Sciences: Theory & Practices*, 10(4):2001–2019, 2001. ISSN 1303-0485. doi: 10.4135/9781849208901.
- Dua Dheeru and Efi Karra Taniskidou. *{UCI} Machine Learning Repository*, 2017.
- Clive W.J. Granger. Outline of forecast theory using generalized cost functions. *Spanish Economic Review*, 1(2):161, 1999. ISSN 14355469. doi: 10.1007/s101080050007.
- Jose Hernandez-Orallo. Soft (Gaussian CDE) regression models and loss functions. *arXiv:1211.1043*, (1), 2012.
- José Hernández-Orallo. ROC curves for regression. *Pattern Recognition*, 46(12):3395–3411, 2013. ISSN 00313203. doi: 10.1016/j.patcog.2013.06.014.
- José Hernández-Orallo. Probabilistic reframing for cost-sensitive regression. *CM Transactions on Knowledge Discovery from Data*, 2014. ISSN 15564681. doi: 10.1145/2641758.

- S H Khan, M Hayat, M Bennamoun, F Sohel, R Togneri, and C V Mar. Cost-Sensitive Learning of Deep Feature Representations from Imbalanced Data. *IEEE Transactions on Neural Networks and Learning Systems*, 29(8):3573–3587, 2018.
- Miroslav Kubat, Robert C. Holte, and Stan Matwin. Machine Learning for the Detection of Oil Spills in Satellite Radar Images. *Machine Learning*, 30:195–215, 1998. ISSN 08856125. doi: 10.1023/A:1007452223027.
- TH Lee. Loss Functions in Time Series Forecasting. *International encyclopedia of the social sciences*,, pages 495–502, 2008.
- Emanuel Parzen. On Estimation of a Probability Density Function and Mode. *The Annals of Mathematical Statistics*, 33(3):1065–1076, 1962. ISSN 0003-4851. doi: 10.1214/aoms/1177704472.
- Rita Ribeiro. Utility-based Regression. PhD thesis, 2011.
- M. Rudemo. Empirical choice of histograms and kernel density estimators. *Scandinavian Journal of Statistics*, 9(2):65–78, 1982. ISSN 03036898. doi: 10.2307/4615859.
- Lamyaa Sadouk, Taoufiq Gadi, and El Hassan Essoufi. A novel cost-sensitive learning approach and metric for regression in imbalanced domains. *Expert Systems*, 2021, 2021. doi: 10.1111/exsy.12680.
- David W. Scott and George R. Terrell. Biased and unbiased cross-validation in density estimation. *Journal of the American Statistical Association*, 82(400): 1131–1146, dec 1987. ISSN 1537274X. doi: 10.1080/01621459.1987.10478550.
- Stephen V Stehman. Selecting and interpreting measures of thematic classification accuracy. *Remote Sensing of Environment*, 62:77—89, 1997.
- Luís Torgo. Regression error characteristic surfaces. In *Proceeding of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining - KDD '05*, page 697, 2005. ISBN 159593135X. doi: 10.1145/1081870.1081959.
- Luis Torgo and Rita Ribeiro. Utility-Based Regression. In *European Conference on Principles of Data Mining and Knowledge Discovery*, pages 597–604, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg. doi: 10.1007/978-3-540-74976-9_63.
- Luis Torgo and Rita Ribeiro. Precision and recall for regression. In *International Conference on Discovery Science*, pages 332–346, 2009. ISBN 3642047467. doi: 10.1007/978-3-642-04747-3_26.

Luís Torgo, Paula Branco, Rita P. Ribeiro, and Bernhard Pfahringer. Resampling strategies for regression. *Expert Systems*, 32(3):465–476, 2015. ISSN 14680394. doi: 10.1111/exsy.12081.

M. Woodroffe. On choosing a delta-sequence. *The Annals of Mathematical Statistics*, 41(5):1665–1671, 1970. ISSN 0003-4851.

Arnold Zellner. Bayesian Estimation and Prediction Using Asymmetric Loss Functions. *Journal of the American Statistical Association*, 81(394):446–451, jun 1986. ISSN 0162-1459. doi: 10.1080/01621459.1986.10478289.

Huimin Zhao, Atish P. Sinha, and Gaurav Bansal. An extended tuning method for cost-sensitive regression and forecasting. *Decision Support Systems*, 51(3): 372–383, 2011. ISSN 01679236. doi: 10.1016/j.dss.2011.01.003.

Part VI

Learning from outliers

In regression analysis, the presence of outliers in the data set can strongly distort the classical least squares (known as “L2”) estimator and lead to unreliable results (due to the large abnormal error registered by outliers compared to the error of the majority of the training samples). To deal with this, several robust-to-outliers methods in Robust Statistics have been proposed in the statistical literature. However, in the context of deep regression networks, very few efforts have been carried out to deal with outliers and most deep regression algorithms make use of the traditional L2 loss function. In this part, we consider the issue of training deep neural networks in the context of robust regression. As such, we introduce a robust deep regression model which is based on a novel robust loss function (Sadouk et al., 2020). With this latter, our model is able to adapt to an outlier distribution, without requiring any hard threshold on the proportion of outliers in the training set. Experimental evaluations on a head pose estimation dataset show that our model generalizes well to noisy datasets, compared to other state-of-the-art techniques.

Chapter 1

Introduction

Statistical estimation is one of the fundamental tools in numerous fields in engineering and science. Most of the techniques rely on strong assumptions about the distribution of the measurements. Most often, these techniques are derived based on the assumption of a Gaussian-distributed noise and yield poor results when there are even small deviations from that assumption. To overcome those problems, robust estimation theory has been introduced by considering a large family of statistical models as well as possible outliers that deviate from the general model Huber (2011). A robust regression method has a high breakdown point, which is the smallest amount of outlier contamination that an estimator can handle before yielding poor results. In other words, an estimator is called robust if a large deviation from the assumed statistical model (recorded from an outlier) has a low impact on the overall performance.

In the last years, deep neural networks revolutionized related fields of research. As such, the goal of this chapter is to apply these robust regression methods in the context of deep learning. In this paper, we introduce a novel loss function within a Convolutional Neural Network (ConvNet) regressor that addresses datasets with outliers by reducing the impact of outliers in the model fitting process. Indeed, in the context of feed-forward neural networks, we can employ a robust estimator (instead of the regular LS or L2 loss function) for minimizing the cost/error function, whereby corrupted training samples (e.g., outliers) with unusually large errors are down-weighted such that they minimally influence the training backpropagation process.

1.1 Robust Regression

A classical statistical inference problem is linear regression which consists of estimating a vector of unknown parameters given a noisy linear observation model.

The Least Squares (LS) method is usually employed to solve such problem but it performs badly in the presence of outliers. To tackle this issue, Robust Regression has been introduced. Indeed, it has long been studied in statistics Huber (2011); Maronna et al. (2018); Rousseeuw and Leroy (2005) and in computer vision Black and Rangarajan (1996); Meer et al. (1991). The most common robust statistical techniques are: the M-estimators, sampling methods, trimming methods and robust clustering. M-estimators Huber (2011) minimize the sum of a positive-definite function of the residuals and attempt to reduce the influence of large residual values. The minimization is carried out with weighted least squares techniques, with no proof of convergence for most M-estimators. Sampling methods Meer et al. (1991) such as least-median-of-squares or random sample consensus (RANSAC), estimate the model parameters by solving a system of equations defined for a randomly chosen data subset. The main drawback of such methods is that they require complex data-sampling procedures and it is tedious to use them for estimating a large number of parameters. Trimming methods Rousseeuw and Leroy (2005), such as Least Trimmed Squares (LTS), rank the residuals and down-weight or discard data points associated with large residuals. They are typically cast into a (non-linear) weighted least squares optimization problem, where the weights are modified at each iteration, leading to iteratively re-weighted least squares problems. Robust statistics have also been addressed in the framework of mixture models and a number of robust mixture models were proposed, such as Gaussian mixtures with a uniform noise component Banfield and Raftery (1993); Coretto and Hennig (2016), heavy-tailed distributions Forbes and Wraith (2014), trimmed likelihood estimators Galimzianova et al. (2015); Neykov et al. (2007), or weighted-data mixtures Gebru et al. (2016).

1.2 Deep Learning for Regression

In the context of regression where continuous values are to be estimated, several deep learning techniques have been recently introduced. For the human pose estimation task, studies Li and Chan (2014); Pfister et al. (2014); Toshev and Szegedy (2014) attempted to estimate positions of the body joints on the image plane. As for facial landmark detection task, the work of Sun et al. (2013) predicts image locations of facial points. In the scheme of object and text detection, the goal is to predict regressed values which represent a bounding box for localization Jaderberg et al. (2016); Szegedy et al. (2013).

However, most of the research community still keeps one element nearly completely fixed: when it comes to regression, most of the studies train their deep learning models based on the standard Least Squares loss function, which is sensi-

tive to outliers. Despite the large literature on regression-based deep learning, only three works attempted to provide a robust deep regressor which tackles the issue of outliers Belagiannis et al. (2015); Diskin et al. (2017); Lathuilière et al. (2018). In Belagiannis et al. (2015), authors achieve robustness by choosing the Tukey’s bi-weight M-estimator as the minimizing loss function. In Diskin et al. (2017), a robust deep regressor is derived by unfolding a gradient descent method for a generalized least trimmed squares objective. This regressor is trained on triplets of unknown parameters, linear models and noisy observations with outliers. Another attempt toward a robust deep regressor was made by Lathuilière et al. (2018) which makes use of an optimization algorithm that alternates between the unsupervised detection of outliers using expectation-maximization, and the supervised training with cleaned samples using stochastic gradient descent.

This part comes as an extension of works Belagiannis et al. (2015); Diskin et al. (2017); Lathuilière et al. (2018). The main contribution of this part is to improve the performance of deep regressors in the presence of outliers with a robust regression approach based on training a ConvNet with novel loss function. In chapter 2, we describe basic components of our approach. We choose to demonstrate our approach on a pose estimation dataset. Accordingly, we describe experimental details (chapter 3), then we discuss experiments and results (chapter 4). Finally, chapter 5 concludes our work.

Chapter 2

Methodology

The neural network is fed by an input image $\mathbf{x} : \Omega \rightarrow \mathbb{R}$ and its corresponding label (target) which is a vector $\mathbf{y} = (y_1, \dots, y_J)$ composed of J elements such that $y_j \in \mathbb{R}$. Having a set of training data with outliers $\{(\mathbf{x}^s, \mathbf{y}^s)\}_{s=1}^S$ composed of S samples (e.g., instances), $\mathbf{x}^{(s)}$ the input vector and $\mathbf{y}^{(s)}$ the output vector, we wish to train a regression model represented by a function $\omega(\cdot)$, under the minimization of a cost function $L(\cdot)$ (also called an objective function) using backpropagation and stochastic gradient descent. Training this network generates tuned parameters $\boldsymbol{\theta}$ which define a mapping between \mathbf{x}^s and \mathbf{y}^s , represented by:

$$\hat{\mathbf{y}}^{(s)} = \omega(\mathbf{x}^{(s)}, \boldsymbol{\theta}) \quad (2.1)$$

where $\hat{\mathbf{y}}^{(s)}$ is the network output e.g., the estimated output value of the network at an instance s .

The training process of a neural network is accomplished through the minimization of $L(\cdot)$ which compares between the target vector and network output vector as follows,

$$\min_{\boldsymbol{\theta}} L(\boldsymbol{\theta}) \quad (2.2)$$

where

$$L(\boldsymbol{\theta}) = \frac{1}{N} \sum_{i=1}^N L^{(i)}(\boldsymbol{\theta}), L^{(i)}(\boldsymbol{\theta}) = \sum_{j=1}^J \ell(r_j^{(i)}) \quad (2.3)$$

where N is the number of instances per batch, J the number of elements within the target vector, $\ell(\cdot)$ is the loss function, and $r_j^{(i)}$ the residual of the j^{th} value of the target vector at an instance i which is defined as,

$$r_j^{(i)} = y_j^{(i)} - \hat{y}_j^{(i)} \quad (2.4)$$

where $y_j^{(i)}$ and $\hat{y}_j^{(i)}$ represent the j^{th} element (value) of the target (true label) and output (estimated) vector respectively at an instance i .

In this chapter, we start by giving an overview of popular loss functions $\ell(\cdot)$ for regression including the standard one and the robust ones (Section 2.1). Next, Section 2.2 defines our novel loss function.

2.1 Background on popular loss functions for regression

L-estimates. The two most common and standard estimators are the Least absolute deviations (LAD) and ordinary Least Squares (OLS). The LAD regression estimator (also known as L1-estimator) minimizes the sum of the absolute values of the residuals and is defined as,

$$\ell(r_j^{(i)}) = |r_j^{(i)}| \quad (2.5)$$

It is quite robust and is generally not affected by outliers, but its derivatives are not continuous which makes it impractical to use as a loss function for training neural networks. On the other hand, the least square (LS) estimate (also denoted as L2-estimate) which minimizes the sum of squared residuals,

$$\ell(r_j^{(i)}) = (r_j^{(i)})^2 \quad (2.6)$$

gives a more stable and closed form solution. Nonetheless, the L2-estimate gives outliers excessive weight by squaring the value of the residual and tries to adjust the model according to these outlier values, even on the expense of other samples.

