جامعة سيدي محمد بن عبد الله بفاس
UNIVERSITÉ SIDI MOHAMED BEN ABDELLAH DE FES

كلية العلوم والتقنيات فاس
FACULTÉ DES SCIENCES ET TECHNIQUES DE FÈS

N° d'ordre : 51/2022

Centre des Etudes Doctorales : Sciences et Techniques et Sciences Médicales
Formation Doctorale : Sciences de l'ingénieur Sciences Physiques, Mathématiques et Informatique
Discipline : Mathématiques et Informatique
Spécialité : Mathématiques Appliquées et Informatique
Laboratoire : Modélisation et Structures Mathématique

Thèse pour l'obtention de DOCTORAT

Regularization of Deep Neural Networks: Hyperparameters estimation, redundancy optimization, and sparsity induction. Applications to real problems.

Préparée par : Mr. HSSAYNI EL HOUSSAINE

Soutenue le 04 Juillet 2022 devant le jury :

Nom et Prénom	Grade	Etablissement	Qualité
Elhoussine AZROUL	PES	Faculté des Sciences Dhar El mahraz Fès	Président
Abdellatif EL AFIA	PES	ENSIAS, Université Mohammed V de Rabat	Rapporteur & examinateur
Mohammed SABBANE	PES	Faculté des Sciences Meknès	Rapporteur & examinateur
Karim EL MOUTAOUAKKIL	PH	Faculté Polydisciplinaire de Taza	Rapporteur & examinateur
Mohamed LAZAAR	PH	ENSIAS, Université Mohammed V de Rabat	Examinateur
Khalid HADDOUCH	PH	Ecole Nationale des Sciences Appliquées de Fès	Examinateur
Rachid EL KHAOULANI EL IDRISSI	PH	Faculté des Sciences et Techniques de Fès	Examinateur
Mohamed ETTAOUIL	PES	Faculté des Sciences et Techniques de Fès	Directeur de thèse
Nour-Eddine JOUDAR		ENSAM, Université Mohammed V de Rabat	Invité

جامعة سيدي محمد بن عبد الله بفاس
UNIVERSITÉ SIDI MOHAMED BEN ABDELLAH DE FES
كلية العلوم والتقنيات فاس
FACULTÉ DES SCIENCES ET TECHNIQUES DE FÈS

Résumé de la thèse

Au cours de la dernière décennie, l'apprentissage profond, en particulier les réseaux de neurones profonds, ont démontré de grandes performances dans de nombreuses tâches difficiles d'apprentissage automatique et d'intelligence artificielle telles que la classification, le regroupement et la prédiction. Récemment, les réseaux de neurones convolutifs (CNNs) sont devenus l'un des modèles d'apprentissage profond les plus importants et les plus puissants, qui ont constamment montré des résultats prometteurs et un succès significatif dans de nombreux domaines et applications. Cependant, les réseaux de neurones convolutionnels performants nécessitent souvent un grand nombre de paramètres, ce qui entraîne une quantité importante de mémoire et un coût de calcul plus élevé. Cela peut produire certains phénomènes indésirables, notamment, l'overfitting et la redondance des paramètres.

Dans cette thèse, notre étude se concentre sur la régularisation des réseaux de neurones profonds en utilisant différentes approches. En effet, motivée par le succès de la méthode de régularisation DropConnect, une analyse théorique est développée afin d'estimer ses hyperparamètres. Cette estimation est basée sur l'optimisation de l'écart de généralisation en vertu de la complexité de Rademacher. La deuxième approche concerne une nouvelle vision de la régularisation basée sur l'optimisation combinatoire. En effet, un nouveau modèle d'optimisation est proposé pour contrôler et visualiser la redondance dans les réseaux convolutifs profonds. Et comme dernier axe dans cette thèse, une nouvelle approche basée sur l'apprentissage profond est proposée pour la classification des séries temporelles.

Pour chaque problème étudié dans ce travail, un certain nombre de résultats numériques sont présentés et comparés avec plusieurs approches de la littérature.

Mots clés : Estimation des hyperparamètres, optimisation non-linéaire, apprentissage profond, réseaux de neurones convolutifs, sur-apprentissage, régularisation, dropout, dropconnect, complexité de Rademacher, généralisation, classification des séries temporelles.

كلية العلوم و التقنيات فاس
ⵜⴰⵖⵉⵡⴰⵏⵜ ⵏ ⵜⵎⴰⵙⵙⴰⵏⵉⵏ ⴷ ⵜⵙⵏⵉⴳⵉⵜⵉⵏ
Faculté des Sciences et Techniques de Fès

جامعة سيدي محمد بن عبد الله
ⵜⴰⵙⴷⴰⵡⵉⵜ ⵙⵉⴷⵉ ⵎⵓⵃⵎⵎⴰⴷ ⴱⵏ ⵄⴱⴷⵓⵍⵍⴰⵀ
Université Sidi Mohamed Ben Abdellah

Regularization of Deep Neural Networks : Hyperparameters estimation, redundancy optimization and sparsity induction. Applications to real problems

HSSAYNI El houssaine

Thesis supervisor : ETTAOUIL Mohamed

Faculty of Sciences and Techniques
Sidi Mohamed Ben Abdellah University
Fez, Morocco

A dissertation submitted to the University of Sidi Mohamed Ben Abdellah in accordance with the requirements of the degree of
DOCTOR IN APPLIED MATHEMATICS AND ARTIFICIAL INTELLIGENCE

2022

Contents

Acknowledgement **v**

Abstract **vii**

Acronyms **ix**

Lists of Figures **xii**

Lists of Tables **xiii**

Résumé **1**

0.1 Introduction 2
0.2 Apprentissage profond 4
0.2.1 Réseaux de neurones convolutionnels 4
0.2.2 Optimisation pour l'apprentissage profond 8
0.3 Estimation des hyperparamètres de DropConnect par l'optimisation de l'écart de généralisation 10
0.3.1 Régularisation des réseaux de neurones profonds 10
0.3.2 Contrôle de l'écart de généralisation en vertu de la complexité de Rademacher pour DropConnect 14
0.3.3 Estimation de l'hyperparamètre de DropConnect 16
0.4 Contrôle et visualisation de la redondance dans les CNNs par l'optimisation combinatoire 21
0.4.1 Nouveau modèle d'optimisation de la redondance dans les CNNs . 22
0.4.2 Visualisation de la redondance dans les CNNs 25
0.4.3 Résolution du modèle d'optimisation obtenu 25
0.5 Optimisation des réseaux de neurones profonds pour la classification des séries chronologiques 27
0.5.1 Préliminaires 27
0.5.2 Représentation des séries chronologiques 29
0.5.3 Classification-Optimisation des séries chronologiques 32
0.6 Conclusion 36

Introduction **39**

1 Deep Learning **41**
1.1 Introduction . . . 41
1.2 Deep Learning generalities . . . 42
1.2.1 What Is Deep Learning? . . . 42
1.2.2 History of Deep Learning . . . 42
1.3 Deep Convolutional neural networks . . . 43
1.3.1 Convolution layer . . . 43
1.3.2 Non-linear non-saturating activation functions . . . 49
1.3.3 Subsampling layer . . . 51
1.3.4 Classification layers . . . 53
1.4 Loss functions in deep learning . . . 54
1.4.1 Mean Square Loss / L^2 Loss . . . 54
1.4.2 Mean Absolute Error . . . 54
1.4.3 Hinge loss . . . 54
1.4.4 Negative log-likelihood . . . 55
1.4.5 Binary Cross-Entropy Loss . . . 55
1.4.6 Cross-Entropy Loss . . . 56
1.5 Optimization for Training Deep neural networks . . . 56
1.5.1 Stochastic Gradient Descent . . . 57
1.5.2 Stochastic Gradient Descent with Momentum . . . 59
1.5.3 Optimization Algorithms with Adaptive Learning Rates . . . 60
1.6 Conclusion . . . 64

2 Deep neural networks regularization and generalization: A mathematical approach **65**
2.1 Introduction . . . 65
2.2 Regularization for DNNs . . . 66
2.2.1 L^1 and L^2 regularization . . . 66
2.2.2 Batch normalization . . . 67
2.2.3 Dropout regularization . . . 67
2.2.4 DropConnect regularization . . . 68
2.3 Genralization gap optimization using Rademacher complexity . . . 69
2.3.1 Rademacher complexity . . . 69
2.3.2 Deep neural networks generalization gap . . . 74
2.4 Conclusion . . . 75

3 Estimation of DropConnect hyperparameter using generalization gap and Rademacher complexity **77**
3.1 Abstract . . . 77
3.2 Introduction . . . 78
3.3 Estimation of DropConnect hyperparameter using Rademacher complexity and generalization gap . . . 79

3.3.1 Rademacher DropConnect complexity and estimation of generalization gap . . . 80
3.3.2 Adaptive DropConnect: An improved regularization technique . . . 82
3.4 Experiments . . . 87
3.4.1 Description of datasets . . . 88
3.4.2 Evaluation metrics . . . 88
3.4.3 Description of experiments . . . 89
3.4.4 Results and discussion . . . 89
3.5 Conclusion . . . 93

4 Optimization and localization of redandancy in CNNs using a mixed non linear programming model 95
4.1 abstract . . . 95
4.2 Introduction . . . 96
4.3 Proposed optimization model for CNN redundancy reduction . . . 97
4.3.1 Modelling the redundancy optimization problem . . . 98
4.3.2 Modelling the localization of redundancy in CNNs . . . 101
4.3.3 Adaptation of GA for solving the obtained model . . . 102
4.4 Experiments . . . 104
4.4.1 Description of datasets . . . 104
4.4.2 Evaluation metrics . . . 104
4.4.3 Numerical study of the hyperparameter λ . . . 105
4.4.4 Performance of proposed approach . . . 106
4.4.5 Localization of redundancy in CNNs . . . 108
4.4.6 Comparison with other methods . . . 109
4.5 Conclusion . . . 110

5 Optimization and regularization of deep neural networks for time series classification 111
5.1 Abstract . . . 111
5.2 Introduction . . . 112
5.3 Preliminaries . . . 114
5.3.1 Time series representation background . . . 114
5.3.2 Convolutional neural networks . . . 115
5.4 Proposed framework . . . 117
5.4.1 Representation phase . . . 117
5.4.2 Classification-Optimization phase . . . 121
5.5 Experiments . . . 125
5.5.1 Description of datasets . . . 125
5.5.2 Evaluation metrics . . . 125
5.5.3 Description of experiments . . . 126
5.5.4 Results and discussion . . . 126
5.6 Conclusion . . . 130

Conclusions and Future work 137

Acknowledgment

This account of gratitude is something special to me. It is not just a formal note but more than that. It would make possible for me to express my feelings for all those who had been at my side during this important project of my life and I never found a chance to do that till now.

I was lucky to have Pr. **Mohamed ETTAOUIL** as my advisor. He is a kind hearted and highly professional person. From the very start till the end, his contribution and guidance were of great value. Whenever, I needed help, he was there for assistance. His precise and pertinent comments always helped me get out of difficult situations. I am thankful for the extraordinary experiences he arranged for me and for providing opportunities to grow professionally.

With equal fervour, I want to extend a heart-felt thank to my committee members, Pr **Abdellatif EL AFIA**, Pr. **Karim EL MOUTAOUAKKIL** and Pr **Mohammed SABBANE** for their reviews and helpful comments throughout the process as well as Pr. **Elhoussine AZROUL** for presiding the doctoral committee. Further, a special thanks to the professors: **Mohamed LAZAAR**, **Khalid HADDOUCH** and **Rachid EL KHAOULANI EL IDRISSI**, for the honor they did me by agreeing to examine this thesis and joining the jury. This thesis would not have been possible without Pr. **Nour-Eddine Joudar**, whose his guidance from the initial step enabled me to develop an understanding of the subject. I would like to take the opportunity to express my deepest gratitude to him.

I am grateful for my parents whose constant love and support keep me motivated and confident, and who remind me of what is important in life. My accomplishments and success are because they believed in me and they kept me all the time in their prayers. I wish one day I get a chance to pay back a little for what my parents did for me. My brothers and sisters had always been there for me during all kind of circumstances. I am thankful for their encouragement and kindness.

At the end, I would like to express my deepest gratitude to my colleagues and friends for their unfailing support and continuous encouragement.
I express my sincere appreciation to all those who have helped me throughout my thesis.

Abstract

During the last decade, deep learning, in particular, deep neural networks have demonstrated great performance in many challenging machine learning and artificial intelligence tasks such as classification, clustering, and prediction. Recently, convolutional neural networks (CNN), become one of the extraordinary important and powerful deep learning models that have constantly shown hopeful results and significant success in large areas and applications. However, successful CNNs often require a large number of parameters, which leads to a significant amount of memory and a higher computational cost. This may produce some undesirable phenomena, notably the overfitting and parameters redundancy. In this thesis, our study focuses on regularization of deep neural networks using different approaches. In fact, motivating by the success of DropConnect method, a theoretical analysis is developed in order to estimate its hyperparameters. This estimation is based on optimizing the generalization gap using Rademacher complexity. This generated at the end a new adaptive regularization method. The second approach consists of regularizing CNNs by reducing the parameters redundancy. In this context, an optimization model is proposed to control and localize the redundancy over the networks. In addition, the time series classification is treated using deep learning as the last contribution of this work. For each problem studied in this work, a number of numerical results are presented and compared against several approaches from the literature.

Mots-clés: hyperparameters estimation, redundancy optimization, sparsity induction, deep neural networks, convolutional neural networks, overfitting, regularization, dropout, dropconnect, Rademacher complexity, gap generalization, time series classification.

Acronyms

- ANN : Artificial Neural Network
- DNN : Deep Neural Network
- CNN : Convolutional Neural Network
- DL : Deep Learning
- GA : Genetic Algorithm
- MINLP : Mixed Integer Non Linear Programming
- TSC : Time Series Classification
- KRR : Kernels Redundancy Reduction
- SGD : Stochastic Gradient Descent

List of Figures

1 Description des réseaux de neurones convolutionels 5
2 La fonction d'activation ReLU 6
3 Dropout régularisation 13
4 DropConnect régularisation 13
5 Un exemple de l'utilisation de U pour réduire la redondance dans le CNN. 23
6 Organigramme du modèle proposé 26
7 La représentation des séries temporelles proposée 30
8 L'approche proposée pour la classification des séries temporelles 35

1.1 A Venn diagram describing deep learning as a subfield of machine learning which is in turn a subfield of artificial intelligence 42
1.2 Description of convolutional neural networks. 44
1.3 Convolution process 46
1.4 Illustration of the flipped delta matrix 47
1.5 Convolution process 48
1.6 ReLU activation function 50
1.7 Illustration of the hybrid pooling. (a) The operation in the training stage. (b) The operation in the testing stage. **[73]** 53
1.8 Effect of momentum in smoothing zigzag updates 60

2.1 Dropout regularization 68
2.2 DropConnect regularization 69

3.1 Accuracies variations durind DNN1 training using DropConnect, A-DropConnect, and without Drop 93

4.1 An illustrative example of the use of U in reducing the redundancy in CNN. 99
4.2 Flow chart of the proposed model 101
4.3 Variation of optimized number of kernels and accuracy according to λ: model CNN:16 $\times$ 32 105
4.4 Variation of optimized number of kernels and accuracy according to λ: model CNN:32 $\times$ 64 106
4.5 Variation of optimized number of kernels and accuracy according to λ: model CNN:64 $\times$ 128 106

4.6 Local redundancy distribution within ResNet . . . 108
4.7 Local redundancy distribution within VGG19 . . . 109
4.8 Local redundancy distribution within Net in Net. . . . 109

5.1 Proposed framework for time series classification . . . 118
5.2 Time series representation phase . . . 120
5.3 Suggested CNN architecture for time series classification . . . 127
5.4 Variation of the error rate according to the reduced dimensions m: case of $Lightning-7$ dataset . . . 128
5.5 Variation of the error rate according to the reduced dimensions m: case of $Beef$ dataset . . . 128
5.6 Variation of the error rate according to the reduced dimensions m: case of $Adiac$ dataset . . . 129
5.7 Parameters number in CNN and in the corresponding optimized CNN* for each time series dataset . . . 131

List of Tables

1 Notations 22

3.1 DNN architectures used for validation and evaluation of A-DropConnect . 89
3.2 CNN architectures used for evaluation of A-DropConnect applied in the classification of NORB and SVHN datasets 90
3.3 Classification evaluation of MNIST and Fasion-MNIST using different DNNs regularized with DropConnect and A-DropConnect. 91
3.4 Mean of accuracies for MNIST classification using DropConnect with fixed and adaptive hyperparameter 91
3.5 Mean of accuracies for Fashion-MNIST classification using DropConnect with fixed and adaptive hyperparameter 91
3.6 Comparison of accuracies mean for MNIST and Fashion-MNIST classification using different drop methods 92
3.7 Classification performance evaluation using different regularization methods in the classification of NORB dataset 93
3.8 Classification performance evaluation using different regularization methods in the classification of SVHN dataset 94

4.1 Notations 98
4.2 Evaluation metrics obtained using ResNet and the optimized ResNet* in the classification of Cifar10 and Cifar100 107
4.3 Evaluation metrics obtained using VGG19 and the optimized VGG19* in the classification of Cifar10 and Cifar100 107
4.4 Evaluation metrics obtained using Net-in-Net and the optimized Net-in-Net* in the classification of Cifar10 and Cifar100 108
4.5 Performance in term of accuracy of different methods on the datasets. . . . 110

5.1 Description of the used time series datasets 132
5.2 Comparison of NCR-CNN with and without using redundancy optimization on different datasets 133
5.3 Error rates of different approaches on 17 UCR time series datasets 134
5.4 Evaluation metrics of different approaches using 17 UCR time series datasets 135

Résumé

RÉGULARISATION DES RÉSEAUX DE NEURONES PROFONDS: ESTIMATION DES HYPERPARAMÈTRES, OPTIMISATION DE LA REDONDANCE ET INDUCTION DE LA SPARSITÉ. APPLICATIONS AUX PROBLÈMES RÉELS

0.1 Introduction

Aujourd'hui, l'intelligence artificielle est entrain de coloniser tous les secteurs de l'économie, de l'industrie, de la communication, de la santé, et des transports avec la voiture autonome [**20**, **31**, **64**]. Beaucoup d'observateurs ne parlent plus d'une évolution technologique, mais d'une révolution [**68**]. L'apprentissage automatique ou le machine learning est l'un des piliers fondamentaux de l'intelligence artificielle, dont l'objectif est de construire des machines capables de prendre des décisions relevant d'une certaine forme de l'intelligence, des machines capables d'accomplir des tâches généralement assurées par les animaux et les humains : percevoir, raisonner et agir [**6**, **63**, **79**]. Dans ses débuts, le machine learning focalisait sur des approches dont la construction d'un extracteur de caractéristiques efficace est très difficile. En outre, avec l'arrivée du big data, les approches classiques de l'apprentissage automatique souffrent devant l'augmentation des données [**87**, **100**]. Ceci a nécessité de continuer à développer et de proposer des approches alternatives. Depuis quelques années, les algorithmes d'apprentissage profond ou deep learning ont révolutionné ce domaine, en parvenant à réaliser des tâches considérées jusqu'alors comme difficiles. Actuellement, l'apprentissage profond a pris le devant de la scène, et devenu le noyau dur de l'intelligence artificielle [**45**, **46**, **74**]. Ce deep learning caractérise un réseau de neurones artificiel, dont l'architecture et le fonctionnement sont inspirés de ceux du cerveau humain [**20**, **71**]. Bien que les découvertes en neurosciences ont nourri la recherche en intelligence artificielle et en apprentissage profond en particulier, le développement de deep learning nécessitait des analyses mathématiques théoriques pour suivre les attentes du monde réel [**30**, **50**]. Grâce à la solidité de ses bases théoriques et à la robustesse de ses algorithmes, les réseaux de neurones profonds sont devenus le sujet de recherche le plus traité par les spécialistes de l'intelligence artificielle [**45**]. Ces recherches ont permis de réaliser d'importants progrès dans différents secteurs, à savoir : la robotique, les véhicules autonomes, le traitement de la parole et la compréhension du langage naturel [**29**, **48**, **49**]. Ces progrès impressionnants ont été accomplis à travers le développement de différentes architectures profondes [**16**, **46**, **53**]. Le réseau de neurones convolutionel (CNN) est l'une de ces architectures qui a réalisé des taches extraordinaires [**10**, **42**, **107**]. Aussi, tout comme le cerveau humain, les CNNs, et les réseaux de neurones profonds en général se caractérisent par un très grand nombre de paramètres. Par conséquence, l'apprentissage d'un réseau profond peut produire certains phénomènes indésirables, notamment l'overfitting [**61**]. C'est le problème qui surgit quand on utilise un modèle trop compliqué et qu'on n'a pas assez de données d'apprentissage. Le système a une capacité suffisante pour apprendre les données sans découvrir la régularité sous-jacente [**25**, **56**, **61**]. L'overfitting ou le surapprentissage est le challenge actuel de l'apprentissage profond, et c'est exactement l'objectif de cette thèse. Pour faire face à ce problème majeur, plusieurs méthodes de régularisation ont été développées. Ces méthodes peuvent être divisées en plusieurs classes, à savoir : les méthodes de régularisations basés sur la pénalisation de la fonction objective et des méthodes de régularisation stochastiques. L^1 et L^2 représentent deux méthodes compétitives de la première classe [**50**, **108**]. La régularisation stochastique consiste à injecter des variables aléatoires dans le modèle pour renforcer sa généralisation. Dropout et DropConnect sont deux méthodes de régularisation aléatoires qui ont achevé des performances très significatives [**61**, **111**,

123].
Le but de cette thèse est d'aborder le problème de régularisation dans les réseaux de neurones profonds par plusieurs approches. Dans la première contribution, nous investissons les résultats surprenants de la complexité de Rademacher pour élaborer une estimation de l'hyperparamètre de la méthode DropConnect. La deuxième approche est consacrée au contrôle et à la visualisation de la redondance dans les CNNs à l'aide d'un modèle d'optimisation combinatoire. Comme dernière contribution, les réseaux de neurones profonds sont optimisés dans le cadre de la classification des séries chronologiques.
Ce document est structuré de la manière suivante :
Le **Chapitre 1** est consacré à la présentation des réseaux de neurones profonds et leur apprentissage. Cette représentation montre toute la diversité et la richesse des travaux qui s'inscrivent dans ce domaine. Tout d'abord, nous décrivons le concept du deep learning. Après, nous introduisons le noyau des réseaux profonds qui est le réseau de neurones convolutif, ainsi que les composantes principales qui construisent son architecture. Ensuite, nous expliquons les fondements théoriques de l'apprentissage des réseaux profonds, en introduisant les différents algorithmes d'optimisation qui ont montré leur efficacité dans plusieurs contextes [**79**, **18**, **21**, **22**].
Le **Chapitre 2** s'attache à présenter la vision mathématique de la régularisation et la généralisation des réseaux de neurones profonds. Dans la première section, nous décrivons les différentes méthodes de régularisation connues efficaces et compétitives. Dans la deuxième section, nous développons une analyse théorique de la généralisation en se basant sur la complexité de Rademacher [**58**, **103.105**].
Dans le **Chapitre 3**, la théorie exposée dans le chapitre précédent est investie pour élaborer une estimation de l' hyermeramètre de la méthode de régularisation DropConnect. Cette estimation est basée sur la minimisation de la complexité de Rademacher en tant qu'une borne sup de l'écart de généralisation. La technique proposée aboutit aux résultats satisfaisants en comparaison avec d'autres méthodes de la littérature.
Le **Chapitre 4** concerne une nouvelle vision de la régularisation, basée sur l'optimisation combinatoire. En effet, nous proposons un modèle d'optimisation pour contrôler et visualiser la redondance dans les réseaux convolutifs profonds. En outre, nous intégrons la norme L^1 pour renforcer la régularisation. Le modèle suggéré est un problème d'optimisation à variables mixtes et contraintes linéaires. Pour résoudre le problème obtenu, nous proposons de combiner l'optimisation par descente de gradient stochastique (DGS) pour résoudre la partie continue, et l'algorithme génétique pour l'optimisation combinatoire.
Dans le **Chapitre 5**, les réseaux de neurones profonds sont mis en oeuvre pour la classification des séries chronologiques. Dans ce contexte, nous introduisons les séries chronologiques, ainsi que leurs propriétés principales dans la première section. Après, nous proposons une représentation efficace pour transformer une série chronologique à une matrice de caractéristiques pertinentes. Ensuite, nous proposons un réseau de neurones profond renforcé par un modèle d'optimisation afin de réduire la redondance des paramètres. Le modèle ainsi construit permet de classifier les séries temporelles avec une grande efficacité.
Le dernier chapitre conclut les travaux développés dans cette thèse et propose des futures directions de recherche en apprentissage profond.

0.2 Apprentissage profond

Grâce à ses capacités émergentes et ses applications incontournables, l'apprentissage profond a pris le devant de la scène dans le domaine de l'intelligence artificielle [**45**, **46**, **49**]. C'est un domaine de recherche extrêmement actif, dont le fonctionnement est inspiré de celui du cerveau humain [**71**]. Il utilise des algorithmes pour traiter les données et imiter le processus de réflexion chez un humain. Extrêmement puissants, les algorithmes de l'apprentissage profond sont à l'origine de la plupart des progrès récents, et permettent de réaliser des taches complexes, à savoir, la classification et la prédiction des séries temporelles [**47**, **64**, **126**], la prévision de la consommation d'énergie du bâtiment [**3**]. Ces percées impressionnantes ont été l'objectif du développement des réseaux de neurones profonds, qui sont le noyau dur de l'apprentissage profond. Parmi ces modèles profonds, le réseau de neurones convolutif à démontré sa performance émergente dans plusieurs applications provient du monde réel [**66**, **77**, **107**].

0.2.1 Réseaux de neurones convolutionnels

Le réseau de neurones convolutif (CNN) a été inventé par LeCun en 1998 [**71**]. Sa découverte a permis de réaliser des avancées extraordinaires dans le domaine de l'intelligence artificielle [**68**, **74**, **107**]. C'est un réseau d'apprentissage profond dont le fonctionnement et la topologie sont inspirés du cortex visuel [**71**]. Grâce à sa domination sur le domaine, il est connu aussi sous le nom de deep learning [**45**]. Contrairement aux méthodes d'apprentissage classiques, les CNNs intègrent la phase d'extraction des caractéristiques dans le processus d'apprentissage. Cette propriété impressionnante, qui est l'origine de tous les progrès accomplis par l'apprentissage profond, est assurée par l'unique caractéristique des CNNs. En effet, CNN se caractérise par des couches spécifiques permettant d'introduire des liens partiels pour réduire le nombre des paramètres et renforcer le partage des caractéristiques communes [**10**, **50**, **132**]. Généralement, un CNN est composé de deux parties : une première pour l'extraction des caractéristiques à l'aide d'une successivement des couches de convolution et des couches de sur-échantillonnage (pooling). La deuxième partie consiste à classifier les caractéristiques pertinentes reçues de la première phase à travers des couches entièrement connectées. La figure 1 illustre la topologie générale d'un réseau de neurones convolutif. Dans la suite, nous détaillons le principe, le fonctionnement, ainsi que les opérateurs utilisés pour chaque couche.

0.2.1.1 Couche de convolution

La prise en compte du voisinage d'un pixel est une notion connue de la communauté d'intelligence artificielle et est depuis longtemps exploitée, notamment par les algorithmes reposant sur l'extraction de caractéristiques. Ces algorithmes ont été basés sur des opérateurs qui introduisent une invariance par translation. Ces derniers ont été généralisés par un opérateur mathématique, appelé opérateur de convolution. Il est donc naturel d'exploiter ces opérateurs de convolutions dans les réseaux de neurones.
La définition de l'opérateur de convolution (ou produit de convolution) dans le cas continu, discret, ainsi que son exploitation dans les CNNs, sont décrits comme suit.

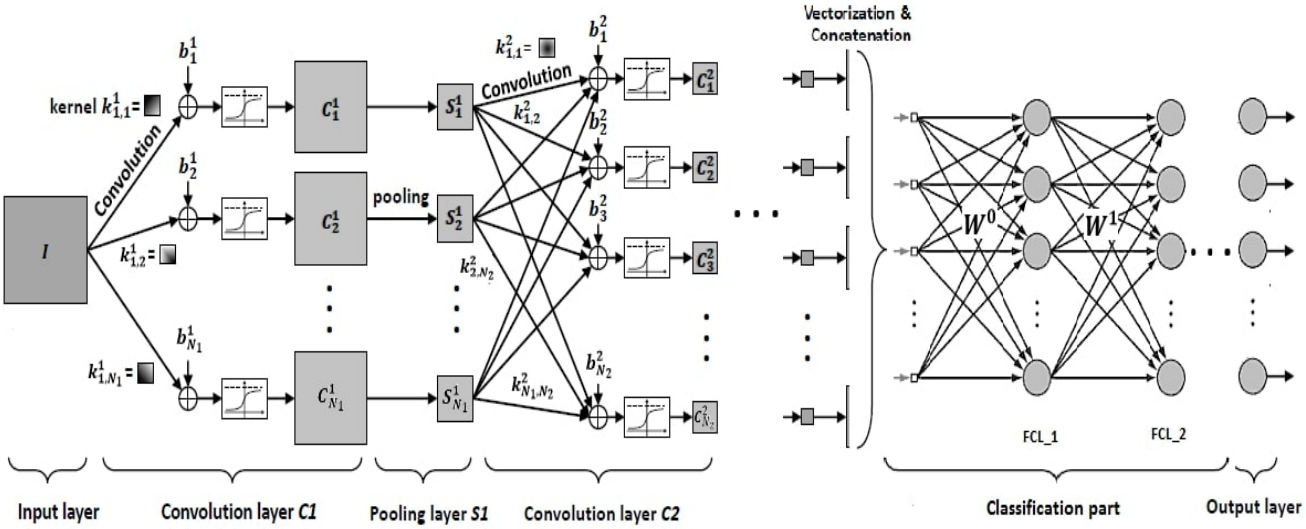

Figure 1 – Description des réseaux de neurones convolutionels

- **Produit de convolution dans le cas continu**
  Soient f et g deux fonctions dans $L^1(IR^2)$, on note par $f * g$ le produit de convolution de f et g et est donné par

$$\forall (x,y) \in IR^2, \;\; f*g(x,y) = \int_{IR}\int_{IR} f(x-x', y-y')g(x',y')dx'dy'.$$

  qui est évidemment une fonction de $L^1(IR^2)$.

- **Produit de convolution dans le cas discret:**
  En se basant sur la convolution continue, nous proposons l'extension discrète de la convolution pour le cas des suites finies.
  Soient $(f_{m,n})_{1\leq m\leq N, 1\leq n\leq N}$ et $(g_{m,n})_{1\leq m\leq N, 1\leq n\leq N}$ deux suites finies. On note par $f * g$ la suite suivante

$$(f*g)_{m,n} = \sum_{i=1}^{N}\sum_{j=1}^{N} f_{m-i,n-j}g_{i,j}$$

- **Produit de convolution pour l'apprentissage profond**
  Dans l'apprentissage profond, la couche de convolution effectue un produit de convolution entre deux matrices, la première représente une sous partie des données en entrée (champ réceptif) et la deuxième représente un filtre qui contient les paramètres d'apprentissage. Une opération de convolution génère une troisième matrice notée par la carte des caractéristiques. La formule suivante décrit la convolution de la matrice I par le noyau de convolution K

$$S(i,j) = (I*K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n)$$

Le processus d'une convolution illustre la stratégie de réduction de dimensionnalité d'un CNN, où chaque case (neurone) de la carte de caractéristique courante est connectée seulement à un sous ensemble des neurones en entrées (champs récepteurs). De plus, l'application du même filtre sur toute la carte des caractéristiques lui permet de découvrir les attributs précédemment détectés dans différentes zones de l'image.

À la fin de chaque opération de convolution, une fonction d'activation est appliquée sur la couche de convolution résultante afin d'améliorer la généralisation [**116**].

0.2.1.2 Fonctions d'activation

Le choix des fonctions d'activation dans les réseaux profonds a un effet significatif sur la dynamique de l'apprentissage et la performance du modèle. En outre, une stratégie clé permettant mieux d'entrainer les réseaux de neurones profonds, consiste à utiliser des fonctions d'activation non saturantes pour réduire le problème de fuite du gradient [**46**, **113**, **116**]. Les choix populaires qui se saturent uniquement dans le domaine négatif sont l'unité linéaire rectifiée (ReLU), sa variante lisse et non linéaire Softplus, et les unités linéaires exponentielles (ELU et SELU). D'autres fonctions sont non saturantes dans tout le domaine réel, comme la ReLU linéaire paramétrique (PReLU) [**116**]. Avant de donner l'expression de la fonction ReLU, nous donnons d'abord la définition d'une fonction non saturante.