M-estimates. Compared to the L-estimates which are not robust with respect to bad leverage points (outliers), the M-estimates are one of the robust regression estimation methods whose goal is to reduce the influence of large residual values. M-estimators act like LS except that the squares minimization is modified by a sum of robust penalty functions of the residuals. They are a generalization of the maximum likelihood estimator proposed in Huber (2011) and they are defined as, $\ell(r_j^{(i)}) = \rho(r_j^{(i)})$ such that ρ is a symmetric function with unique minimum at zero. An example of such estimate is the Huber loss, a function function that is quadratic in small values of $r_j^{(i)}$ but grows linearly for large values of $r_j^{(i)}$, as shown in Fig.2.2.b. This function and its partial derivative with respect to $r_j^{(i)}$ (Fig.2.1.b) are given by equations 2.7 and 2.8 respectively,

$$\ell(r_j^{(i)}) = \begin{cases} \frac{1}{2}(r_j^{(i)})^2 & \text{if } |r_j^{(i)}| \leq c \\ c|r_j^{(i)}| - \frac{1}{2}c^2 & \text{otherwise} \end{cases} \quad (2.7)$$

$$\frac{\partial \ell(r_j^{(i)})}{\partial r_j^{(i)}} = \begin{cases} r_j^{(i)} & \text{if } |r_j^{(i)}| \leq c \\ c \operatorname{sign}(r_j^{(i)}) & \text{otherwise} \end{cases} \quad (2.8)$$

where c is the hyper-parameter that controls how small the loss should be to go from the linear to the quadratic forms (c usually being set to 1.345). For instance, Huber loss approaches L1-estimate when $c \approx 0$ and L2-estimate when $c \approx \infty$ (large numbers). However, despite the fact that the Huber loss is convex, differentiable and robust to outliers, setting its parameter c is not an easy task.

Another robust M-estimate which is more robust than Huber loss is the Tukey's biweight function (plotted in Fig.2.2.b) which is defined as,

$$\ell(r_j^{(i)}) = \begin{cases} \frac{c^2}{6} (1 - (1 - (\frac{r_j^{(i)}}{c^2})^2)^3) & \text{if } |r_j^{(i)}| \leq c \\ \frac{c^2}{6} & \text{otherwise} \end{cases} \quad (2.9)$$

, and whose partial derivative with respect to $r_j^{(i)}$ (plotted in Fig.2.1.b) is given by,

$$\ell(r_j^{(i)}) = \begin{cases} r_j^{(i)} (1 - (\frac{r_j^{(i)}}{c^2})^2)^2 & \text{if } |r_j^{(i)}| \leq c \\ 0 & \text{otherwise} \end{cases} \quad (2.10)$$

where c is a tuning constant (which if is set to 4.6851, gives approximately 95% asymptotic efficiency as L2 minimization on the standard normal distribution of residuals). This function has the property of suppressing the influence of outliers during backpropagation by reducing the magnitude of their gradient close to zero, as shown in Fig.2.1.b. However, even though it is robust to outliers, it is non-convex and non-differentiable.

2.2 Our proposed loss function

The idea is to come up with a robust loss function that has advantages over existent robust loss functions (mentioned above) and that generalizes well on deep learning models. Our loss function has a partial derivative with respect to the residual which is inspired from the sigmoid function. Instead of having a partial derivative that looks like step function, as it is the case for the L1 loss partial derivative, we want a smoother version of it that is similar to the smoothness of the sigmoid activation function. To this matter, we will start by defining our loss function partial derivative. Then, from this latter, we will come up with the loss function itself.

Partial derivative of our loss function. Let's consider the sigmoid function as a function of $r_j^{(i)}$ which, as a reminder, is given by $\sigma(r_j^{(i)}) = 1/(1 + e^{-r_j^{(i)}})$. We can

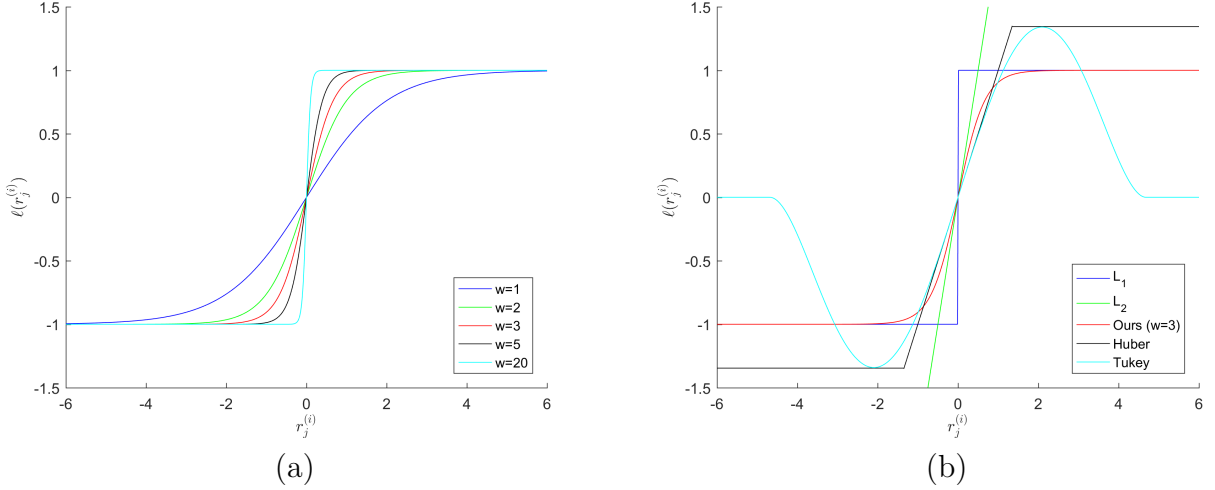


Figure 2.1: Visualization of: (a) multiple variants of the partial derivative of our loss function with respect to the residual with varying weighting factor ($w = \{1, 2, 3, 5, 20\}$), (b) partial derivatives (with respect to the residual) of some commonly used loss functions: L1, L2, Huber (with $c = 1.345$) and Tukey (with $c = 4.685$) as well as ours (with $w = 3$).

add steepness or smoothness to this sigmoid function by adding a weighting factor w which gives us,

$$\sigma_w(r_j^{(i)}) = \frac{1}{1 + e^{-wr_j^{(i)}}} \quad (2.11)$$

The goal is to come up with a loss function whose partial derivative w.r.t $r_j^{(i)}$ acts like a smooth step function being around -1 for large negative residuals and around +1 for large positive residuals (just like the gradient of Huber and L1). And, Knowing that the function σ_w is in the range $[0, 1]$, we rescale this latter to become within the range $[-1, 1]$ and to obtain the following partial derivative of $\ell(\cdot)$,

$$\frac{\partial \ell(r_j^{(i)})}{\partial r_j^{(i)}} = -1 + 2 \frac{1}{1 + e^{-wr_j^{(i)}}} \quad (2.12)$$

where $\lim_{r_j^{(i)} \rightarrow -\infty} \frac{\partial \ell(r_j^{(i)})}{\partial r_j^{(i)}} = -1$, $\lim_{r_j^{(i)} \rightarrow +\infty} \frac{\partial \ell(r_j^{(i)})}{\partial r_j^{(i)}} = +1$, and $\frac{\partial \ell(r_j^{(i)})}{\partial r_j^{(i)}} = 0$ at $r_j^{(i)} = 0$. Figure 2.1.b illustrates this partial derivative as well as partial derivatives of other previously mentioned loss functions (L1, L2, Huber and Tukey).

Our loss function. By computing the loss function itself by integrating its gradient, we arrive at,

$$\ell(r_j^{(i)}) = (r_j^{(i)}) + \frac{2}{w} \log(1 + e^{-w*r_j^{(i)}}) - \frac{2}{w} \log 2 \quad (2.13)$$

where the term $\frac{2}{w} \log 2$ is added in order to have the loss function equal to 0 at $r_j^{(i)} = 0$. This loss function as well as loss functions mentioned in the previous section are depicted in Figure 2.2.b.

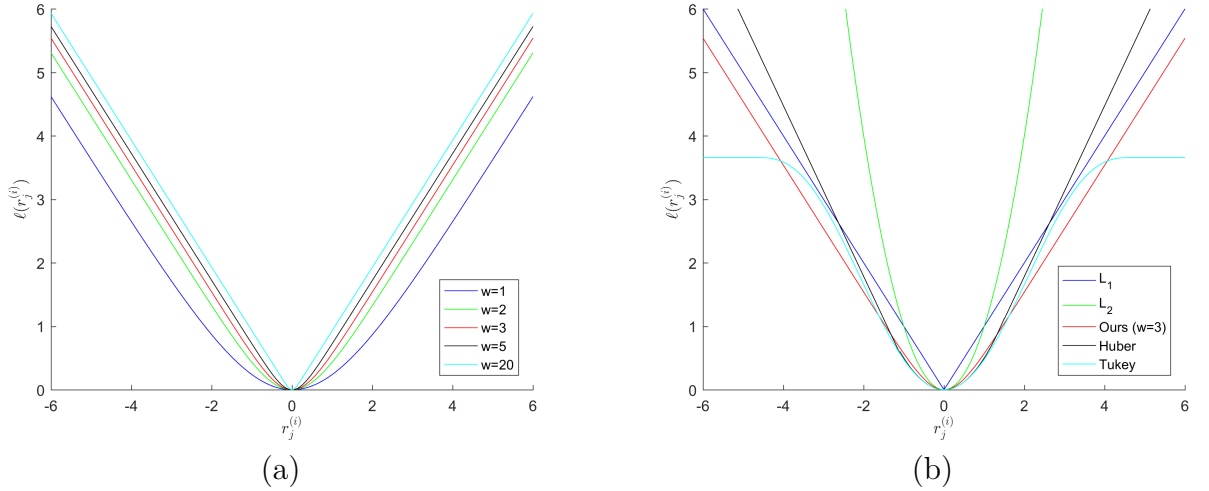


Figure 2.2: Visualization of: (a) multiple variants of our loss function with varying weighting factor ($w = \{1, 2, 3, 5, 20\}$), (b) our loss function (with $w = 3$) as well as the commonly used loss functions: L1, L2, Huber and Tukey.

Varying the weighting factor w . Through Figure 2.2.a, we can visualize multiple variants of our loss function with varying weighting factor ($w = \{1, 2, 3, 5, 20\}$). We notice that, as w increases, the loss function gets steeper and approaches the L1 loss function, thereby attributing relatively high errors to good estimated outputs (i.e., for $r_j^{(i)} \approx 0$). On the other hand, as w decreases, the loss function flattens and gives less penalty to outliers (to bad estimated output for which $r_j^{(i)}$ is very far from 0). This is confirmed by plots of partial derivatives of those variants (see Fig. 2.1.a).

2.3 Comparing our loss function to other loss functions

By looking at Table 2.1 which compares between existent loss functions and ours, this latter seems to have the following advantages: (i) unlike Huber and Tukey loss functions which are composed of 2 functions, our loss function is composed of a single function, (ii) unlike the Huber loss which has a hard threshold C imposed, our loss function offers a smoother transition at $r_j^{(i)} = C$ with no threshold required, and (iii) as opposed to the Tukey loss, our loss function offers a convex optimization that guarantees one optimal solution (globally optimal), whereas the Tukey loss has a non-convex optimization which may result in multiple locally optimal points and may take a lot of time to identify whether the solution is global. Hence, the efficiency in time of the convex optimization problem is much better when training

Table 2.1: Comparison with existent loss functions for regression.

Properties	L2	L1	Huber	Tukey	Ours
Convex	Yes	Yes	Yes	No	Yes
Differentiable	Yes	No	Yes	Yes	Yes
Robust to outliers	No	Yes	Yes	Yes	Yes
Bounded	No	No	No	Yes	No
# constraints	0	0	1	1	0

a neural network.

Chapter 3

Experimental setup

Dataset. We evaluate our loss function on the 2D human pose estimation task. To this matter, we conduct our experiments on the LSP dataset (?), a publicly available dataset. This dataset contains 2000 pose annotated images of mostly sports people gathered from Flickr. Each image has been annotated with 14 joint locations (illustrated by yellow dots in the output image of Figure 3.1) which are represented by pixel coordinates (x and y coordinates). The purpose is to train our model to estimate the 2D body skeleton as a set of joints. To do so, we assume that each individual is localized within a bounding box with normalized body pose coordinates.

Data processing. This step is similar to Belagiannis et al. (2015). Indeed, after rescaling input images to 120×80 , we normalize them by subtracting the mean image (taken from the training images) from them. Moreover, in order to prevent overfitting of the ConvNet, data augmentation is conducted by performing random rotations and flipping and by adding a small Gaussian noise to the label vector \mathbf{y} of the augmented instances. As for the label vector, it is rescaled to the range $[0, 1]$.

Training details. The ConvNet model hyper-parameters are set according to the architecture shown in Figure 3.1. Other hyper-parameters are set as follows: 0.01 for the learning rate, 0.9 for the momentum, 0.5 for the dropout, 230 for the batch size. Training of the ConvNet is performed using a 5-fold cross validation.

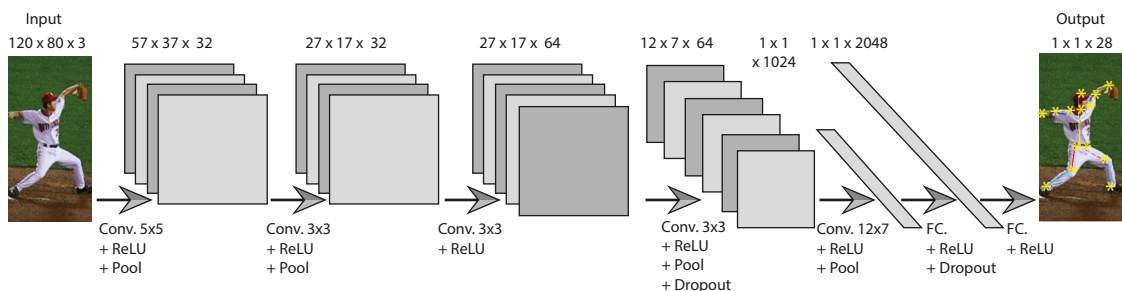


Figure 3.1: Architecture of our Convolutional Neural Network.

Performance measure. The evaluation metric used in our study is the mean

pixel error (MPE) which consists of averaging over errors of pixel coordinates of all 14 joints locations.

Chapter 4

Experiments and results

4.1 Baseline evaluation

Using the LSP dataset, we train our ConvNet with our loss function based on different values of w (weighting factor) $\{2.25, 3, 3.5, 5\}$. Figure 4.1 depicts the convergence of each ConvNet and shows that best performance is obtained when $w = 3.5$. This confirms our earliest assumption that: (i) a large w makes our loss function steeper at $r_i = 0$ and look like the L1 loss, which therefore over-penalizes small residuals and also over-penalizes large residuals (as seen in Fig. 2.2.b), and (ii) a small w flattens out our loss function. This results in under-penalizing large residuals (which is a good property for getting rid of outliers) but also in under-penalizing small residuals (which is not a good property for inliers since the network does not learn properly from these latter).

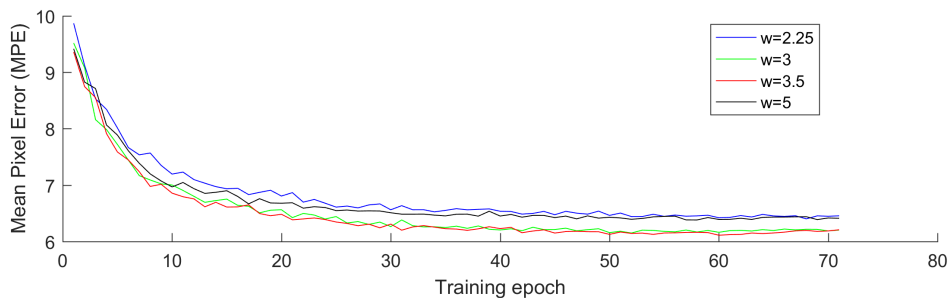


Figure 4.1: Convergence of different variants of ConvNets trained with our loss function with different values of w (weighting factor) $\{2.25, 3, 3.5, 5\}$.

4.2 Comparison with other techniques

In this section, performance of our approach based on our loss function is compared to that of other commonly used loss functions. For this purpose, we train a ConvNet using each of the following loss functions: the standard L2 loss function as well as

Table 4.1: Comparative results between the ConvNets trained on commonly used loss functions for regression as well as ours, in terms of Mean Pixel Error (MPE).

L2	Huber	Tukey Belagiannis et al. (2015)	Ours (w=3.5)
6.2720	6.3376	5.9587	6.1104

the two common robust loss functions: Huber and Tukey Belagiannis et al. (2015). Comparative results are summarized in Table 4.1. Through obtained results, we show that our approach yields a better performance than the classical approach (ConvNet with L2 loss function) and has comparable results to state-of-the-art robust deep regression techniques.

Chapter 5

Conclusion

In this part, we proposed a robust deep regression model that addresses the issue of outliers thanks to the use of a novel robust loss function. After defining our loss function, we showed that this latter has advantages over common robust loss functions. Experimental validation conducted on the human pose estimation task showed that our robust deep regression ConvNet: (i) results in a better generalization and a faster convergence than the standard deep regression, (ii) has higher or comparable performance results to ConvNets trained on robust loss functions. Finally, our robust deep regression model is simple and could be easily used for handle regression given datasets with outliers.

Bibliography

- Jeffrey D Banfield and Adrian E Raftery. Model-Based Gaussian an Non-Gaussian Clustering. *Biometrics*, pages 803–821, 1993.
- Vasileios Belagiannis, Christian Rupprecht, Gustavo Carneiro, and Nassir Navab. Robust optimization for deep regression. In *Proceedings of the IEEE International Conference on Computer Vision*, volume 2015 Inter, pages 2830–2838, 2015. ISBN 9781467383912. doi: 10.1109/ICCV.2015.324.
- Michael J. Black and Anand Rangarajan. On the unification of line processes, outlier rejection, and robust statistics with applications in early vision. *International Journal of Computer Vision*, 19(1):57–91, jul 1996. ISSN 09205691. doi: 10.1007/BF00131148.
- Pietro Corretto and Christian Hennig. Robust Improper Maximum Likelihood: Tuning, Computation, and a Comparison With Other Methods for Robust Gaussian Clustering. *Journal of the American Statistical Association*, 111(516):1648–1659, oct 2016. ISSN 0162-1459. doi: 10.1080/01621459.2015.1100996.

Tzvi Diskin, Gordana Draskovic, Frederic Pascal, and Ami Wiesel. Deep robust regression. In *IEEE 7th International Workshop on Computational Advances in Multi-Sensor Adaptive Processing (CAMSAP)*, pages 1–5, 2017.

Florence Forbes and Darren Wraith. A new family of multivariate heavy-tailed distributions with variable marginal amounts of tailweight: application to robust clustering. *Statistics and Computing*, 24(6):971–984, nov 2014. ISSN 0960-3174. doi: 10.1007/s11222-013-9414-4.

Alfia Galimzianova, Franjo Pernuš, Boštjan Likar, and Žiga Špiclin. Robust estimation of unbalanced mixture models on samples with outliers. *IEEE transactions on pattern analysis and machine intelligence*, 37(11):2273–85, 2015.

Israel Dejene Gebreu, Xavier Alameda-Pineda, Florence Forbes, and Radu Horaud. EM algorithms for weighted-data clustering with application to audio-visual scene analysis. *IEEE transactions on pattern analysis and machine intelligence*, 38(12):2402–15, 2016.

PJ Huber. *Robust statistics*. Springer Berlin Heidelberg, 2011.

Max Jaderberg, Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Reading Text in the Wild with Convolutional Neural Networks. *International Journal of Computer Vision*, 116(1):1–20, jan 2016. ISSN 0920-5691. doi: 10.1007/s11263-015-0823-z.

Stéphane Lathuilière, Pablo Mesejo, Xavier Alameda-Pineda, and Radu Horaud. DeepGUM: Learning Deep Robust Regression with a Gaussian-Uniform Mixture Model. In *Proceedings of the European Conference on Computer Vision (ECCV)*, volume 11209 LNCS, pages 202–217, 2018. ISBN 9783030012274. doi: 10.1007/978-3-030-01228-1_13.

Sijin Li and Antoni B. Chan. 3D human pose estimation from monocular images with deep convolutional neural network. In *Asian Conference on Computer Vision*, volume 9004, pages 332–347, 2014. ISBN 9783319168074. doi: 10.1007/978-3-319-16808-1_23.

RA Maronna, RD Martin, VJ Yohai, and M Salibián-Barrera. *Robust statistics: theory and methods (with R)*. Wiley, 2018.

Peter Meer, Doron Mintz, Azriel Rosenfeld, and Dong Yoon Kim. *Robust regression methods for computer vision: A review*, apr 1991. ISSN 09205691.

- Neyko Neykov, Peter Filzmoser, R Dimova, and Plamen Neytchev. Robust fitting of mixtures using the trimmed likelihood estimator. *Computational Statistics & Data Analysis*, 52(1):299–308., 2007.
- Tomas Pfister, Karen Simonyan, James Charles, and Andrew Zisserman. Deep convolutional neural networks for efficient pose estimation in gesture videos. In *Deep convolutional neural networks for efficient pose estimation in gesture videos*, volume 9003, pages 538–552, 2014. ISBN 9783319168647. doi: 10.1007/978-3-319-16865-4_35.
- Peter J Rousseeuw and Annick M Leroy. *Robust statistics for outlier detection*, volume 589. John wiley & sons, jan 2005. ISBN 1942-4787. doi: 10.1002/widm.2.
- Lamyaa Sadouk, Taoufiq Gadi, and El Hassan Essoufi. Robust Loss Function for Deep Learning Regression with Outliers. In *Advances in Intelligent Systems and Computing*, volume 1076, pages 359–368. Springer, 2020. ISBN 9789811509469. doi: 10.1007/978-981-15-0947-6_34.
- Yi Sun, Xiaogang Wang, and Xiaoou Tang. Deep Convolutional Network Cascade for Facial Point Detection. In *IEEE conference on computer vision and pattern recognition*, pages 3476–3483, 2013. doi: 10.1109/CVPR.2013.446.
- Christian Szegedy, Alexander Toshev, and Dumitru Erhan. Deep Neural Networks for Object Detection. In *Advances in Neural Information Processing Systems 26 (NIPS 2013)*, pages 2553–2561, 2013. ISBN 978-1-4799-5118-5. doi: 10.1109/CVPR.2014.276.
- Alexander Toshev and Christian Szegedy. DeepPose: Human Pose Estimation via Deep Neural Networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1653–1660, 2014. doi: 10.1109/CVPR.2014.214.

Conclusion

1.1 Conclusion

In this thesis, we explored solving different machine learning tasks with artificial and deep neural networks. As a branch of machine learning, we investigated a particular area in the design space of machine learning algorithms in which datasets are homogeneous, imbalanced, small in size (having few data) or noisy (containing outliers). The previous parts have introduced new algorithms and implementations for the classification of such datasets to increase the state-of-the-art accuracy.

In Part 2, a thorough review covering techniques in the area of deep learning is presented. It also introduces basic concepts about artificial and deep neural networks.

Part 3 addressed the issue of classifying homogeneous data especially homogeneous time-series and images. The first chapter dealt with time-series by introducing data-, and algorithm-level approaches for classification tasks. In the data-level, we proposed to turn time-series into the frequency domain using the Stockell Transform to allow the deep neural network (namely the ConvNet) to extract the most valuable features out of the time-series inputs. In the algorithm-level, we introduced a deep learning model with an adaptable first convolutional layer whose filter size varies depending on the overall input signals' variations. In the second chapter, classification of images was involved where we provided an application of ConvNets on a new task which is the Handwritten Character Tifinagh recognition, resulting in the state-of-the-art accuracy results at the time of publication.

Part 4 demonstrated ways to alleviate the problem of lack of data when training a Convolutional Neural Network. First we showed that using transfer learning with fine-tuning along with a source domain more global and general than the target domain improves the overall classification rate on the target domain task. Indeed, applying Phoenician handwritten characters as a source domain to further recognize current alphabets' characters improved the classification performance. Second, in another contribution, we applied a novel transfer learning approach based on a ConvNet pre-trained on a source domain similar or different but related to the target domain and based on an SVM classifier, resulting in state-of-the-art results on ASD

signal classification benchmarks at the time of publication. Third, we introduced an ensemble learning framework which combines ConvNets trained differently, some of which employing knowledge transfer. This framework was implemented on sketch image recognition. Finally, we defined novel augmentation and voting ConvNet approaches and chose to run simulations on sketch recognition too.

In Part 5, we tackled the problem of imbalanced data by suggesting approaches for both classification and regression tasks. Regarding classification, we introduced a cost-sensitive learning approach that can be applied on Multi-Layer Perceptrons or Convolutional Neural Networks, achieving promising performance in classifying largely imbalanced datasets. Experiments were conducted on several 1D datasets' tasks as well as the Traffic flow prediction task. As for regression tasks, we elaborated a robust learning regression approach based on a cost-sensitive learning loss function for neural networks. This approach demonstrated a faster convergence and better generalization than the classical method, and has comparable or slightly higher results than sampling techniques. Simulation were run on multiple 1D datasets. Furthermore, we also developed new evaluation strategies for regression models that handle imbalanced datasets, including scalar and graphical-based measures.