Définition 1

Une fonction à valeurs réelles $f : \mathbb{R} \longrightarrow \mathbb{R}$ est dite fonction non saturante si $|\lim_{x \longrightarrow +\infty} f(x)| = +\infty$ ou $|\lim_{x \longrightarrow -\infty} f(x)| = +\infty$

La première fonction d'activation non saturante proposée pour les réseaux de neurones profonds est la fonction ReLU (Rectified Linear Unit), définie par :

$$f(x) = ReLU(x) = \begin{cases} x & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases}$$

La figure 2 représente la courbe de la fonction ReLU.

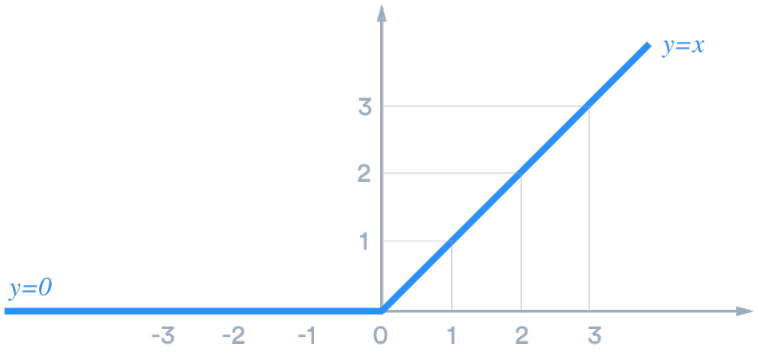

Figure 2 – La fonction d'activation ReLU

D'autres fonctions d'activation, ainsi que les propriétés de chaque fonction sont présentées dans le paragraphe 1.3.2.

0.2.1.3 Couche de sur-échantillonnage (pooling)

Dans un réseau de neurones convolutif, les couches de convolution sont alternées avec des couches de sous-échantillonnage (appelé pooling en anglais). Ces dernières réduisent la taille des cartes de caractéristiques résultantes de la couche de convolution. Le but de cette réduction est d'améliorer la précision par la sélection des attributs dominants. De plus, l'optimisation du nombre des paramètres permet de réduire la taille du modèle et d'optimiser la complexité temporelle [**67**, **73**, **109**].
Il existe plusieurs types d'opérations de sur-échantillonnage :

- **Max Pooling** [**117**] est la méthode de sur-échantillonnage la plus populaire, qui renvoie la valeur maximale du champ réceptif. Cette opération est décrite par l'équation (1).

$$s_k = \max_{a_i \in R_k} a_i \quad for \quad k = 1, ..., K \tag{1}$$

  Où a_i sont les éléments du champ réceptif R_k et s_k la valeur résultante correspondante.
  Max Pooling vise à extraire des caractéristiques locales telles que des lignes, des bords et des textures dans la carte des caractéristiques.

- **Average Pooling** [**72**] renvoie la moyenne arithmétique des éléments du champ réceptif. Cette opération est décrite par l'équation (2)

$$s_k = \frac{1}{\#R_k} \sum_{a_i \in R_k} a_i \quad for \quad k = 1, ..., K \tag{2}$$

  Cette méthode peut extraire des caractéristiques globales dans une carte de caractéristiques donnée en lissant le champ réceptif.

- **Stochastic Pooling**[**120**] est une méthode de sur-échantillonnage améliorée, qui se déroule comme suit : Dans la première, les probabilités p_j sont calculées pour chaque région en normalisant les activations au sein des régions, comme indiqué dans l'expression (3).

$$p_j = \frac{a_j}{\sum_{a_i \in R_k} a_i} \quad for \quad j = 1, ..., |R_k| \tag{3}$$

Ensuite, les sorties sont sélectionnées en fonction des probabilités calculées par la distribution multinomiale déterminée avec les probabilités p_j pour $j = 1, ..., |R_k|$ comme suit:

$$s_k = a_l \quad where \quad l \sim P(p_1, ..., p_{|R_k|}) \tag{4}$$

Dans la phase de test, au lieu de la moyenne normale, on adopte la forme probabiliste de la moyenne, comme indiqué dans la formule (5).

$$s_k = \sum_{a_j \in R_k} p_j a_j \quad for \quad k = 1, ..., K \tag{5}$$

Cette méthode est moins sujette au sur-apprentissage en raison de la stratégie stochastique utilisée, et elle est plus performante que les méthodes de sur-échantillonnage max-pooling et average-pooling [**120**].

0.2.1.4 Couche entièrement connectée

Une fois que nous avons atteint la dernière couche d'extraction des caractéristiques, toutes les cartes de caractéristiques sont converties en un vecteur qui, à son tour, est présenté comme une entrée à un classificateur composé de couches entièrement connectées, où les sorties sont données par l'expression :

$$Z^k = \begin{pmatrix} Z_1^k \\ Z_2^k \\ \vdots \\ Z_{n_{FC_k}}^k \end{pmatrix} = \begin{pmatrix} f\Big(\sum_{i=1}^{n_{k-1}} W_{1i}^k \times Z_i^{k-1}\Big) \\ f\Big(\sum_{i=1}^{n_{k-1}} W_{2i}^k \times Z_i^{k-1}\Big) \\ \vdots \\ f\Big(\sum_{i=1}^{n_{k-1}} W_{n_{FC_k} i}^k \times Z_i^{k-1}\Big) \end{pmatrix}$$

La fonction Softmax est ensuite appliquée pour calculer la sortie du réseau.

$$Y = softmax(Z_{FC}^N)$$

où N_{FC} est le nombre de couches cachées.

0.2.2 Optimisation pour l'apprentissage profond

L'optimisation en général est une tâche extrêmement difficile. Traditionnellement, l'apprentissage automatique évite la difficulté de l'optimisation générale en concevant soigneusement la fonction objective et les contraintes pour s'assurer que le problème

d'optimisation est convexe. Par contre, lors de l'apprentissage des réseaux de neurones, nous devons faire face au cas général non convexe. De plus, l'apprentissage des réseaux de neurones profonds nécessite de grands ensembles d'apprentissage pour une bonne généralisation, mais de grands ensembles d'apprentissage sont également plus coûteux en termes de calcul. Pour résoudre ce problème, plusieurs algorithmes d'optimisation ont été développés en tenant compte de ces défis. Parmi ces algorithmes, on cite la descente de gradient stochastique et ses variantes [**18**, **22**, **79**, **84**].
La descente de gradient stochastique (DGS) est probablement l'algorithme d'optimisation le plus utilisé pour l'apprentissage automatique en général et pour les réseaux de neurones profonds en particulier. La DGS est une extension de l'algorithme de descente de gradient qui a été développée pour traiter des problèmes d'optimisation non linéaire [**71**]. L'idée principale derrière cet algorithme est que le gradient est une espérance. L'espérance peut être estimée approximativement à l'aide d'un petit ensemble d'échantillons. Plus précisément, à chaque étape de l'algorithme, nous pouvons échantillonner un mini-groupe d'exemples $\mathbb{B} = \left\{ \boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m')} \right\}$ tirés uniformément de l'ensemble d'apprentissage. La taille du minibatch m' est généralement choisie pour être un nombre relativement faible d'exemples, allant de 1 à quelques centaines. De manière cruciale, m' est généralement maintenu fixe lorsque la taille de l'ensemble d'apprentissage m augmente. Nous pouvons adapter un ensemble d'apprentissage contenant des milliards d'exemples en utilisant des mises à jour calculées sur seulement une centaine d'exemples.
L'estimation du gradient est formée par l'expression suivante :

$$g = {}^{1}_{m'} \nabla_{W} \sum_{i=1}^{m'} L\left(\boldsymbol{x}^{(i)}, y^{(i)}, \boldsymbol{W}\right)$$

en utilisant des exemples du minibatch $\mathbb{B}$. L'algorithme de descente de gradient stochastique suit ensuite le gradient estimé en descendant :

$$\boldsymbol{W} \leftarrow \boldsymbol{W} - \epsilon g$$

où ϵ est le pas d'apprentissage.

L'algorithme 1 montre comment suivre cette estimation de descente du gradient.

Algorithm 1: Descente de gradient stochastique (DGS)

- **Entrée :**

  Paramètre initial $\boldsymbol{W}$, pas d'apprentissage ϵ.

- **Sortie :**

  Paramètre optimisé $\boldsymbol{W}$*

tant que critère d'arrêt non rempli, **faire**
. échantillonner un mini-lot de m exemples uniformément de l'ensemble d'apprentissage

$\left\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\right\}$ avec les cibles correspondantes : $\boldsymbol{y}^{(i)}$.
. Calculer l'estimation du gradient : $\hat{\boldsymbol{g}} \leftarrow +\frac{1}{\mathrm{m}} \nabla_{\boldsymbol{W}} \sum_i L\left(f\left(\boldsymbol{x}^{(i)}; \boldsymbol{W}\right), \boldsymbol{y}^{(i)}\right)$
. Appliquer la mise à jour : $\boldsymbol{W} \leftarrow \boldsymbol{W} - \epsilon \hat{\boldsymbol{g}}$
fin tant que

$$\boldsymbol{W}^* \leftarrow \boldsymbol{W}$$

Un paramètre crucial pour l'algorithme DGS est le pas d'apprentissage. Auparavant, nous avons décrit DGS en utilisant un pas d'apprentissage fixe ϵ. En pratique, il est nécessaire de diminuer progressivement le pas d'apprentissage au cours du temps. Nous désignons donc le pas d'apprentissage à l'itération k par ϵ_k.
Dans ce contexte, plusieurs extensions de l'algorithme de DGS ont été développées dans le but d'adapter automatiquement le pas d'apprentissage lors de la descente de gradient. Parmi ces extensions, on cite, le DGS avec momontum [**18**], AdaGrad [**22**], RMSprop [**84**] et Adam [**21**].

0.3 Estimation des hyperparamètres de DropConnect par l'optimisation de l'écart de généralisation

Aussi, tout comme le cerveau humain, les CNNs, et les réseaux de neurones profonds en général, se caractérisent par un très grand nombre de paramètres. Par conséquence, l'apprentissage de ces réseaux profonds peut produire certains phénomènes indésirables, notamment, l'overfitting [**59**, **61**, **97**]. C'est le problème qui surgit quand on utilise un modèle trop compliqué et qu'on n'a pas assez de données d'apprentissage. Le système a une capacité suffisante pour apprendre les données sans découvrir la régularité sous-jacente [**46**]. Afin d'éviter ce défi, les méthodes de régularisation sont recommandées.
La régularisation des réseaux de neurones profond est le thème principal abordé dans cette thèse. dans cette section, nous décrivons les méthodes de régularisation les plus compétitives de la littérature. Ensuite, nous dérivons une borne de l'écart de généralisation en fonction de la complexité de Rademecher, afin de contrôler la généralisation de notre modèle. Cette fonction de contrôle est ensuite utilisée pour estimer les hyperparamètres de la méthode de régularisation DropConnect.

0.3.1 Régularisation des réseaux de neurones profonds

L'overfitting ou le surapprentissage est le challenge actuel de l'apprentissage profond. Pour faire face à ce problème majeur, plusieurs méthodes de régularisation ont été développées. Ces méthodes peuvent être regroupées en plusieurs classes, à savoir : les méthodes de régularisations basées sur la pénalisation de la fonction objective et des méthodes de régularisation stochastiques. L^1 et L^2 représentent deux méthodes compétitives de la première classe [**97**, **108**, **129**]. La régularisation stochastique consiste à injecter des variables

aléatoires dans le modèle pour renforcer sa généralisation [**110**]. Dropout et DropConnect sont deux méthodes de régularisation aléatoires qui ont achevé des performances très significatives [**111**, **123**, **130**].

0.3.1.1 L^1 et L^2 régularisation

Une des formes de régularisation couramment utilisée pour éviter le sur-apprentissage est de pénaliser les poids du réseau. Cette pénalisation permet de limiter la complexité du modèle et de réduire le risque de sur-apprentissage [**50**].
Cela revient à ajouter un terme $\lambda\Omega(\mathbb{W})$ à la fonction objective pour pénaliser les poids $\mathbb{W}$. L'équation (6) est le résultat de l'ajout de ce terme à la fonction de coût. Le coefficient λ permet de contrôler l'amplitude de la régularisation.

$$\arg\min_{\mathbb{W}} \frac{1}{m} \sum_{i} L\left(F\left(x^{(i)}; \mathbb{W}\right), y^{(i)}\right) + \lambda\Omega(\mathbb{W}) \tag{6}$$

Généralement, les fonctions de régularisation utilisées sont la régression Lasso [**108**] et la régression de Tikhonov, souvent appelées régularisation L_1 ou la norme L_2 respectivement. Le terme de pénalisation présent dans l'équation (6), pour la régularisation L_1 s'écrit :

$$\lambda\Omega(\mathbb{W}) = \lambda\|\mathbb{W}\|_1 \tag{7}$$

et pour la régularisation L_2 :

$$\lambda\Omega(\mathbb{W}) = \lambda\|\mathbb{W}\|_2 \tag{8}$$

Même si les termes des équations (7) et (8) pénalisent tous les deux $\mathbb{W}$, l'effet de ces deux formes de régularisation sur l'apprentissage est différent. En effet, l'utilisation de la norme L_2 aura pour effet de réduire les poids dont la covariance avec l'entrée est faible, favorisant ainsi les poids qui caractérisent au mieux la variance du modèle. Cela permet de rendre la fonction de coût le plus convexe possible et facilite la recherche du minimum global. Pourtant, l'utilisation de la norme L_1 aura pour effet de créer un modèle parcimonieux en faisant tendre une partie des poids du modèle vers zéro pour sélectionner uniquement une partie des neurones du réseau dans l'apprentissage et aussi dans l'inférence [**50**, **108**, **129**].

0.3.1.2 Batch normalization

Dans la phase d'apprentissage, les données sont généralement normalisées dans le but de les rendre comparables en termes de caractéristiques. Cependant, il y a un changement dans la distribution des activations du réseau en raison de la modification des paramètres du réseau pendant l'apprentissage. Dans ce contexte, la normalisation par lots (BN) est l'une des approches efficaces qui améliorent considérablement la convergence pendant l'apprentissage [**34**]. Elle consiste à normaliser en moyenne et en variance les sorties des couches du réseau.

Considérons un mini lot $B = \{X_1, X_2, \ldots, X_m\}$ de dimension m, et $\{Y_1, Y_2 \ldots, Y_m\}$ sont leurs transformations linéaires. La normalisation du lot $BN_{\gamma,\beta}$ se rapporte à la transformation :

$$\begin{aligned} BN_{\gamma,\beta}: \quad & \{X_1, X_2 \ldots, X_m\} \longrightarrow \{Y_1, Y_2 \ldots, Y_m\} \\ & X_i \longrightarrow Y_i = \gamma \cdot \hat{X}_i + \beta \end{aligned}$$

où $\hat{X}_i = \frac{X_i - \mu_B}{\sqrt{\sigma_B^2 + \varepsilon}}$ est la valeur normalisée de X_i et

$$(\mu_B, \sigma_B^2) = (\frac{1}{m}\sum_{i=1}^{m} X_i, \frac{1}{m}\sum_{i=1}^{m} (X_i - \mu_B)^2)$$

désignent la moyenne et la variance de X.

0.3.1.3 Dropout

Le dropout est une autre forme de régularisation couramment utilisée, facile à utiliser et permettant de réduire le risque de sur-apprentissage [**59**, **61**, **74**, **123**, **130**]. Son principe, décrit dans la figure 3, est de forcer à zéro la sortie de certains neurones choisis aléatoirement dans le réseau. Les neurones forcés à zéro, ou éteints, changent à chaque itération. Mathématiquement, si x et W désignent respectivement l'entrée et les poids d'une couche, alors la sortie de cette couche en appliquant le Dropout est exprimée par :

$$Y_{dropout} = m \circ f(W.x) \tag{9}$$

où $\circ$ désigne le produit de Hadamard, f est une fonction d'activation non linéaire, et m est un vecteur binaire, dans lequel chaque élément m_j est tiré indépendamment de la distribution de Bernoulli avec le paramètre ρ :

$$m_j \sim Bernoulli(\rho) \tag{10}$$

Cette technique permet de réduire le nombre important de liens entre les neurones et la spécialisation de certains au détriment d'autres, ce qui améliorera la généralisation et évitera le surapprentissage [**25**, **57**]. Le processus d'abandon génère différentes architectures dans chaque itération et la performance totale est définie par la moyenne des performances de ces architectures. Cela permet de réduire la variance entre ces modèles et d'améliorer la généralisation [**59**, **61**, **104**].

0.3.1.4 DropConnect

DropConnect est une technique de régularisation qui généralise le Dropout [**111**]. En effet, les connexions d'une couche entièrement connectée, plutôt que chaque unité de sortie, peuvent être abandonnées avec une probabilité ρ. DropConnect peut être considéré comme un générateur de sparsité dynamique dans le modèle (figure 4). En d'autres termes, la sparsité au sein du réseau est produite par des connexions qui sont choisies au hasard pendant l'étape d'apprentissage. Notez que le terme "dynamique" signifie que les

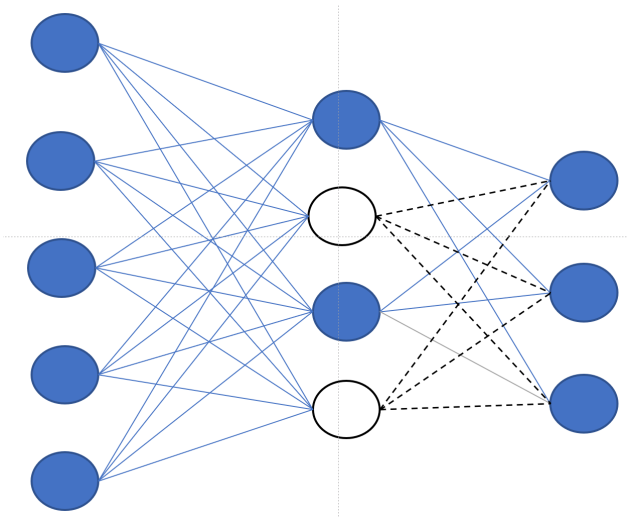

Figure 3 – Dropout régularisation

paramètres W ne sont pas fixes comme le cas dans les formes éparses classiques. Mathématiquement, la sortie d'une couche DropConnect peut être exprimée par :

$$Y_{DropConnect} = f((M \circ W).x) \tag{11}$$

où M est un masque binaire dont chaque élément, M_{ij} est tiré indépendamment de la distribution de Bernoulli avec le paramètre ρ :

$$M_{ij} \sim Bernoulli(\rho) \tag{12}$$

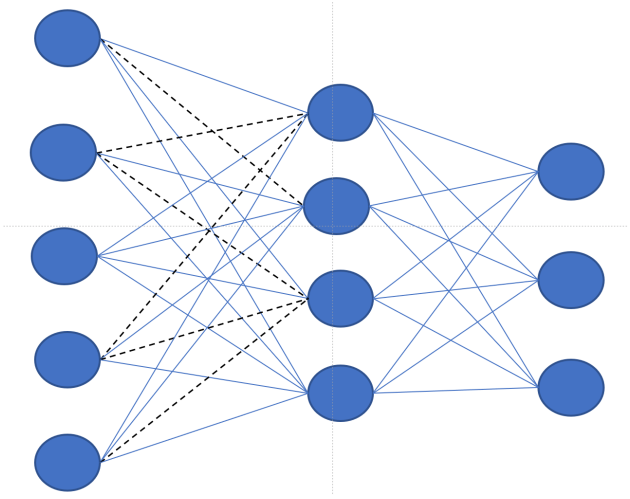

Figure 4 – DropConnect régularisation

Le choix de l'hyperparamètre ρ a été discuté analytiquement dans le cas du dropout comme dans [**35**, **54**, **104**], pourtant dans le cas de DropConnect, il n'est pas encore clair

et nécessite une analyse théorique dans le but de son estimation. Dans ce sens, nous exploitons les résultats surprenants, fournis par la complexité de Rademacher dans la minimisation de l'écart de généralisation, pour fournir une estimation de l'hyperparamètre de DropConnect.

0.3.2 Contrôle de l'écart de généralisation en vertu de la complexité de Rademacher pour DropConnect

Considérons un réseau de neurones profond composé de N couches cachées $\{\mathfrak{h}_1, \mathfrak{h}_2, ..., \mathfrak{h}_N\}$. Chaque couche cachée $\mathfrak{h}_k$ a n_k neurones, et $W^k = \{W_1^k, W_2^k, ..., W_{n_{k+1}}^k\}$ représente les poids entre les couches $\mathfrak{h}_k$ et $\mathfrak{h}_{k+1}$ avec $W_i^k \in \mathbb{R}^{n_k}$.
Soit $D = \{(x_1, y_1), (x_2, y_2), ..., (x_m, y_m)\}$ un ensemble de données. Dans la suite, nous commençons par donner quelques préliminaires théoriques essentiels et nécessaires pour développer une estimation de l'hyperparamètre de DropConnect.
Nous commençons par définir l'erreur et l'écart de généralisation pour un réseau de neurones profond.

Définition 2

L'erreur de généralisation du réseau en appliquant dropconnect est définie par :

$$R_{dc.exp} = E_{(X,Y),M}\Big[l\big(F(\mathbb{W}, X, M), Y\big)\Big] \tag{13}$$

et le risque empirique associé est défini par :

$$R_{dc.emp} = \frac{1}{m}\sum_{j=1}^{m} l(F(\mathbb{W}, x_j, M_j), y_j) \tag{14}$$

avec l et $F(\mathbb{W}, x_k, M_k)$ désignent respectivement la fonction de perte et la sortie du réseau pour l'échantillon k. $M_k = \{M_k^1, M_k^2, ..., M_k^N\}$ avec M_k^l représente le masque binaire pour l'échantillon k appliqué aux connexions entre $\mathfrak{h}_{l-1}$ et $\mathfrak{h}_l$ i.e,

$$M_k^l \in \mathcal{M}_{n_l, n_{l-1}}(\{0,1\}) \; ; \quad M_{kij}^l \sim Bernoulli(\rho)$$

En vertu de $R_{dc.exp}$ et $R_{dc.emp}$, l'écart de généralisation est exprimé par :

$$G_{gap} = |R_{dc.exp} - R_{dc.emp}| \tag{15}$$

Dans le cas de surapprentissage, $R_{dc.exp}$ est largement supérieur à $R_{dc.emp}$. par conséquent, l'écart de généralisation s'écrit sous forme

$$G_{gap} = R_{dc.exp} - R_{dc.emp}$$

.
En se basant sur la définition de la complexité de Rademacher [**53**, **103**, **104**], nous donnons la définition suivante :

Définition 3

La complexité empirique de Rademacher pour DropConnect et la complexité de Rademacher pour DropConnect sont respectivement définies par :

$$\widehat{\mathcal{R}}_m(\mathcal{G}, X, M) = E_\sigma\Big[\sup_{g\in\mathcal{G}} \frac{1}{m}\sum_{j=1}^{m} \sigma_j g(x_j, M_j)\Big] \tag{16}$$

$$\mathcal{R}_m(\mathcal{G}) = E_{X,M}\Big[\widehat{\mathcal{R}}_m(\mathcal{G}, X, M)\Big]$$

tels que $(\sigma_1, \sigma_2, ..., \sigma_m)$ sont des variables de Rademacher i.i.d. et $\mathcal{G}$ est un espace de fonctions à valeurs réelles.

Les lemmes suivants sont nécessaires pour l'optimisation de l'écart de généralisation. Ainsi, les détails de leurs preuves sont explicitement présentés dans le Chapitre 2.

Lemme 1

Pour un espace fonctionnel $\mathcal{G}$, on définit

$$absconv(\mathcal{G}) = \Big\{\sum \beta_k g_k : \quad g_k \in \mathcal{G} \text{ and } \sum|\beta_k| = 1\Big\}$$

Ensuite, nous avons:

$$\widehat{\mathcal{R}}_m(\mathcal{G}, X, M) = \widehat{\mathcal{R}}_m(absconv(\mathcal{G}), X, M) \tag{17}$$

Lemme 2

Si f désigne une fonction continue $L-$lipschitzienne avec $f(0) = 0$. Alors nous avons

$$E_\sigma\Big[\sup_{g\in\mathcal{G}} \frac{1}{m}\sum_{j=1}^{m} \sigma_j f(g(x^j, M^j))\Big] \leq LE_\sigma\Big[\sup_{g\in\mathcal{G}} \frac{1}{m}\sum_{j=1}^{m} \sigma_j g(x^j, M^j)\Big] \tag{18}$$

Lemme 3

Soit $l(.,.)$ une fonction continue $D-$lipschitzienne par rapport à sa première variable. Alors nous avons

$$\mathcal{R}_m\Big(l\big(F(\mathbb{W}, X, M), Y\big)\Big) \leq D\mathcal{R}_m\Big(F(\mathbb{W}, X, M), Y\Big) \tag{19}$$

Lemme 4

L'écart de généralisation est contrôlé par une quantité en fonction de la complexité de Rademacher pour DropConnect, exprimée par :

$$G_{gap} \leq 2\mathcal{R}_m\Big(l\big(F(\mathbb{W}, X, M), Y\big)\Big) + A\sqrt{\ln(2/\eta)/m} \tag{20}$$

où $1-\eta$ est la probabilité de tenir Eq. (20), et A est une borne constante de l.

Dans le paragraphe suivant, nous visons à construire une borne supérieure de $\mathcal{R}_m\Big(l\big(F(\mathbb{W}, X, M), Y\big)\Big)$ liée au masque de DropConnect M. Par la suite, nous minimisons cette borne en développant à la fin une estimation de l'hyperparamètre de DropConnect ρ.

0.3.3 Estimation de l'hyperparamètre de DropConnect

Théorème 1

Supposons que l'ensemble des paramètres $\mathbb{W}$ satisfait les hypothèses suivantes:

$$\begin{cases} \bullet & \|W^N\|_1 \leq B_N \\ \bullet & \max_k \|W_k^j\|_1 \leq B_j for 1 \leq k \leq N-1 \\ \bullet & \max_k \|W_k^q\|_2 \leq C_q \end{cases}$$

tels que B_j et C_q sont deux constantes. Alors,

$$\mathcal{R}_m\Big(l\big(F(\mathbb{W}, X, M), Y\big)\Big) \leq C(M) = \frac{DC_q}{m}\Big(\prod_{k=q}^{N} B_k\Big)L^{N-q}E_X\left[\sqrt{E_M\Big\|\sum_{j=1}^{m} M_{j,1}^q \odot \psi_j^q\Big\|_2^2}\right] \tag{21}$$

Proof.

On pose $\widehat{R}(M) = \widehat{\mathcal{R}}_m\Big(l\big(F(\mathbb{W}, X, M), Y\big)\Big)$, alors on a :

$$\begin{aligned} \widehat{R}(M) &= \widehat{\mathcal{R}}_m\Big(l\big(F(\mathbb{W}, X, M), Y\big)\Big) \\ &\leq D\widehat{\mathcal{R}}_m\Big(F(\mathbb{W}, X, M), Y\Big) \\ &= \frac{D}{m}E_\sigma\Big[\sup_{W^N}\sum_{j=1}^{m}\sigma_j F(\mathbb{W}, x^j, M^j)\Big] \\ &= \frac{D}{m}E_\sigma\Big[\sup_{W^N}\sum_{j=1}^{m}\sigma_j W^N.\psi_j^N\Big] \\ &\leq \frac{D}{m}B_N E_\sigma\Big[\sup_{W^N}\frac{W^N}{\|W^N\|_1}.\sum_{j=1}^{m}\sigma_j\psi_j^N\Big] \end{aligned}$$

en considérant $\frac{W_k^N}{\|W^N\|_1}$ comme β_k dans le Lemme 1, on obtient :

$$\widehat{R}(M) \leq \frac{D}{m} B_N E_\sigma \left[\sup_{W_1^{N-1}} \sum_{j=1}^{m} \sigma_j f\left((M_{j,1}^{N-1} \odot W_1^{N-1}).\psi_j^{N-1} \right) \right] \tag{22}$$

En combinant le lemme 2 et la propriété L-lipschitzienne de la fonction d'activation, on obtient :

$$\begin{aligned}
\widehat{R}(M) &\leq \frac{D}{m} B_N L E_\sigma \left[\sup_{W_1^{N-1}} \sum_{j=1}^{m} \sigma_j (M_{j,1}^{N-1} \odot W_1^{N-1}).\psi_j^{N-1} \right] \\
&= \frac{D}{m} B_N L E_\sigma \left[\sup_{W_1^{N-1}} \sum_{j=1}^{m} \sigma_j W_1^{N-1}.(M_{j,1}^{N-1} \odot \psi_j^{N-1}) \right] \\
&\leq \frac{D}{m} B_N B_{N-1} L E_\sigma \left[\sup_{W_1^{N-1}} \sum_{j=1}^{m} \sigma_j \frac{W_1^{N-1}}{\|W_1^{N-1}\|_1}.(M_{j,1}^{N-1} \odot \psi_j^{N-1}) \right]
\end{aligned}$$

en considérant $\frac{W_{1,k}^{N-1}}{\|W_1^{N-1}\|_1}$ comme β_k dans le lemme 1, on obtient:

$$\begin{aligned}
\widehat{R}(M) &\leq \frac{D}{m} B_N B_{N-1} L E_\sigma \left[\sup_{W_1^{N-1}} \sum_{j=1}^{m} \sigma_j M_{j,1,1}^{N-1} \psi_j^{N-1}(1) \right] \\
&= \frac{D}{m} B_N B_{N-1} L E_\sigma \left[\sup_{W_1^{N-1}} \sum_{j=1}^{m} \sigma_j M_{j,1,1}^{N-1} f\left((M_{j,1}^{N-2} \odot W_1^{N-2}).\psi_j^{N-2} \right) \right] \\
&= \frac{D}{m} B_N B_{N-1} L E_\sigma \left[\sup_{W_1^{N-1}} \sum_{j=1}^{m} \sigma_j f\left(M_{j,1,1}^{N-1} (M_{j,1}^{N-2} \odot W_1^{N-2}).\psi_j^{N-2} \right) \right]
\end{aligned}$$

A partir du lemme 2, on obtient :

$$\begin{aligned}
\widehat{R}(M) &\leq \frac{D}{m} B_N B_{N-1} L^2 E_\sigma \left[\sup_{W_1^{N-2}} \sum_{j=1}^{m} \sigma_j M_{j,1,1}^{N-2} (M_{j,1}^{N-2} \odot W_1^{N-2}).\psi_j^{N-2} \right] \\
&= \frac{D}{m} B_N B_{N-1} L^2 E_\sigma \left[\sup_{W_1^{N-2}} \sum_{j=1}^{m} \sigma_j W_1^{N-2}.(M_{j,1,1}^{N-1} (M_{j,1}^{N-2} \odot \psi_j^{N-2})) \right] \\
&\leq \frac{D}{m} B_N B_{N-1} B_{N-2} L^2 E_\sigma \left[\sup_{W_1^{N-2}} \sum_{j=1}^{m} \sigma_j \frac{W_1^{N-2}}{\|W_1^{N-2}\|_1}.(M_{j,1,1}^{N-1} (M_{j,1}^{N-2} \odot \psi_j^{N-2})) \right] \\
&= \frac{D}{m} B_N B_{N-1} B_{N-2} L^2 E_\sigma \left[\sup_{W_1^{N-3}} \sum_{j=1}^{m} \sigma_j M_{j,1,1}^{N-1} M_{j,1,1}^{N-2} f\left((M_{j,1}^{N-3} \odot W_1^{N-3}).\psi_j^{N-3} \right) \right] \\
&= \frac{D}{m} B_N B_{N-1} B_{N-2} L^2 E_\sigma \left[\sup_{W_1^{N-3}} \sum_{j=1}^{m} \sigma_j f\left(M_{j,1,1}^{N-1} M_{j,1,1}^{N-2} (M_{j,1}^{N-3} \odot W_1^{N-3}).\psi_j^{N-3} \right) \right]
\end{aligned}$$

Récursivement, on obtient le résultat suivant :

$$\widehat{R}(M) \leq \frac{D}{m}(\prod_{k=q}^{N} B_k)L^{N-q}E_\sigma\left[\sup_{W_1^q}\sum_{j=1}^{m}\sigma_j\Big(\prod_{s=q+1}^{N-1}M_{j,1,1}^{s}\Big)(M_{j,1}^{q}\odot W_1^q).\psi_j^q\right] \tag{23}$$

On suppose que DropConnect n'est appliqué qu'à la couche $\mathfrak{h}_q$, alors on a $M_{j,1,1}^{s}=1$ pour tout $s\neq q$ et par conséquent

$$\prod_{s=q+1}^{N-1}M_{j,1,1}^{s}=1$$

En effet,

$$\begin{aligned}\widehat{R}(M) &\leq \frac{D}{m}(\prod_{k=q}^{N} B_k)L^{N-q}E_\sigma\left[\sup_{W_1^q}\sum_{j=1}^{m}\sigma_j(M_{j,1}^{q}\odot W_1^q).\psi_j^q\right]\\ &= \frac{D}{m}(\prod_{k=q}^{N} B_k)L^{N-q}E_\sigma\left[\sup_{W_1^q}W_1^q.\sum_{j=1}^{m}\sigma_j M_{j,1}^{q}\odot\psi_j^q\right]\end{aligned}$$