Part 6 took into account the outliers present within datasets by suggesting a robust deep regression model (a ConvNet) based on a novel robust loss function. Conducting our model on the human pose estimation task showed the superiority in performance of our model over the standard deep regression and a higher or comparable performance to ConvNets trained on the famous robust loss functions.

1.2 Towards the Future

Much remains to be yet accomplished, even in the restricted domain of deep learning . Despite our efforts toward using artificial and deep neural networks for extracting the best features out of the different types of data, the diversity of types of data still poses many challenges. More research should be conducted to answer the questions of how to learn good features from data. Furthermore, more application scenarios and problem settings are also worth investigation especially in the field of medicine in which data is scarce and presents outliers.

List of publications

We provide below a list of publications whose content was used in this thesis.

1.1 Journal papers

1. Lamyaa Sadouk, Taoufiq Gadi and El Hassan Essoufi, A Novel Deep Learning Approach for Recognizing Stereotypical Motor Movements within and across Subjects on the Autism Spectrum Disorder, *Computational Intelligence and Neuroscience*, 2018 (2018), 116.
2. Lamyaa Sadouk, Taoufiq Gadi, El Hassan Essoufi and Abdelhak Bassir, Handwritten Phoenician Character Recognition and Its Use to Improve Recognition of Handwritten Alphabets with Lack of Annotated Data, *International Journal of Advanced Trends in Computer Science and Engineering*, 9(1).1 (2020), 17181.
3. Lamyaa Sadouk, Taoufiq Gadi and El Hassan Essoufi, A Novel cost-sensitive learning approach and metric for regression in imbalanced domains, *Expert Systems*. 2021;e12680. <https://doi.org/10.1111/exsy.12680>, (2021).
4. Lamyaa Sadouk, Taoufiq Gadi, El Hassan Essoufi and Abdelhak Bassir, A Cost-Sensitive Learning Approach applied on Shallow and Deep Neural Networks for Classification of Imbalanced Data, Unpublished paper submitted to *Computational Intelligence and Neuroscience*, (2020).

1.2 Books

- Lamyaa Sadouk, CNN Approaches for Time Series Classification, in *Time Series Analysis*, 2018 .

1.3 Conference papers

1. Lamyaa Sadouk, Taoufiq Gadi and El Hassan Essoufi, Sketch Recognition Using a Hybrid Model Ensemble of Deep CNNs, in International Conference on Computing, Wireless and Communication Systems, 2016.
2. Lamyaa Sadouk, Taoufiq Gadi and El Hassan Essoufi, A Novel Approach of Deep Convolutional Neural Networks for Sketch Recognition, in Advances in Intelligent Systems and Computing, 2016, pp. 99112.
3. Lamyaa Sadouk and Taoufiq Gadi, Convolutional Neural Networks for Human Activity Recognition in Time and Frequency-Domain, Advances in Intelligent Systems and Computing, 2019, dclvi.
4. Lamyaa Sadouk, Taoufiq Gadi and El Hassan Essoufi, Handwritten Tifinagh Character Recognition Using Deep Learning Architectures, in Proceedings of the 1st International Conference on Internet of Things and Machine Learning, 2017, p. 59.
5. Lamyaa Sadouk, Taoufiq Gadi and El Hassan Essoufi, Robust Loss Function for Deep Learning Regression with Outliers, in Advances in Intelligent Systems and Computing (Springer, 2020), 35968.

Notation

1.1 List of commonly used abbreviations

Table 1.1: Notation.

Symbol	Meaning
HAR	Human Activity Recognition
ASD	Autism Spectrum Disorder
SMM	Stereotypical Motor Movements
ST	Stockwell Transform
FFT	Fast Fourier Transform
HMM	Hidden Markov Model
SGD	Stochastic Gradient Descent
SVM	Support Vector Machine
HOG	Histogram of Oriented Gradient
DBN	Deep Belief Network
SIFT	Scale-invariant Feature Transform
RBM	Restricted Boltzmann Machine
ConvNet	Convolutional Neural Network
TL	Transfer Learning
ADAM	Adaptive moment estimation (a method for stochastic optimization)
ReLU	Rectified Linear Unit
FC	Fully Connected layer
Conv	Convolutional layer
Maxpool	Maxpooling layer
ℓ	loss function
E	Expectation
N	number of samples
\mathbf{W}	weight matrix
\mathbf{x}	input vector
θ	network parameter
$\sigma(\cdot)$	sigmoid function

1.2 General math notation

Table 1.2: Notation.

Symbol	Meaning
ω	function
ℓ	loss function
E	Expectation
N	number of samples
\mathbf{W}	weight matrix
\mathbf{x}	input vector
θ	network parameter
$\sigma(\cdot)$	sigmoid function

1.3 Abbreviations for the DBN

Table 1.3: Notation.

Symbol	Meaning
\mathbf{b}	bias of the hidden units
\mathbf{c}	bias of the visible units
I	number of visible units
J	number of hidden units
\mathbf{v}	visible units
\mathbf{h}	hidden units
Z	energy partition function
α	learning rate
Δ	change (variation) of a given variable. For example ΔW_{ij} represents the weight change

Appendix

1.1 History of Deep learning

Timeline 1: Deep learning timeline

1958	● Perceptron: Foundation of NNs	(Rosenblatt, 1958)
1962	● Backpropagation: Foundation of training multi-layered NNs	(Dreyfus, 1962)
1980, 1998	● Convolutional NNs: NN architecture specialized for processing spatial data	(Fukushima, 1980; Lecun et al., 1998)
2006	● RBM Pretraining: Breakthrough allowing to train deep NNs	(Hinton et al., 2006)
2009	● Semantic Hashing: Using RBMs for fast search for similar text documents	(Salakhutdinov and Hinton, 2009)
2010	● Glorot Initialization: Scaled random initialization almost as good as pretraining	(Glorot and Bengio, 2010)
may, 2010	● mcRBM: RBM variant learning more complex features	(Marc'Aurelio Ranzato Geoffrey, 2010)
June, 2010	● Hessian-free Learning: Second-order optimization to train Rectified Linear Units deep networks without pretraining	(Martens, 2010)
2011	● Rectified Linear Units (ReLU): Nonsaturating nonlinearity helps training deep NNs	(Glorot et al., 2011)
July, 2012	● Dropout: Generic solution to reduce overfitting for deep NNs	(Hinton et al., 2012)
dec., 2012	● AlexNet: Wins object recognition challenge with deep CNN trained on raw pixels, using dropout	(Krizhevsky et al., 2012)
June, 2013	● Leaky Rectified Linear Units: ReLU variant with a nonzero gradient for negative inputs	(Maas et al., 2013)
June, 2013	● Nesterov Momentum: Demonstration that Nesterov momentum improves NN training	(Sutskever et al., 2013)
dec, 2013	● Saliency Maps: Inspect which inputs a deep NN used for a prediction	(Zeiler and Fergus, 2014)
dec, 2013	● Saxe Initialization: Random orthogonal initialization	(Saxe et al., 2013)
May, 2015	● VGG-Net: Second place in object recognition challenge, with only 3×3 convolutions	(Simonyan and Zisserman, 2015)
May, 2015	● ADAM: Optimization scheme improving over Nesterov momentum in some cases	(Kingma and Ba, 2015)
May, 2015	● Guided Backpropagation: Better interpretable saliency maps	(Springenberg et al., 2015)
dec, 2015	● He Initialization: Scaled random initialization for networks with rectified linear units	(Kaiming et al., 2015)
July, 2015	● Batch Normalization: Regularization allowing faster training of deep NNs, with reduced overfitting	(Ioffe and Szegedy, 2015)
2016	● Residual Networks: Wins object recognition challenge with network architecture allowing to train CNNs of 1000 layers	(He et al., 2016)

1.2 Details of the Code

- The project for “A Novel Deep Learning Approach for Recognizing Stereotypical Motor Movements within and across Subjects on the Autism Spectrum Disorder” can be found the URL: https://github.com/lsadouk/code_SMMs
- The Handwritten Phoenician Character database (HPCDB) is available at : <https://osf.io/4j9b6/>
- The project for “Handwritten Phoenician Character Recognition and Its Use to Improve Recognition of Handwritten Alphabets with Lack of Annotated Data” can be found the URL: https://github.com/lsadouk/Phoenician_recognition_code
- The project for “A Novel cost-sensitive learning approach and metric for regression in imbalanced domains” can be found the URL: https://github.com/lsadouk/imbalanced_regression
- The project for “A Cost-Sensitive Learning Approach applied on Shallow and Deep Neural Networks for Classification of Imbalanced Data” can be found the URL: https://github.com/lsadouk/imbalanced_classification

1.3 Supplementary material

1.3.1 Supplementary material for Part II

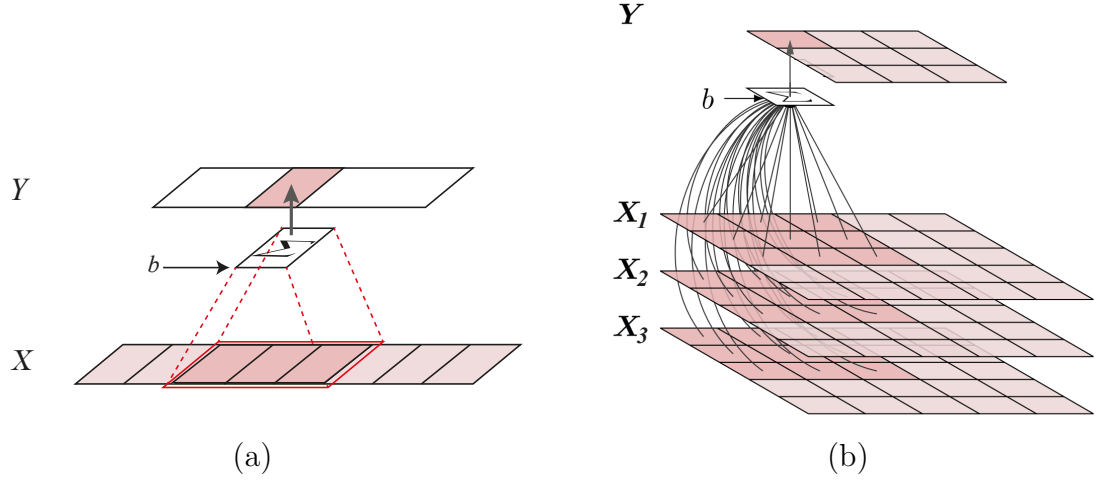


Figure 1.1: Visualization of convolution applied on a 1D input (a) and a 3D input (b).

1.3.2 Supplementary material for Chapter 1

Table 1.4: A summary of the Phoenician ConvNet architecture parameters.

Id	Layer	Type	Filter size	Filter Num	Stride	Pad	Output Size	#Parameters
0		Input	-	-	-	-	60×60	0
1		Conv	5×5	20	1	0	56×56	520
2	L1	ReLu	-	-	-	-	56×56	0
3		Maxpool	2×2	-	2	0	28×28	0
4		Conv	4×4	50	1	0	25×25	16,050
5	L2	ReLu	-	-	-	-	25×25	0
6		Maxpool	2×2	-	2	0	12×12	0
7		Conv	3×3	150	1	1	12×12	67,650
8	L3	ReLu	-	-	-	-	12×12	0
9		Conv	3×3	150	1	1	12×12	202,650
10	L4	ReLu	-	-	-	-	12×12	0
11		Maxpool	2×2	-	2	0	6×6	0
12		Conv (FC1)	6×6	500	1	0	1×1	2,700,500
13	L5	ReLu	-	-	-	-	1×1	0
14		Dropout	-	-	-	-	1×1	0
15		Conv (FC2)	1×1	500	1	0	1×1	250,500
16	L6	ReLu	-	-	-	-	1×1	0
17		Dropout	-	-	-	-	1×1	0
18	L7	Conv (FC3)	1×1	33	1	0	1×1	11,022

Table 1.5: The confusion matrix for the recognition of each Phoenician character.

	𐤀	𐤁	𐤂	𐤃	𐤄	𐤅	𐤆	𐤇	𐤈	𐤉	𐤊	𐤋	𐤌	𐤍	𐤎	𐤏	𐤐	𐤑	𐤒	𐤓	𐤔	
𐤀	162																					
𐤁		159									1								2			
𐤂		1	156														5					
𐤃				158			1				1					1			1			
𐤄		1			159					1					1							
𐤅						162																
𐤆							162															
𐤇								160							1		1					
𐤈		1			2			1	157						1							
𐤉										161												1
𐤊	1										161											
𐤋												158	3									1
𐤌												1	161									
𐤍	1						1			1					158				1			
𐤎																162						
𐤏			1											1			160					
𐤐																		160	1			1
𐤑																1				159		
𐤒		8	2	1																	151	
𐤓												1										161
𐤔																						
𐤕																						162

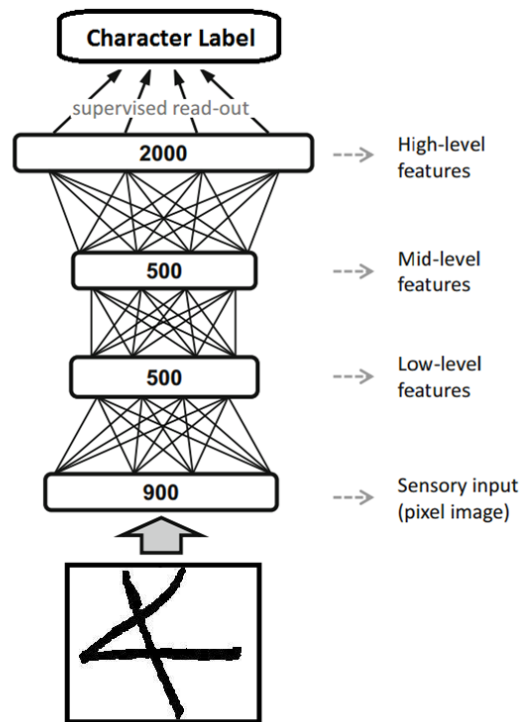


Figure 1.2: Graphical representation of the DBN model for handwritten Phoenician character recognition. Undirected and directed connections stand for unsupervised and supervised learning respectively. Numbers on each layer denote the size of that layer.

Table 1.6: Learning parameters of the DBN. Let's note that learning parameters are set according to suggestions published by Hinton (Hinton et al., 2006).

Phase	Learning rate	Momentum	Dropout	Number of epochs	Batch size
Unsupervised phase	0.0001	0.7	0.3	200	200
Supervised phase	0.0001	0.9		100	200

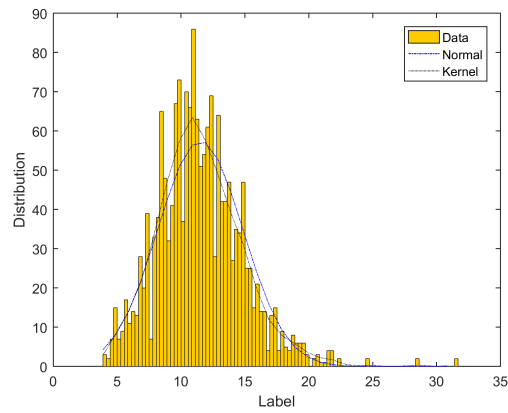
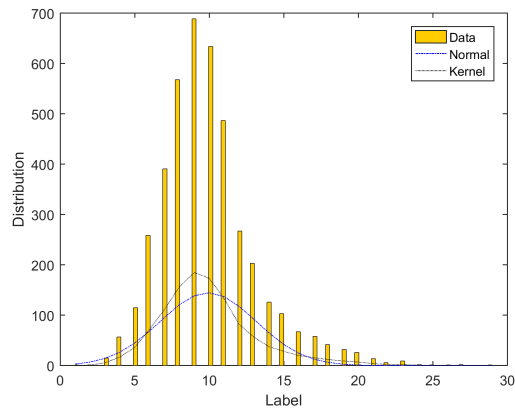
1.3.3 Supplementary material for Chapter 2

Datasets

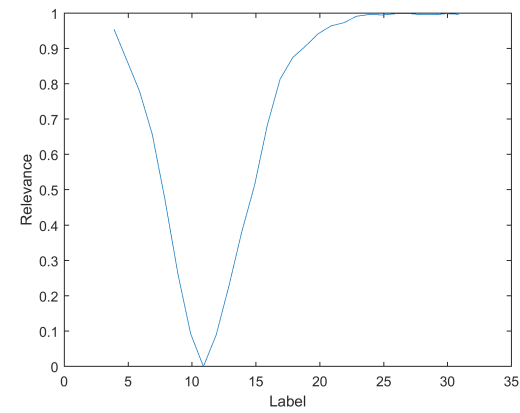
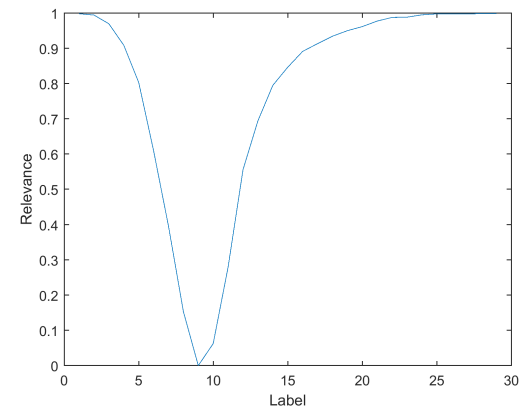
abalone

accel

(a)



(b)



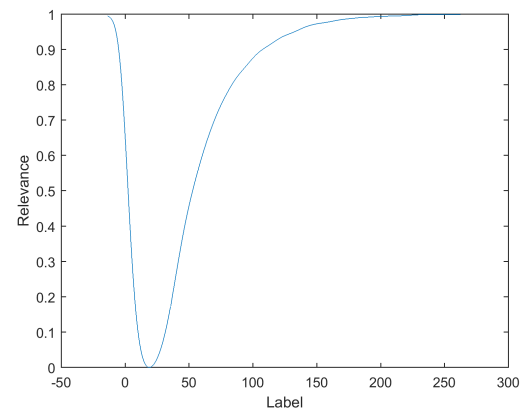
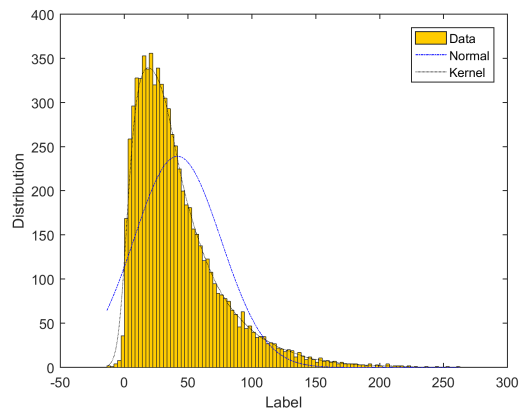
Plot (a): histograms of the target variable distribution, NDF (blue plot) and KDE (black plot) of each dataset. Plot (b): KDR of each dataset.

Datasets

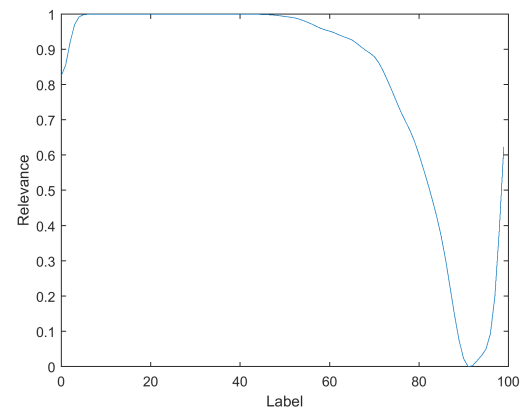
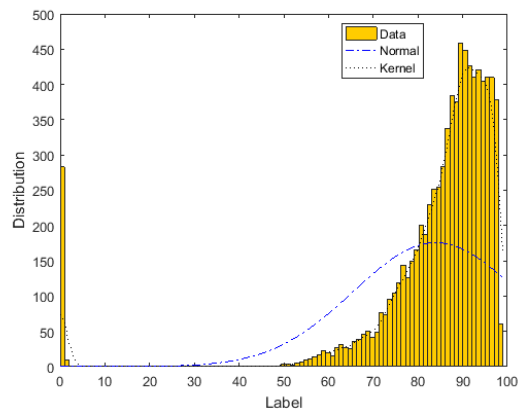
(a)

(b)

heat



cpuSM



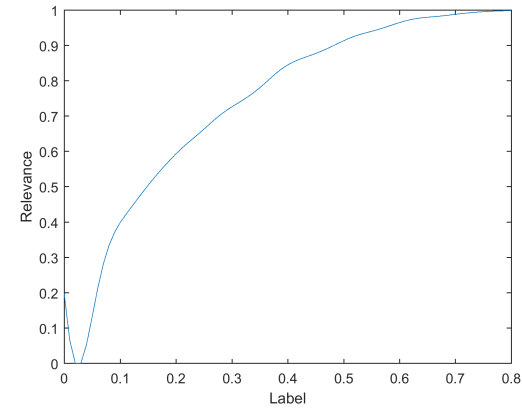
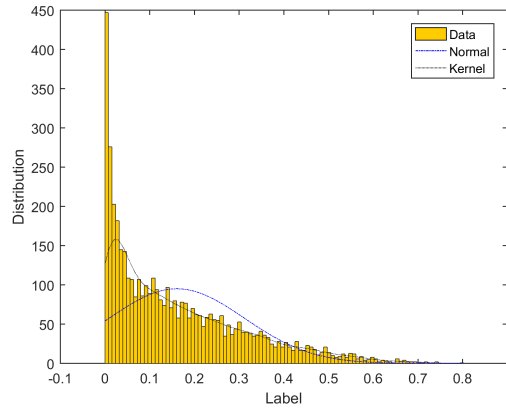
Plot (a): histograms of the target variable distribution, NDF (blue plot) and KDE (black plot) of each dataset. Plot (b): KDR of each dataset.

Datasets

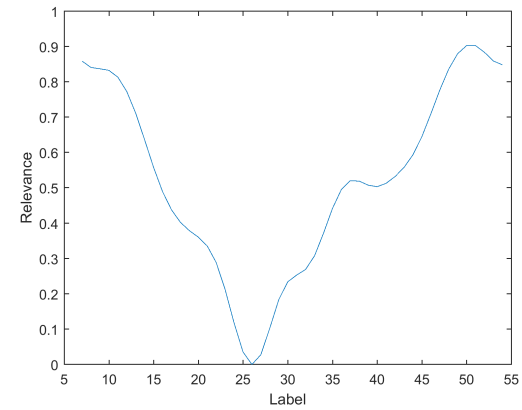
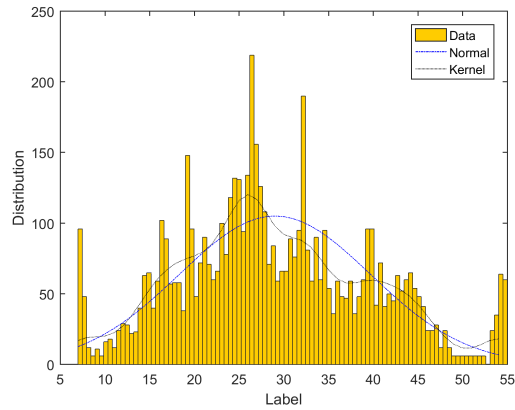
(a)

(b)

bank8FM



parkinson



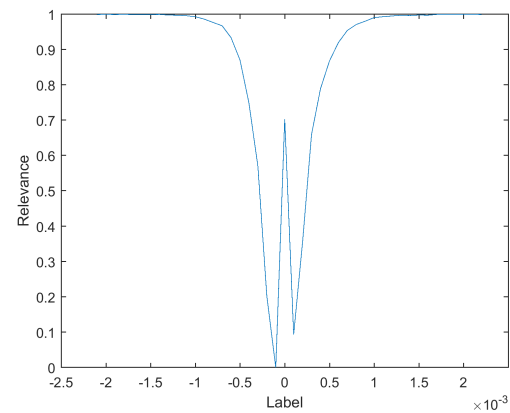
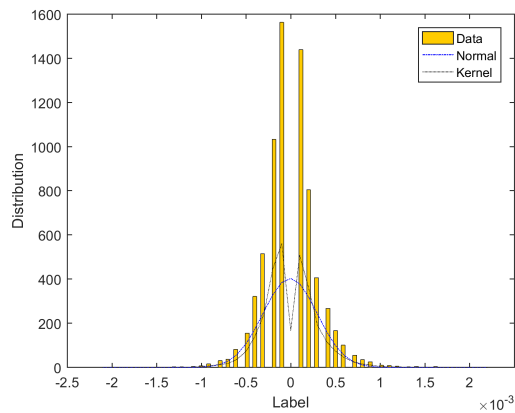
Plot (a): histograms of the target variable distribution, NDF (blue plot) and KDE (black plot) of each dataset. Plot (b): KDR of each dataset.