En se basant sur l'inégalité de Cauchy-Schwarz, on obtient :

$$\widehat{R}(M) \leq \frac{DC_q}{m}(\prod_{k=q}^{N} B_k)L^{N-q}E_\sigma\left[\|\sum_{j=1}^{m}\sigma_j M_{j,1}^{q}\odot\psi_j^q\|_2\right] \tag{24}$$

Ensuite, la variante probabiliste de l'inégalité de Jensen, fournit une borne supérieure du terme $E_\sigma\left[\|\sum_{j=1}^{m}\sigma_j M_{j,1}^{q}\odot\psi_j^q\|_2\right]$. Celle-ci est développée de la manière suivante :

$$\begin{aligned}E(M) &= E_\sigma\left[\|\sum_{j=1}^{m}\sigma_j M_{j,1}^{q}\odot\psi_j^q\|_2\right]\\ &= E_\sigma\left[\sum_{j_1=1}^{m}\sum_{j_2=1}^{m}\sigma_{j_1}\sigma_{j_2}\Big(M_{j_1,1}^{q}\odot\psi_{j_1}^q\Big).\Big(M_{j_2,1}^{q}\odot\psi_{j_2}^q\Big)\right]^{1/2}\\ &\leq \left[\sum_{j_1=1}^{m}\sum_{j_2=1}^{m}E_{\sigma_{j_1}\sigma_{j_2}}\sigma_{j_1}\sigma_{j_2}\Big(M_{j_1,1}^{q}\odot\psi_{j_1}^q\Big).\Big(M_{j_2,1}^{q}\odot\psi_{j_2}^q\Big)\right]^{1/2}\\ &= \left\|\sum_{j=1}^{m}M_{j,1}^{q}\odot\psi_j^q\right\|_2\end{aligned}$$

La dernière égalité est obtenue en utilisant la propriété des variables de Rademacher qui vérifient $E_{\sigma_{j_1}\sigma_{j_2}}\sigma_{j_1}\sigma_{j_2}=0$ pour $j_1\neq j_2$ et $E_{\sigma_j\sigma_j}\sigma_j\sigma_j=1$.
Finalement, on obtient l'inégalité suivante :

$$\widehat{R}(M) \leq \frac{DC_q}{m}(\prod_{k=q}^{N} B_k)L^{N-q}\left\|\sum_{j=1}^{m}M_{j,1}^{q}\odot\psi_j^q\right\|_2 \tag{25}$$

qui est donc utilisé pour estimer la complexité de Rademacher pour DropConnect $R(M)$

$$R(M) \leq \frac{DC_q}{m}(\prod_{k=q}^{N} B_k)L^{N-q}E_{X,M}\left[\left\|\sum_{j=1}^{m} M_{j,1}^q \odot \psi_j^q\right\|_2\right] \tag{26}$$

$$= \frac{DC_q}{m}(\prod_{k=q}^{N} B_k)L^{N-q}E_{X,M}\left[\sqrt{\left\|\sum_{j=1}^{m} M_{j,1}^q \odot \psi_j^q\right\|_2^2}\right] \tag{27}$$

$$\leq \frac{DC_q}{m}(\prod_{k=q}^{N} B_k)L^{N-q}E_{X}\left[\sqrt{E_M\left\|\sum_{j=1}^{m} M_{j,1}^q \odot \psi_j^q\right\|_2^2}\right] \tag{28}$$

$$= C(M) \tag{29}$$

De même, nous démontrons que

$$R(M) \leq \frac{DC_q}{m}(\prod_{k=q}^{N} B_k)L^{N-q}E_{X}\left[\sqrt{E_M\left\|\sum_{j=1}^{m} M_{j,k}^q \odot \psi_j^q\right\|_2^2}\right] \tag{30}$$

$$\tag{31}$$

for any $1 \leq k \leq n_{q+1}$

□

Théorème 2

Dans les conditions du théorème précédent, nous avons

$$\min C(M) = \frac{\left(\sum_{k=1}^{n_q} \alpha(k)\right)^2}{n_q - \sum_{k=1}^{n_q} \rho_1^k} - \sum_{k=1}^{n_q}\left(\alpha(k)\right)^2 + \sum_{k=1}^{n_q}\beta(k) \tag{32}$$

Avec

$$\alpha(k) = \sqrt{\sum_{j=1}^{m}\left(\psi_j^q(k)\right)^2}, \qquad \beta(k) = \left(\sum_{j=1}^{m}\psi_j^q(k)\right)^2 \tag{33}$$

De plus, le minimum de la fonction $C(M)$ est atteint par:

$$\rho_1^k = 1 - \frac{\alpha(k)}{\max\limits_k \alpha(k)} \tag{34}$$

Proof. Afin de minimiser $C(M)$, nous utilisons l'expression équivalente notée par $\widehat{C}(M)$, avec

$$\widehat{C}(M) = E_M\left\|\sum_{j=1}^{m} M_{j,1}^q \odot \psi_j^q\right\|_2^2$$

En effet, le reste des paramètres dans $C(M)$ sont des constantes. De plus, $E_X[g(.)]$ et $g(.)$ ont la même monotonie lorsque la distribution de X est fixée. Ainsi, nous avons

$$\begin{aligned}
\widehat{C}(M) &= E_M \Big\| \sum_{j=1}^{m} M_{j,1}^q \odot \psi_j^q \Big\|_2^2 \\
&= \sum_{k=1}^{n_q} \Big[E_M \Big(\sum_{j=1}^{m} M_{j,1,k}^q * \psi_j^q(k) \Big)^2 \Big] \\
&= \sum_{k=1}^{n_q} \Big[Var \Big(\sum_{j=1}^{m} M_{j,1,k}^q * \psi_j^q(k) \Big) + \Big(E_M \Big(\sum_{j=1}^{m} M_{j,1,k}^q * \psi_j^q(k) \Big) \Big)^2 \Big] \\
&= \sum_{k=1}^{n_q} \Big[\sum_{j=1}^{m} Var \Big(M_{j,1,k}^q \Big) * \Big(\psi_j^q(k) \Big)^2 + \Big(\sum_{j=1}^{m} E_M \Big(M_{j,1,k}^q \Big) * \psi_j^q(k) \Big)^2 \Big] \\
&= \sum_{k=1}^{n_q} \Big[\sum_{j=1}^{m} \Big(E_M \Big((M_{j,1,k}^q)^2 \Big) - \Big(E_M (M_{j,1,k}^q) \Big)^2 \Big) * \Big(\psi_j^q(k) \Big)^2 + \Big(\sum_{j=1}^{m} E_M \Big(M_{j,1,k}^q \Big) * \psi_j^q(k) \Big)^2 \Big] \\
&= \sum_{k=1}^{n_q} \Big[\Big(\frac{1}{1-\rho_1^k} - 1 \Big) \sum_{j=1}^{m} \Big(\psi_j^q(k) \Big)^2 + \Big(\sum_{j=1}^{m} \psi_j^q(k) \Big)^2 \Big] \\
&= \sum_{k=1}^{n_q} \frac{1}{1-\rho_1^k} \sum_{j=1}^{m} \Big(\psi_j^q(k) \Big)^2 - \sum_{k=1}^{n_q} \sum_{j=1}^{m} \Big(\psi_j^q(k) \Big)^2 + \sum_{k=1}^{n_q} \Big(\sum_{j=1}^{m} \psi_j^q(k) \Big)^2
\end{aligned}$$

Remarque :
Pour simplifier les expressions, nous avons multiplié chaque masque M_j^k par $\frac{1}{1-\rho_j^k}$.
Par conséquence, nous avons

$$E_M(M_{j,1,k}^q) = 1 \quad and \quad E_M \Big((M_{j,1,k}^q)^2 \Big) = \frac{1}{1-\rho_1^k}$$

Nous posons

$$\alpha(k) = \sqrt{\sum_{j=1}^{m} \Big(\psi_j^q(k) \Big)^2}, \qquad \beta(k) = \Big(\sum_{k=1}^{n_q} \psi_j^q(k) \Big)^2 \tag{35}$$

$\widehat{C}(M)$ est ensuite reformulé comme suit

$$\widehat{C}(M) = \sum_{k=1}^{n_q} \frac{\Big(\alpha(k) \Big)^2}{1-\rho_1^k} - \sum_{k=1}^{n_q} \Big(\alpha(k) \Big)^2 + \sum_{k=1}^{n_q} \beta(k)$$

En utilisant l'inégalité de Cauchy-Schwarz, nous obtenons

$$\widehat{C}(M) \geq \frac{\left(\sum_{k=1}^{n_q} \alpha(k)\right)^2}{n_q - \sum_{k=1}^{n_q} \rho_1^k} - \sum_{k=1}^{n_q} \left(\alpha(k)\right)^2 + \sum_{k=1}^{n_q} \beta(k) \tag{36}$$

La dernière inégalité est vérifiée si et seulement si la condition suivante est satisfaite :

$$\frac{\alpha(1)}{1-\rho_1^1} = \frac{\alpha(2)}{1-\rho_1^2} = ... = \frac{\alpha(n_q)}{1-\rho_1^{n_q}} = \frac{\sum_{k=1}^{n_q} \alpha(k)}{n_q - \sum_{k=1}^{n_q} \rho_1^k} \tag{37}$$

Par conséquence, le minimum de $\widehat{C}(M)$ est exprimé par :

$$\min \widehat{C}(M) = \frac{\left(\sum_{k=1}^{n_q} \alpha(k)\right)^2}{n_q - \sum_{k=1}^{n_q} \rho_1^k} - \sum_{k=1}^{n_q} \left(\alpha(k)\right)^2 + \sum_{k=1}^{n_q} \beta(k) \tag{38}$$

Puisque $0 \leq \rho_1^k \leq 1$ et en utilisant l'hypothèse de Cauchy-Schwarz donnée par Eq. 37, le minimum de $\widehat{C}(M)$ peut être présenté comme suit :

$$\min \widehat{C}(M) = \max_k \alpha(k) * \sum_{k=1}^{n_q} \alpha(k) - \sum_{k=1}^{n_q} \left(\alpha(k)\right)^2 + \sum_{k=1}^{n_q} \beta(k) \tag{39}$$

$\widehat{C}(M)$ atteint son minimum si ρ_1^k satisfait

$$\rho_1^k = 1 - \frac{\alpha(k)}{\max_k \alpha(k)} \tag{40}$$

□

L'expression (40) représente une estimation dépendante des données de l'hyperparamètre de DropConnect ρ.

0.4 Contrôle et visualisation de la redondance dans les CNNs par l'optimisation combinatoire

Dans cette contribution, nous proposons une nouvelle vision de la régularisation basée sur l'optimisation combinatoire. En effet, un nouveau modèle d'optimisation est proposé pour contrôler et visualiser la redondance dans les réseaux convolutifs profonds. En outre, ce modèle est renforcé par la norme de Lasso L^1 pour renforcer la régularisation. Le modèle suggéré est un problème d'optimisation à variables mixtes dont la résolution est effectuée par la composition de la méthode de décente de gradient stochastique (SGD) et l'algorithme génétique (AG).

Table 1 – Notations

Variables et paramètres	
X	L'entrée du réseau
$\mathbb{W}$	L'ensemble des paramètres du CNN
W	Tenseur des poids
K	Tenseur des noyaux de convolution
C_l	Tenseur de sorties de la l ème couche de convolution
N_{conv}	Nombre des couches de convolution dans le réseau
N_{FC}	Nombre des couches entièrement connectées
N_l	Nombre des cartes de caractéristiques dans la l ème couche de convolution
n_l	Nombre de neurones dans la couche entièrement connectée l
$D \times D$	Taille des noyaux de convolution.
$conv - layer_I$	Ensemble de couches de convolution dans le réseau
$conv - layer_l$	la l ème couche de convolution
U	Variables de décision
$\alpha_\sigma(I)$	Ensemble ordonné des proportion de la redondance.

0.4.1 Nouveau modèle d'optimisation de la redondance dans les CNNs

Pour formaliser le modèle, nous avons besoin de définir les paramètres ci-dessous:

0.4.1.1 Fonction objective

L'idée derrière l'approche que nous proposons est de transformer le CNN standard avec sa fonction d'apprentissage basique, en un CNN amélioré avec une fonction d'erreur en variables mixtes. Pour cette raison, dans un premier temps, nous définissons la variable binaire suivante :

$$u_{ij}^k = \begin{cases} 0 & \text{si le noyau de convolution } K_{ij}^k \text{ ne contribue pas} \\ 1 & \text{sinon} \end{cases} \tag{41}$$

Son rôle principal est de contrôler la contribution des noyaux dans la tâche d'apprentissage en utilisant une approche combinatoire. Cet ensemble de variables est introduit dans le processus de convolution via la formule suivante :

$$C_l^k = f\left(\sum_{j=1}^{N_{k-1}} (u_{j,l}^k . K_{j,l}^k) * X_j^{k-1} \right) \tag{42}$$

Un exemple illustratif du fonctionnement de U dans la réduction de la redondance des noyaux dans le CNN est montré dans la figure 5. Dans lequel, pour l'ensemble des noyaux qui relaient une sortie d'une couche précédente et toute une couche actuelle, la solution obtenue U indique que seuls trois noyaux ne sont pas redondants.

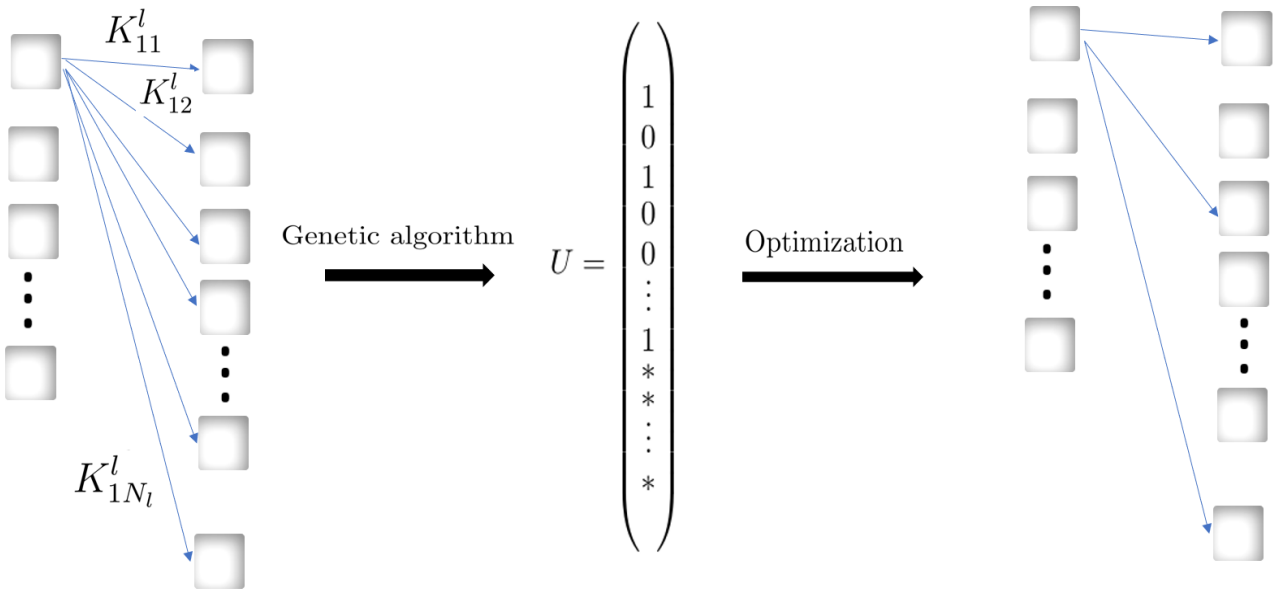

Figure 5 – Un exemple de l'utilisation de U pour réduire la redondance dans le CNN.

En tenant compte des variables U, le problème de réduction de la redondance des noyaux peut être formalisé par la formule :

$$\underset{\mathbb{W},U}{\operatorname{argmin}} \{E(W,U) = l\,(\mathbb{W},U) + \lambda\|U\|_1\} \tag{43}$$

où $l\,(\mathbb{W},U)$ est la fonction de perte liée à la précision (par exemple, la perte d'entropie croisée pour les tâches de classification), λ est un hyperparamètre scalaire qui contrôle l'équilibre entre le terme de perte et le terme de régularisation et $\|U\|_1$ est la norme l^1 de U, qui représente le nombre de noyaux contribuant effectivement à l'apprentissage du réseau.
Le problème d'optimisation ci-dessus peut être formalisé sous la forme d'une optimisation sous contrainte comme :

$$\underset{\mathbb{W},U}{\operatorname{argmin}} \{l\,(\mathbb{W},U) \quad \text{sous contrainte : } \|U\|_1 \leq R\}$$

où R est le niveau de sparsité (c'est-à-dire le nombre de noyaux non nuls).

0.4.1.2 Contraintes

L'utilisation de U pour les couches convolutionnelles nécessite une considération particulière. En effet, pour éviter l'élimination totale d'une carte de caractéristiques de sortie dans les couches convolutionnelles, il est nécessaire de conserver au moins un noyau dans chaque opération de convolution. Ainsi, nous proposons la contrainte suivante :

$$\sum_{i=1}^{N_{l-1}} u_{i,j}^l \geq 1 \tag{44}$$

pour $j = 1, ..., N_l$ et $l = 1, ..., N_{conv}$.

En ce qui concerne les variables du problème d'apprentissage $\mathbb{W}$, on peut les considérer comme un tenseur donné par $\mathbb{W} = K \cup W$.
où K représente l'ensemble des noyaux convolutifs (c'est-à-dire les paramètres de la partie d'extraction des caractéristiques):

$$K = \left((K_{i,j}^l)_{p,q} \right)_{\substack{l=1,...,N_{conv} \\ i=1,...,N_k \\ j=1,...,N_{k-1} \\ p,q=1,...,D}}$$

où $(K_{i,j}^k)_{p,q} \in \mathbb{R}$
et W c'est l'ensemble des poids (les paramètres de partie de classification) :

$$W = \left(W_{i,j}^k \right)_{\substack{k=1,...,N_{FC} \\ i=1,...,n_k \\ j=1,...,n_{k-1}}}$$

tel que $W_{i,j}^k \in \mathbb{R}$

0.4.1.3 Modèle d'optimisation obtenu

Le modèle d'optimisation proposé pour réduire la redondance dans les réseaux de neurones convolutionels est donné par:

$$(P) \begin{cases} \underset{\mathbb{W},U}{Min} \quad E(\mathbb{W}, U) = l(\mathbb{W}, U) + \lambda \|U\|_1 \\ \textit{Sous contraintes} : \\ \quad \sum_{i=1}^{N_{l-1}} u_{i,j}^l \geq 1 \quad \textit{pour} \;\; j = 1, ..., N_l \quad \textit{et} \; l = 1, ..., N_{conv}. \\ \quad \mathbb{W} = K \cup W \\ \quad K = \left((K_{i,j}^l)_{p,q} \right)_{\substack{l=1,...,N_{conv} \\ i=1,...,N_{l-1} \\ j=1,...,N_l \\ p,q=1,...,D}} \quad \textit{avec} \; (K_{i,j}^l)_{p,q} \in \mathbb{R} \\ \quad W = \left(W_{i,j}^l \right)_{\substack{l=1,...,N_{FC} \\ i=1,...,n_{l-1} \\ j=1,...,n_l}} \quad \textit{avec} \; W_{i,j}^l \in \mathbb{R} \\ \quad U = \left(u_{i,j}^l \right)_{\substack{l=1,...,N_{conv} \\ i=1,...,N_{l-1} \\ j=1,...,N_l}} \quad \textit{avec} \; u_{i,j}^l \in \{0, 1\} \end{cases}$$

0.4.2 Visualisation de la redondance dans les CNNs

Considérons l'ensemble $I = \{1, 2, ..., N_{conv}\}$ qui désigne les indices des couches convolutives dans un modèle CNN, le vecteur des uns $e^i = (1, 1, ..., 1)$ qui indique le vecteur des uns de même taille de U_i qui est $N_{i-1} \times N_i$. Nous définissons le taux de redondance dans chaque couche convolutionnelle du CNN comme suit :

$$R_i = \frac{||e^i - U_i^*||_1}{||e^i||_1} \qquad \text{pour } i = 1, 2, ..., N_{conv} \tag{45}$$

Ensuite, la proportion de redondance de chaque couche convolutionnelle est définie comme suit :

$$\alpha_i = \frac{R_i}{\sum_{j=1}^{N_{conv}} R_j} \qquad \text{pour } i = 1, 2, ..., N_{conv} \tag{46}$$

Considérons l'application définie par:

$$\begin{array}{rl} \sigma : & \{1, 2, .., N_{conv}\} \longrightarrow \{1, 2, .., N_{conv}\} \\ & i \longrightarrow \sigma(i) \end{array}$$

vérifiant $\alpha_{\sigma(i)} \geq \alpha_{\sigma(j)}$ pour $i < j$.

Nous obtenons donc l'ensemble

$$\alpha_{\sigma(I)} = \{\alpha_{\sigma(1)}, \alpha_{\sigma(2)}, ..., \alpha_{\sigma(N_{conv})}\} \tag{47}$$

qui représente un ensemble ordonné de proportions de redondance. Ainsi, la condensation d'une certaine proportion η de redondance peut être formulée comme le problème suivant :

$$s = \underset{m \in I}{Min}\{m \ / \ \sum_{j=1}^{m} \alpha_{\sigma(j)} > \eta\} \tag{48}$$

ce qui signifie que la proportion η de redondance est répartie sur les s premières couches $conv - layer_\sigma$.

0.4.3 Résolution du modèle d'optimisation obtenu

La résolution du modèle obtenu est divisée en deux phases : la minimisation et l'optimisation. Dans la première phase, nous utilisons une approche d'apprentissage pour entraîner les CNNs à l'aide d'un algorithme d'apprentissage spécifique, tel que la descente de gradient stochastique ou l'optimiseur d'Adam [**19**]. Dans la seconde phase, nous exploitons la solution $\mathbb{W}^*$ obtenue lors de la première tâche pour générer un nouveau problème d'optimisation $\min_U E(\mathbb{W}^*, U)$ qui sera résolu à l'aide de l'algorithme génétique évolutionnaire (figure 6).

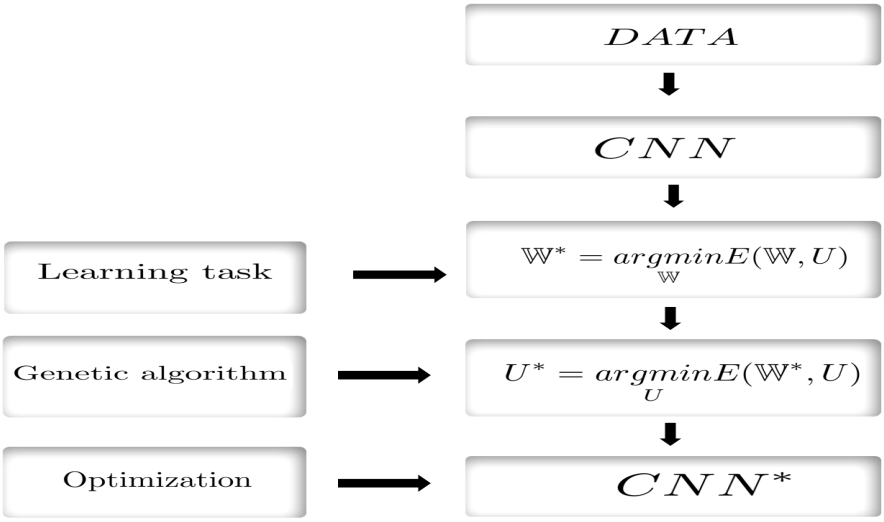

Figure 6 – Organigramme du modèle proposé

A la fin du processus de résolution, la solution obtenue $(\mathbb{W}^*, U^*)$ forme un CNN^* optimal avec un nombre de noyaux réduit (figure 6).
Les principales étapes de l'approche proposée sont présentées dans le pseudocode suivant :

Algorithme 2: Localisation-Optimisation de la redondance dans les CNNs

- **Entrée:**
  -CNN, Dataset
- **Sortie:**
  - Paramètres optimales $\mathbb{W}^*$ et U^* et la distribution de redondance ordonnée $\alpha_{\sigma(I)}$

— **Définie:**
  — Les variables de décision U en utilisant Eq. 41
  — La fonction objective $E(\mathbb{W}, U)$ en utilisant Eq. 43
— **Step 1: CNN Training**
  — $\mathbb{W}^* = \underset{\mathbb{W}}{Argmin} \quad E(\mathbb{W}, U)$.
— **Step 2 : CNN Redundancy optimization**
  — $U^* = \underset{U}{Argmin} \quad E(\mathbb{W}^*, U)$ en utilisant l'AG.
  — Calculer $\alpha_{\sigma(I)}$ en utilisant Eq. 45, 46, 47.
— **Return**
  — $CNN^* = (\mathbb{W}^*, U^*)$
  — $\alpha_{\sigma(I)}$

0.5 Optimisation des réseaux de neurones profonds pour la classification des séries chronologiques

La classification des séries temporelles (TSC) a commencé à susciter beaucoup d'attention récemment, principalement en raison des applications importantes des séries temporelles dans le monde réel, comme dans l'industrie financière **[64]**, environnement écologique **[28]** et énergies renouvelables **[115]**.
Plusieurs approches ont été développées pour la classification des séries temporelles. Ces approches peuvent être divisées en deux catégories : les approches classiques et les approches basées sur l'apprentissage profond **[1, 122, 128]**. Ces dernières ont apporté une amélioration très significative en comparaison avec les approches classiques **[39, 62]**. Pour cette raison, nous focalisons à développé une approche basée sur l'optimisation des réseaux de neurones profonds pour une meilleure classification des séries chronologiques. La méthode proposée consiste en deux phases (représentation et classification). Dans la première, la représentation par nuage normal est proposée pour transformer la série temporelle brute en une représentation bidimensionnelle générant à la fin un ensemble de caractéristiques pertinentes de la série. Dans la phase de classification, un réseau de neurones profond est proposé et renforcé par un modèle d'optimisation liée à son architecture, pour éliminer les paramètres redondants et générer à la fin une classification robuste des séries chroniques.
Pour clarifier l'approche proposée, nous proposons, dans la section suivante, quelques préliminaires liés à la représentation des séries chronologiques.

0.5.1 Préliminaires

Au cours des deux dernières décennies, la représentation et la classification des séries chronologiques ont été considérées comme deux des problèmes les plus difficiles de l'exploration de données. Dans un premier temps, nous donnons les définitions suivantes.

Définition 4

Une série temporelle X est une séquence de valeurs réelles $X = (x_1, x_2, ..., x_n)$; où n est la longueur de la série X.

Définition 5

Soit X une série temporelle de longueur n. Une représentation de série temporelle X est définie comme un modèle $\bar{X}$ de dimension réduite m $(m \ll n)$, où $\bar{X}$ contient les caractéristiques pertinentes de X ou se rapproche étroitement de X.

Diverses approches de représentation des séries temporelles ont été proposées dans la littérature **[47]**. Parmi ces approches, la représentation par nuage est l'un des modèles de représentation les plus efficaces et les plus difficiles. Ce dernier a été introduit par Li **[40]** comme un nouveau modèle cognitif de l'incertitude qui est basé sur la théorie des probabilités et de la théorie des ensembles flous. L'un des modèles de nuage les plus importants est le modèle de nuage normal, dans lequel les phénomènes satisfont à la

distribution normale [**12**]. Le modèle du nuage normal est défini par :

Définition 6

Soit U un ensemble universel décrit par des nombres précis et $C(Ex, En, He)$ un concept qualitatif relatif à U. Si un nombre quantitatif $x(x \in U)$ est une réalisation aléatoire du concept C, et x suit la distribution normale

$$x \sim N\left(Ex, En'^2\right)$$

où

$$En' \sim N\left(En, He^2\right)$$

et le degré de certitude de x sur U est

$$\mu(x) = e^{-\frac{(x-Ex)^2}{2(En')^2}}$$

alors la distribution de x sur U est un nuage normal, et chaque x est défini comme une goutte de nuage.

Le modèle de nuage normal possède les propriétés suivantes :

. L'ensemble universel U peut être unidimensionnel ou multidimensionnel, et le modèle de nuage normal unidimensionnel possède la "courbe d'espérance"

$$y = \exp\left\{-\frac{(x-Ex)^2}{2(En)^2}\right\}$$

. Pour tout $x \in U$, la correspondance de x à $[0, 1]$ est multiple. Le degré de certitude de x sur C est une distribution normale plutôt qu'un nombre fixe.

. Le nuage est composé de gouttes de nuage. Plus il y a de gouttes de nuage, mieux les caractéristiques globales de ce concept sont représentées.

. Plus la goutte de nuage x est probable, plus le degré de certitude est élevé et donc plus la contribution de la goutte de nuage x au concept est importante.

Ces caractéristiques des nuages normaux signifient qu'il est raisonnable d'utiliser le nuage normal bidimensionnel pour représenter les séries temporelles. Le nuage normal peut intégrer efficacement le caractère aléatoire et flou des concepts via les trois paramètres numériques suivants :

. Ex (Expectation). L'espérance mathématique des gouttes du nuage distribuées dans l'ensemble universel. En d'autres termes, Ex est le point le plus représentatif du concept qualitatif.

. En (Entropie). La mesure de l'incertitude du concept qualitatif. En est déterminée à la fois par le caractère aléatoire et le caractère flou du concept.

. He (Hyper-entropie). He est la mesure de l'incertitude de l'entropie.

Le modèle de nuage normal bidimensionnel est une extension du modèle de nuage normal, qui peut être défini comme suit :

Définition 7

Soient $U\{X, Y\}$ un ensemble universel à deux dimensions décrit par des nombres précis et $C(Ex, Ey, Enx, Eny, Hex, Hey)$ un concept qualitatif relatif à U. Si un nombre quantitatif $(x, y), (x \in X, y \in Y)$ est une réalisation aléatoire du concept C, et (x, y) suit la distribution normale

$$(x, y) \sim N\left((Ex, Ey), \left(Enx'^2, Eny'^2\right)\right)$$

où $\left(\text{ En}x'^2, \text{En}y\ '^2\right)$ est une réalisation aléatoire d'une distribution normale bidimensionnelle

$$\left(\text{ Enx}'^2,\ \text{Eny}'^2\right) \sim N\left(\left(\text{ Enx }, \text{ Eny }\right), \left(\text{ Hex}^2,\ \text{Hey}^2\right)\right)$$

et le degré de certitude de (x, y) sur U est

$$\mu(x, y) = \exp\left\{-\frac{(x - Ex)^2}{2\left(Enx'\right)^2} - \frac{(y - Ey)^2}{2\left(Eny'\right)^2}\right\}$$

alors la distribution de (x, y) sur U est un nuage normal à deux dimensions, et chaque (x, y) est défini comme une goutte de nuage.

Deux transformations de nuage, à savoir la transformation de nuage en avant (FCT) et la transformation de nuage en arrière (BCT), sont utilisées pour réaliser la transformation cognitive bidirectionnelle entre l'intension et l'extension d'un concept [**31**]. La FCT est utilisée pour effectuer la transformation de l'intension à l'extension d'un concept, tandis que la BCT met en œuvre la transformation de l'extension à l'intension [**99**]. L'algorithme bidimensionnel de transformation de nuage normal en arrière (T-BNCT) est décrit par l'algorithmes 2.

0.5.2 Représentation des séries chronologiques

Encouragés par l'efficacité de la représentation de nuage, nous proposons d'utiliser une représentation bidimensionnelle basée sur le modèle de nuage, appelée représentation normale de nuage en deux dimensions [**99**], pour transformer une série chronologique brute en une représentation 2D. Cette représentation peut être divisée en quatre étapes fondamentales comme l'illustre la figure 7.

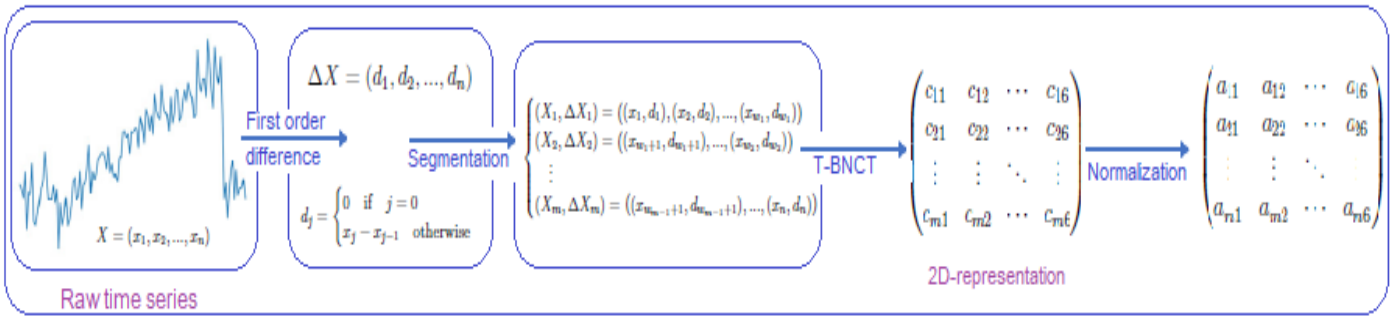

Figure 7 – La représentation des séries temporelles proposée

Considérons une série temporelle brute $X = (x_1, x_2, ..., x_n)$ et m est la dimension réduite. Les quatre étapes de la représentation suggérée sont décrites comme suit :

- Étape 1 : Calculer la différence de premier ordre ΔX

$$\Delta X = (d_1, d_2, ..., d_n)$$

où

$$d_j = \begin{cases} 0 & \text{if} \quad j = 1 \\ x_j - x_{j-1} & \text{sinon} \end{cases}$$

- Étape 2 : Segmentation de X et de ΔX en m segments simultanément.

$$\begin{cases} (X_1, \Delta X_1) = \big((x_1, d_1), (x_2, d_2), ..., (x_{w_1}, d_{w_1})\big) \\ (X_2, \Delta X_2) = \big((x_{w_1+1}, d_{w_1+1}), ..., (x_{w_2}, d_{w_2})\big) \\ \quad \vdots \\ (X_m, \Delta X_m) = \big((x_{w_{m-1}+1}, d_{w_{m-1}+1}), ..., (x_n, d_n)\big) \end{cases}$$

- Étape 3 : Effectuer la représentation du nuage normal 2D :
  Transformer chaque paire de sous-séquences $(X_j, \Delta X_j)$ en un nuage normal 2D C_j en utilisant l'algorithme T-BNCT décrit ci-après (Algorithme 3).
  Ensuite, la matrice des caractéristiques C est obtenue en considérant chaque nuage normal 2D C_j comme la $j$ème ligne de la matrice C.
- Etape 4 : Normalisation
  La dernière étape de la phase de représentation permet de convertir la matrice des caractéristiques C en valeurs de gris. Pour cela, nous appliquons la normalisation min-max afin d'obtenir la matrice finale A comme exprimé ci-dessous :

$$A = \frac{C - \min(C)}{\max(C) - \min(C)} \times 255$$

L'algorithme T-BNCT utilisé dans la troisième étape est résumé dans le pseudocode suivant, tel que présenté dans [**100**]:

Algorithme 3: T-BNCT

- **Entrée:**

  Echantillons (x_i, y_i) $(i = 1, 2, \ldots, n)$.

- **Sortie:**

  Caractéristiques numériques $(Ex, Ey, Enx, Eny, Hex, Hey)$.

Étape 1 : Calculer la moyenne et la variance pour chaque dimension de l'échantillon, c'est-à-dire,

$$\bar{X} = \frac{1}{n}\sum_{i=1}^{n} x_i, \quad Sx^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{X})$$

$$\bar{Y} = \frac{1}{n}\sum_{i=1}^{n} y_i, \quad Sy^2 = \frac{1}{n-1}\sum_{i=1}^{n}(y_i - \bar{Y})$$

Étape 2 : Calculer l'espérance Ex et Ey :

$$Ex = \bar{X}, \quad Ey = \bar{Y}$$

Étape 3 : Calcul de l'entropie Enx et Eny :

$$\text{Enx} = \sqrt{\frac{\pi}{2}} \times \frac{1}{n}\sum_{i=1}^{n} |x_i - Ex|$$

$$\text{Eny} = \sqrt{\frac{\pi}{2}} \times \frac{1}{n}\sum_{i=1}^{n} |y_i - Ey|$$

Étape 4 : Calculer l'hyper-entropie Hex et Hey :

$$Hex = \sqrt{Sx^2 - Enx^2}$$

$$Hey = \sqrt{Sy^2 - Eny^2}$$

0.5.3 Classification-Optimisation des séries chronologiques

Dans cette phase, nous commençons par décrire le modèle d'optimisation pour la réduction de la redondance dans le CNN, puis nous l'adaptons au TSC en utilisant la représentation normale des nuages.

0.5.3.1 Modèle d'optimisation de la redondance des paramètres pour la classification des séries temporelles

L'utilisation de *CNN* pour des tâches de classification est généralement appliquée en utilisant les paramètres par défaut, c'est-à-dire sans étude de la redondance dans ces modèles. De plus, la distribution de la redondance dans les CNNs n'est pas clairement décrite [**78**]. Dans cette section, l'utilisation du CNN pour la classification des séries temporelles est renforcée par un modèle d'optimisation afin de contrôler la redondance de ses paramètres. Le modèle proposé présente une nouvelle vision de la localisation et de la réduction de la redondance en définissant un ensemble de variables de décision.

$$u_{ij}^{k} = \begin{cases} 0 & \text{si le noyau de convolution } K_{ij}^{k} \text{ ne contribue pas} \\ 1 & \text{sinon} \end{cases}$$

$$v_{ij}^{k} = \begin{cases} 0 & \text{si le poids } W_{ij}^{k} \text{ ne contribue pas} \\ 1 & \text{sinon} \end{cases}$$

Le rôle principal de ces variables est de contrôler la contribution de chaque paramètre dans l'apprentissage en utilisant une approche combinatoire. Cet ensemble de variables est techniquement introduit dans le processus de propagation comme suit

$$C_{l}^{k} = f\left(\sum_{j=1}^{N_{k-1}} (u_{j,l}^{k}.K_{j,l}^{k}) * X_{j}^{k-1} \right) \tag{49}$$

$$Z_{j}^{k} = f\Big(\sum_{i=1}^{n_{k-1}} (v_{ji}^{k} \times W_{ji}^{k}) \times Z_{i}^{k-1} \Big) \tag{50}$$

En prenant en considération les variables U et V, le problème de réduction de la redondance des paramètres peut être formalisé comme suit :

$$\min_{\mathbb{W},U,V} \{ l\left(\mathbb{W}, U, V\right) \}$$

où $l\left(\mathbb{W}, U, V\right)$ est une fonction de perte (par exemple, la perte d'entropie croisée).

•Contraintes

L'intégration des variables binaires U et V dans le réseau nécessite une réflexion spécifique. En effet, dans le processus de convolution, la génération d'une carte de caractéristiques de sortie nécessite au moins un noyau convolutif. Mathématiquement, ceci peut être exprimé comme suit

$$\sum_{i=1}^{N_{h-1}} u_{i,j}^{h} \geq 1$$

pour $j = 1, ..., N_l$ et $h = 1, ..., N_{conv}$.

De plus, pour éviter la suppression totale des connexions entre neurones dans les couches de classification, il est nécessaire de conserver au moins une connexion pour chaque neurone. Ainsi, cette contrainte peut être formulée comme suit :

$$\sum_{i=1}^{n_{h-1}} v_{i,j}^{h} \geq 1$$

pour $j = 1, ..., n_h$ et $h = 1, ..., N_{FC}$.

Les paramètres de CNN $\mathbb{W}$ représentent une union de deux tenseurs K et W. Ainsi, K présente les variables de CNN dans la partie d'extraction des caractéristiques.

$$K = \left((K_{i,j}^{l})_{p,q} \right)_{\substack{l=1,...,N_{conv} \\ i=1,...,N_k \\ j=1,...,N_{k-1} \\ p,q=1,...,D}}$$

Chaque élément de ce tenseur est une variable réelle $(K_{i,j}^{k})_{p,q} \in \mathbb{R}$
Le deuxième tenseur de paramètres W est composé de variables du CNN au sein de couches entièrement connectées :

$$W = \left(W_{i,j}^{k} \right)_{\substack{k=1,...,N_{FC} \\ i=1,...,n_k \\ j=1,...,n_{k-1}}}$$

tel que $W_{i,j}^{k} \in \mathbb{R}$

• **Modèle d'optimisation obtenu pour la classification des séries temporelles**

Le modèle d'optimisation proposé pour la classification des séries chronologiques est donné par:

$$
(P)\begin{cases}
\underset{\mathbb{W},U,V}{Min} \quad E(\mathbb{W},U,V) = l\left(\mathbb{W},U,V\right) \\
\textit{Sous contraintes} : \\
\sum\limits_{i=1}^{N_{h-1}} u_{i,j}^{h} \geq 1 \quad \textit{pour} \;\; j = 1,...,N_h \quad \text{et} \;\; h = 1,...,N_{conv}. \\
\sum\limits_{i=1}^{n_{h-1}} v_{i,j}^{h} \geq 1 \quad \textit{pour} \;\; j = 1,...,n_h \quad \text{et} \;\; h = 1,...,N_{FC}. \\
\mathbb{W} = K \cup W \\
K = \left((K_{i,j}^{h})_{p,q} \right)_{\substack{h=1,...,N_{conv} \\ i=1,...,N_{h-1} \\ j=1,...,N_h \\ p,q=1,...,D}} \quad \text{où} \quad (K_{i,j}^{h})_{p,q} \in \mathbb{R} \\
W = \left(W_{i,j}^{h} \right)_{\substack{h=1,...,N_{FC} \\ i=1,...,n_{h-1} \\ j=1,...,n_h}} \quad \text{où} \quad W_{i,j}^{h} \in \mathbb{R} \\
U = \left(u_{i,j}^{h} \right)_{\substack{h=1,...,N_{conv} \\ i=1,...,N_{h-1} \\ j=1,...,N_h}} \quad \text{où} \quad u_{i,j}^{h} \in \{0,1\} \\
V = \left(v_{i,j}^{h} \right)_{\substack{h=1,...,N_{FC} \\ i=1,...,n_{h-1} \\ j=1,...,n_h}} \quad \text{où} \quad v_{i,j}^{h} \in \{0,1\}
\end{cases}
$$

Le modèle obtenu appartient au problème de programmation non linéaire à variables mixtes (MINLP) avec les paramètres de CNN comme variables continues et les variables d'optimisation binaires.
La résolution du modèle proposé est effectuée en utilisant deux tâches : minimisation et optimisation. Dans la première, un algorithme d'apprentissage spécifique, tel que la descente de gradient stochastique, est utilisé pour l'établissement des paramètres. Dans la seconde, nous optons pour l'algorithme génétique évolutionnaire pour résoudre la partie combinatoire.
Le CNN^* obtenu est ensuite utilisé pour classifier les données reçues de la représentation normale de nuages. L'approche proposée pour la classification des séries temporelles est présentée explicitement dans la figure 8.

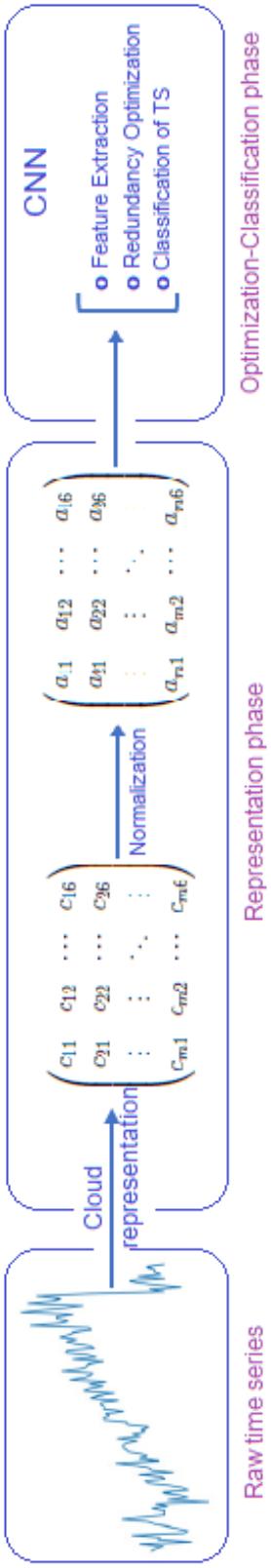

Figure 8 – L'approche proposée pour la classification des séries temporelles

0.6 Conclusion

Depuis son invention, l'apprentissage profond ou le deep learning a révolutionné le domaine de l'intelligence artificielle et devenu un axe de recherche très actif. Ceci revient à ses réalisations incontournables dans tous les secteurs du monde réel. Grâce à ces propriétés émergentes, l'apprentissage profond est devenu l'alternative la plus privilégiée pour la résolution des problèmes difficiles en machine learning. Les réseaux de neurones profonds sont à la base de divers progrès accomplis par l'apprentissage profond. Parmi ces réseaux, le réseau de neurones convolutif (CNN) a apporté des améliorations majeures au domaine de l'intelligence artificielle. C'est un modèle dont l'architecture et le fonctionnement sont inspirés du cortex visuel, et dont son développement nécessitait des études mathématiques approfondies. Ce réseau qui intègre une phase d'extraction des caractéristiques dans le processus d'apprentissage, a réalisé des tâches impressionnantes dans le domaine des technologies modernes. Cependant, les réseaux de neurones convolutifs performants nécessitent souvent un grand nombre de paramètres. Par conséquence, l'apprentissage de ces réseaux peut produire des phénomènes indésirables, notamment, le sur-apprentissage, la redondance des paramètres, et un coût de calcul très élevé. Pour faire face à ces défis, plusieurs méthodes de régularisation ont été développées. Parmi ces méthodes, on cite L^1 et L^2 régularisation, Dropout, DroConnect et d'autres qui ont été proposées dans le but d'éviter le sur-apprentissage et améliorer la performance des modèles.
Dans cette thèse, nous avons développé et amélioré quelques méthodes de régularisation existantes. Ensuite, nous avons proposé une vision différente de la régularisation, basée sur la réduction de la redondance en utilisant l'optimisation combinatoire. Ainsi que nous avons apporté une nouvelle approche pour la classification des séries temporelles, basée sur l'optimisation et la régularisation des réseaux de neurones profonds.

Dans le **Chapitre 1**, nous avons présenté les réseaux de neurones et l'apprentissage profonds. Cette représentation montre toute la diversité et la richesse des travaux qui s'inscrivent dans ce domaine. Tout d'abord, nous avons décrit le concept du deep learning. Après, nous avons introduit le noyau des réseaux profonds qui est le réseau de neurones convolutif, ainsi que les composantes principales qui construisent son architecture. Ensuite, nous avons expliqué les fondements théoriques de l'apprentissage des réseaux profonds, en introduisant les différents algorithmes d'optimisation qui ont montré leurs efficacités dans plusieurs contextes.

Dans le **Chapitre 2**, nous avons présenté la vision mathématique de la régularisation et la généralisation des réseaux de neurones profonds. Dans la première section, nous avons introduit les différentes méthodes de régularisation connues efficaces et compétitives. Dans la deuxième section, nous avons développé une analyse théorique de la généralisation en se basant sur la complexité de Rademacher.

Dans le **Chapitre 3**, nous avons proposé une estimation de l'hyperparamètre de la méthode de régularisation DropConnect. En effet, en se basant sur la théorie exposée dans le Chapitre 2, nous avons dérivé une borne de l'écart de généralisation en fonction de la

complexité de Rademacher. Cette borne est ensuite investie pour estimer l'hyperparamètre de DropConnect. Les résultats obtenus par la méthode de régularisation proposée sont satisfaisants numériquement et visuellement.

Le **Chapitre 4** a concerné une nouvelle vision de la régularisation basée sur l'optimisation combinatoire. En effet, nous avons proposé un modèle d'optimisation pour contrôler et visualiser la redondance dans les réseaux convolutifs profonds. De plus, nous avons intégré la norme L^1 pour renforcer et contrôler le niveau de sparsité dans le réseau. Le modèle suggéré est un problème d'optimisation avec variables mixtes et contraintes linéaires. Pour la résolution du problème obtenu, nous avons proposé l'algorithme de descente de gradient stochastique (SGD) et ses variantes, pour résoudre la partie continue, ainsi que l'algorithme génétique est investi pour l'optimisation combinatoire.

Suite au succès de l'apprentissage profond dans divers secteurs, nous avons décidé de traiter l'une de ses applications impressionnantes comme dernière contribution de cette thèse. En effet, dans le **Chapitre 5**, nous avons développé une approche basée sur l'apprentissage profond pour la classification des séries chronologiques. Dans ce contexte, nous avons introduit les séries chronologiques, ainsi que leurs propriétés principales dans la première section. Après, nous avons introduit une représentation efficace pour transformer une série chronologique à une matrice de caractéristiques pertinentes. Ensuite, nous avons proposé un réseau de neurones profond renforcé par un modèle d'optimisation pour réduire la redondance des paramètres. L'approche proposé a généré à la fin un modèle optimal capable de classifier les séries temporelles avec une meilleure efficacité.

Perspectives

Les résultats de recherche issus de cette thèse ont ouvert plusieurs directions de recherche, qui ont un champ d'investigations plus approfondies. En effet, l'estimation de l'hyperparamètre de DropConnect utilise une analyse liée au risque pour optimiser l'écart de généralisation, ce qui fournit une piste de réflexion importante pour la suite de la recherche. Dans ce sens, nos travaux futurs incluent la dérivation d'une estimation plus compacte de l'écart de généralisation et le développement d'approches de régularisation plus stables et précises. De plus, on observe que le modèle d'optimisation de la redondance proposé se focalise sur la réduction des noyaux redondants. Cependant, d'autres paramètres du réseau sont en redondance. Dans ce contexte, nous souhaitons développer d'autre modèles d'optimisation pour réduire plus les phénomènes indésirables tels que le surapprentissage et la redondance des paramètres. Enfin, l'application traitée comme dernière contribution génère plusieurs suggestions sur les futures recherches dans la classification des séries temporelles et les réseaux de neurones profonds.

Introduction

During the last decade, deep learning, in particular, deep neural networks have demonstrated a great performance in many challenging machine learning and artificial intelligence tasks such as classification, clustering and prediction [**87**]. These models have drawn much attention and led to great researches in this field including deep boltzmane machine (DBM), which was first proposed as a concept by Smolensky, and published by Hinton [**9**]. Deep bleif networks (DBN) proposed by Hinton [**9**] and convolutional neural network (CNN) introduced firstly for computer vision task by LeCun [**71**]. An overview of other deep neural networks has been presented in [**16**].These networks have achieved tremendous success and significant performance in various applications in real research fields [**5**, **29**, **48**, **66**, **127**].

Deep convolutional neural networks (CNNs) have become the most influential learning models that has shown great results and considerable success in many applications such as object detection for medical images [**40**], time series forecasting [**5**], microscopy image analysis [**24**], building energy consumption forecast [**3**], automatic ship classification [**32**], etc. The reason behind this high performance is due to its special characteristics such as parameters sharing, localized sparse connectivity, and the features extraction using the convolution operator [**71**].

However, in order to reach a good performance of prediction and achieve more challenging goals, the CNN architecture becomes more complicated and contains a large number of parameters. As a result, training a deep CNN becomes more difficult and brings three fundamental problems; Firstly, the training and the inference are cumbersome and take a long time [**107**]. Secondly, the training leads to overfitting problem [**74**], and finally, the redundancy problem that slows the network performance. In fact, as mentioned in [**86**, **101**], some weights learned in the networks are correlated, which means that some of the weights can either be predicted or be unnecessary to learn.

To deal with these issues, several regularization methods have been developed to improve the generalization of deep networks. These methods can be divided into two classes: regularization approaches based on objective function penalization and stochastic regularization methods. L^1 and L^2 represent two competitive methods of the first class [**50**]. Stochastic regularization consists in injecting random variables into the model to strengthen its generalization. Dropout and DropConnect are two random regularization methods that have achieved very significant performances [**74**, **111**].

The aim of this thesis is to elaborate the regularization problem in deep neural networks by several approaches. First, we focus on stochastic regularization methods and propose an estimation of DropConnect hyperparameters. The second vision is devoted to the control and visualization of redundancy in CNNs using a combinatorial optimization model. As a last contribution, an optimized deep model is proposed for time series classification. This thesis is structured as follows:

Chapter 1 is devoted to the presentation of Learning and Deep Neural Networks. This presentation shows the diversity and richness of the work in this field. First, we describe the concept of deep learning. Then, we introduce the core of deep networks which is the convolutional neural network, as well as the main components that build its architecture. Then, we explain the theoretical foundations of deep network learning, introducing the different optimization algorithms that have shown their efficiency in several contexts.

Chapter 2 focuses on presenting the mathematical vision of regularization and the generalization of deep neural networks. In the first section, we introduce the different known efficient and competitive regularization methods. In the second section, we develop a theoretical analysis of generalization based on Rademacher's complexity.

In **Chapter 3**, we propose an estimate of the hyermerameters of the DropConnect regularization method. Based on the theory outlined in Chapter 3, we will extract a generalization gap bound based on Rademacher complexity. Then, we propose an estimate for the DropConnect hyperparameters by minimizing the bound on the generalization gap. The results obtained by the proposed regularization method are numerically and visually satisfactory.

Chapter 4 concerns a new vision of regularization based on combinatorial optimization. Indeed, we propose an optimization model to control and visualize redundancy in deep convolutional networks. Thus, we integrate the L^1 norm to strengthen the regularization. The suggested model is an optimization problem with mixed variables and linear constraints. For the solution of the obtained problem, we propose the stochastic gradient decent (SGD) algorithm and its extensions to solve the continuous part, thus, the genetic algorithm is invested for combinatorial optimization.

Chapter 5 deals with a very important application of deep neural networks, that is the classification of time series. In this context, we introduce time series and their main properties in the first section. Then, we propose an efficient representation to transform a time series to a matrix of relevant features. Then, we propose a deep neural network enhanced by an optimization model to reduce redundancy in the proposed network.

The last chapter concludes the work developed in this thesis and proposes future research directions in deep neural networks regularization and time series analysis.

Chapter 1

Deep Learning

1.1 Introduction

Deep learning allows computational models that are composed of multiple processing layers to learn representations of data with multiple levels of abstraction [**45**, **46**, **49**]. These methods have dramatically improved the state-of-the-art in speech recognition, visual object recognition, object detection and many other domains such as drug discovery and genomics [**29**, **48**, **49**]. Furthermore, its outstanding ability to learn very complex, both linear and non-linear relationships directly from the data makes deep neural network perfectly suitable to perform intelligent tasks similar to those performed by human brain [**45**]. Recently, Convolutional neural network (CNN), one of the popular deep learning models which originally introduced for computer vision task, has constantly shown promising results in large areas and applications [**66**, **71**, **107**]. With the advancement of GPU-based parallel computing and the availability of large public datasets on the Internet, such as Imagenet, CNN marks its superiority over the conventional computer vision approaches in large competition [**74**]. Furthermore, due to the large number of parameters, training of deep neural networks is cumbersome and takes a lot of time, sometimes training a neural network may take days or even weeks. This limits the application of deep learning system in various research fields such as big data. Because of these reasons, there is a need for efficient and improved training speed to carry out such application especially when considering the parallelization of CNN's. Stochastic gradient descent (SGD) and its variants are the most widely used algorithms for the training of deep learning models [**18**, **22**, **79**].
In this chapter, we first gives some generalities about deep learning and especially deep neural networks. Next, we introduce convolutional neural network as one of the most important and successful deep learning models. Loss functions widely used in deep neural networks are introduced in Section 1.4. Optimization as an extremely challenging task, is object of Section 1.5, in which we firstly introduce the formulation of optimization for training deep neural networks, then, we present competitive optimization algorithms and we highlight advantages and drawbacks of each algorithm.

1.2 Deep Learning generalities

1.2.1 What Is Deep Learning?

Deep learning has been a challenge to define for many because it has changed forms slowly over the past decade. One useful definition specifies that deep learning deals with a "neural network with more than two layers". The problematic aspect to this definition is that it makes deep learning sound as if it has been around since the 1980s. We feel that neural networks had to transcend architecturally from the earlier network styles (in conjunction with a lot more processing power) before showing the spectacular results seen in more recent years [**91**, **92**]. Following are some of the facets in this evolution of neural networks:

- More neurons than previous networks
- More complex ways of connecting layers/neurons in NNs
- Explosion in the amount of computing power available to train
- Automatic feature extraction

Deep learning is a subfield of machine learning, which is, in turn, a subfield of artificial intelligence (AI). Figure 1.1 represents a graphical depiction of this relationship.

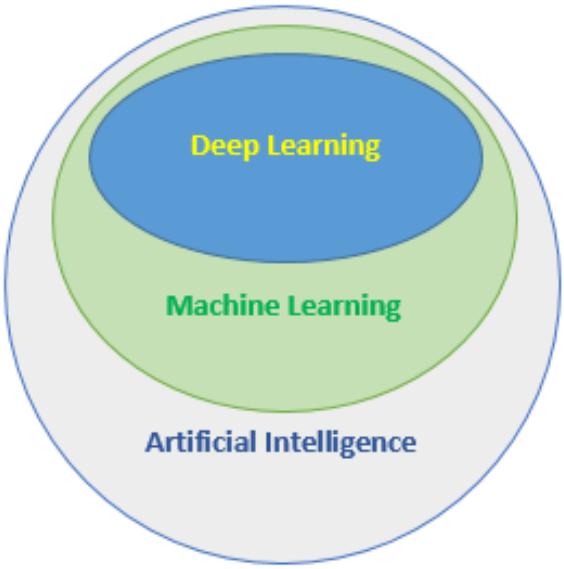

Figure 1.1 – A Venn diagram describing deep learning as a subfield of machine learning which is in turn a subfield of artificial intelligence

1.2.2 History of Deep Learning

Deep learning is based on the concept of artificial neural networks, or computational systems that mimic the way the human brain functions [**46**]. And so, our brief history of

deep learning must start with those neural networks.

- 1943: Warren McCulloch and Walter Pitts create a computational model for neural networks based on mathematics and algorithms called threshold logic.
- 1958: Frank Rosenblatt creates the perceptron, an algorithm for pattern recognition based on a two-layer computer neural network using simple addition and subtraction. He also proposed additional layers with mathematical notations, but these wouldn't be realised until 1975.
- 1980: Kunihiko Fukushima proposes the Neoconitron, a hierarchical, multilayered artificial neural network that has been used for handwriting recognition and other pattern recognition problems.
- 1989: Scientists were able to create algorithms that used deep neural networks, but training times for the systems were measured in days, making them impractical for real-world use.
- 1992: Juyang Weng publishes Cresceptron, a method for performing 3-D object recognition automatically from cluttered scenes.
- Mid-2000s: The term "deep learning" begins to gain popularity after a paper by Geoffrey Hinton and Ruslan Salakhutdinov showed how a many-layered neural network could be pre-trained one layer at a time.

In the next Section, we give a description in details of convolutional neural networks as one fo the most competitive deep learning models.

1.3 Deep Convolutional neural networks

Convolutional neural network is a special class of multilayer feedforward neural networks inspired by the neurobiological process in the animal visual cortex [**71**]. It allows handling images directly without preprocessing. Generally, CNN is composed of two parts: feature extraction and classification parts (as shown in Fig.1.2). In the feature extraction, the CNN receives an image pattern in the input and transforms it into a set of smaller images (called feature maps) representing the local features of the input. It is composed of a sequence of convolutional and pooling (subsampling) layers. In convolutional layers, the features are extracted from local regions of input using several filters called kernels. Next, a non linear function (activation function) is applied to generate the output. The resulting activations are then passed to the pooling layer, in which, we achieve a spatial invariance by reducing the resolution of feature maps. In the classification part, the last output in features extraction is presented as input to a fully connected classifier that can end up with a Softmax output layer [**10**]. The functioning of each part of CNNs is described below.

1.3.1 Convolution layer

The structure of a convolutional layer is composed of one or more convolutional feature maps, in which, the characteristics are extracted using the following formula:

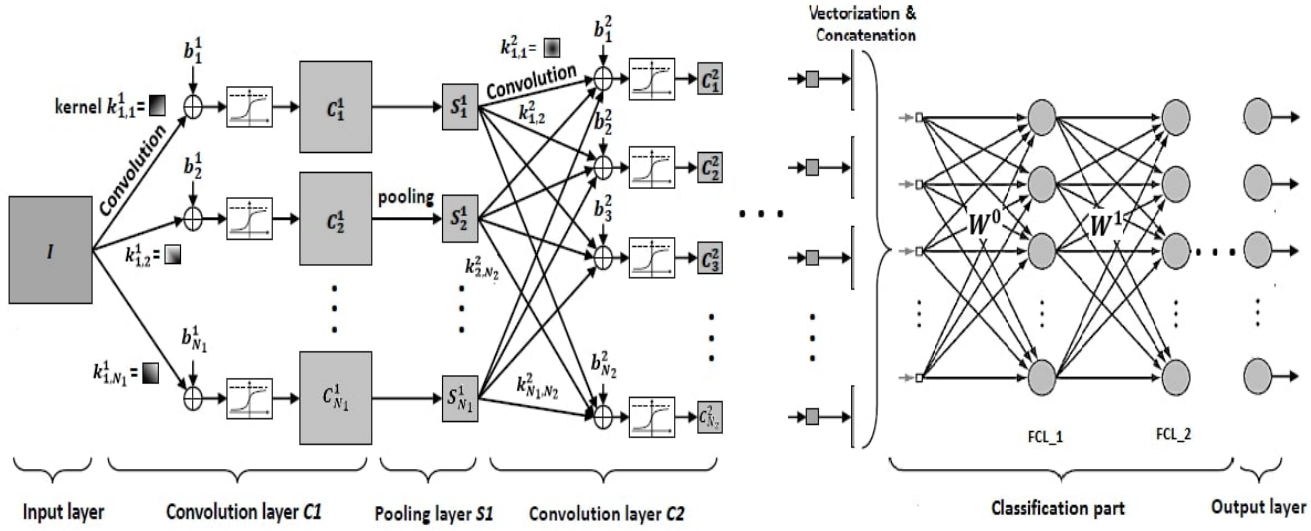

Figure 1.2 – Description of convolutional neural networks.

1.3.1.1 Convolution Operation

Convolution is a mathematical operation performed on two functions and is written as $(f * g)$, where f and g are two functions.

- **Continuous convolution:**
  Let f and g are two functions in $L^1(IR^2)$, we denote by $f * g$ the convolution product of f by g is expressed as follows:

$$\forall (x,y) \in IR^2, \;\; f * g(x,y) = \int_{IR} \int_{IR} f(x-x', y-y') g(x',y') dx' dy'.$$

  which is obviously a function in $L^1(IR^2)$.

- **Discrete convolution:**
  Based on the continuous convolution, we provide the discrete extension of the continuous convolution for the case of finite sequences.
  Let $(f_{m,n})_{1 \leq m \leq N, 1 \leq n \leq N}$ and $(g_{m,n})_{1 \leq m \leq N, 1 \leq n \leq N}$ two finite sequences. We denote by $f * g$ the following sequence:

$$(f * g)_{m,n} = \sum_{i=1}^{N} \sum_{j=1}^{N} f_{m-i,n-j} g_{i,j}$$

- **Convolution in deep learning:**
  In deep learning applications, the input is usually a multidimensional array of data and the kernel is usually a multidimensional array of parameters that are adapted

by the learning algorithm. We will refer to these multidimensional arrays as tensors. Because each element of the input and kernel must be explicitly stored separately, we usually assume that these functions are zero everywhere but the finite set of points for which we store the values. This means that in practice we can implement the infinite summation as a summation over a finite number of array elements.
Finally, we often use convolutions over more than one axis at a time. For example, if we use a two-dimensional image I as our input, we probably also want to use a two-dimensional kernel K :

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n)$$

Convolution is commutative, meaning we can equivalently write:

$$S(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i-m, j-n)K(m,n)$$

Usually the latter formula is more straightforward to implement in a machine learning library, because there is less variation in the range of valid values of m and n.

1.3.1.2 Importance and motivation of convolution

Convolution leverages three important ideas that can help improve a machine learning system: **sparse interactions**, **parameter sharing** and **equivariant representations** [**46**, **50**]. Moreover, convolution provides a means for working with inputs of variable size. We now describe each of these ideas in turn.

- **Sparse interactions:**
  Traditional neural network layers use matrix multiplication by a matrix of parameters with a separate parameter describing the interaction between each input unit and each output unit. This means every output unit interacts with every input unit. Convolutional networks, however, typically have **sparse interactions** (also referred to as sparse connectivity or sparse weights). This is accomplished by making the kernel smaller than the input.
- **Parameter sharing:**
  Parameter sharing refers to using the same parameter for more than one function in a model. In a convolutional neural net, each member of the kernel is used at every position of the input (except perhaps some of the boundary pixels, depending on the design decisions regarding the boundary). The parameter sharing used by the convolution operation means that rather than learning a separate set of parameters for every location, we learn only one set.
- **Equivariant representations:**
  In the case of convolution, the particular form of parameter sharing causes the layer to have a property called equivariance to translation. To say a function is equivariant means that if the input changes, the output changes in the same way. Specifically, a function $f(x)$ is equivariant to a function g if $f(g(x)) = g(f(x))$.
  In the case of convolution, if we let g be any function that translates the input, i.e., shifts it, then the convolution function is equivariant to g.