Datasets

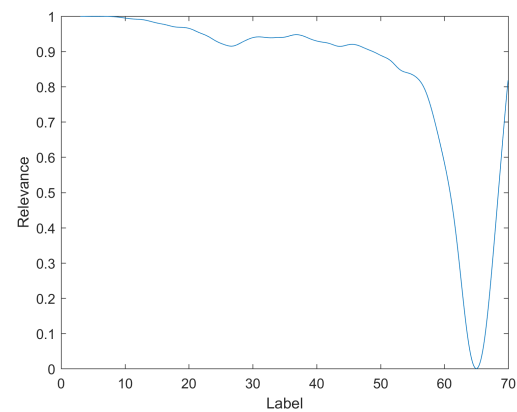
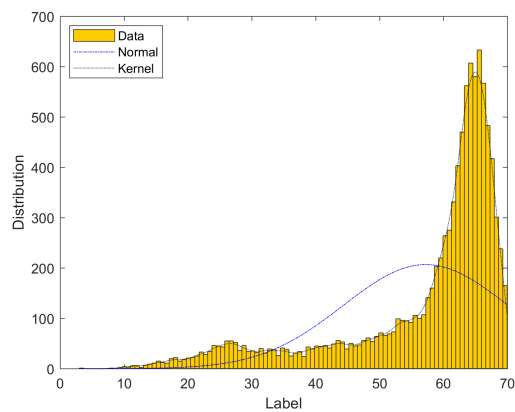
(a)

(b)

dAiler



H101_North_D7



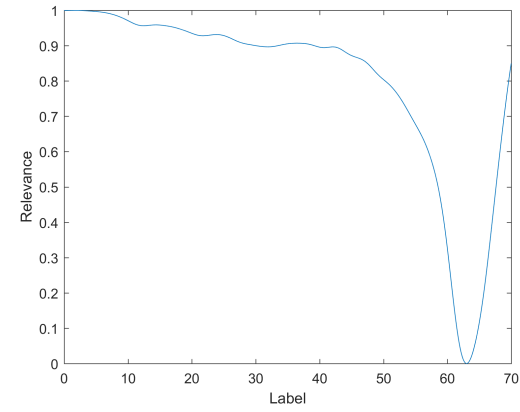
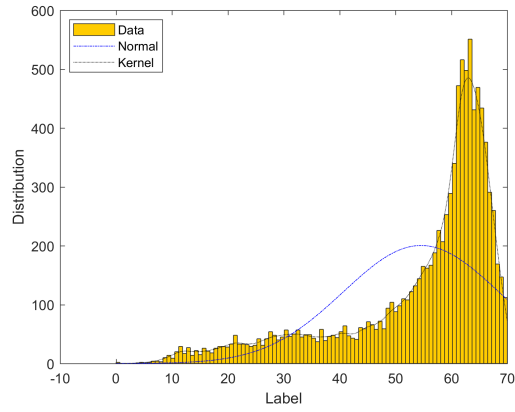
Plot (a): histograms of the target variable distribution, NDF (blue plot) and KDE (black plot) of each dataset. Plot (b): KDR of each dataset.

Datasets

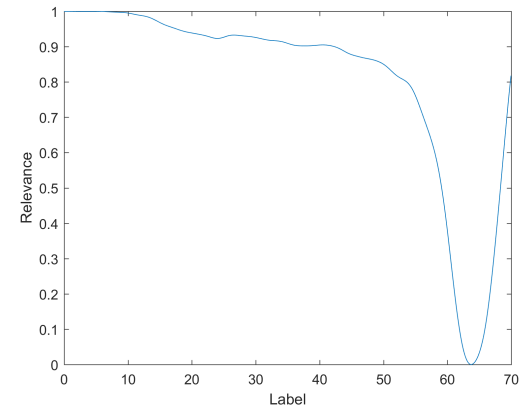
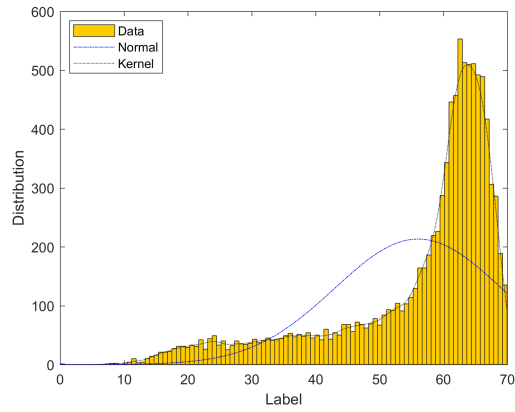
(a)

(b)

I5_South_D7



I5_North_D7

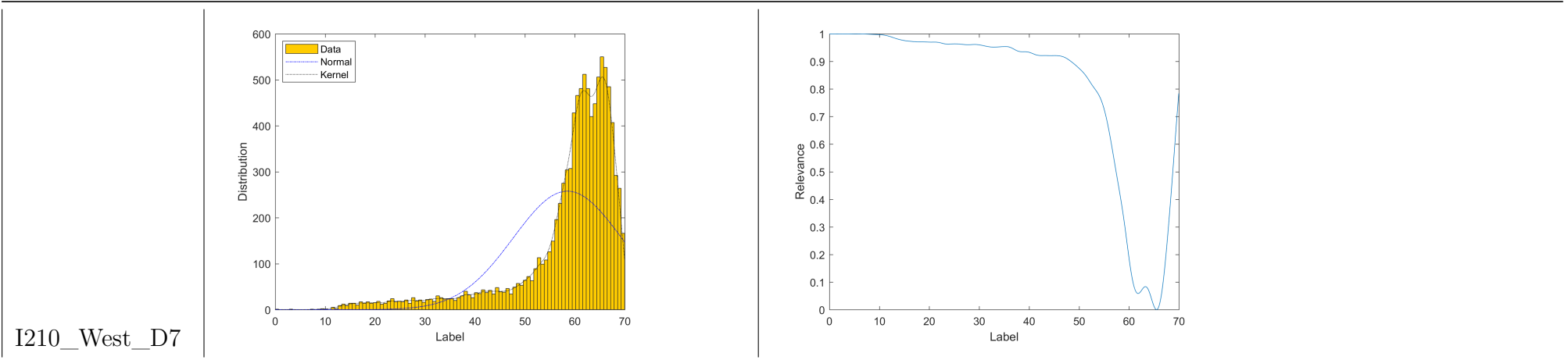


Plot (a): histograms of the target variable distribution, NDF (blue plot) and KDE (black plot) of each dataset. Plot (b): KDR of each dataset.

Datasets

(a)

(b)



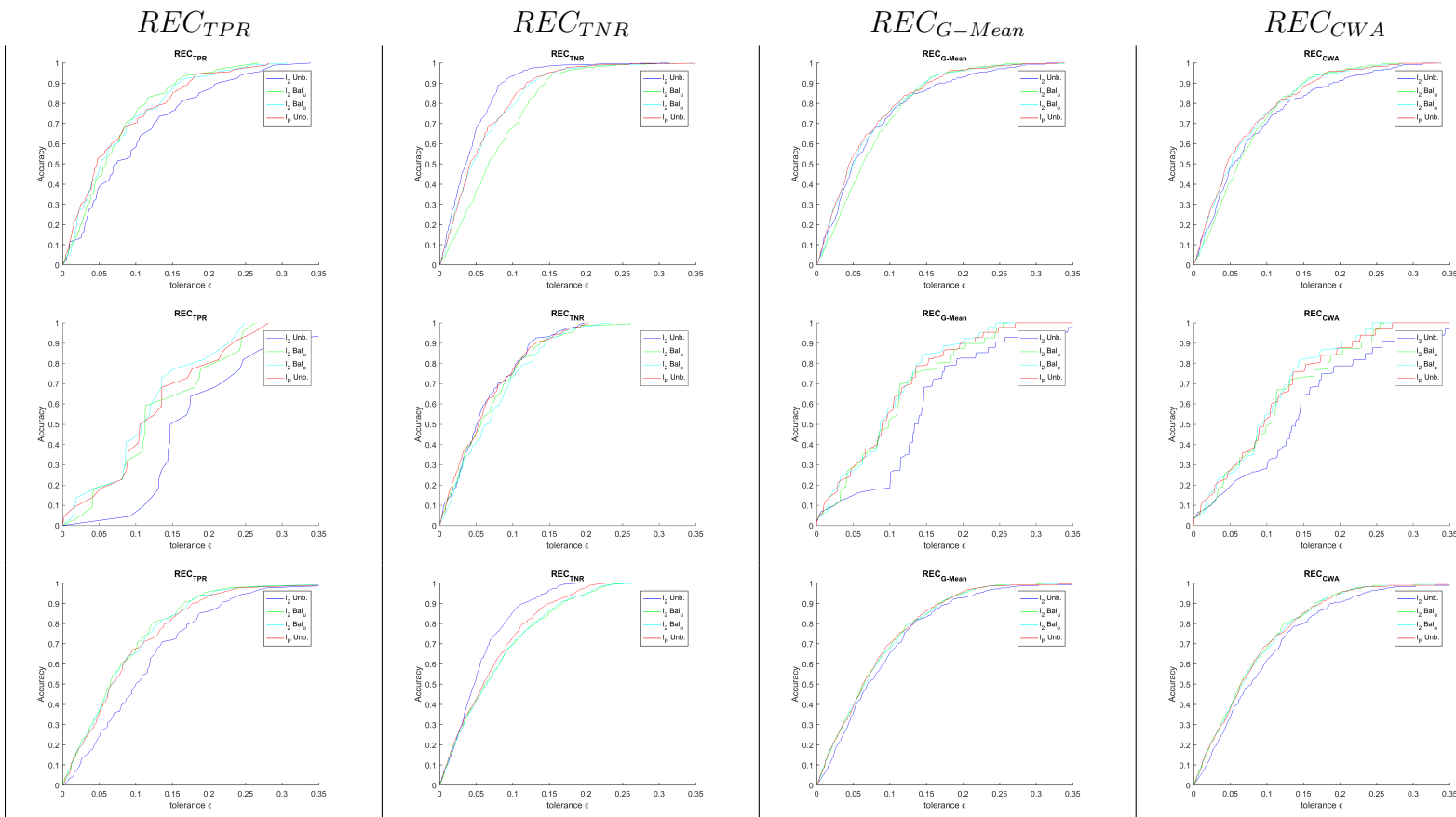
Plot (a): histograms of the target variable distribution, NDF (blue plot) and KDE (black plot) of each dataset. Plot (b): KDR of each dataset.

Datasets

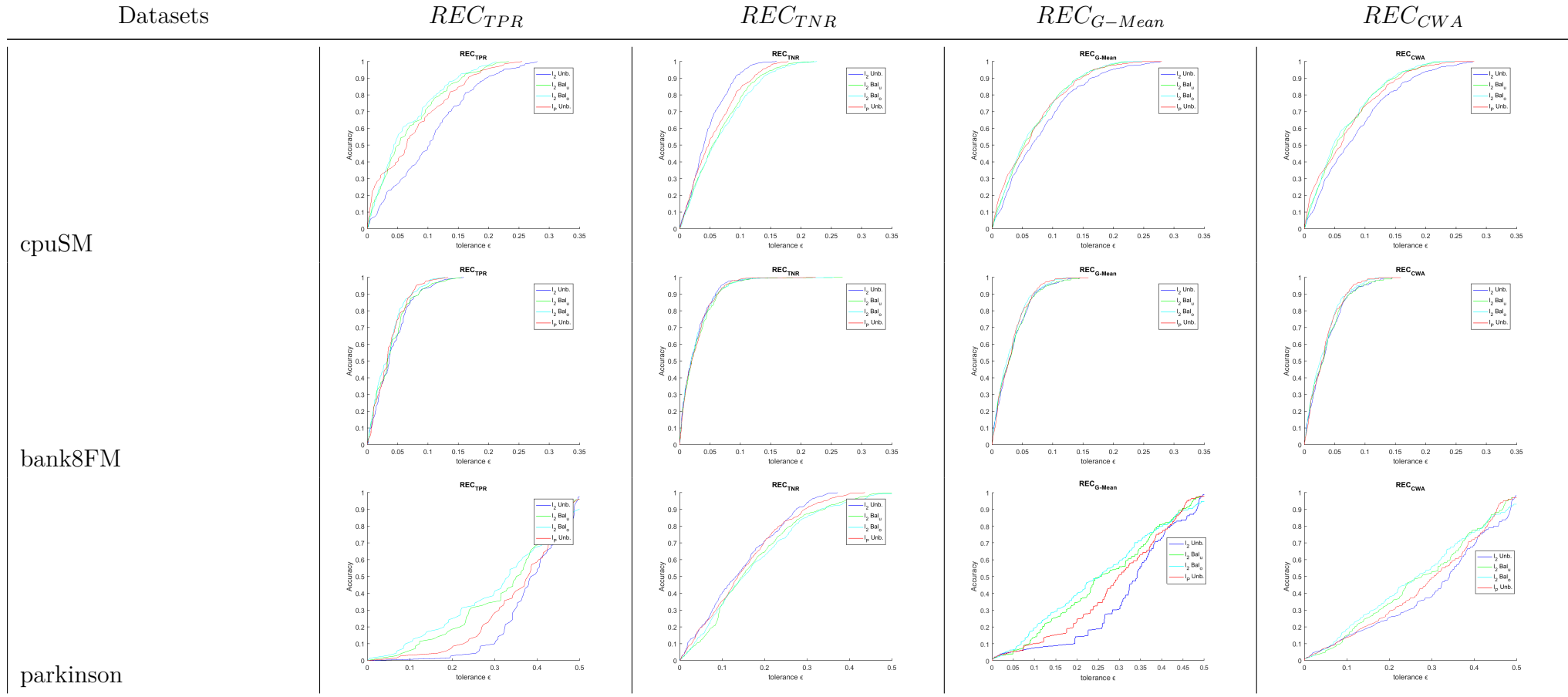
abalone

accel

heat



REC curves REC_{TPR} , REC_{TNR} , REC_{G-Mean} and REC_{CWA} at test time for experiments $l_2Unb.$, l_2Bal_u , l_2Bal_o , $l_pUnb.$ on the 11 datasets.



REC curves REC_{TPR} , REC_{TNR} , REC_{G-Mean} and REC_{CWA} at test time for experiments $l_2Unb.$, l_2Bal_u , l_2Bal_o , $l_pUnb.$ on the 11 datasets.

Datasets

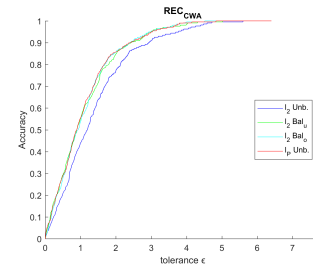
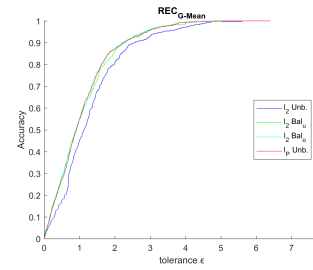
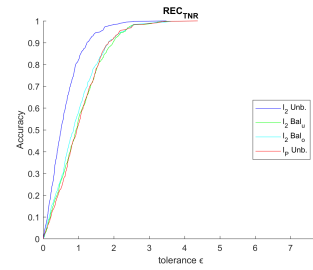
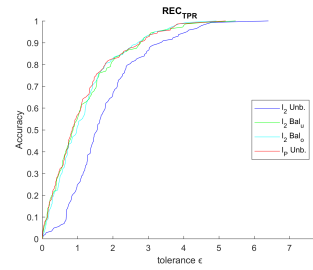
REC_{TPR}

REC_{TNR}

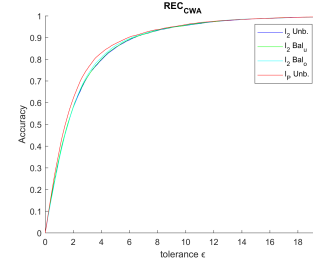
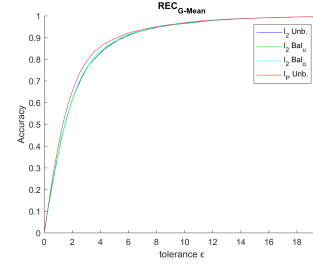
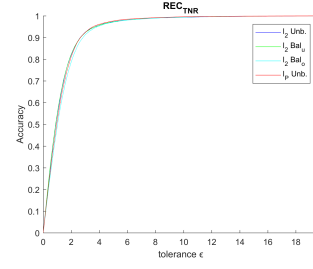
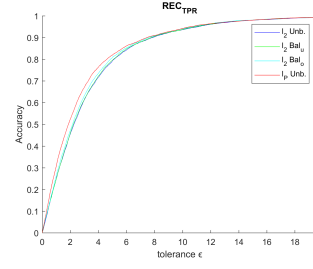
REC_{G-Mean}

REC_{CWA}

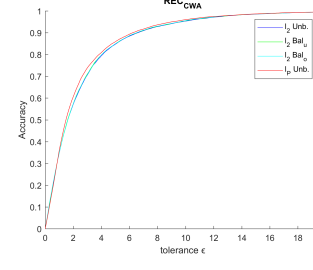
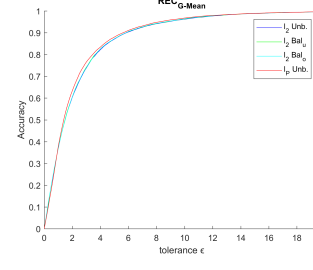
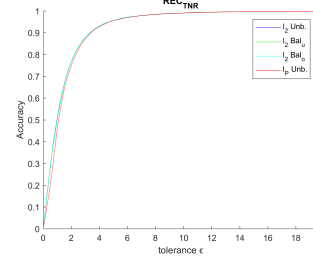
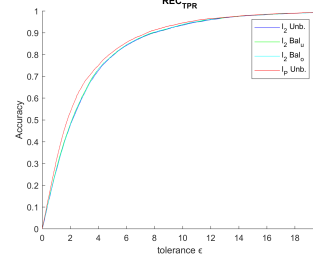
dAiler



H101_North_D7



I5_South_D7



REC curves REC_{TPR} , REC_{TNR} , REC_{G-Mean} and REC_{CWA} at test time for experiments $l_2Unb.$, l_2Bal_u , l_2Bal_o , $l_pUnb.$ on the 11 datasets.

Datasets

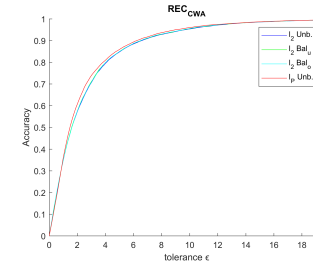
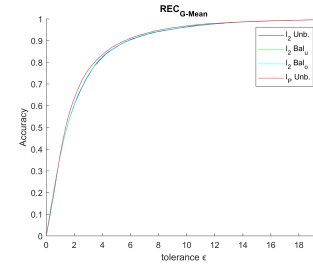
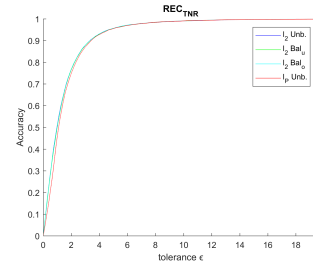
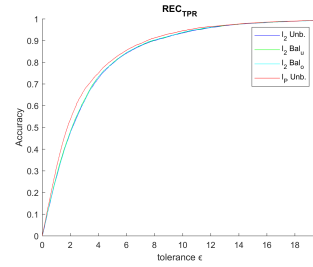
REC_{TPR}

REC_{TNR}

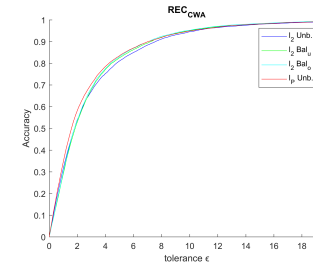
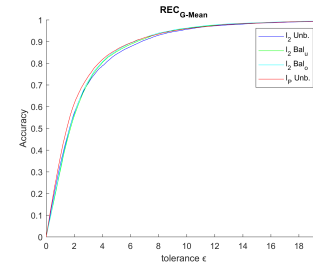
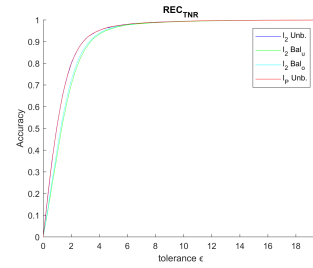
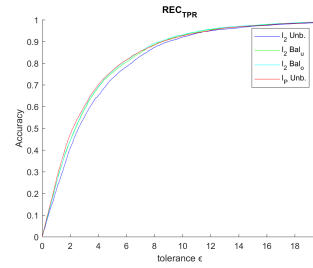
REC_{G-Mean}

REC_{CWA}

I5_North_D7



I210_West_D7



REC curves REC_{TPR} , REC_{TNR} , REC_{G-Mean} and REC_{CWA} at test time for experiments $l_2Unb.$, l_2Bal_u , l_2Bal_o , $l_pUnb.$ on the 11 datasets.

Bibliography

- Stuart Dreyfus. The numerical solution of variational problems. *Journal of Mathematical Analysis and Applications*, 5(1):30–45, 1962. ISSN 10960813. doi: 10.1016/0022-247X(62)90004-5.
- Kunihiko Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36(4):193–202, apr 1980. ISSN 0340-1200. doi: 10.1007/BF00344251.
- Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *13th International Conference on Artificial Intelligence and Statistics*, volume 9, pages 249–256, 2010. ISBN 9781937284275. doi: 10.1.1.207.2059.
- Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep Sparse Rectifier Neural Networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 315–323, 2011. ISBN 1532-4435. doi: 10.1.1.208.6449.
- Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. In *Proceedings of the 29th IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016. ISBN 978-1-4673-8851-1. doi: 10.1109/CVPR.2016.90.
- Geoffrey E. Hinton, Simon Osindero, and Yee-Whye Teh. A Fast Learning Algorithm for Deep Belief Nets. *Neural Computation*, 18(7):1527–1554, jul 2006. ISSN 0899-7667. doi: 10.1162/neco.2006.18.7.1527.
- Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arxiv.org*, 2012. ISSN 9781467394673. doi: arXiv:1207.0580.
- Sergey Ioffe and Christian Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *The 32nd International Conference on Machine Learning (ICML)*, pages 448–456, feb 2015. ISBN 1532-4435. doi: 10.1.1.208.6449.
- He Kaiming, Zhang Xiangyu, Ren Shaoqing, and Sun Jian. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. In *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV)*, volume 498, pages 1026–1034, 2015. ISBN 9781467383912. doi: 10.1016/j.bbrc.2018.01.076.

- Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. In Proceedings of the 3rd International Conference on Learning Representations (ICLR), dec 2015. ISBN 9781450300728. doi: 10.1063/1.4902458.
- Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In Advances in neural information processing systems, pages 1097–1105, 2012. ISBN 9780415468442. doi: 10.1061/(ASCE)GT.1943-5606.0001284.
- Yann Lecun, Leon Bottou, Yoshua Bengio, and Patrick Ha. Gradient-Based Learning Applied to Document Recognition. In Proceedings of the IEEE 86.11, pages 2278–2324, 1998.
- Andrew L. Maas, Awni Y. Hannun, and Andrew Y Ng. Rectifier nonlinearities improve neural network acoustic models. In Proceedings of the 30th International Conference on Machine Learning (ICML), pages 315–323, 2013. ISBN 0162-4962. doi: 10.1016/0010-0277(84)90022-2.
- E.H. Marc’Aurelio Ranzato Geoffrey. Modeling Pixel Means and Covariances Modeling Pixel Means and Covariances Using Factorized Third-Order Boltzmann Machines. IEEE, 2010.
- James Martens. Deep learning via Hessian-free optimization. In ICML, volume 27, pages 735–742, 2010. ISBN 9781605589077. doi: 10.1155/2011/176802.
- Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. Psychological Review, 65(6):386, 1958.
- Ruslan Salakhutdinov and Geoffrey Hinton. Semantic hashing. International Journal of Approximate Reasoning, 50(7):969–978, 2009. ISSN 0888613X. doi: 10.1016/j.ijar.2008.11.006.
- Andrew M. Saxe, James L. McClelland, and Surya Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. arXiv preprint arXiv:1312.612, dec 2013.
- Karen Simonyan and Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. In Proceedings of the 3rd International Conference on Learning Representations, sep 2015. ISBN 1097-0142 (Electronic)\n0008-543X (Linking). doi: 10.2146/ajhp170251.
- Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. Striving for Simplicity: The All Convolutional Net. Inorkshop of the

3rd International Conference on Learning Representations (ICLR), 2015. ISBN 9781600066634. doi: 10.1163/_q3_SIM_00374.

Ilya Sutskever, James Martens, George E Dahl, and Geoffrey E Hinton. On the importance of initialization and momentum in deep learning. In Proceedings of the 30th International Conference on Machine Learning (ICML), volume 28, pages 1139–1147, 2013. ISBN 978-1-4799-0356-6. doi: 10.1109/ICASSP.2013.6639346.

Matthew Zeiler and Rob Fergus. Visualizing and Understanding Convolutional Networks. In European conference on computer vision, pages 818–833, 2014. ISBN 978-3-319-10590-1. doi: 10.1111/j.1475-4932.1954.tb03086.x.