1.3.1.3 Backpropagation of Convolution Layers

Since convolutional kernels are considered parameters of CNNs, training of these models requires the gradient of convolution layer with respect to its parameters and its inputs. For this purpose, we are looking to compute $\frac{\partial E}{\partial K^l_{m',n'}}$ which can be interpreted as the measurement of how the change in a single pixel $K_{m',n'}$ in the weight kernel affects the loss function E.

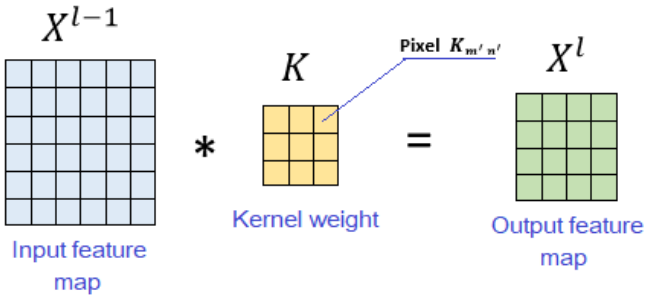

Figure 1.3 – Convolution process

During forward propagation, the convolution operation ensures that the yellow pixel $K_{m',n'}$ in the weight kernel makes a contribution in all the products (between each element of the weight kernel and the input feature map element it overlaps). This means that pixel $K_{m',n'}$ will eventually affect all the elements in the output feature map.
Convolution between the input feature map of dimension $H \times W$ and the weight kernel of dimension $k_1 \times k_2$ produces an output feature map of size $(H - k_1 + 1)$ by $(W - k_2 + 1)$. The gradient component for the individual weights can be obtained by applying the chain rule in the following way:

$$\begin{aligned}\frac{\partial E}{\partial K^l_{m',n'}} &= \sum_{i=0}^{H-k_1} \sum_{j=0}^{W-} \frac{\partial E}{\partial x^l_{i,j}} \frac{\partial x^l_{i,j}}{\partial K^l_{m',n'}} \\ &= \sum_{i=0}^{H-k_1} \sum_{j=0}^{W-k_2} \frac{\partial x^l_{i,j}}{\partial K^l_{m',n'}}\end{aligned}$$

In Eq. 10, $x^l_{i,j}$ is equivalent to $\sum_m \sum_n K^l_{m,n} o^{l-1}_{i+m,j+n} + b^l$ and expanding this part of the equation gives us:

$$\frac{\partial x^l_{i,j}}{\partial K^l_{m',n'}} = \frac{\partial}{\partial K^l_{m',n'}} \left(\sum_m \sum_n K^l_{m,n} o^{l-1}_{i+m,j+n} + b^l \right)$$

Further expanding the summations in Eq. 11 and taking the partial derivatives for all the components results in zero values for all except the components where $m = m'$ and

$n = n'$ in $K^l_{m,n} o^{l-1}_{i+m,j+n}$ as follows:

$$\begin{aligned}\frac{\partial x^l_{i,j}}{\partial K^l_{m',n'}} &= \frac{\partial}{\partial K^l_{m',n'}} \left(K^l_{0,0} o^{l-1}_{i+0,j+0} + \cdots + K^l_{m',n'} o^{l-1}_{i+m',j+n'} + \cdots + b^l \right) \\ &= \frac{\partial}{\partial K^l_{m',n'}} \left(K^l_{m',n'} o^{l-1}_{i+m',j+n'} \right) \\ &= o^{l-1}_{i+m',j+n'}\end{aligned}$$

Substituting Eq. 12 in Eq. 10 gives us the following results:

$$\begin{aligned}\frac{\partial E}{\partial K^l_{m',n'}} &= \sum_{i=0}^{H-k_1} \sum_{j=0}^{W-k_2} \delta^l_{i,j} o^{l-1}_{i+m',j+n'} \\ &= \text{rot}_{180^\circ} \left\{ \delta^l_{i,j} \right\} * o^{l-1}_{m',n'}\end{aligned}$$

The dual summation in Eq. 13 is as a result of weight sharing in the network (same weight kernel is slid over all of the input feature map during convolution). The summations represents a collection of all the gradients $\delta^l_{i,j}$ coming from all the outputs in layer l.
Obtaining gradients w.r.t to the filter maps, we have a cross-correlation which is transformed to a convolution by "flipping" the delta matrix $\delta^l_{i,j}$ (horizontally and vertically) the same way we flipped the filters during the forward propagation. An illustration of the flipped delta matrix is shown Figure 1.5.

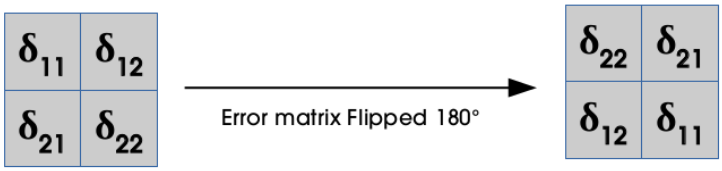

Figure 1.4 – Illustration of the flipped delta matrix

During the reconstruction process, the deltas $(\delta_{11}, \delta_{12}, \delta_{21}, \delta_{22})$ are used. These deltas are provided by an equation of the form:

$$\delta^l_{i,j} = \frac{\partial E}{\partial x^l_{i,j}}$$

At this point we are looking to compute $\frac{\partial E}{\partial x^l_{i',j'}}$ which can be interpreted as the measurement of how the change in a single pixel $x_{i',j'}$ in the input feature map affects the loss function E.
From the diagram above, we can see that region in the output affected by pixel $x_{i',j'}$ from the input is the region in the output bounded by the dashed lines where the top left

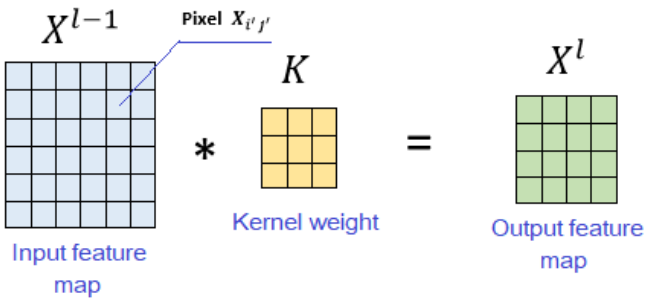

Figure 1.5 – Convolution process

corner pixel is given by $(i'-k_1+1, j'-k_2+1)$ and the bottom right corner pixel is given by (i', j'). Using chain rule and introducing sums give us the following equation:

$$\begin{aligned}\frac{\partial E}{\partial x^l_{i',j'}} &= \sum_{i,j\in Q} \frac{\partial E}{\partial x^{l+1}_Q} \frac{\partial x^{l+1}_Q}{\partial x^l_{i',j'}} \\ &= \sum_{i,j\in Q} \delta^{l+1}_Q \frac{\partial x^{l+1}_Q}{\partial x^l_{i',j'}}\end{aligned}$$

Q in the summation above represents the output region bounded by dashed lines and is composed of pixels in the output that are affected by the single pixel $x_{i',j'}$ in the input feature map. A more formal way of representing Eq. 16 is:

$$\begin{aligned}\frac{\partial E}{\partial x^l_{i',j'}} &= \sum_{m=0}^{k_1-1}\sum_{n=0}^{k_2-1} \frac{\partial E}{\partial x^{l+1}_{i'-m,j'-n}} \frac{\partial x^{l+1}_{i'-m,j'-n}}{\partial x^l_{i',j'}} \\ &= \sum_{m=0}^{k_1-1}\sum_{n=0}^{k_2-1} \delta^{l+1}_{i'-m,j'-n} \frac{\partial x^{l+1}_{i'-m,j'-n}}{\partial x^l_{i',j'}}\end{aligned}$$

In the region Q, the height ranges from $i'-0$ to $i'-(k_1-1)$ and width $j'-0$ to $j'-(k_2-1)$. These two can simply be represented by $i'-m$ and $j'-n$ in the summation since the iterators m and n exists in the following similar ranges from $0 \le m \le k_1-1$ and $0 \le n \le k_2-1$.
In Eq. 17, $x^{l+1}_{i'-m,j'-n}$ is equivalent to $\sum_{m'}\sum_{n'} K^{l+1}_{m',n'} o^l_{i'-m+m',j'-n+n'} + b^{l+1}$ and expanding this part of the equation gives us:

$$\begin{aligned}\frac{\partial x^{l+1}_{i'-m,j'-n}}{\partial x^l_{i',j'}} &= \frac{\partial}{\partial x^l_{i',j'}} \left(\sum_{m'}\sum_{n'} K^{l+1}_{m',n'} a^l_{i'-m+m',j'-n+n'} + b^{l+1} \right) \\ &= \frac{\partial}{\partial x^l_{i',j'}} \left(\sum_{m'}\sum_{n'} K^{l+1}_{m',n'} f\left(x^l_{i'-m+m',j'-n+n'}\right) + b^{l+1} \right)\end{aligned}$$

Further expanding the summation in Eq. 17 and taking the partial derivatives for all the components results in zero values for all except the components where $m' = m$ and $n' = n$

so that $f\left(x^l_{i'-m+m',j'-n+n'}\right)$ becomes $f\left(x^l_{i',j'}\right)$ and $K^{l+1}_{m',n'}$ becomes $w^{l+1}_{m,n}$ in the relevant part of the expanded summation as follows:

$$\begin{aligned}\frac{\partial x^{l+1}_{i'-m,j'-n}}{\partial x^l_{i',j'}} &= \frac{\partial}{\partial x^l_{i',j'}}\left(K^{l+1}_{m',n'} f\left(x^l_{0-m+m',0-n+n'}\right) + \cdots + K^{l+1}_{m,n} f\left(x^{l'}_{i',j'}\right) + \cdots + b^{l+1}\right) \\ &= \frac{\partial}{\partial x^l_{i',j'}}\left(K^{l+1}_{m,n} f\left(x^l_{i',j'}\right)\right) \\ &= K^{l+1}_{m,n}\frac{\partial}{\partial x^l_{i',j'}}\left(f\left(x^l_{i',j'}\right)\right) \\ &= K^{l+1}_{m,n} f'\left(x^l_{i',j'}\right)\end{aligned}$$

Substituting Eq. 19 in Eq. 17 gives us the following results:

$$\frac{\partial E}{\partial x^l_{i',j'}} = \sum_{m=0}^{k_1-1}\sum_{n=0}^{k_2-1} \delta^{l+1}_{i'-m,j'-n} K^{l+1}_{m,n} f'\left(x^l_{i',j'}\right)$$

For backpropagation, we make use of the flipped kernel and as a result we will now have a convolution that is expressed as a cross-correlation with a flipped kernel:

$$\begin{aligned}\frac{\partial E}{\partial x^l_{i',j'}} &= \sum_{m=0}^{k_1-1}\sum_{n=0}^{k_2-1} \delta^{l+1}_{i'-m,j'-n} K^{l+1}_{m,n} f'\left(x^l_{i',j'}\right) \\ &= \text{rot}_{180^\circ}\left\{\sum_{m=0}^{k_1-1}\sum_{n=0}^{k_2-1} \delta^{l+1}_{i'+m,j'+n} K^{l+1}_{m,n}\right\} f'\left(x^l_{i',j'}\right) \\ &= \delta^{l+1}_{i',j'} * \text{rot}_{180^\circ}\left\{K^{l+1}_{m,n}\right\} f'\left(x^l_{i',j'}\right)\end{aligned}$$

1.3.2 Non-linear non-saturating activation functions

The choice of activation functions in deep networks has a significant effect on the training dynamics and task performance. Moreover, a key strategy to enable training of deep neural networks is to use non-saturating activation functions to reduce the vanishing gradient problem [**46**, **113**, **116**]. Popular choices that saturate only in the negative domain are the rectified linear unit (ReLU), its smooth, non-linear variant, Softplus, and the exponential linear units (ELU and SELU). Other functions are non-saturating across the entire real domain, like the linear parametric ReLU (PReLU) [**116**]. Before given the expression of these functions, we firstly give the definition of a non-saturating function.

Definition 1

A real-valued function $f : \mathbb{R} \longrightarrow \mathbb{R}$ is called a non-saturating function iff $|\lim_{x\longrightarrow+\infty} f(x)| = +\infty$ or $|\lim_{x\longrightarrow-\infty} f(x)| = +\infty$

The first non-saturating activation function proposed for deep neural networks, is the Rectified Linear Unit (ReLU) function which is defined as follow:

$$f(x) = ReLU(x) = \begin{cases} x & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases}$$

Figure 1.6 represents the curve of ReLU function.

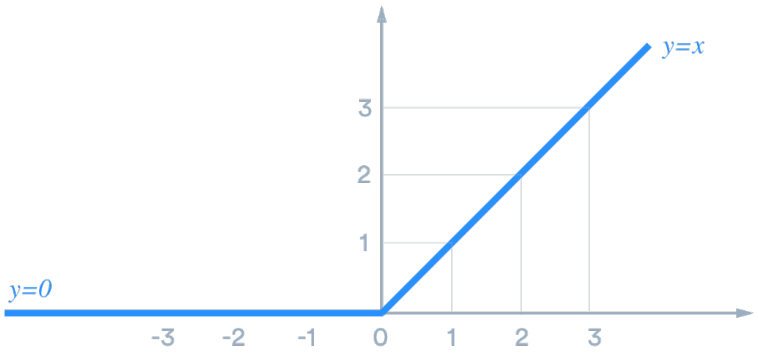

Figure 1.6 – ReLU activation function

A soft version of ReLU is also a non-saturating function called Softplus which is expressed as follow

$$Softplus(x) = \ln(1 + \exp(x))$$

An other non-saturating version of ReLU was recently introduced is the PReLU function

$$f(x) = PReLU(x) = \begin{cases} x & \text{if } x \geq 0 \\ a & \text{if } x < 0 \end{cases}$$

where, a is a coefficient determining the slope of the negative part. In PReLU, the parameter a is learned together with the weights in the network. If the parameter is fixed, PReLU becomes equivalent to LReLU. Here, we fixed the parameter in each simulation.
ELU falls in the category of saturating activation functions, but takes both positive and negative values and it is locally non- saturated around the origin

$$f(x) = ELU(x) = \begin{cases} x & \text{if } x \geq 0 \\ \alpha.(\exp(x) - 1) & \text{if } x < 0 \end{cases}$$

where, α is a parameter determining the saturating value in the negative domain, usually set to 1. The gradient of ELU in the origin is α.
SELU is the scaled variant of ELU, as it introduces a second, scaling parameter, λ

$$f(x) = SELU(x) = \lambda. \begin{cases} x & \text{if } x \geq 0 \\ \alpha.(\exp(x) - 1) & \text{if } x < 0 \end{cases}$$

A fully non-saturating function called $Soft++$, extending both Softplus and PReLU, was recently proposed in [**116**] and defined as follows:

$$Soft++(x) = \ln(1 + \exp(k.x)) + \frac{x}{c} - \ln(2)$$

where, k is an exponent coefficient and c is a slope coefficient which sets the slope in the negative domain. The constant $\ln(2)$ term removes the offset of the $Soft++$ function in the origin.
Finally, it is very important to note that:

- Convergence of networks with any activation function depends critically on the particular dataset and network architecture, emphasizing the need for parametrization, which allows to adapt the activation function to the particular problem.
- non-linearity around the origin improves learning and generalization.
- in many cases, non-saturation across the entire real domain further improves performance.

1.3.3 Subsampling layer

The subsampling (pooling) process applies to the feature maps obtained from the convolution and activation function. It is a crucial component of CNN that helps to avoid overfitting phenomena and generating the summaries of the input feature maps [**67**, **73**, **109**]. The pooling layer includes several types, such as the max-pooling layer which is the most widely-used subsampling and the average pooling layer [**117**, **72**]. These two types and other pooling methods are described as follows:

- **Max Pooling** [**117**] is the most popular pooling method, which takes the largest activation in the pooling region as given in Eq.5.2.

$$s_k = \max_{a_i \in R_k} a_i \quad for \quad k = 1, ..., K \tag{1.1}$$

  Max Pooling is aimed for extracting local features such as lines, edges, and textures in the feature map.

- **Average Pooling** [**72**] takes the arithmetic mean value of activations in the pooling region as given in Eq.5.3.

$$s_k = \frac{1}{\#R_k} \sum_{a_i \in R_k} a_i \quad for \quad k = 1, ..., K \tag{1.2}$$

  This method can extract global features in a given feature map by smoothing the pooling region.

- **Stochastic Pooling[120]** is an improved pooling method, which processed as follow: In the first, the probabilities p_j is computed for each region by normalizing the activations within the regions, as given in Eq.1.3.

$$p_j = \frac{a_j}{\sum_{a_i \in R_k} a_i} \quad for \quad j = 1, ..., |R_k| \tag{1.3}$$

Then the outputs are selected based on the probabilities calculated by multinomial distribution determined with the probabilities p_j for $j = 1, ..., |R_k|$ as follows:

$$s_k = a_l \quad where \quad l \sim P(p_1, ..., p_{|R_k|}) \tag{1.4}$$

In the testing phase, instead of the normal averaging, the probabilistic form of averaging is adopted, as given in Eq.1.5.

$$s_k = \sum_{a_j \in R_k} p_j a_j \quad for \quad k = 1, ..., K \tag{1.5}$$

This method is less prone to overfitting due to the stochastic strategy used, and outperforms the max and average pooling[**120**].

- **Hybrid Pooling [73]** is another improved pooling strategy, which introduces a probability to choose the average or the max pooling technique for each convolutional feature map. In the training phase, for each feature map in the convolutional layer, the max pooling is used with probability $1 - p$ and the average pooling with probability p for all the pooling regions:

$$S = \begin{cases} S_{avg} & \text{with probability } p \\ S_{max} & \text{with probability } 1 - p \end{cases} \tag{1.6}$$

where $S_{avg} = \{s_1^{avg}, ..., s_K^{avg}\}$ with $s_k^{avg} = \frac{1}{|R_k|} \sum_{a_i \in R_k} a_i$ and $S_{max} = \{s_1^{max}, ..., s_K^{max}\}$ with $s_k^{max} = \max_{a_i \in R_k} a_i$.
In the testing phase, the output for each pooling maps is obtained by the expected value of S in Eq.1.6, which is given as follows:

$$S = S_{hyb} = pS_{avg} + (1 - p)S_{max} \tag{1.7}$$

This method has shown that is effective for increasing the generalization ability of CNNs.

where $\{a_1, a_2, .., a_{\#R_k}\}$ designates the kth pooling region.

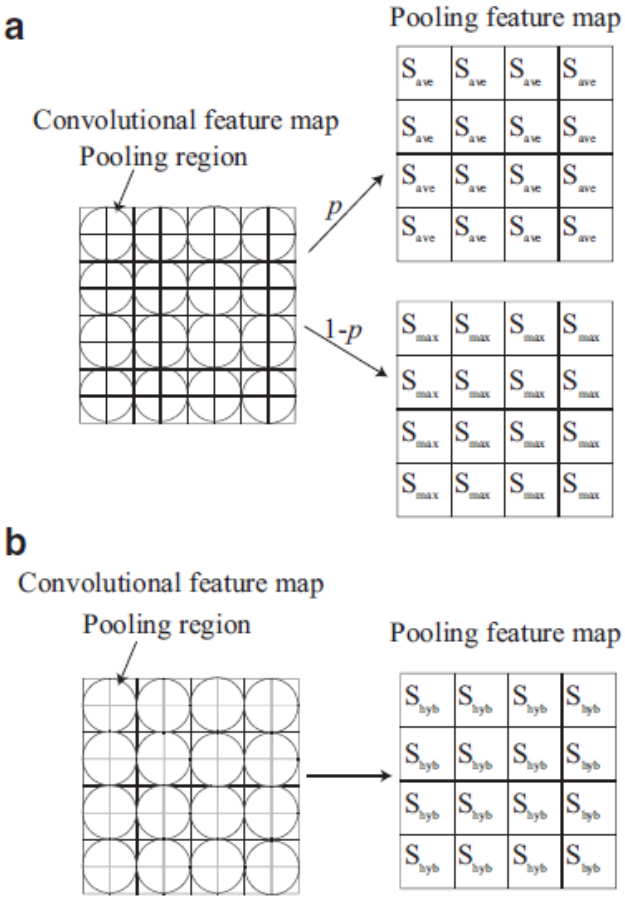

Figure 1.7 – Illustration of the hybrid pooling. (a) The operation in the training stage. (b) The operation in the testing stage. [73]

1.3.4 Classification layers

Once we achieve the last layer in the features extraction, all the feature maps are converted to a one dimension (1D) array which, in turn, is presented as an input to a classifier composed of fully-connected layers where the outputs are given as follows:

$$Z^k = \begin{pmatrix} Z_1^k \\ Z_2^k \\ \vdots \\ Z_{n_{FC_k}}^k \end{pmatrix} = \begin{pmatrix} f\Big(\sum_{i=1}^{n_{k-1}} W_{1i}^k \times Z_i^{k-1}\Big) \\ f\Big(\sum_{i=1}^{n_{k-1}} W_{2i}^k \times Z_i^{k-1}\Big) \\ \vdots \\ f\Big(\sum_{i=1}^{n_{k-1}} W_{n_{FC_k} i}^k \times Z_i^{k-1}\Big) \end{pmatrix}$$

The Softmax function is then used to calculate the network output,

$$Y = softmax(Z_{FC}^N)$$

where N_{FC} is the number of hidden layers.

1.4 Loss functions in deep learning

In the learning process, the parameters of a deep neural network are subject to minimize a certain objective function, which represents a measure of proximity between the prediction of the network and the associated target. This is also known under the equivalent names of cost function, loss function, , or error function [**94**, **91**, **92**]. In the following, we shall describe some of the most familiar cost functions used in deep neural networks.

1.4.1 Mean Square Loss / L^2 Loss

It is more often used regression loss that is computed by taking the average squared difference between actual and predicted observations. It mainly takes into consideration the average magnitude of error ignoring the direction. Due to squaring the predictions that are distant from the true values are penalized laboriously in comparison to less diverged predictions. It is easy to compute gradients because of the mathematical properties there in L^2 Loss. The MSE (also termed the "quadratic loss" or "l^2 error") formula can be written as follow

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \tilde{y}_i)^2$$

where y_i and $\tilde{y}_i$ are respectively the expected and predicted values.

1.4.2 Mean Absolute Error

Regression metric which measures the average magnitude of errors in a group of predictions, without considering their directions. In other words, it's a mean of absolute differences among predictions and expected results where all individual deviations have even importance. The MAE formula can be written as follow

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \tilde{y}_i)^2$$

1.4.3 Hinge loss

The hinge loss function is calculated on the score $f(\mathbf{x})$ of the class, as opposed to the final prediction $\hat{y}$. It attempts to rectify zero-one loss for the case of binary classification by defining loss that is directly proportional to the degree of score for incorrect predictions. It is computed as

$$\mathcal{L}_{\text{hinge}}(y, f(\mathbf{x})) = \max(0, 1 - yf(\mathbf{x})) = [1 - yf(\mathbf{x})]_+$$

Simply put, the hinge loss is equal to the positive value of $[1 - yf(\mathbf{x})]$. Hence the notation, $[1 - yf(\mathbf{x})]_+$. This formulation is applicable when the actual class labels are represented

as positive and negative class (and not 0 or 1 class). This means, $y \in \{-1, 1\}$. It is the loss function used in support vector machines

1.4.4 Negative log-likelihood

The hinge loss is non-differentiable at specific points. A continuously differentiable alternative is the negative log-likelihood (NLL) loss function. A good model should maximize the log-likelihood of the data. Conversely, the negative of the log-likelihood should be low for a good model. Consider a binary classification problem with the classes C_1 and C_2. Let $y_i = 1$ if $\mathbf{x}_i$ belongs to the class C_1, and zero if it belongs to the class C_2. If $\mathbb{W}$ denotes the parameters of the model, then, the likelihood of the training data $\mathbb{L}$ given these parameters is

$$P(\mathbb{L} \mid \mathbb{W}) = \prod_{l=1}^{L} P\left(\hat{y} = 1 \mid \mathbf{x}_l, \mathbb{W}\right)^{y_l} \left(1 - P(\hat{y} = 1 \mid \mathbf{x}l, \mathbb{W})\right)^{1-y_l}$$

As mentioned earlier, maximizing the log-likelihood is equivalent to minimizing the negative of the log-likelihood, the loss.

$$\text{NLL}(\mathbb{L} \mid \mathbb{W}) = -\sum_{l=1}^{L} \left[y_l \log P\left(\hat{y} = 1 \mid \mathbf{x}_l, \mathbb{W}\right) + (1 - y_l) \log(1 - P(\hat{y} = 1 \mid \mathbf{x}l, \mathbb{W}))\right]$$

Thus, the per instance loss is

$$\mathcal{L}_{\text{NLL}}\left(y_l, \mathbb{W}\right) = -y_l \log P\left(\hat{y} = 1 \mid \mathbf{x}_l, \mathbb{W}\right) - (1 - y_l) \log(1 - P(\hat{y} = 1 \mid \mathbf{x}l, \mathbb{W}))$$

Negative log-likelihood is the loss-function used for training the logistic regression classifier (Specifically though, there we merely maximize the log-likelihood, but that's really the same thing). Owing to its differentiability it is also commonly used as a loss function in train deep neural networks.

1.4.5 Binary Cross-Entropy Loss

The negative log-likelihood is differentiable but it requires probabilistic outputs for estimating $P(\hat{y} = 1 \mid \mathbb{W})$ and $P(\hat{y} = 0 \mid \mathbb{W})$. An alternative, with a similar formulation, for non-probabilistic models is the binary cross-entropy (BCE) loss that directly works with the score of the positive class $f(\mathbf{x}, \mathbb{W})$, where, $\mathbb{W}$ is the set of parameters of the model. The per-instance BCE Loss is

$$\mathcal{L}_{\text{BCE}}\left(y_l, \mathbf{x}_l, \mathbb{W}\right) = -y_l \log f\left(\mathbf{x}_l, \mathbb{W}\right) - (1 - y_l) \log\left(1 - f\left(\mathbf{x}_l, \mathbb{W}\right)\right)$$

Just like other losses, the per-instance BCE loss is aggregated as a sum over all the examples in the training set (or minibatch, if optimizing using minibatch stochastic gradient descent).
Note that the formulation is similar to the negative log-likelihood loss, with the probability $P(\hat{y} \mid \mathbb{W})$ now replaced with $f\left(\mathbf{x}_l, \mathbb{W}\right)$ the score of the positive class. For a training

instance $\mathbf{x}_l$, only one of the two terms is active depending on the value of y_l and the other becomes zero. Suppose $f_y(\mathbf{x}, \mathbb{W})$ denotes the score of the model for the class y, so that $sum_{y\in 0,1} f_y(\mathbf{x}, \mathbb{W}) = 1$. In this case, we can also rewrite the BCE loss alternatively as

$$\mathcal{L}_{\text{BCE}}\left(y_l, \mathbf{x}_l, \mathbb{W}\right) = -\log f_{y_l}\left(\mathbf{x}_l, \mathbb{W}\right)$$

It is called binary cross-entropy because its formulation is similar to the cross-entropy between two discrete distributions p and q, calculated as $\mathcal{H}[p, q] = -\sum_{x\in\mathcal{X}} p(x) \log q(x)$. In our case, the cross-entropy is between the distribution of y and the distribution of $f(\mathbf{x}, \mathbb{W})$. The BCE loss is differentiable, but has a numerical challenge. If $f(\mathbf{x}, \mathbb{W}) = 0$ or $f(\mathbf{x}, \mathbb{W}) = 1$, then one of terms becomes $\log 0$ which is mathematically undefined. Some packages like PyTorch get over this issue by setting $\log 0 = \infty$ and then clamping the value of the loss to be greater than -100, an arbitrary choice that works just fine.

1.4.6 Cross-Entropy Loss

A natural extension of the binary cross entropy loss to multiclass problems is the cross-entropy loss. Typically, multi-class classifiers, such as those implemented with neural networks, will arrive at a score for each class $m = 1, \ldots, M$ as $f_m(\mathbf{x}, \mathbb{W})$. The cross-entropy loss first calculates the softmax of these scores to normalize the scores, so that they sum to 1. Then, the negative of the logarithm of this softmax is used to compute the crossentropy. The per-instance cross-entropy loss is calculated as

$$\mathcal{L}_{\text{cross-entropy}}\ \left(y_l, \mathbf{x}_l, \mathbb{W}\right) = -\log \frac{\exp\left(f_{y_l}\left(\mathbf{x}_l, \mathbb{W}\right)\right)}{\sum_{m=1}^{M} \exp\left(f_{C_m}\left(\mathbf{x}_l, \mathbb{W}\right)\right)}$$

Note that this formulation is similar to the alternative BCE loss formulation that we described in the formula of BCE. In modern deep learning, the cross-entropy loss is the default recommendation for a good differentiable loss function for training classifiers.

1.5 Optimization for Training Deep neural networks

Optimization in general is an extremely difficult task. Traditionally, machine learning has avoided the difficulty of general optimization by carefully designing the objective function and constraints to ensure that the optimization problem is convex. When training neural networks, we must confront the general non-convex case. Even convex optimization is not without its complications.
A recurring problem in deep learning is that large training sets are necessary for good generalization, but large training sets are also more computationally expensive. For example, the negative conditional log-likelihood of the training data can be written as

$$J(\boldsymbol{W}) = \mathbb{E}_{\mathbf{x},\mathrm{y}\sim\hat{p}_{\text{data}}} L(\boldsymbol{x}, y, \boldsymbol{W}) = \frac{1}{m}\sum_{i=1}^{m} L\left(\boldsymbol{x}^{(i)}, y^{(i)}, \boldsymbol{W}\right)$$

where L is the per-example loss $L(\boldsymbol{x}, y, \boldsymbol{W}) = -\log p(y \mid \boldsymbol{x}; \boldsymbol{W})$. For these additive cost functions, gradient descent requires computing

$$\nabla_{\boldsymbol{W}} J(\boldsymbol{W}) = \frac{1}{m} \sum_{i=1}^{m} \nabla_{\boldsymbol{W}} L\left(\boldsymbol{x}^{(i)}, y^{(i)}, \boldsymbol{W}\right)$$

The computational cost of this operation is $O(m)$. As the training set size grows to billions of examples, the time to take a single gradient step becomes prohibitively long.

To address this issue, several optimization algorithms have been developed taking into account this challenge. In the rest of this section, we present the most successful and competitive optimization algorithms for deep neural network training.

1.5.1 Stochastic Gradient Descent

Nearly all of deep learning is powered by one very important algorithm: stochastic gradient descent or SGD [**79**]. Stochastic gradient descent is probably the most used optimization algorithm for machine learning in general and for deep neural networks in particular. SGD is an extension of the gradient descent algorithm which has been developed to deal with the above mentioned issue.

The main idea of this algorithm is using an unbiased estimate of the gradient by taking the average gradient on a minibatch of m examples drawn i.i.d from the data generating distribution.

The insight of stochastic gradient descent is that the gradient is an expectation. The expectation may be approximately estimated using a small set of samples. Specifically, on each step of the algorithm, we can sample a minibatch of examples $\mathbb{B} = \left\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m')}\right\}$ drawn uniformly from the training set. The minibatch size m' is typically chosen to be a relatively small number of examples, ranging from 1 to a few hundred. Crucially, m' is usually held fixed as the training set size m grows. We may fit a training set with billions of examples using updates computed on only a hundred examples. The estimate of the gradient is formed as

$$\boldsymbol{g} = {}^{1}_{m'} \nabla_{\boldsymbol{W}} \sum_{i=1}^{m'} L\left(\boldsymbol{x}^{(i)}, y^{(i)}, \boldsymbol{W}\right)$$

using examples from the minibatch $\mathbb{B}$. The stochastic gradient descent algorithm then follows the estimated gradient downhill:

$$\boldsymbol{W} \leftarrow \boldsymbol{W} - \epsilon \boldsymbol{g}$$

where ϵ is the learning rate.

Algorithm 1 shows how to follow this estimate of the gradient downhill.

Algorithm 1: Stochastic gradient descent (SGD)

- **Input:**

  Initial parameter $\boldsymbol{W}$, Learning rate ϵ_k.
- **Output:**

  Optimized parameter $\boldsymbol{W}^*$

while stopping criterion not met **do**
. Sample a minibatch of m examples from the training set $\left\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\right\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
. Compute gradient estimate: $\hat{\boldsymbol{g}} \leftarrow +\frac{1}{\mathrm{m}} \nabla_{\boldsymbol{W}} \sum_i L\left(f\left(\boldsymbol{x}^{(i)}; \boldsymbol{W}\right), \boldsymbol{y}^{(i)}\right)$
. Apply update: $\boldsymbol{W} \leftarrow \boldsymbol{W} - \epsilon \hat{\boldsymbol{g}}$
end while

$$\boldsymbol{W}^* \leftarrow \boldsymbol{W}$$

A crucial parameter for the SGD algorithm is the learning rate. Previously, we have described SGD as using a fixed learning rate ϵ. In practice, it is necessary to gradually decrease the learning rate over time, so we new denote the learning rate at iteration k as ϵ_k.