List of Figures

1.1	Overview of Deep Learning techniques.	18
1.2	Example of: (a) a Feedforward neural network and (b) a Feedback neural network.	20
2.1	Illustration of a convolutional units with (a) a 1D input and (b) a 3D input.	29
2.2	Typical activation functions for hidden layers with plots (a), (b) and (c) corresponding to <i>Sigmoid</i> , <i>Tanh</i> and <i>ReLU</i> activation units. . .	31
2.3	Computational graph of the minimization of the objective function $L(\mathbf{y}^i, \hat{\mathbf{y}}^i)$, divided into two components: $a_n = g(\mathbf{x}^i, \theta)$ and $\ell(\mathbf{y}^i, \hat{\mathbf{y}}^i)$. . .	35
2.4	Visualization of: (a) a gradient descent with a large learning rate and, (b) a gradient descent with a small learning rate.	39
2.5	Nesterov momentum. Instead of evaluating the gradient at the current position (red circle), we know that our momentum is about to carry us to the tip of the green arrow. With Nesterov momentum we therefore instead evaluate the gradient at this “looked-ahead” position.	41
2.6	Visualization of: (a) a biological neuron and, (b) an artificial one (single-layer perceptron) $\phi(\mathbf{b} + \mathbf{W}^T \mathbf{x})$	43
2.7	Visualization of $\phi(\mathbf{b} + \mathbf{W}^T \mathbf{x})$	44
2.8	Visualization of $\phi_2(\mathbf{b}_2 + \mathbf{W}_2^T \phi_1(\mathbf{b}_1 + \mathbf{W}_1^T \mathbf{x}))$	45
2.9	Visualization of max-pooling	47
2.10	A simple ConvNet architecture with one set of convolutional, max-pooling and fully connected layers	48
3.1	Schematic representation of a Restricted Boltzmann Machine (RBM).	56
3.2	Markov chain Monte Carlo using alternating Gibbs sampling in a restricted Boltzmann machine (RBM). The chain is initialized with the data input vector \mathbf{x}	59
3.3	Structure of a DBN with three hidden layers.	61

3.4	DBN training with one input layer \mathbf{x} , and three hidden layers $\mathbf{h1}$, $\mathbf{h2}$, $\mathbf{h3}$. From left to right, purple color represents layers already trained, while cyan the RBM being trained.	61
1.1	The overall frequency-domain ConvNet architecture. Symbols “C”, “S”, “U” in the parentheses of the layer tags refer to convolution, subsampling and unification operations respectively. Numbers before and after “@” refer to the feature map size. Note that <i>ReLU</i> layers located after <i>C1</i> , <i>C2</i> and <i>F1</i> are not showed due to the limitation of space.	80
1.2	Plots (a) and (b) display time domain acceleration signal samples taken from the SMM dataset. The red, green and blue curves stand for x, y and z signals respectively. The size of samples is 90×1 (e.g., 1 second long).	81
1.3	Plots (a) and (b) represent frequency domain acceleration signal samples taken from the SMM dataset. The red, green and blue curves stand for x, y and z signals respectively. The size of samples is 50×1 (e.g., 50 frequency points).	82
1.4	Figures (a) and (b) show histograms and box plots which represent the frequency distribution of 30 peak lengths present within 30 randomly selected time domain signals and 30 randomly selected frequency domain signals respectively.	83
1.5	Plots (a) and (b) display weights of two filters within 1 st convolutional layer of the frequency-domain ConvNet model. The red, green and blue curves stand for weights of x, y and z signals respectively.	86
1.6	Plots (a) and (b) represent weights of two filters contained in the 1st convolutional layer of the time-domain ConvNet model. The red, green and blue curves represent weights of x, y and z signals respectively.	87
1.7	Effect of the size of 1 st convolutional layer kernel on SMM recognition performance.	88
2.1	Filter weights on the first layer for the three ConvNet architectures: (a) 4 by 4 filter weights for the architecture adopted for 30×30 images, (b) 5 by 5 filter weights for the architecture adopted for 60×60 images, (c) 7 by 7 filter weights for the architecture adopted for 100×100 images.	104

2.2	Structure of the chosen DBN architecture (500-500-2000-31). Undirected connections stand for unsupervised learning, whereas directed arrows on the top layer entail supervised learning. The number on each layer denotes the layer size.	104
2.3	20 sample weights taken from the units of the 1 st hidden layer for three DBN architectures. Each sample is a 900 raw vector of a single 1st hidden unit sample that is reshaped into a 30×30 matrix. (a) weights for the 500-500-2000-31 architecture, (b) weights for the 500-1000-2000-31 architecture, (c) weights for the 1000-1000-2000-31 architecture.	106
2.4	Some samples of the character \odot (class=23) badly written in the database.	107
1.1	Phoenician text inscribed on stone, 1st millennium BCE.	118
1.2	(a) consists of the two forms (“A” and “B”) given to writers, where each form has a different style of alphabet writing. (b) is an example of two sample forms filled by two writers.	120
1.3	Example of images of character ‘Kaph’ K stored in our database.	121
1.4	Architecture of the proposed ConvNet.	122
1.5	Examples of badly written characters in our database.	123
1.6	Some existing alphabets.	125
1.7	Classification results of the RI ConvNet, TL using Phoenician ConvNet, TL using Latin ConvNet, TL using Digits ConvNet, and TL using Cifar ConvNet techniques as a function of the number of training instances n per target dataset.	128
2.1	The overall transfer learning framework. Abbreviations “conv.” and “FC” stand for convolutional and fully connected layers respectively.	135
3.1	(a) Examples of hand-sketched objects, and (b) three samples of a stickman object drawn with different levels of abstraction.	146
3.2	Ensemble learning framework at test time using either direct output vectors $\hat{\mathbf{y}}_1, \dots, \hat{\mathbf{y}}_n$ or output probabilities $\mathbf{p}_1, \dots, \mathbf{p}_n$ coming out the softmax layer (the softmax being represented by dashed boxes).	149
3.3	Overall steps of our experiment.	155

3.4	Visualization of the learned filters in the first layer of each model. Left: first layer filters of our ConvNet model from scratch (64 filters in total, each having a size of 15 by 15 pixels). Middle: first layer filters of our fine-tuned model of VGG-f (64 filters in total, each having a size of 11 by 11 pixels). Right: first layer filters of our fine-tuned model of VGG-deep (64 filters in total, each having a size of 3 by 3 pixels). We can see the similarities and differences between the three models.	156
4.1	Illustration of the overall steps within our deep learning framework.	166
4.2	(a) Original sample of an airplane object, (b) Transformed samples of the airplane generated via our skewing augmentation technique. .	167
1.1	Plots (a) and (b) represent different loss functions at an instance i with respect to the network output \hat{y}_k^i when $y_k^i = 1$ and $y_k^i = 0$ respectively. $\sigma(\cdot)$ denotes the sigmoid function. Curves that may not appear in plots are equal to 0.	186
1.2	Plots in (a) and (b) represent respectively standard and cost-sensitive partial derivatives of different loss functions with respect to \hat{y}_k^i at instance i for $y_k^i = 1$. Instance i is chosen to have a high relevance $\phi(m^i) = 0.9$, and τ is set to 1.	194
2.1	Plot (a) is a histogram that represents the “cpuSM” target distribution, grouped into bins of interval 0.01. Plots (b) and (c) are the NDF $f(\cdot)$ and the NDR function $\phi(\cdot)$ of the same distribution. . . .	214
2.2	Plot (a) and (b) are the KDE $f(\cdot)$ and the KDR function $\phi(\cdot)$ of the same target distribution of the “cpuSm” dataset.	216
2.3	Red and blue plots represent the value of ℓ and ℓ_p respectively with respect to the residual r^i (x-axis) for two scenarios: a rare event i having a low CPU performance label $y^i = 0.3$ (a), and a frequency event having a high CPU performance label $y^i = 0.9$ (b).	219
2.4	Blue and red plots represent the derivatives of ℓ and ℓ_p functions respectively with respect to residual r^i for two scenarios: an instance i with a rare label $y^i = 0.3$ (a), and another instance i with a frequent label $y^i = 0.9$ (b).	220

2.1	Visualization of: (a) multiple variants of the partial derivative of our loss function with respect to the residual with varying weighting factor ($w = \{1, 2, 3, 5, 20\}$), (b) partial derivatives (with respect to the residual) of some commonly used loss functions: L1, L2, Huber (with $c = 1.345$) and Tukey (with $c = 4.685$) as well as ours (with $w = 3$).	256
2.2	Visualization of: (a) multiple variants of our loss function with varying weighting factor ($w = \{1, 2, 3, 5, 20\}$), (b) our loss function (with $w = 3$) as well as the commonly used loss functions: L1, L2, Huber and Tukey.	257
3.1	Architecture of our Convolutional Neural Network.	259
4.1	Convergence of different variants of ConvNets trained with our loss function with different values of w (weighting factor) $\{2.25, 3, 3.5, 5\}$.	261
1.1	Visualization of convolution applied on a 1D input (a) and a 3D input (b).	275
1.2	Graphical representation of the DBN model for handwritten Phoenician character recognition. Undirected and directed connections stand for unsupervised and supervised learning respectively. Numbers on each layer denote the size of that layer.	276

List of Tables

2.1	List of loss functions for classification. \mathbf{y}^i and $\hat{\mathbf{y}}^i$ are the true label and network output vectors respectively. \cdot_k denotes the k th dimension (element) of a given vector, and $\sigma(\cdot)$ denotes a probability estimate (softmax function or the sigmoid function).	33
1.1	Number of instances (N) per subject per study after data extraction in time and frequency domains.	79
1.2	Performance rates of time-, and frequency-domain ConvNets for the SMM recognition (in terms of F1-score, denoted as “F1 sc.”) and Human Activity Recognition referred to as “HAR” (in terms of accuracy). Highest rates are in bold.	85
1.3	Results of our models and other models on 6 subjects of Study1 and Study2 datasets. For each study, two to three video sessions were performed per subject except for Subject6 (in Study2) who had only one session recorded; therefore, experiments could not be performed, which is indicated by “_”.	88
2.1	ConvNets architecture.	101
2.2	Comparative results using different Transfer Learning ConvNets. . .	103
2.3	Results of different DBN architectures.	105
2.4	Individual recognition rate for each character for both DBN (architecture 1000-1000-2000) and ConvNet models.	108
2.5	Comparison between our method and other existing approaches. Images used are taken from AMHCD. The abbreviation % stands for the percentage with respect to the total size of images used.	109
1.1	Phoenician alphabet with their correspondents (names and values) in Latin Characters.	119
1.2	Classification rate of each character in the PHCDB database during the validation phase.	123
1.3	Comparison of different used classifiers.	124

1.4	Comparative results using different Transfer Learning ConvNets. . .	127
2.1	Results of our transfer learning approach as well as other ConvNet approaches per domain (time or frequency) per subject and per study.	138
2.2	Characteristics and resources used for the different techniques implemented in experiments.	138
3.1	Training hyper-parameters of ConvNet models. The abbreviations 'LR' and 'Mom.' stand for the learning rate and the momentum respectively.	152
3.2	Architecture of the ConvNet model trained from scratch.	153
3.3	Results of the hybrid model using different model ensemble architectures, i.e. different methods of combining models at test time. . . .	157
3.4	Comparison of different sketch classification methods as well as the human recognition performance.	158
4.1	Results of our trained model at test time using different characteristics. For each image, we perform rotation and rescaling to get n variants of the image. Then, these images are fed into our network, producing n outputs. After that, we see which output has the highest number of votes. Finally, this output is compared to the true label of the image.	169
4.2	Comparison of different sketch classification methods as well as the human recognition performance.	170
1.1	Characteristics of the 1D datasets.	197
1.2	Training hyper-parameters for the 1D datasets in (a) and the 2D datasets in (b). $[n1, n2, n3]$ defines the architecture of the MLP for the 1D datasets, with $n1$, $n2$ and $n3$ denoting the number of neurons in the first, second and third layer respectively. In (b), the architecture of the ConvNet is defined by the layer type, its kernel size, stride, and depth.	198
a	MLP training hyper-parameters	198
b	ConvNet training hyper-parameters	198
1.3	Mean classification results of neural networks over 3 runs using the standard version (denoted as "Std.") and the cost-sensitive version of different loss functions in terms of average values of g-mean (in %). Best rates per loss function and per dataset are in bold.	200
1.4	Comparative results between different methods in terms of the G-mean performance metric (in %).	202

2.1	Confusion matrix.	221
2.2	A description of the 7 datasets (N : number of instances; $p.total$: number of features in the input; $p.nom$: number of nominal features; $p.num$: number of numeric features; $nRare$: number of rare cases using the Kernel Density Relevance (KDR) with $\phi(y) > 0.7$; Rare: $100 \times nRare/N$).	232
2.3	Description of networks used in our study. (N : number of instances; $nRare$: number of rare cases using the KDR with $t_E = 0.7$; $\%Rare$: $100 \times nRare/N$). Starting postmile, ending postmile and postmile length are in miles.	234
2.4	Training hyperparameters for each dataset. [$n1; n2; n3$] defines the architecture of the MLP for each dataset, where $n1$, $n2$ and $n3$ denote the number of neurons in the first, second and third layer respectively.	235
2.5	Results of the 11 datasets, in terms of the AOC of the REC_{G-Mean} and REC_{CWA} curves. The best results per AOC and per dataset are marked in bold.	236
2.6	Results for the four experiments applied on the 7 datasets as well as the traffic flow datasets, in terms of scalar measures GME and CWE . The best results per measure and per dataset are in bold.	237
2.7	Convergence speeds (in terms of the number of backpropagations required before convergence) of the different approaches for the 7 datasets and the traffic flow datasets. The least number of backpropagations recorded per dataset is reported in bold.	237
2.1	Comparison with existent loss functions for regression.	258
4.1	Comparative results between the ConvNets trained on commonly used loss functions for regression as well as ours, in terms of Mean Pixel Error (MPE).	262
1.1	Notation.	271
1.2	Notation.	272
1.3	Notation.	272
1.4	A summary of the Phoenician ConvNet architecture parameters.	275
1.5	The confusion matrix for the recognition of each Phoenician character.	276
1.6	Learning parameters of the DBN. Let's note that learning parameters are set according to suggestions published by Hinton (Hinton et al., 2006).	277