This is because the SGD gradient estimator introduces a source of noise (the random sampling of m training examples) that does not vanish even when we arrive at a minimum. By comparison, the true gradient of the total cost function becomes small and then 0 when we approach and reach a minimum using batch gradient descent, so batch gradient descent can use a fixed learning rate.

Lemma 1.1

A sufficient condition to guarantee convergence of SGD is that

$$\sum_{k=1}^{\infty} \epsilon_k = \infty, \quad \text{and}$$

$$\sum_{k=1}^{\infty} \epsilon_k^2 < \infty$$

In practice, it is common to decay the learning rate linearly until iteration τ :

$$\epsilon_k = (1 - \alpha)\epsilon_0 + \alpha\epsilon_\tau$$

with $\alpha = \frac{k}{\tau}$. After iteration τ, it is common to leave ϵ constant.

1.5.2 Stochastic Gradient Descent with Momentum

The method of momentum [**18**] is designed to accelerate learning, especially in the face of high curvature, small but consistent gradients, or noisy gradients. The momentum algorithm accumulates an exponentially decaying moving average of past gradients and continues to move in their direction.

Formally, the momentum algorithm introduces a variable v that plays the role of velocity. It is the direction and speed at which the parameters move through parameter space. The velocity is set to an exponentially decaying average of the negative gradient. The name momentum derives from a physical analogy, in which the negative gradient is a force moving a particle through parameter space, according to Newton's laws of motion. Momentum in physics is mass times velocity. In the momentum learning algorithm, we assume unit mass, so the velocity vector v may also be regarded as the momentum of the particle. A hyperparameter $\alpha \in [0, 1)$ determines how quickly the contributions of previous gradients exponentially decay. The update rule is given by:

$$\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon \nabla_{\boldsymbol{W}} \left(\frac{1}{m} \sum_{i=1}^{m} L \left(f \left(\boldsymbol{x}^{(i)}; \boldsymbol{W} \right), \boldsymbol{y}^{(i)} \right) \right)$$
$$\boldsymbol{W} \leftarrow \boldsymbol{W} + \boldsymbol{v}$$

The velocity v accumulates the gradient elements

$$\nabla_{\boldsymbol{W}} \left(\frac{1}{m} \sum_{i=1}^{m} L \left(f \left(\boldsymbol{x}^{(i)}; \boldsymbol{W} \right), \boldsymbol{y}^{(i)} \right) \right)$$

The larger α is relative to ϵ, the more previous gradients affect the current direction. The SGD algorithm with momentum is given in algorithm 2.

Algorithm 2: Stochastic gradient descent (SGD) with momentum

- **Input:**

  Initial parameter $\boldsymbol{W}$, Learning rate ϵ , momentum parameter α , initial velocity $\boldsymbol{v}$.

- **Output:**

  Optimized parameter $\boldsymbol{W}^*$

---

while stopping criterion not met **do**

. Sample a minibatch of m examples from the training set $\left\{ \boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)} \right\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.

. Compute gradient estimate: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{W}} \sum_i L \left(f \left(\boldsymbol{x}^{(i)}; \boldsymbol{W} \right), \boldsymbol{y}^{(i)} \right)$

. Compute velocity update: $\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon \boldsymbol{g}$

. Apply update: $W \leftarrow W + v$
end while

$$W^* \leftarrow W$$

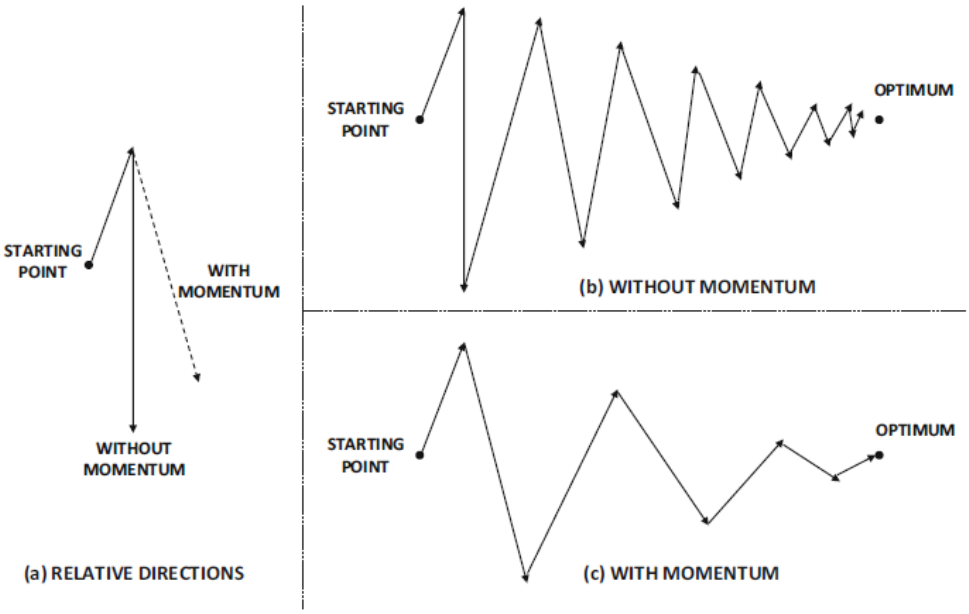

Figure 1.8 – Effect of momentum in smoothing zigzag updates

An example of the use of momentum is illustrated in Figure 1.8. It is evident from Figure 1.8(a) that momentum increases the relative component of the gradient in the correct direction. The corresponding effects on the updates are illustrated in Figure 1.8(b) and (c). It is evident that momentum-based updates can reach the optimal solution in fewer updates.

1.5.3 Optimization Algorithms with Adaptive Learning Rates

In deep neural networks optimization, the learning rate was reliably one of the hyper-parameters that is the most difficult to set because it has a significant impact on model performance. To address this challenge, a number of incremental (or mini-batch-based) methods have been introduced that adapt the learning rates of model parameters. In this subsection, we briefly review three successful and competitive, of these algorithms.

1.5.3.1 Adaptive Gradient Algorithm (AdaGrad optimizer)

AdaGrad algorithm is an adaptive optimization algorithm which individually adapts the learning rates of all model parameters by scaling them inversely proportional to the square root of the sum of all of their historical squared values [**22**]. The parameters with the largest partial derivative of the loss have a correspondingly rapid decrease in their learning rate, while parameters with small partial derivatives have a relatively small decrease in their learning rate. Algorithm 3 describes the overall steps of the AdaGrad algorithm [**22**].

Algorithm 3: The AdaGrad algorithm

- **Input:**

  Initial parameter $\boldsymbol{W}$, Global learning rate ϵ ,
Small constant δ, perhaps 10^{-7}, for numerical stability.

- **Output:**

  Optimized parameter $\boldsymbol{W}^*$

. Initialize gradient accumulation variable $\boldsymbol{r} = \boldsymbol{0}$
while stopping criterion not met **do**
. Sample a minibatch of m examples from the training set $\left\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\right\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
. Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{W}} \sum_i L\left(f\left(\boldsymbol{x}^{(i)}; \boldsymbol{W}\right), \boldsymbol{y}^{(i)}\right)$
. Accumulate squared gradient: $\boldsymbol{r} \leftarrow \boldsymbol{r} + \boldsymbol{g} \odot \boldsymbol{g}$
. Compute update: $\Delta \boldsymbol{W} \leftarrow -\frac{\epsilon}{\delta + \sqrt{\boldsymbol{r}}} \odot \boldsymbol{g}$.
(Division and square root applied element-wise)
. Apply update: $\boldsymbol{W} \leftarrow \boldsymbol{W} + \Delta \boldsymbol{W}$
end while

$$\boldsymbol{W}^* \leftarrow \boldsymbol{W}$$

1.5.3.2 Root Mean Square Propagation (RMSprop optimizer)

AdaGrad algorithm performs well for some but not all deep neural networks. Indeed, the accumulation of squared gradients from the beginning of training can result in a premature and excessive decrease in the effective learning rate. Motivated by this challenge, Hinton et al. proposed an extension of AdaGrad called Root Mean Square Propagation

(RMSprop) algorithm to perform better in the non-convex setting by changing the gradient accumulation into an exponentially weighted moving average [**84**]. The description of RMSprop is given in Algorithm 4.

Algorithm 4: The RMSProp algorithm

- **Input:**

  Initial parameter $\boldsymbol{W}$, Global learning rate ϵ , decay rate ρ.
Small constant δ, usually 10^{-6}, used to stabilize division by small numbers.

- **Output:**

  Optimized parameter $\boldsymbol{W}^*$

. Initialize accumulation variables $r = 0$
while stopping criterion not met **do**
. Sample a minibatch of m examples from the training set $\left\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\right\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
. Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{W}} \sum_i L\left(f\left(\boldsymbol{x}^{(i)}; \boldsymbol{W}\right), \boldsymbol{y}^{(i)}\right)$
. Accumulate squared gradient: $\boldsymbol{r} \leftarrow \rho \boldsymbol{r} + (1-\rho)\boldsymbol{g} \odot \boldsymbol{g}$
. Compute parameter update: $\Delta \boldsymbol{W} = -\frac{\epsilon}{\sqrt{\delta + r}} \odot \boldsymbol{g}$.
$\left(\frac{1}{\sqrt{\delta+r}}\text{. applied element-wise }\right)$
. Apply update: $\boldsymbol{W} \leftarrow \boldsymbol{W} + \Delta \boldsymbol{W}$
end while

$$\boldsymbol{W}^* \leftarrow \boldsymbol{W}$$

1.5.3.3 Adam optimizer

Adam optimizer is one of the most successful and competitive optimization algorithm which has achieved relevant results in training deep neural networks [**21**]. Adam is yet an adaptive learning rate optimization algorithm that combines the advantages of the two efficient methods previously introduced: RMSProp and momentum. It exponentially smooths the first-order gradient in order to incorporate momentum into the update and also directly addresses the bias inherent in exponential smoothing when the running estimate of a smoothed value is unrealistically initialized to 0. Algorithm 5 gives a description of the Adam optimizer.

Algorithm 5: The Adam algorithm

- **Input:**

  Initial parameter $\boldsymbol{W}$, Step size ϵ(Suggested default: 0.001)
Exponential decay rates for moment estimates, ρ_1 and ρ_2 in $[0, 1)$.
(Suggested defaults: 0.9 and 0.999 respectively)
Small constant δ used for numerical stabilization. (Suggested default: 10^{-8})

- **Output:**

  Optimized parameter $\boldsymbol{W}^*$

. Initialize 1st and 2nd moment variables $s = 0, r = 0$
. Initialize time step $t = 0$
while stopping criterion not met **do**
. Sample a minibatch of m examples from the training set $\left\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\right\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
. Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{W}} \sum_i L\left(f\left(\boldsymbol{x}^{(i)}; \boldsymbol{W}\right), \boldsymbol{y}^{(i)}\right) \quad t \leftarrow t + 1$
. Update biased first moment estimate: $s \leftarrow \rho_1 s + (1 - \rho_1)\, \boldsymbol{g}$
. Update biased second moment estimate: $\boldsymbol{r} \leftarrow \rho_2 \boldsymbol{r} + (1 - \rho_2)\, \boldsymbol{g} \odot \boldsymbol{g}$
. Correct bias in first moment: $\hat{s} \leftarrow \frac{s}{1-\rho_1^t}$
. Correct bias in second moment: $\hat{\boldsymbol{r}} \leftarrow \frac{1}{1-\rho_2^t}$
. Compute update: $\Delta \boldsymbol{W} = -\epsilon \frac{\hat{s}}{\sqrt{\hat{r}}+\delta}$ (operations applied element-wise)
. Apply update: $\boldsymbol{W} \leftarrow \boldsymbol{W} + \Delta \boldsymbol{W}$
end while

$$\boldsymbol{W}^* \leftarrow \boldsymbol{W}$$

1.6 Conclusion

Recently, deep learning, in particular, deep neural networks have demonstrated a great performance in many challenging machine learning and artificial intelligence tasks such as classification, clustering and prediction. In this Chapter, the first section consecrated to define deep learning in general as one of the most challenging machine learning fields, As well as, a brief history of deep learning has been presented. In Section 1.3, we have introduced convolutional neural network as one of the most important and successful deep learning models. Indeed, each components of CNN is detailed in this Section. Section 1.4 is object of loss functions widely used in deep neural networks. Optimization as an extremely challenging task, is object of Section 1.5, in which we firstly introduced the formulation of optimization for training deep neural networks, then, we presented competitive optimization algorithms and we highlighted advantages and drawbacks of each algorithm.

Chapter 2

Deep neural networks regularization and generalization: A mathematical approach

2.1 Introduction

Deep neural networks as presented in the previous Chapter, characterized by a huge number of learnable parameters [**16**, **30**, **50**, **90**]. Furthermore, the efficient deep learning models are with large and complex architectures [**68**, **74**, **98**]. As a results, learning these models prone to overfitting problems, especially when trained on a relatively small training set. This leads to poor performance and they lack the generalization ability to unseen data [**61**, **97**, **59**]. To deal effectively with this issue, regularization is a set of techniques that can prevent overfitting in deep neural networks by controlling the variance type error, and thus improves the accuracy of a deep learning model when facing completely new data from the problem domain [**49**]. The main idea behind this is to make the model more simple by imposing some constraints. Generally speaking, it reduces the variance type error and improves the generalization of the solution [**60**]. There are many forms of regularization, including the Euclidean-type regularization L_2 which favors low coefficients while a lasso-type regularization L_1 (used when the number of input variables is high) will promote the "sparsity" of the representation by pushing the algorithm to take into account only a small part of the data, ignoring the others [**108**].
Recently, Dropout, proposed in [**74**], is one of the most successful regularization methods that has been applied widely to DNN. The basic idea is to randomly drop neurons from the deep neural network during training with probability ρ sampled from a Bernoulli distribution. Dropout has proved its efficiency in many applications [**35**, **73**, **54**]. However, the theoretical understanding of dropout is still far from clear.
The rest of this Chapter is organized as follow: In the next section, we present the most competitive and efficient regularization techniques for DNNs. In Section 2.3, we develop a theoretical analysis related to the Rademacher complexity in order to exploit it to derive an estimation of dropconnect hyperparameter in the next Chapter.

2.2 Regularization for DNNs

A central problem in deep learning is how to make an algorithm that will perform well not just on the training data, but also on new inputs. Many strategies used in deep learning are explicitly designed to reduce the test error, possibly at the expense of increased training error. These strategies are known collectively as regularization. As we will see there are a great many forms of regularization for deep neural networks. In fact, developing more effective regularization strategies has been one of the major research efforts in the field [**34**, **109**, **111**, **129**]. We can categorize common regularization approaches into the following classes, based on their central idea:

- Approaches which regularize the network using data level techniques (e.g., data augmentation);
- Approaches which introduce stochastic behavior in the neural activations (e.g., dropout and drop connect);
- Approaches which normalize batch statistics in the feature activations (e.g., batch normalization);
- Approaches which use decision level fusion to avoid over-fitting (e.g., ensemble model averaging);
- Approaches which introduce constraints on the network weights (e.g., l^1 norm, l^2 norm, max-norm, and elastic net constraints);
- Approaches which use guidance from a validation set to halt the learning process (e.g., early stopping).

In the remainder of this Section, we discuss most important of these approaches.

2.2.1 L^1 and L^2 regularization

One form of regularization commonly used to optimize learning is to penalize the weights of the network. This penalty allows to limit the complexity of the model and to reduce the risk of overfitting. This amounts to adding a term $\alpha\Omega(\mathbb{W})$ to the objective function to penalize the weights $\mathbb{W}$. The equation 2.1 is the result of adding this term to the cost function. The coefficient λ controls the amplitude of the regularization.

$$\arg\min_{\mathbb{W}} \frac{1}{m} \sum_i L\left(F\left(x^{(i)}; \mathbb{W}\right), y^{(i)}\right) + \lambda\Omega(\mathbb{W}) \tag{2.1}$$

Generally the used regularization functions are the Lasso regression [**108**] and the Tikhonov regression, often called L_1 regularization or L_2 norm respectively. The penalty term in equation (2.1), for the L_1 regularization is written:

$$\lambda\Omega(\mathbb{W}) = \lambda\|\mathbb{W}\|_1 \tag{2.2}$$

and for the L_2 regularization:

$$\lambda\Omega(\mathbb{W}) = \lambda\|\mathbb{W}\|_2 \tag{2.3}$$

Although the terms of equations (2.2) and (2.3) both penalize $\mathbb{W}$, the effect of these two regularization forms is different. Indeed, the use of the L_2 norm will have the effect of reducing the weights whose covariance with the input is low, thus favoring the weights that best characterize the variance of the model. This makes the cost function J_{train} as convex as possible and makes it easier to find the global minimum. Using the L_1 norm will have the effect of creating a parsimonious model by tending part of the model weights towards zero to select only part of the neurons in the network in learning and also in inference [**50**, **108**].

2.2.2 Batch normalization

In the learning task, usually, the data are normalized for the purpose of making them comparable in terms of the features. However, there is a change in the distribution of network activations due to the change in network parameters during training. In this context, batch normalization (BN) is one of the successful approaches that greatly improve convergence during training [**34**]. It consists in normalizing on average and in variance the outputs of the layers of the network; Lets consider a mini batch $B = \{X_1, X_2, \ldots, X_m\}$ of size m, and $(y_1, y_2, \ \ldots, y_m)$ are their linear transformations. Batch normalization $BN_{\gamma,\beta}$ is referred to the transform:

$$\begin{aligned} BN_{\gamma,\beta}: \ & \{X_1, X_2 \ldots, X_m\} \longrightarrow \{Y_1, Y_2 \ldots, Y_m\} \\ & X_i \longrightarrow Y_i = \gamma \cdot \hat{X}_i + \beta \end{aligned}$$

where $\hat{X}_i = \frac{X_i - \mu_B}{\sqrt{\sigma_B^2 + \varepsilon}}$ is the normalized value of X_i and

$$(\mu_{\mathrm{B}}, \sigma_B^2) = (\frac{1}{m}\sum_{i=1}^{m} X_i, \frac{1}{m}\sum_{i=1}^{m} (X_i - \mu_B)^2)$$

denote the mean and the variance of X.

2.2.3 Dropout regularization

Dropout was first explained in [**74**], but one could find more details about it in [**61**] and especially in [**123**]. Dropout is applied to each layer of the fully connected in the following manner: according to a dropout parameter ρ ranging from 0 to 1 (to be interpreted as a Bernoulli probability), a set of units are dropped in each epoch. This behavior forces the network to learn redundancies so it is better in isolating the necessary properties of the dataset. Mathematically, if x and W denote respectively the input and weights of a layer, then the output of this layer after applying Dropout is expressed as:

$$Y_{dropout} = m \circ f(W.x) \tag{2.4}$$

where $\circ$ denotes the Hadamard product, f is a non linear activation function, and m is a binary vector, in which each element m_j is drawn independently from Bernoulli distribution with parameter ρ:

$$m_j \sim Bernoulli(\rho) \tag{2.5}$$

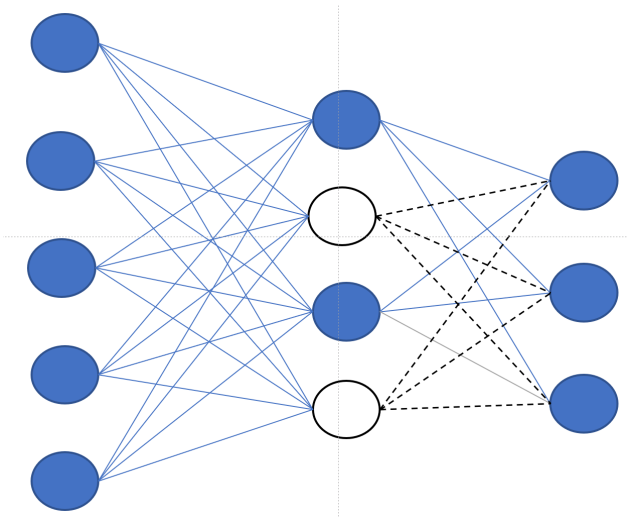

Figure 2.1 – Dropout regularization

An example of a Dropout regularizer applied to a a deep neural network is illustrated in Fig.2.1.

2.2.4 DropConnect regularization

DropConnect is a regularization technique that generalizes Dropout [**111**]. In fact, connections of a fully connected layer, rather than each output unit, can be dropped with probability ρ. DropConnect can be considered as a dynamic sparsity generator within the model (Fig. 2.2). In other words, the sparsity within the network is produced by connections that are chosen at random during the training stage. Note that the term "dynamic" means that the parameters W are not fixed as known in the classical sparse forms. Mathematically, the output of a DropConnect layer can be expressed by:

$$Y_{DropConnect} = f((M \circ W).x) \tag{2.6}$$

where M is a binary mask with each element, M_{ij} is drawn independently from Bernoulli distribution with parameter ρ :

$$M_{ij} \sim Bernoulli(\rho) \tag{2.7}$$

The hyperparameter ρ choice has been analytically discussed in the case of dropout such as in [**35**, **104**], but in the case of DropConnect, it is still unclear and needs a theoretical understanding for the purpose of its estimation. In this sense, we exploit the surprising results, provided by Rademacher complexity in minimizing generalization gap, to provide an estimation of DropConnect hyperparameter.

In the next Section, we develop a theoretical analysis related to the Rademacher complexity in order to exploit it to derive an estimation of dropconnect hyperparameter in the next Chapter.

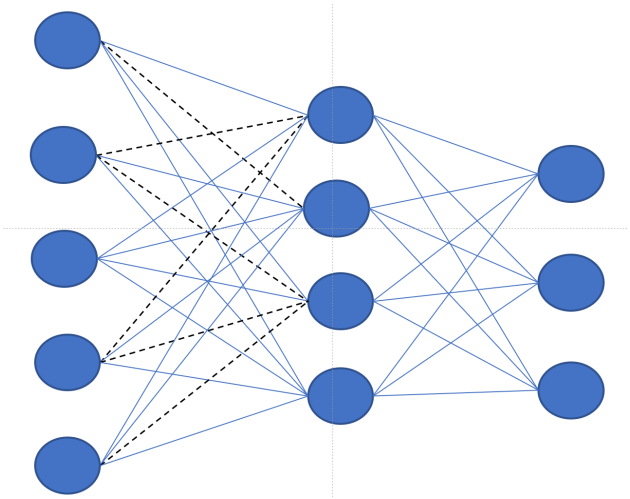

Figure 2.2 – DropConnect regularization

2.3 Genralization gap optimization using Rademacher complexity

In this section, we firstly introduce Rademacher complexity as an alternative notion of the complexity of a function class. Moreover, its fundamental proprieties are developed and demonstrated. Next, we present the generalization gap notion of deep neural networks. Finlay, we derive an estimation of generalization gap using rademacher complexity.

2.3.1 Rademacher complexity

Firstly, We introduce the definition of Rademacher complexity of a function class

Definition 2: Rademacher complexity

Let μ be a probability distribution on a set $\mathcal{X}$ and suppose that $X_1, \ldots, X_n$ are independent samples selected according to μ. Let F be a class of functions mapping from $\mathcal{X}$ to $\mathbb{R}$. Define the random variable

$$\hat{R}_n(F) = \mathbf{E}_\sigma \left[\sup_{f \in F} \left| \frac{2}{n} \sum_{i=1}^{n} \sigma_i f(X_i) \right| \mid X_1, \ldots, X_n \right] \tag{2.8}$$

where $\sigma_1, \ldots, \sigma_n$ are independent uniform $\{\pm 1\}$-valued random variables. Then the Rademacher complexity of F is

$$R_n(F) = \mathbf{E}_X \hat{R}_n(F) \tag{2.9}$$

$R_n(F)$ quantifies the extent to which some function in the class F can be correlated with a noise sequence of length n.

Before developing fundamental results related to Rademacher complexity, we highlight two important theorems: theorem of Vapnik and Chervonenkis [**134**] and McDiarmid's Inequality [**133**]

Theorem 2.1: (Vapnik and Chervonenkis, 1971)

Let F be a class of $\{\pm 1\}$-valued functions defined on a set $\mathcal{X}$. Let P be a probability distribution on $\mathcal{X} \times \{\pm 1\}$, and suppose that $(X_1, Y_1), \ldots, (X_n, Y_n)$ and (X, Y) are chosen independently according to P. Then, there is an absolute constant c such that for any integer n, with probability at least $1 - \delta$ over samples of length n, every f in F satisfies

$$P(Y \neq f(X)) \leq \hat{P}_n(Y \neq f(X)) + c\sqrt{\frac{\mathrm{VCdim}(F)}{n}}$$

where $\mathrm{VCdim}(F)$ denotes the Vapnik-Chervonenkis dimension of F,

$$\hat{P}_n(S) = \frac{1}{n}\sum_{i=1}^{n} 1_S\left(X_i, Y_i\right)$$

and 1_S is the indicator function of S.

Theorem 2.2: (McDiarmid's Inequality, 1989)

Let $X_1, \ldots, X_n$ be independent random variables taking values in a set A, and assume that $f : A^n \to \mathbb{R}$ satisfies

$$\sup_{x_1,\ldots,x_n,x_i' \in A} \left|f\left(x_1, \ldots, x_n\right) - f\left(x_1, \ldots, x_{i-1}, x_i', x_{i+1}, \ldots x_n\right)\right| \leq c_i$$

for every $1 \leq i \leq n$. Then, for every $t > 0$

$$P\left\{f\left(X_1, \ldots, X_n\right) - \mathbf{E}f\left(X_1, \ldots, X_n\right) \geq t\right\} \leq e^{-2t^2/\sum_{i=1}^{n} c_i^2}$$

The following result is an example of the usefulness of Rademacher complexity.

Theorem 2.3

Let P be a probability distribution on $\mathcal{X} \times \{\pm 1\}$, let F be a set of $\{\pm 1\}$-valued functions defined on $\mathcal{X}$, and let $(X_i, Y_i)_{i=1}^n$ be training samples drawn according to P^n. With probability at least $1 - \delta$, every function f in F satisfies

$$P(Y \neq f(X)) \leq \hat{P}_n(Y \neq f(X)) + \frac{R_n(F)}{2} + \sqrt{\frac{\ln(1/\delta)}{2n}}$$

Proof. We set $\mathcal{L}(Y, f(X)) = 1(Y \neq f(X))$, then, for all $f \in F$,

$$P(Y \neq f(X)) = \mathbf{E}\mathcal{L}(Y, f(X)) \leq \hat{\mathbf{E}}_n\mathcal{L}(Y, f(X)) + \sup_{h \in \mathcal{L} \circ F}\left(\mathbf{E}h - \hat{\mathbf{E}}_n h\right)$$

In this case, when (X_i, Y_i) changes, the supremum changes by no more than $1/n$, so McDiarmid's inequality implies that with probability at least $1-\delta$, every $f \in F$ satisfies

$$\mathbf{E}\mathcal{L}(Y, f(X)) \leq \hat{\mathbf{E}}_n \mathcal{L}(Y, f(X)) + \mathbf{E} \sup_{h \in \mathcal{L} \circ F} \left(\mathbf{E}h - \hat{\mathbf{E}}_n h\right) + \sqrt{\frac{\ln(1/\delta)}{2n}}.$$

Then, we have

$$\begin{aligned}
\mathbf{E} \sup_{h \in \mathcal{L} \circ F} \left(\mathbf{E}h - \hat{\mathbf{E}}_n h\right) &\leq \mathbf{E} \sup_{h \in \mathcal{L} \circ F} \frac{2}{n} \sum_{i=1}^{n} \sigma_i h\left(X_i, Y_i\right) \\
&= \mathbf{E} \sup_{f \in F} \frac{2}{n} \sum_{i=1}^{n} \sigma_i \mathbf{1}\left(Y_i \neq f\left(X_i\right)\right) \\
&= \mathrm{E}\sup_{f \in F} \frac{2}{n} \sum_{i=1}^{n} \sigma_i \left(1 - Y_i f\left(X_i\right)\right)/2 \\
&= \mathbf{E} \sup_{f \in F} \frac{1}{n} \sum_{i=1}^{n} \sigma_i f\left(X_i\right) \\
&= \frac{R_n(F)}{2}
\end{aligned}$$

where we have used the fact that $Y_i, f\left(X_i\right) \in \{\pm 1\}$, and that the conditional distribution of $\sigma_i Y_i$, given Y_i, is the same as the distribution of σ_i.

□

The following theorem represents a major result related to Redmacher complexity

Theorem 2.4

Consider a loss function $\mathcal{L} : \mathcal{Y} \times \mathcal{A} \to [0,1]$ and a dominating cost function $\phi : \mathcal{Y} \times \mathcal{A} \to [0,1]$. Let F be a class of functions mapping from $\mathcal{X}$ to $\mathcal{A}$ and let $(X_i, Y_i)_{i=1}^n$ be independently selected according to the probability measure P. Then, for any integer n and any $0 < \delta < 1$, with probability at least $1-\delta$ over samples of length n, every f in F satisfies

$$\mathbf{E}\mathcal{L}(Y, f(X)) \leq \hat{\mathbf{E}}_n \phi(Y, f(X)) + R_n(\tilde{\phi} \circ F) + \sqrt{\frac{8\ln(2/\delta)}{n}}$$

where $\tilde{\phi} \circ F = \{(x, y) \mapsto \phi(y, f(x)) - \phi(y, 0) : f \in F\}$

Proof. Since ϕ dominates $\mathcal{L}$, for all $f \in F$ we can write

$$\begin{aligned}
\mathbf{E}\mathcal{L}(Y, f(X)) \leq& \mathbf{E}\phi(Y, f(X)) \\
\leq& \hat{\mathbf{E}}_n \phi(Y, f(X)) + \sup_{h \in \phi \circ F} \left(\mathbf{E}h - \hat{\mathbf{E}}_n h\right) \\
=& \hat{\mathbf{E}}_n \phi(Y, f(X)) + \sup_{h \in \tilde{\phi} \circ F} \left(\mathbf{E}h - \hat{\mathbf{E}}_n h\right) \\
&+ \mathbf{E}\phi(Y, 0) - \hat{\mathbf{E}}_n \phi(Y, 0)
\end{aligned}$$

When an (X_i, Y_i) pair changes, the random variable $\sup_{h\in\tilde{\phi}\circ F}\left(\mathbf{E}h - \hat{\mathbf{E}}_n h\right)$ can change by no more than $2/n$. McDiarmid's inequality implies that with probability at least $1-\delta/2$,

$$\sup\left(\mathbf{E}h - \hat{\mathbf{E}}_n h\right) \leq \mathbf{E}\sup\left(\mathbf{E}h - \hat{\mathbf{E}}_n h\right) + \sqrt{2\ln(2/\delta)/n}.$$

A similar argument, together with the fact that $\mathbf{E}\hat{\mathbf{E}}_n\phi(Y,0) = \mathbf{E}\phi(Y,0)$, shows that with probability at least $1-\delta$,

$$\mathbf{E}\mathcal{L}(Y, f(X)) \leq \hat{\mathbf{E}}_n\phi(Y, f(X)) + \mathbf{E}\sup_{h\in\tilde{\phi}\circ F}\left(\mathbf{E}h - \hat{\mathbf{E}}_n h\right) + \sqrt{\frac{8\ln(2/\delta)}{n}}.$$

It remains to show that the second term on the right hand side is no more than $R_n(\tilde{\phi}\circ F)$. If $(X_1', Y_1'), \ldots, (X_n', Y_n')$ are independent random variables with the same distribution as (X, Y), then

$$\begin{aligned}
\mathbf{E}\sup_{h\in\tilde{\phi}\circ F}\left(\mathbf{E}h - \hat{\mathbf{E}}_n h\right) &= \mathbf{E}\sup_{h\in\tilde{\phi}\circ F}\mathbf{E}\left[\frac{1}{n}\sum_{i=1}^{n} h\left(X_i', Y_i'\right) - \hat{\mathbf{E}}_n h \mid (X_i, Y_i)\right] \\
&\leq \mathbf{E}\sup_{h\in\tilde{\phi}\circ F}\left(\frac{1}{n}\sum_{i=1}^{n} h\left(X_i', Y_i'\right) - \hat{\mathbf{E}}_n h\right) \\
&= \mathbf{E}\sup_{h\in\tilde{\phi}\circ F}\frac{1}{n}\sum_{i=1}^{n}\sigma_i\left(h\left(X_i', Y_i'\right) - h\left(X_i, Y_i\right)\right) \\
&\leq 2\mathbf{E}\sup_{h\in\tilde{\phi}\circ F}\frac{1}{n}\sum_{i=1}^{n}\sigma_i h\left(X_i, Y_i\right) \\
&\leq R_n(\tilde{\phi}\circ F).
\end{aligned}$$

□

In the next theorem, we aim to develop some fundamental proprieties of Rademacher complexity

Theorem 2.5

Let $F, F_1, \ldots, F_k$ and H be classes of real functions. Then

1. If $F \subseteq H$, then, $R_n(F) \leq R_n(H)$
2. $R_n(F) = R_n(\text{conv}\, F) = R_n(\text{absconv}\, F)$
3. For every $c \in \mathbb{R}$, $R_n(cF) = |c| R_n(F)$
4. If $\phi : \mathbb{R} \to \mathbb{R}$ is Lipschitz with constant L_ϕ and satisfies $\phi(0) = 0$, then
$$R_n(\phi \circ F) \leq 2L_\phi R_n(F)$$
5. For any uniformly bounded function
$$h, R_n(F + h) \leq R_n(F) + \|h\|_\infty / \sqrt{n}$$
6. For $1 \leq q < \infty$, let $\mathcal{L}_{F,h,q} = \{|f - h|^q \mid f \in F\}$, where h is uniformly bounded. If $\|f - h\|_\infty \leq 1$ for every $f \in F$, then,
$$R_n\left(\mathcal{L}_{F,h,q}\right) \leq 2q\left(R_n(F) + \|h\|_\infty / \sqrt{n}\right)$$
7. $R_n\left(\sum_{i=1}^k F_i\right) \leq \sum_{i=1}^k R_n\left(F_i\right)$

Proof. Parts 1 and 3 are immediate from the definitions. To see part 2, notice that for every $x_1, \ldots, x_n$ and $\sigma_1, \ldots, \sigma_n$,

$$\begin{aligned}
&\sup_{f \in \text{ absconv } F} \left| \sum_{i=1}^n \sigma_i f\left(x_i\right) \right| \\
&= \sup_{f \in \text{conv } F} \left| \sum \sigma_i f\left(x_i\right) \right| \\
&= \max\left(\sup_{f \in \text{conv } F} \sum \sigma_i f\left(x_i\right), \sup_{f \in \text{conv } F} -\sum \sigma_i f\left(x_i\right) \right) \\
&= \max\left(\sup_{f \in F} \sum \sigma_i f\left(x_i\right), \sup_{f \in F} -\sum \sigma_i f\left(x_i\right) \right) \\
&= \sup_{f \in F} \left| \sum \sigma_i f\left(x_i\right) \right|.
\end{aligned}$$

The inequality of part 4 is due to Corollary 3.17 in the work of Ledoux and Talagrand

[135]. As for part 5 , note that for every realization of $X_1, \ldots, X_n$,

$$\begin{aligned}
\mathbf{E} \sup_{f \in F} \left| \sum_{i=1}^{n} \sigma_i \left(f\left(x_i\right) + h\left(x_i\right) \right) \right| & \\
& \leq \mathbf{E} \sup_{f \in F} \left| \sum_{i=1}^{n} \sigma_i f\left(x_i\right) \right| + \mathbf{E} \left| \sum_{i=1}^{n} \sigma_i h\left(x_i\right) \right| \\
& \leq \mathrm{E} \sup_{f \in F} \left| \sum_{i=1}^{n} \sigma_i f\left(x_i\right) \right| + \left(\sum_{i=1}^{n} h\left(x_i\right)^2 \right)^{\frac{1}{2}},
\end{aligned}$$

where the last inequality follows since for any function g, $\mathbf{E}|g| \leq \left(\mathbf{E}g^2\right)^{1/2}$. Hence,

$$R_n(F + h) \leq R_n(F) + \|h\|_\infty / \sqrt{n},$$

as claimed. To see part 6 , notice that $\phi(x) = |x|^q$ is a Lipschitz function which passes through the origin with a Lipschitz constant q. By parts 4 and 5, we obtain

$$R_n\left(\mathcal{L}_{F,h,q}\right) \leq 2qR_n(F - h) \leq 2q\left(R_n(F) + \frac{\|h\|_\infty}{\sqrt{n}}\right).$$

Finally, part 7 follows from the triangle inequality.

□

2.3.2 Deep neural networks generalization gap

Definition 3

Lets $\mathcal{L}$ the loss function used to measure the performance of deep neural network. The expected risk is defined as

$$R_{exp} = \mathbf{E}_{(Y,X)} \mathcal{L}(Y, f(X)) \tag{2.10}$$

The empirical risk is defined as

$$R_{emp} = \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}(y_i, f(x_i)) \tag{2.11}$$

where $(y_1, x_1), (y_2, x_2), \cdots, (y_n, x_n)$ are drawn i.i.d. from distribution (Y, X)

Definition 4

Generalization gap is a quantity that measures the generalization ability of DNNs. It is defined as

$$G_{gap} = |R_{exp} - R_{emp}| \tag{2.12}$$

Once a deep neural network is overfitting, R_{exp} will be much larger than R_{emp} and its generalization gap is written as

$$G_{gap} = R_{exp} - R_{emp} \tag{2.13}$$

Based on theorem 2.4 and using the proprieties of Rademacher complexity, we obtain the following theorem which presents the generalization gap bound using Rademacher complexity.

Theorem 2.6: (Estimation of generalization gap)

Let's assume that the loss function $\mathcal{L}$ is bounded with constant B and all training samples are i.d.d sampled. Then, the following inequality holds with probability $1 - \delta$:

$$G_{gap} \leq 2R_n(\mathcal{L}(Y, f(X)) + B\sqrt{\frac{\ln(2/\delta)}{n}} \tag{2.14}$$

2.4 Conclusion

Deep neural networks are a promising tool for solving real-world problems. However, in order to receive a good performance of prediction and accomplish more difficult goals, the deep learning architecture becomes deeper and has a large number of parameters. As a result, training a deep neural network becomes more complicated and brings three fundamental problems; Firstly, the training and the inference are cumbersome and take a long time. Secondly, the training leads to overfitting problem, and finally, the redundancy problem that slows the network performance. To deal effectively with these issues, regularization is a set of techniques that can prevent overfitting in deep neural networks by controlling the variance type error, and thus improves the accuracy of a deep learning model when facing completely new data from the problem domain. There are many forms of regularization, including the Euclidean-type regularization L_2, L_1 regularization, Batch normalization, Dropout, and DroConnect as presented in Section 2.2. In Section 2.3, we have developed a theoretical analysis related to the Rademacher complexity in order to exploit it to derive an estimation of dropconnect hyperparameter in the next Chapter.

Chapter 3

Estimation of DropConnect hyperparameter using generalization gap and Rademacher complexity

3.1 Abstract

Deep neural networks (DNNs) have solved numerous challenging reels problems. However, successful DNNs often require a large number of parameters, which may produce some undesirable phenomena, notably overfitting. DropConnect, which is a generalization of Dropout, is one of the successful stochastic regularizers that prevent overfitting in deep neural networks. Indeed, it allows dropping some parameters according to a fixed probability, generating at the end dynamic sparse DNNs. The study of the DropConnect hyperparameter is still not estimated and needs a theoretical understanding. In this context, we propose an estimation of the DropConnect hyperparameter using the gap generalization and the Rademacher complexity. This estimation gives rise to a new DropConnect technique named Adaptive DropConnect (A-DropConnect), in which the studied hyperparameter is updated using the data-dependent during training. Efficiency of A-DropConnect is demonstrated by several experiments. Numerical results demonstrate that the proposed method yields significant improvement in classification performance compared to the state of the art.

Keywords: Hypermarameter estimation. Deep neural networks. DropConnect. Regularization. Rademacher complexity. Generalization gap.

3.2 Introduction

A deep neural network (DNN) has obtained a great attention for solving engineering problems including computer vision, natural language processing, speech recognition, and many others [**52**, **51**]. A deep neural network is an artificial neural network (ANN) with several hidden layers and deeper combinations, which is responsible for learning from data with impressive capability. However, the biggest issue faced by machine learning experts is the overfitting problem. Indeed, it refers to the phenomenon where a neural network learns from data but also from patterns, considered as noise, that are not related to the problem. Thus, overfitting describes a complex model with a very high variance type error.
To deal effectively with this issue, regularization is a set of techniques that can prevent overfitting in deep neural networks by controlling the variance type error, and thus improves the accuracy of a deep learning model when facing completely new data from the problem domain. The main idea behind this is to make the model more simple by imposing some constraints. Generally speaking, it reduces the variance type error and improves the generalization of the solution. There are many forms of regularization, including the Euclidean-type regularization L_2 which favors low coefficients while a lasso-type regularization L_1 (used when the number of input variables is high) will promote the "sparsity" of the representation by pushing the algorithm to take into account only a small part of the data, ignoring the others [**108**].
Recently, Dropout, proposed in [**74**], is one of the most successful regularization methods that has been applied widely to DNN. The basic idea is to randomly drop neurons from the deep neural network during training with probability ρ sampled from a Bernoulli distribution. Dropout has proved its efficiency in many applications [**35**, **73**, **54**]. However, the theoretical understanding of dropout is still far from clear. For this purpose, Dropout has been analytically studied by several researchers including the Bayesian approach which interpreted the dropout properties [**57**]. An empirical analysis of dropout in linear networks has been discussed by David et al. who respond to several questions related to Dropout [**25**]. In [**59**], Wager et al. showed that the dropout regularizer is first-order equivalent to an L_2 regularizer. An extensive study of the properties of dropout's uncertainty has been performed by Gal et al. [**57**]. Recently, Rademacher complexity has been employed to provide some theoretical results of the Dropout functioning. In fact, the generalization gap of DNNs, which is defined as the absolute difference between the empirical and theoretical risks [**68**], has been expressed by two offsets Rademacher Complexities [**60**]. Rademacher has been applied to the Dropout mechanism in [**58**]. Vladimir et al. have built a data-dependent upper bound on the risk in function learning problems, these bounds are related to the local norms of the Rademacher process [**105**]. Rademacher complexity was also studied for different types of dropouts and has been reduced in a polynomial in shallow neural networks and exponentially for DNNs [**58**]. The generalization gap has been optimized using Rademacher complexity to formulate a new Drop technique [**104**].
Unlike Dropout, where a randomly selected subset of activations are set to zero within each layer, DropConnect is another regularization form that generalizes Dropout by randomly dropping the weights [**111**]. It proved its efficiency in many applications such as

the regularization of extreme learning machine [**56**], learning vector quantizer [**131**] and mobile sensing networks [**7**]. Despite this wealth of applications, the mathematical study of DropConnect, as well as its functioning, is far from clear, with some exceptions such as the regularization of Neural Networks using DropConnect proposed by Wan et al. [**111**], and the Monte Carlo DropConnect (MC-DropConnect), in which the authors have developed a theoretical framework to approximate Bayesian inference for DNNs by imposing a Bernoulli distribution on the model weights [**55**].
Since DropConnect is essentially relying on the choice of Bernouilli hyperparameter, and motivated by the lack of its theoretical study, we propose in this chapter an estimation of the DropConnect hyperparameter using both gap generalization and Rademacher complexity. Indeed, we firstly define a DropConnect Rademacher complexity by injecting Bernouilli masks into the network. Next, we formulate an upper bound of generalization gap using a constraint function related to DropConnect hyperparameter. This later is finally estimated from the optimization of the upper bound generating an adaptive DropConnect (A-DropConnect). Experimental analysis shows significant improvements in classification performance on the MNIST, Fashion-MNIST, NORB, and SVHN datasets. The main contributions of this chapter are summarized as follows:

- Derivation of a generalization gap upper bound using DropConnect Rademacher complexity.
- Minimizing the obtained upper bound which is a constraint function of the DropConnect hyperparameters.
- Estimation of DropConnect hyperparameter using the gap generalization optimization.

The remainder of this chapter is organized as follows: In Section 3.3, the estimation of the DropConnect hyperparameter is elaborated through several theoretical studies. Section 3.4 details the experiments and presents the numerical results. Section 3.5 provides a brief conclusion and directions for future work.

3.3 Estimation of DropConnect hyperparameter using Rademacher complexity and generalization gap

Let $D = \{(x_1, y_1), (x_2, y_2), ..., (x_m, y_m)\}$ be a dataset and let consider a deep neural network composed of N hidden layers $\{\mathfrak{h}_1, \mathfrak{h}_2, ..., \mathfrak{h}_N\}$. Each hidden layer $\mathfrak{h}_k$ has n_k neurons and $W^k = \{W_1^k, W_2^k, ..., W_{n_{k+1}}^k\}$ represents the weights between layers $\mathfrak{h}_k$ and $\mathfrak{h}_{k+1}$ with $W_i^k \in \mathbb{R}^{n_k}$.
In the next, we begin by giving some essential theoretical preliminaries needed to develop the DropConnect hyperparmeter estimation.

3.3.1 Rademacher DropConnect complexity and estimation of generalization gap

Based on the definition of expected and empirical risks adapted to Dropout [**58**], we respectively define these errors for DropConnect ($R_{dc.exp}$ and $R_{dc.emp}$) as follows:

$$R_{dc.exp} = E_{(X,Y),M}\Big[l\big(F(\mathbb{W}, X, M), Y\big)\Big] \tag{3.1}$$

and

$$R_{dc.emp} = \frac{1}{m}\sum_{j=1}^{m} l(F(\mathbb{W}, x_j, M_j), y_j) \tag{3.2}$$

where l and $F(\mathbb{W}, x_k, M_k)$ denote respectively loss function and the network output for the kth sample. $M_k = \{M_k^1, M_k^2, ..., M_k^N\}$ with M_k^l represents the binary mask for the kth sample applied to the connections between $\mathfrak{h}_{l-1}$ and $\mathfrak{h}_l$ i.e,

$$M_k^l \in \mathcal{M}_{n_l, n_{l+1}}(\{0,1\}) \; ; \quad M_{kij}^l \sim Bernoulli(\rho)$$

using $R_{dc.exp}$ and $R_{dc.emp}$, the generalization gap is expressed by:

$$G_{gap} = |R_{dc.exp} - R_{dc.emp}| \tag{3.3}$$

where in the case of overfitting, $R_{dc.exp}$ will be much larger than $R_{dc.emp}$ and its generalization gap is written as $G_{gap} = R_{dc.exp} - R_{dc.emp}$.

Definition 5

Let $g : \mathbb{R} \to \mathbb{R}$ a real-valued function. g is called P-Lipschitz continuous function if there exists a positive real constant P such that, for all real y_1 and y_2,

$$|g(y_1) - g(y_2)| \leq P|y_1 - y_2|$$

Moreover, a real-valued two variable function $h(y, t)$ is called P-Lipschitz continuous with its first argument if there exists a positive real constant P such that, for all y_1 and y_2:

$$|h(y_1, t) - h(y_2, t)| \leq P|y_1 - y_2| \tag{3.4}$$

Unlike the explicit definition of Rademacher Dropout complexity in which the mask q is not related to the rank of samples [**58**, **104**], our proposed Rademacher DropConnect complexity takes into consideration this technical particularity. It is defined as follows:

Definition 6

The empirical DropConnect Rademacher complexity and DropConnect Rademacher complexity are respectively defined as:

$$\widehat{\mathcal{R}}_m(\mathcal{G}, X, M) = E_\sigma\Big[\sup_{g\in\mathcal{G}} \frac{1}{m}\sum_{j=1}^{m} \sigma_j g(x_j, M_j)\Big] \tag{3.5}$$

$$\mathcal{R}_m(\mathcal{G}) = E_{X,M}\Big[\widehat{\mathcal{R}}_m(\mathcal{G}, X, M)\Big]$$

where $(\sigma_1, \sigma_2, ..., \sigma_m)$ are Rademacher variables and $\mathcal{G}$ is a real-valued function space.

The following Lemmas are mandatory for the generalization gap. Thus, the details of theirs proofs are explicitly presented in **[103, 58]**.

Lemma 3.1

For a function space $\mathcal{G}$, we define

$$absconv(\mathcal{G}) = \Big\{\sum \beta_k g_k : \quad g_k \in \mathcal{G} \text{ and } \sum|\beta_k| = 1\Big\}$$

Then we have:

$$\widehat{\mathcal{R}}_m(\mathcal{G}, X, M) = \widehat{\mathcal{R}}_m(absconv(\mathcal{G}), X, M) \tag{3.6}$$

Lemma 3.2

If f denotes a L-Lipschitz continuous function and $f(0) = 0$. Then, we have

$$E_\sigma\Big[\sup_{g\in\mathcal{G}} \frac{1}{m}\sum_{j=1}^{m} \sigma_j f(g(x^j, M^j))\Big] \leq LE_\sigma\Big[\sup_{g\in\mathcal{G}} \frac{1}{m}\sum_{j=1}^{m} \sigma_j g(x^j, M^j)\Big] \tag{3.7}$$

Lemma 3.3

Let $l(.,.)$ is a D-Lipschitz continuous with its first argument. Thus, we have

$$\mathcal{R}_m\Big(l\big(F(\mathbb{W}, X, M), Y\big)\Big) \leq D\mathcal{R}_m\Big(F(\mathbb{W}, X, M), Y\Big) \tag{3.8}$$

Lemma 3.4

The upper bound of generalization gap is expressed, using Rademacher DropConnect complexity, as follows

$$G_{gap} \leq 2\mathcal{R}_m\Big(l\big(F(\mathbb{W}, X, M), Y\big)\Big) + A\sqrt{\ln(2/\eta)/m} \tag{3.9}$$

where $1-\eta$ is the probability of holding Eq. (3.9), and A is a constant bound of l.

In the next, we aims to build an upper bound of $\mathcal{R}_m\Big(l\big(F(\mathbb{W}, X, M), Y\big)\Big)$ that is related to DropConnect masks M. Thereafter, we minimize this bound developing at the end an estimation of DropConnect hyperparameter ρ.

3.3.2 Adaptive DropConnect: An improved regularization technique

3.3.2.1 Estimation of deep neural network generalization gap

During training, the input ψ_j^k to each layer $\mathfrak{h}_k$ is expressed as follows

$$\begin{aligned}\psi_j^k &= f\Big((M_j^{k-1} \odot W^{k-1}).\psi_j^{k-1}\Big)\\ &= \{\psi_j^k(1), \psi_j^k(2), ..., \psi_j^k(n_k)\}\end{aligned}$$

which is explicitly given by

$$\psi_j^k = \left\{f\Big((M_{j,1}^{k-1} \odot W_1^{k-1}).\psi_j^{k-1}\Big), f\Big((M_{j,2}^{k-1} \odot W_2^{k-1}).\psi_j^{k-1},\Big), ..., f\Big((M_{j,n_k}^{k-1} \odot W_{n_k}^{k-1}).\psi_j^{k-1},\Big)\right\} \tag{3.10}$$

The following theorem provides an upper bound of Rademacher DropConnect complexity.

Theorem 3.1

Let assume that $\mathbb{W}$ satisfies the following assumptions

$$\begin{cases} \bullet & \|W^N\|_1 \leq B_N \\ \bullet & \max\limits_k \|W_k^j\|_1 \leq B_j for 1 \leq k \leq N-1 \\ \bullet & \max\limits_k \|W_k^q\|_2 \leq C_q \end{cases}$$

where B_j and C_q are two constants. Then,

$$\mathcal{R}_m\Big(l\big(F(\mathbb{W}, X, M), Y\big)\Big) \leq C(M) = \frac{DC_q}{m}\Big(\prod_{k=q}^{N} B_k\Big)L^{N-q}E_X\left[\sqrt{E_M\Big\|\sum_{j=1}^{m} M_{j,1}^q \odot \psi_j^q\Big\|_2^2}\right] \tag{3.11}$$

Proof.

$$\begin{aligned}
\widehat{R}(M) &= \widehat{\mathcal{R}}_m\Big(l\big(F(\mathbb{W}, X, M), Y\big)\Big) \\
&\leq D\widehat{\mathcal{R}}_m\Big(F(\mathbb{W}, X, M), Y\Big) \\
&= \frac{D}{m}E_\sigma\Big[\sup_{W^N}\sum_{j=1}^{m}\sigma_j F(\mathbb{W}, x^j, M^j)\Big] \\
&= \frac{D}{m}E_\sigma\Big[\sup_{W^N}\sum_{j=1}^{m}\sigma_j W^N.\psi_j^N\Big] \\
&\leq \frac{D}{m}B_N E_\sigma\Big[\sup_{W^N}\frac{W^N}{\|W^N\|_1}.\sum_{j=1}^{m}\sigma_j\psi_j^N\Big]
\end{aligned}$$

by considering $\frac{W_k^N}{\|W^N\|_1}$ as β_k in Lemma 3.3.1, we obtain:

$$\widehat{R}(M) \leq \frac{D}{m}B_N E_\sigma\left[\sup_{W_1^{N-1}}\sum_{j=1}^{m}\sigma_j f\Big((M_{j,1}^{N-1}\odot W_1^{N-1}).\psi_j^{N-1}\Big)\right] \tag{3.12}$$

Combining Lemma 3.3.1 and the L-Lipschitz propriety of activation function, we obtain:

$$\begin{aligned}
\widehat{R}(M) &\leq \frac{D}{m}B_N L E_\sigma\left[\sup_{W_1^{N-1}}\sum_{j=1}^{m}\sigma_j(M_{j,1}^{N-1}\odot W_1^{N-1}).\psi_j^{N-1}\right] \\
&= \frac{D}{m}B_N L E_\sigma\left[\sup_{W_1^{N-1}}\sum_{j=1}^{m}\sigma_j W_1^{N-1}.(M_{j,1}^{N-1}\odot\psi_j^{N-1})\right] \\
&\leq \frac{D}{m}B_N B_{N-1} L E_\sigma\left[\sup_{W_1^{N-1}}\sum_{j=1}^{m}\sigma_j\frac{W_1^{N-1}}{\|W_1^{N-1}\|_1}.(M_{j,1}^{N-1}\odot\psi_j^{N-1})\right]
\end{aligned}$$

by considering $\frac{W_{1,k}^{N-1}}{\|W_1^{N-1}\|_1}$ as β_k in lemma 3.3.1, we obtain:

$$\begin{aligned}
\widehat{R}(M) &\leq \frac{D}{m}B_N B_{N-1} L E_\sigma\left[\sup_{W_1^{N-1}}\sum_{j=1}^{m}\sigma_j M_{j,1,1}^{N-1}\psi_j^{N-1}(1)\right] \\
&= \frac{D}{m}B_N B_{N-1} L E_\sigma\left[\sup_{W_1^{N-1}}\sum_{j=1}^{m}\sigma_j M_{j,1,1}^{N-1} f\Big((M_{j,1}^{N-2}\odot W_1^{N-2}).\psi_j^{N-2}\Big)\right] \\
&= \frac{D}{m}B_N B_{N-1} L E_\sigma\left[\sup_{W_1^{N-1}}\sum_{j=1}^{m}\sigma_j f\Big(M_{j,1,1}^{N-1}(M_{j,1}^{N-2}\odot W_1^{N-2}).\psi_j^{N-2}\Big)\right]
\end{aligned}$$

Based on lemma 3.3.1, we obtain:

$$\begin{aligned}
\widehat{R}(M) &\leq \frac{D}{m} B_N B_{N-1} L^2 E_\sigma \left[\sup_{W_1^{N-2}} \sum_{j=1}^{m} \sigma_j M_{j,1,1}^{N-2} (M_{j,1}^{N-2} \odot W_1^{N-2}).\psi_j^{N-2} \right] \\
&= \frac{D}{m} B_N B_{N-1} L^2 E_\sigma \left[\sup_{W_1^{N-2}} \sum_{j=1}^{m} \sigma_j W_1^{N-2}.(M_{j,1,1}^{N-1} (M_{j,1}^{N-2} \odot \psi_j^{N-2})) \right] \\
&\leq \frac{D}{m} B_N B_{N-1} B_{N-2} L^2 E_\sigma \left[\sup_{W_1^{N-2}} \sum_{j=1}^{m} \sigma_j \frac{W_1^{N-2}}{\|W_1^{N-2}\|_1}.(M_{j,1,1}^{N-1} (M_{j,1}^{N-2} \odot \psi_j^{N-2})) \right] \\
&= \frac{D}{m} B_N B_{N-1} B_{N-2} L^2 E_\sigma \left[\sup_{W_1^{N-3}} \sum_{j=1}^{m} \sigma_j M_{j,1,1}^{N-1} M_{j,1,1}^{N-2} f\Big((M_{j,1}^{N-3} \odot W_1^{N-3}).\psi_j^{N-3} \Big) \right] \\
&= \frac{D}{m} B_N B_{N-1} B_{N-2} L^2 E_\sigma \left[\sup_{W_1^{N-3}} \sum_{j=1}^{m} \sigma_j f\Big(M_{j,1,1}^{N-1} M_{j,1,1}^{N-2} (M_{j,1}^{N-3} \odot W_1^{N-3}).\psi_j^{N-3} \Big) \right]
\end{aligned}$$

Recursively, we obtain the following result:

$$\widehat{R}(M) \leq \frac{D}{m} (\prod_{k=q}^{N} B_k) L^{N-q} E_\sigma \left[\sup_{W_1^q} \sum_{j=1}^{m} \sigma_j \Big(\prod_{s=q+1}^{N-1} M_{j,1,1}^{s} \Big) (M_{j,1}^{q} \odot W_1^{q}).\psi_j^{q} \right] \tag{3.13}$$

We suppose that DropConnect is applied only to the $\mathfrak{h}_q$ layer, then we have $M_{j,1,1}^{s} = 1$ for any $s \neq q$ and consequently

$$\prod_{s=q+1}^{N-1} M_{j,1,1}^{s} = 1$$

Indeed,

$$\begin{aligned}
\widehat{R}(M) &\leq \frac{D}{m} (\prod_{k=q}^{N} B_k) L^{N-q} E_\sigma \left[\sup_{W_1^q} \sum_{j=1}^{m} \sigma_j (M_{j,1}^{q} \odot W_1^{q}).\psi_j^{q} \right] \\
&= \frac{D}{m} (\prod_{k=q}^{N} B_k) L^{N-q} E_\sigma \left[\sup_{W_1^q} W_1^{q}. \sum_{j=1}^{m} \sigma_j M_{j,1}^{q} \odot \psi_j^{q} \right]
\end{aligned}$$

Based on Cauchy-Schwarz inequality, we obtain:

$$\widehat{R}(M) \leq \frac{DC_q}{m} (\prod_{k=q}^{N} B_k) L^{N-q} E_\sigma \left[\| \sum_{j=1}^{m} \sigma_j M_{j,1}^{q} \odot \psi_j^{q} \|_2 \right] \tag{3.14}$$

Next, the probabilistic variant of the Jensen inequality, provides un upper bound of the

term $E_\sigma\left[\| \sum_{j=1}^{m} \sigma_j M_{j,1}^q \odot \psi_j^q \|_2\right]$. This is developed in the following manner:

$$\begin{aligned}
E(M) &= E_\sigma\left[\| \sum_{j=1}^{m} \sigma_j M_{j,1}^q \odot \psi_j^q \|_2\right] \\
&= E_\sigma\left[\sum_{j_1=1}^{m}\sum_{j_2=1}^{m} \sigma_{j_1}\sigma_{j_2}\left(M_{j_1,1}^q \odot \psi_{j_1}^q\right).\left(M_{j_2,1}^q \odot \psi_{j_2}^q\right)\right]^{1/2} \\
&\leq \left[\sum_{j_1=1}^{m}\sum_{j_2=1}^{m} E_{\sigma_{j_1}\sigma_{j_2}}\sigma_{j_1}\sigma_{j_2}\left(M_{j_1,1}^q \odot \psi_{j_1}^q\right).\left(M_{j_2,1}^q \odot \psi_{j_2}^q\right)\right]^{1/2} \\
&= \left\| \sum_{j=1}^{m} M_{j,1}^q \odot \psi_j^q \right\|_2
\end{aligned}$$

The last equality is obtained using the property of Rademacher variables that verify $E_{\sigma_{j_1}\sigma_{j_2}}\sigma_{j_1}\sigma_{j_2} = 0$ for $j_1 \neq j_2$ and $E_{\sigma_j\sigma_j}\sigma_j\sigma_j = 1$.
Finally, we get the following inequality:

$$\widehat{R}(M) \leq \frac{DC_q}{m}\left(\prod_{k=q}^{N} B_k\right)L^{N-q}\left\| \sum_{j=1}^{m} M_{j,1}^q \odot \psi_j^q \right\|_2 \tag{3.15}$$

which is therefore employed to estimate the DropConnect Rademacher complexity $R(M)$

$$R(M) \leq \frac{DC_q}{m}\left(\prod_{k=q}^{N} B_k\right)L^{N-q}E_{X,M}\left[\left\| \sum_{j=1}^{m} M_{j,1}^q \odot \psi_j^q \right\|_2\right] \tag{3.16}$$

$$= \frac{DC_q}{m}\left(\prod_{k=q}^{N} B_k\right)L^{N-q}E_{X,M}\left[\sqrt{\left\| \sum_{j=1}^{m} M_{j,1}^q \odot \psi_j^q \right\|_2^2}\right] \tag{3.17}$$

$$\leq \frac{DC_q}{m}\left(\prod_{k=q}^{N} B_k\right)L^{N-q}E_X\left[\sqrt{E_M\left\| \sum_{j=1}^{m} M_{j,1}^q \odot \psi_j^q \right\|_2^2}\right] \tag{3.18}$$

$$= C(M) \tag{3.19}$$

Analogously, we demonstrate that

$$R(M) \leq \frac{DC_q}{m}\left(\prod_{k=q}^{N} B_k\right)L^{N-q}E_X\left[\sqrt{E_M\left\| \sum_{j=1}^{m} M_{j,k}^q \odot \psi_j^q \right\|_2^2}\right] \tag{3.20}$$

$$\tag{3.21}$$

for any $1 \leq k \leq n_{q+1}$

□

3.3.2.2 Estimation of DropConnect hyperparameter

In order to minimize $C(M)$, we use an equivalent expression denoted by

$$\widehat{C}(M) = E_M \left\| \sum_{j=1}^{m} M_{j,1}^q \odot \psi_j^q \right\|_2^2$$

Indeed, the rest of parameters in $C(M)$ are constants. Moreover, $E_X[g(.)]$ and $g(.)$ have the same monotonicity when the distribution of X is fixed. Thus, we have

$$\begin{aligned}
\widehat{C}(M) &= E_M \left\| \sum_{j=1}^{m} M_{j,1}^q \odot \psi_j^q \right\|_2^2 \\
&= \sum_{k=1}^{n_q} \left[E_M \Big(\sum_{j=1}^{m} M_{j,1,k}^q * \psi_j^q(k) \Big)^2 \right] \\
&= \sum_{k=1}^{n_q} \left[Var \Big(\sum_{j=1}^{m} M_{j,1,k}^q * \psi_j^q(k) \Big) + \Big(E_M \Big(\sum_{j=1}^{m} M_{j,1,k}^q * \psi_j^q(k) \Big) \Big)^2 \right] \\
&= \sum_{k=1}^{n_q} \left[\sum_{j=1}^{m} Var \Big(M_{j,1,k}^q \Big) * \Big(\psi_j^q(k) \Big)^2 + \Big(\sum_{j=1}^{m} E_M \Big(M_{j,1,k}^q \Big) * \psi_j^q(k) \Big)^2 \right] \\
&= \sum_{k=1}^{n_q} \left[\sum_{j=1}^{m} \Big(E_M \Big((M_{j,1,k}^q)^2 \Big) - \Big(E_M (M_{j,1,k}^q) \Big)^2 \Big) * \Big(\psi_j^q(k) \Big)^2 + \Big(\sum_{j=1}^{m} E_M \Big(M_{j,1,k}^q \Big) * \psi_j^q(k) \Big)^2 \right] \\
&= \sum_{k=1}^{n_q} \left[\Big(\frac{1}{1-\rho_1^k} - 1 \Big) \sum_{j=1}^{m} \Big(\psi_j^q(k) \Big)^2 + \Big(\sum_{j=1}^{m} \psi_j^q(k) \Big)^2 \right] \\
&= \sum_{k=1}^{n_q} \frac{1}{1-\rho_1^k} \sum_{j=1}^{m} \Big(\psi_j^q(k) \Big)^2 - \sum_{k=1}^{n_q} \sum_{j=1}^{m} \Big(\psi_j^q(k) \Big)^2 + \sum_{k=1}^{n_q} \Big(\sum_{j=1}^{m} \psi_j^q(k) \Big)^2
\end{aligned}$$

Remark :
For the reason of simplifying the expressions, we have multipled each mask M_j^k by $\frac{1}{1-\rho_j^k}$ for the purpose of having

$$E_M(M_{j,1,k}^q) = 1 \quad and \quad E_M \Big((M_{j,1,k}^q)^2 \Big) = \frac{1}{1-\rho_1^k}$$

By assuming that

$$\alpha(k) = \sqrt{\sum_{j=1}^{m} \Big(\psi_j^q(k) \Big)^2}, \qquad \beta(k) = \Big(\sum_{j=1}^{m} \psi_j^q(k) \Big)^2 \tag{3.22}$$

$\widehat{C}(M)$ is then reformulated as

$$\widehat{C}(M) = \sum_{k=1}^{n_q} \frac{\Big(\alpha(k)\Big)^2}{1-\rho_1^k} - \sum_{k=1}^{n_q} \Big(\alpha(k)\Big)^2 + \sum_{k=1}^{n_q} \beta(k)$$

Based on the Cauchy-Schwarz inequality, we obtain

$$\widehat{C}(M) \geq \frac{\Big(\sum_{k=1}^{n_q} \alpha(k)\Big)^2}{n_q - \sum_{k=1}^{n_q} \rho_1^k} - \sum_{k=1}^{n_q} \Big(\alpha(k)\Big)^2 + \sum_{k=1}^{n_q} \beta(k) \tag{3.23}$$

The last inequality is hold if and only if the following condition is satisfied:

$$\frac{\alpha(1)}{1-\rho_1^1} = \frac{\alpha(2)}{1-\rho_1^2} = ... = \frac{\alpha(n_q)}{1-\rho_1^{n_q}} = \frac{\sum_{k=1}^{n_q} \alpha(k)}{n_q - \sum_{k=1}^{n_q} \rho_1^k} \tag{3.24}$$

As a consequence, the min of $\widehat{C}(M)$ is expressed by:

$$\min \widehat{C}(M) = \frac{\Big(\sum_{k=1}^{n_q} \alpha(k)\Big)^2}{n_q - \sum_{k=1}^{n_q} \rho_1^k} - \sum_{k=1}^{n_q} \Big(\alpha(k)\Big)^2 + \sum_{k=1}^{n_q} \beta(k) \tag{3.25}$$

Since $0 \leq \rho_1^k \leq 1$ and using the Cauchy-Schwarz assumption given in Eq. 3.24, the minimum of $\widehat{C}(M)$ can be presented as follow:

$$\min \widehat{C}(M) = \max_k \alpha(k) * \sum_{k=1}^{n_q} \alpha(k) - \sum_{k=1}^{n_q} \Big(\alpha(k)\Big)^2 + \sum_{k=1}^{n_q} \beta(k) \tag{3.26}$$

$\widehat{C}(M)$ reaches minimum when ρ_1^k satisfies

$$\rho_1^k = 1 - \frac{\alpha(k)}{\max_k \alpha(k)} \tag{3.27}$$

The expression Eq.3.27 represents a data dependent estimation of the DropConnect hyperparameter ρ. Indeed, according to its expression, ρ is updated using the data each iteration of the training procedure. To prove the efficiency of this estimation in different DNNs with several datasets, the next section shows the performance of A-DropConnect using several numerical experiments.

3.4 Experiments

In this section, we aim to evaluate and demonstrate the performance of the proposed adaptive DropConnect. For this purpose, we conduct several experiments on image classification task using four datasets including MNIST [**71**], Fashion-MNIST [**63**], NORB [**80**],

and SVHN dataset [**106**]. The experiments are implemented at two levels: In the first one, MNIST and Fashion-MNIST are employed for three proposed DNNs to evaluate the classification using A-DropConnect. These datasets are also used to show the efficiency of DropConnect hyperparameter estimation. At the second level, we aim to demonstrate the effectiveness of the suggested A-DropConnect using more challenging datasets.

3.4.1 Description of datasets

- **MNIST** is a standard dataset often used in the validation of most proposed deep learning techniques [**71**]. It consist of $50,000$ training, $10,000$ test, and $10,000$ validation grayscale images of handwritten digits.
- **Fashion-MNIST** is an extension of MNIST that containing fashion thumbnail images and has all the characteristics of standard MNIST dataset [**63**].
- **SVHN** (Street View House Numbers) is a natural scene number image dataset from house numbers in Google Street View images [**106**]. This is similar to the MNIST dataset except that the SVHN images are RGB color images consisting of 32×32 pixels and many of them contain some distractors at the sides. The SVHN dataset contains $73,257$ images for training, $26,032$ images for testing, and 531,131 additional less difficult samples for extra training.
- **NORB** (NYU Object Recognition Benchmark) widely used for evaluating the performance of a suggested deep learning approach in three-dimensional object recognition [**80**]. It consists of $24,300$ training and $24,300$ test images divided into five categories.

3.4.2 Evaluation metrics

Four different performance measures have been utilized to evaluate the classification performance; accuracy, precision, recall, and F1-score. Accuracy is the proportion of all instances which were classified correctly. It is expressed as follows:

$$Acc = \frac{N_c}{N_d}$$

where N_c and N_d denote, respectively, the number of correctly classified instances and the total number of instances in a dataset.
Precision is the proportion of correctly classified instances among all instances in a class. It is defined as:

$$P_c = \frac{N_c^c}{N_d^c}$$

where N_c^c, N_d^c are the restrictions of N_c and N_d on each class.

The recall is the ratio of correctly classified instances among all instances classified in the corresponding class. Recall value is given by:

$$R_c = \frac{N_c^c}{K_d^c}$$

where K_d^c is the total number of the instances classified as belonging to class c.

F1-Score is a combination of precision and recall measures. F1-Score of a class c is defined as:

$$F_1 - Score = \frac{2R_cP_c}{P_c + R_c}$$

3.4.3 Description of experiments

As mentioned above, two important experiments are implemented. In the first one, we validate and evaluate the performance of our method using MNIST and Fashion-MNIST datasets. To this end, we build three deep neural networks DNN1, DNN2, and DNN3 explicitly described in Table 3.1. We opt for ReLU function to be an activation function

Table 3.1 – DNN architectures used for validation and evaluation of A-DropConnect

DNN1		DNN2		DNN3	
Layers	Size	Layers	Size	Layers	Size
F-C1	(784, 912)	F-C1	(784, 912)	F-C1	(784, 912)
F-C2	(912, 10)	F-C2	(912, 1024)	F-C2	(912, 1024)
		F-C3	(1024, 10)	F-C3	(1024, 2048)
				F-C4	(2048, 10)

for each layer, except the output layer, in which we use the softmax function. We let these models train for 200 epochs using Adam optimizer[**21**] with validation split 0.1, batch size 128, and a learning rate fixed at 0.0001. Since our interest is about a new adaptive regularization method, we train our modesls without dropout (No Drop) and with adding a drop technique such as Dropout [**61**], DropConnect [**111**], MC-DropConnect [**55**], and Rad-Dropout [**104**].
In the second experiment, we aim to demonstrate the efficiency of our method in the classification of other challenging datasets such as NORB [**80**] and SVHN dataset [**106**]. For this purpose, we propose two convolutional neural networks CNN1 and CNN2 explicitly described in Table 3.2.

CNN1 and CNN2 are used respectively to classify NORB and SVHN datasets. These networks are trained using Adam optimizer with validation split 0.1, batch size 128, and learning rate 0.0001. Each CNN is trained without dropout (NO Drop) and with adding drop methods such as Dropout, DropConnect, MC-DropConnect, and Rad-Dropout. In the next subsection, we report, analyze, and discuss the obtained numerical results in each experiment.

3.4.4 Results and discussion

In the first experiment, the efficiency of the adaptive DropConnect was demonstrated through three numerical results. In the first one, the three deep neural networks DNN1,

Table 3.2 – CNN architectures used for evaluation of A-DropConnect applied in the classification of NORB and SVHN datasets

CNN1		CNN2	
Layers	(Input, Output) Size	Layers	(Input, Output) Size
Conv 5×5	$(96 * 96 * 2,\ 92 * 92 * 32)$	Conv 3×3	$(32 * 32 * 3,\ 32 * 32 * 32)$
Max-pooling	$(92 * 92 * 32,\ 46 * 46 * 32)$	Conv 3×3	$(32 * 32 * 3,\ 32 * 32 * 32)$
Conv 3×3	$(46 * 46 * 32,\ 44 * 44 * 64)$	Conv 3×3	$(32 * 32 * 3,\ 32 * 32 * 32)$
Max-pooling	$(44 * 44 * 64,\ 22 * 22 * 64)$	Max-pooling	$(32 * 32 * 3,\ 16 * 16 * 32)$
Fully connected	$(30976,\ 1024)$	Conv 3×3	$(16 * 16 * 32,\ 16 * 16 * 64)$
Fully connected	$(1024,\ 5)$	Conv 3×3	$(16 * 16 * 64,\ 16 * 16 * 64)$
		Conv 3×3	$(16 * 16 * 64,\ 16 * 16 * 64)$
		Max-pooling	$(16 * 16 * 64,\ 8 * 8 * 64)$
		Conv 3×3	$(8 * 8 * 64,\ 8 * 8 * 128)$
		Conv 3×3	$(8 * 8 * 128,\ 8 * 8 * 128)$
		Conv 3×3	$(8 * 8 * 128,\ 8 * 8 * 128)$
		Max-pooling	$(8 * 8 * 128,\ 4 * 4 * 128)$
		Fully connected	$(2048,\ 512)$
		Fully connected	$(512,\ 128)$
		Fully connected	$(128,\ 10)$

DNN2, and DNN3 were used to show the quality of the classification of MNIST and Fashion-MNIST using both standard and adaptive DropConnect (Table 3.3).

As shown, the A-DropConnect provides satisfactory classification results in comparison with the standard DropConnect with a little superiority of DDN2. For instance, the classification of MNIST generates an Accuracy rate of 99.01% using A-DropConnect against 98.79% related to the standard DropConnect. These results are justified by the role of ρ which is a dependent data hyperparameter. In other words, A-DropConnect allows setting ρ adaptively during training without setting it to be fixed ($\rho = 0.5$ as known in the literature). In order to justify the advantage of setting ρ using A-DropConnect, Tables 3.4 and 3.5 offer the mean of classification accuracies of MNIST and Fashion-MNIST applied with the A-DropConnect hyperparameter as well as different static values of ρ changing from 0.2 to 0.7.

As reported in these Tables, the mean of accuracies provided using A-DropConnect is higher than those obtained by the fixed values of ρ. In fact, the auto-adaptation of ρ takes into consideration the characteristics of the data during the training and thus learning the redundancy making, at the end, a system with a better generalization ability.

The last result in this first experiment concerns the comparison of A-DropConnect with other existing Drop techniques including Dropout **[61]**, DropConnect **[111]**, MC-DropConnect **[55]**, and Rad-Dropout **[104]**. For this purpose, the three models DNN1, DNN2, and DNN3 are employed to classify MNIST and Fashion-MNIST using several Drop modes.

As presented in Table 3.6, the mean of accuracies generated by A-DropConnect is supe-

Table 3.3 – Classification evaluation of MNIST and Fasion-MNIST using different DNNs regularized with DropConnect and A-DropConnect.

Proposed models	Evaluation metrics	MNIST		Fashion-MNIST	
		DropConnect	Proposed	DropConnect	Proposed
DNN1	Accuracy	0.9854	0.9893	0.8978	0.9011
	F1 score	0.9854	0.9892	0.8981	0.9026
	Precision	0.9855	0.9893	0.8985	0.9140
	Recall	0.9853	0.9891	0.8976	0.8918
DNN2	Accuracy	0.9879	0.9901	0.8981	0.9080
	F1 score	0.9880	0.9901	0.8980	0.9179
	Precision	0.9880	0.9900	0.9147	0.9269
	Recall	0.9880	0.9902	0.8823	0.9094
DNN3	Accuracy	0.9861	0.9894	0.8980	0.9090
	F1 score	0.9861	0.9894	0.8988	0.9096
	Precision	0.9861	0.9896	0.9107	0.9132
	Recall	0.9861	0.9893	0.8875	0.9061

Table 3.4 – Mean of accuracies for MNIST classification using DropConnect with fixed and adaptive hyperparameter

Models	DropConnect with fixed hyperparametre ρ						A-DropConnect
	$\rho = 0.2$	$\rho = 0.3$	$\rho = 0.4$	$\rho = 0.5$	$\rho = 0.6$	$\rho = 0.7$	
DNN1	0.9848	0.9854	0.9847	0.9854	0.9847	0.9843	0.9893
DNN2	0.9855	0.9868	0.9868	0.9879	0.9861	0.9858	0.9901
DNN3	0.9848	0.9869	0.9865	0.9861	0.9868	0.9868	0.9894

Table 3.5 – Mean of accuracies for Fashion-MNIST classification using DropConnect with fixed and adaptive hyperparameter

Models	DropConnect with fixed hyperparametre ρ						A-DropConnect
	$\rho = 0.2$	$\rho = 0.3$	$\rho = 0.4$	$\rho = 0.5$	$\rho = 0.6$	$\rho = 0.7$	
DNN1	0.8982	0.8952	0.8949	0.8978	0.8938	0.8963	0.9011
DNN2	0.8920	0.8937	0.8941	0.8981	0.8987	0.8964	0.9080
DNN3	0.8924	0.8974	0.8931	0.8980	0.8945	0.8898	0.9090

Table 3.6 – Comparison of accuracies mean for MNIST and Fashion-MNIST classification using different drop methods

Methods	MNIST			Fashion-MNIST		
	DNN1	DNN2	DNN3	DNN1	DNN2	DNN3
No Drop	0.9832	0.9847	0.9850	0.8961	0.8974	0.8964
Dropout [**61**]	0.9849	0.9860	0.9853	0.8974	0.8980	0.8982
DropConnect [**111**]	0.9854	0.9879	0.9861	0.8978	0.8981	0.8980
MC-DropConnect [**55**]	0.9890	0.9887	0.9875	0.8999	0.8998	0.9012
Rad-Dropout [**104**]	0.9862	0.9885	0.9877	0.8990	0.9002	0.8999
A-DropConnect	0.9893	0.9901	0.9894	0.9011	0.9080	0.9090

rior to those given by the other Drop modes. In fact, the DropConnect, which generalizes the Dropout, has proved its efficiency in comparison with Dropout and No Drop since it allows to select a different mask for each training example unlike Dropout which uses only one mask for each mini-batch of examples [**111**]. Moreover, Table 3.6 reports that A-DropConnect outperforms DropConnect, MC-DropConnect, and Rad-Dropout. For instance, A-DropConnect has led to an accuracy of 90.90% for Fashion-MNIST using A-DropConnect applied to DNN3 against 90.12% and 89.99% using MC-DropConnect and Rad-Dropout respectively. This result is justified by the data-dependent form of ρ which produces a dynamic sparse form of the network based on the gap generalization, which finally offers a considerable prediction ability.

Another advantage to using A-DropConnect is the fast convergence to the best accuracy in comparison to the other Drop techniques as shown in Fig. 3.1. This benefit is due to the specific setting of the hyperparameter ρ that regularizes the DNN by neglecting the data noise and finally avoiding the overfitting.

In the second experiment, we aim to prove the performance of A-DropConnect using more challenging datasets. In this context, we use two convolutional neural networks denoted by $CNN1$ and $CNN2$ explicitly described in Table 3.2.

Tables 3.7and 3.8 report the evaluation metrics of the classification of NORB and SVHN. As shown, both of the results are very interesting. Indeed, the accuracies provided by A-DropConnect are very sgnificant and satisfactory in comparison with the other Drop techniques without neglecting MC-DropConnect, which is a very concurrent Drop method and yields a result close to that obtained via A-DropConnect. For example, our method provided, for NORB and SVHN respectively, accuracies of 89.17% and 93.51% for A-DropConnect againt 89% and 93.47% for MC-DropConnect. These results can only confirm the effiency of A-DropConnect and the intrinsically role of the data-dependet expression of the estimated hyperparameter DropConnect.

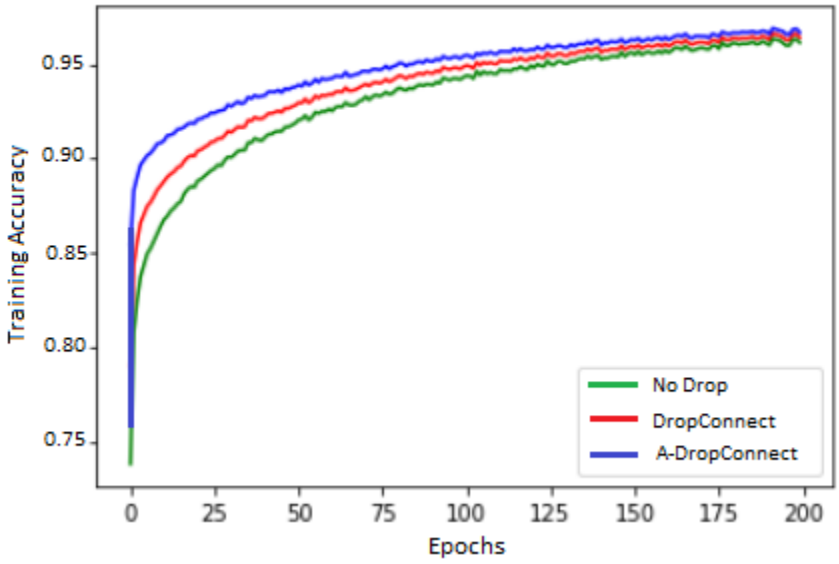

Figure 3.1 – Accuracies variations durind DNN1 training using DropConnect, A-DropConnect, and without Drop

Table 3.7 – Classification performance evaluation using different regularization methods in the classification of NORB dataset

Methods	Accuracy	F1 score	Precision	Recall
No Drop	0.8103	0.8103	0.8104	0.8102
Dropout [**61**]	0.8432	0.8454	0.8453	0.8455
DropConnect [**111**]	0.8393	0.8341	0.8340	0.8342
MC-DropConnect [**55**]	0.8900	0.8980	0.8147	0.8823
Rad-Dropout [**104**]	0.8877	0.8881	0.8885	0.8876
A-DropConnect	0.8917	0.8922	0.8918	0.8924

3.5 Conclusion

Motivating by the efficiency of DropConnect in preventing overfitting in deep neural networks, and due to the lack of the theoretical study that offers a suitable hyperparameter of this successful regularizer. A new DropConnect method named adaptive DropConnect (A-DropConnect) is proposed and analyzed. The suggested technique allows, through Rademacher complexity, to estimate the Bernouilli hyperparameter by minimizing the generalization gap. The performance of A-DropConnect has been evaluated and demonstrated using different public datasets and several deep neural networks. Results have reported the important role of the data-dependent expression of the DropConnect

Table 3.8 – Classification performance evaluation using different regularization methods in the classification of SVHN dataset

Methods	Accuracy	F1 score	Precision	Recall
No Drop	0.8940	0.8938	0.8937	0.8939
Dropout [**61**]	0.9036	0.9015	0.9007	0.9023
DropConnect [**111**]	0.9055	0.9102	0.9097	0.9105
MC-DropConnect [**55**]	0.9347	0.9334	0.9328	0.9340
Rad-Dropout [**104**]	0.9312	0.9314	0.9311	0.9317
A-DropConnect	0.9351	0.9362	0.9341	0.9383

hyperparameter in both data classification and DNNs compatibility. In the future, we would aim to give more theoretical analysis about DropConnect regularizer using other approaches and also to justify the effectiveness of our proposal with other challenging real-world applications.

Chapter 4

Optimization and localization of redandancy in CNNs using a mixed non linear programming model

4.1 abstract

Nowadays, convolutional neural networks (CNNs) have achieved tremendous performance in many machine learning areas. However, using a large number of parameters leads to the redundancy problem, which negatively impacts the performance of CNNs. Indeed, many kernels are redundant and can be taken off from the network without much loss of performance. In this chapter, we propose a new optimization model for localizing and removing the redundancy in CNN. In fact, Unlike numerous existing methods where they only reduce the redundancy, our proposal also allows to localize the distribution of redundancy in CNNs. The suggested model consists of two stages: in the first one, a dataset is used to train a specific CNN generating a learned CNN with optimal parameters. These later are combined with a decision L_1-sparsity optimization model for detecting and reducing the unwanted kernels. At the end, the evolutionary genetic algorithm is adapted to solve the proposed model generating finally an optimal CNN with prior information about the redundancy distribution. The performance of our approach has been shown and demonstrated by several experiments.

Keywords: Convolutional neural networks. convolution kernel. binary optimization. genetic algorithm. image classification. L_1-sparsity.

4.2 Introduction

Deep convolutional neural networks (CNNs) have become the most influential learning models that has shown great results and considerable success in many applications such as object detection for medical images [**40**], time series forecasting [**5**], microscopy image analysis [**24**], building energy consumption forecast [**3**], automatic ship classification [**32**], etc. The reason behind this high performance is due to its special characteristics such as parameters sharing, localized sparse connectivity, and the features extraction using the convolution operator [**71**].
The convolutional neural network (CNN), as a typical deep learning model, is an efficient end-to-end hierarchical learning system. In fact, from an input, several convolutional blocks (usually composed of a convolutional layer, a pooling layer and an activation function) are used to extract the features, which in turn are presented as inputs for a fully connected classifier that can end up with a SoftMax output layer. Encouraged by its advantages, many CNNs have appear in the literature including ResNet [**53**], NetinNet [**90**] and VGGNet [**132**].
However, in order to reach a good performance of prediction and achieve more challenging goals, the CNN architecture becomes more complicated and contains a large number of parameters. As a result, training a deep CNN becomes more difficult and brings three fundamental problems; Firstly, the training and the inference are cumbersome and take a long time [**107**]. Secondly, the training leads to overfitting problem [**74**], and finally, the redundancy problem that slows the network performance. In fact, as mentioned in [**86**, **101**], some weights learned in the networks are correlated, which means that some of the weights can either be predicted or be unnecessary to learn.
Regarding redundancy, various approaches have been designed to reduce redundancy in deep neural network weights. In [**118**], the authors have exploited the Low-rank approximation for reducing redundancy in network weights. Denton et al. reduce the computation cost for redundancy within kernels using the low-rank approximation in the first few convolutional layers of CNN [**119**]. Tai et al. used tensor decomposition to reduce redundancy in the convolution kernels [**43**]. Tucker decomposition was introduced to decompose pre-trained weights in order to remove redundancy [**2**]. Ma et al. proposed a non-convex integrated transformed l_1 regularizer for eliminating redundant connections and unnecessary neurons simultaneously in deep neural networks [**129**]. Reducing redundancy in a specific deep CNN model named SoundNet was discussed by Singh et al. Indeed, they have used the singular value decomposition-based framework to reduce redundant feature maps [**121**]. In [**112**], authors suggested a pruning method based on a novel criterion to measure the redundancy of the CNN parameters through the classification loss. Recently, the authors in [**11**] proposed an approach based on kernel's weights decorrelation to regularize the CNNs for an efficient generalized model. More recently, the authors in [**83**] introduced the $l1/l2$ pseudo-norm on the kernel coefficients to remove the irrelevant filters in CNNs.
Unfortunately, most of these architectures are either manually defined by experts or automatically designed by greedy induction algorithms. In this context, multiscale and multilevel evolutionary optimization and the Gaussian process were combined to optimize some CNN hyperparameters including those related to the architecture [**96**]. In [**98**], the

number of layers in the networks was approximated using particle swarm optimization. Passricha and Aggarwal attempted to automatically optimize CNN architecture's hyper-parameters using the particle swarm optimization for speech recognition [**102**]. In [**44**], Louati et al. proposed a modelling of CNN architecture design as a bi-level optimization problem and suggested solving the obtained model using an adapted version of the generic bi-level EA CODBA.
Most of the previous related works, have handled the redundancy problem without offering its distribution over the CNNs. In other words, no information, about the unnecessary kernels positions, is provided. Moreover, the details of the redundancy condensation is completely unknown. Motivated by this issue, we propose in this chapter a new vision of the redundancy localization and reduction in convolutional neural networks using an adaptive optimization model. Our suggested model consists in associating a binary variable with each convolution kernel in order to localized the unnecessary kernels and controlling the redundancy reduction using the L_1 sparsity induction. Indeed, the combination of a binary variable and its L_1 norm leads to a L_0 semi norm, which is known by its ability to control the models sparsity [**108**]. The obtained model belongs to the mixed integer non linear programming problem (MINLP) with the CNN parameters as continuous variables and the binary optimization variables. Resolution of the proposed model is performed using minimization and optimization tasks. In the first one, a specific learning optimizer, such as stochastic gradient descent, is used for the parameters set. In the second, we opt for the evolutionary genetic algorithm [**20**]. Finally, we perform experiments to verify that our model can achieve better performance than regular CNNs in terms of classification and optimization.
The remainder of this chapter is organized as follows: In the next Section, we describe the modeling steps of the proposed model,as well as the resolution by genetic algorithm. Section 4.4 details the experiments and shows the effectiveness of our model through several numerical and graphical results. Section 4.5 provides a brief conclusion and directions for future work.

4.3 Proposed optimization model for CNN redundancy reduction

In order to detect specifically the redundant kernels in CNNs, we propose a new optimization model which is able ,using the fundamentals of the Mixed Integer Nonlinear Programming (MINLP), to control and localize the redundancy distribution in CNNs. In this context, a set of decision variables are technically introduced in the training loss function and reinforced by a specific constraint. We suggest finally the GA solver to deal with the obtained model. In the next paragraphs, we describe step by step the redundancy optimization modelling, the localization of redundant kernels in CNNs and the resolution by Genetic algorithm.

4.3.1 Modelling the redundancy optimization problem

Firstly, we present some necessary notations in our approach in Table 4.1

Table 4.1 – Notations

Sets and parameters	
X	the network input
Y	the network output
$\mathbb{W}$	set of network parameters.
W	tensor of parameters in the classification part
K	tensor of kernels
C_l	tensor of $l-th$ convolutional layer outputs.
N_{conv}	number of convolutional layers in the network.
N_{FC}	number of fully connected layers.
N_l	number of feature maps in the $l-th$ convolutional layer.
n_l	number of neurones in the fully connected layer number l.
$D \times D$	size of convoltional kernels.
$conv-layer_I$	set of convolutional layers in the network.
$conv-layer_l$	the $l-th$ convolutional layer.
U	decision variables
$\alpha_\sigma(I)$	ordered set of the redundancy proportion.

4.3.1.1 Objective function

The idea behind our proposed approach is transforming the current CNN with its current loss-learning function to an improved one with a mixed variables learning loss function. For this purpose, in first step we define the following binary variable:

$$u_{ij}^k = \begin{cases} 0 & \text{if the convolutional kernel } K_{ij}^k \text{ does not contribute} \\ 1 & \text{else} \end{cases} \tag{4.1}$$

Its main role is controlling the contribution of kernels in the learning task using a combinatorial approach. This set of variables is introduced to the process of convolution as follows:

$$C_l^k = f\left(\sum_{j=1}^{N_{k-1}} (u_{j,l}^k . K_{j,l}^k) * X_j^{k-1} \right) \tag{4.2}$$

An illustrative example of the functioning of U in reducing the kernels redundancy in CNN is shown in (Fig. 4.1). In which, for the set of kernels that relays one output of a previous layer and a whole current layer, the obtained solution U indicates that only three kernels are not redundant.

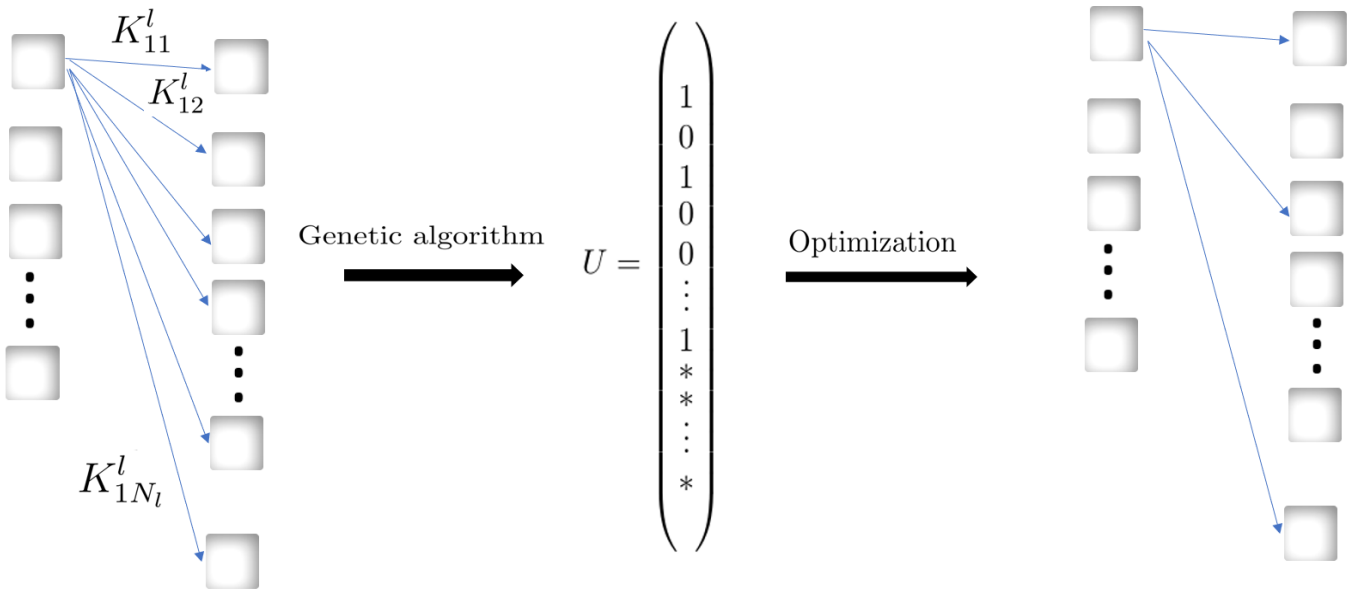

Figure 4.1 – An illustrative example of the use of U in reducing the redundancy in CNN.

Taking into consideration the variables U, The kernels redundancy reducing problem can be formalised as:

$$\underset{\mathbb{W},U}{\operatorname{argmin}} \{E(W,U) = l\,(\mathbb{W},U) + \lambda\|U\|_1\} \tag{4.3}$$

where $l\,(\mathbb{W},U)$ is the accuracy-related loss function (e.g., the cross-entropy loss for classification tasks), λ is a scalar hyperparameter which controls the balance between the loss term and the regularization term and $\|U\|_1$ is the l^1 norm of U, which represents the number of kernels effectively contribute to the network training.
The above optimization problem can be formalised into a constrained optimization form as:

$$\underset{\mathbb{W},U}{\operatorname{argmin}} \{l\,(\mathbb{W},U) \quad \text{subject to : } \|U\|_1 \leq R\}$$

where R is the sparsity level (i.e., the number of non-zero kernels).

4.3.1.2 Constraints

The use of U for the convolutional layers requires a special consideration. In fact, to avoid the whole elimination of an output feature maps in the convolutional layers, it is necessary to keep at least one kernel in each convolution operation. Thus, we suggest the following constraint:

$$\sum_{i=1}^{N_{l-1}} u_{i,j}^l \geq 1 \tag{4.4}$$

for $j = 1, ..., N_l$ and $l = 1, ..., N_{conv}$.

Regarding the learning problem variables $\mathbb{W}$, they can be considered as a tensor given by $\mathbb{W} = K \cup W$.
where K represents the set of convolutionnal kernels (i.e. the parameters in feature extraction part):

$$K = \left((K_{i,j}^{l})_{p,q} \right)_{\substack{l=1,...,N_{conv} \\ i=1,...,N_k \\ j=1,...,N_{k-1} \\ p,q=1,...,D}}$$

where $(K_{i,j}^{k})_{p,q} \in \mathbb{R}$
and W is the set of weights related to the classification part:

$$W = \left(W_{i,j}^{k} \right)_{\substack{k=1,...,N_{FC} \\ i=1,...,n_k \\ j=1,...,n_{k-1}}}$$

such that $W_{i,j}^{k} \in \mathbb{R}$

4.3.1.3 Obtained optimization model

The obtained mathematical model applied to CNN is a mixed integer non linear programming problem (MINLP) presented as follows:

$$(P) \begin{cases} \underset{\mathbb{W},U}{Min} \quad E(\mathbb{W}, U) = l(\mathbb{W}, U) + \lambda \|U\|_1 \\ Subject\ to: \\ \quad \sum_{i=1}^{N_{l-1}} u_{i,j}^{l} \geq 1 \quad for\ \ j = 1, ..., N_l \quad and\ l = 1, ..., N_{conv}. \\ \quad \mathbb{W} = K \cup W \\ \quad K = \left((K_{i,j}^{l})_{p,q} \right)_{\substack{l=1,...,N_{conv} \\ i=1,...,N_{l-1} \\ j=1,...,N_l \\ p,q=1,...,D}} \quad where\ (K_{i,j}^{l})_{p,q} \in \mathbb{R} \\ \quad W = \left(W_{i,j}^{l} \right)_{\substack{l=1,...,N_{FC} \\ i=1,...,n_{l-1} \\ j=1,...,n_l}} \quad where\ W_{i,j}^{l} \in \mathbb{R} \\ \quad U = \left(u_{i,j}^{l} \right)_{\substack{l=1,...,N_{conv} \\ i=1,...,N_{l-1} \\ j=1,...,N_l}} \quad where\ u_{i,j}^{l} \in \{0, 1\} \end{cases}$$

The resolution task is divided into two phases: minimization and optimization. In the first one, we use a learning approach to train the CNNs using a specific optimizer, such as the stochastic gradient descent or Adam optimizer [**19**]. In the second phase, we use the solution $\mathbb{W}^*$ obtained from the first task to generate a new optimization problem

$\min_{U} E(\mathbb{W}^*, U)$ that will be solved using the evolutionary genetic algorithm (Fig.4.2). This is the purpose of the following section .

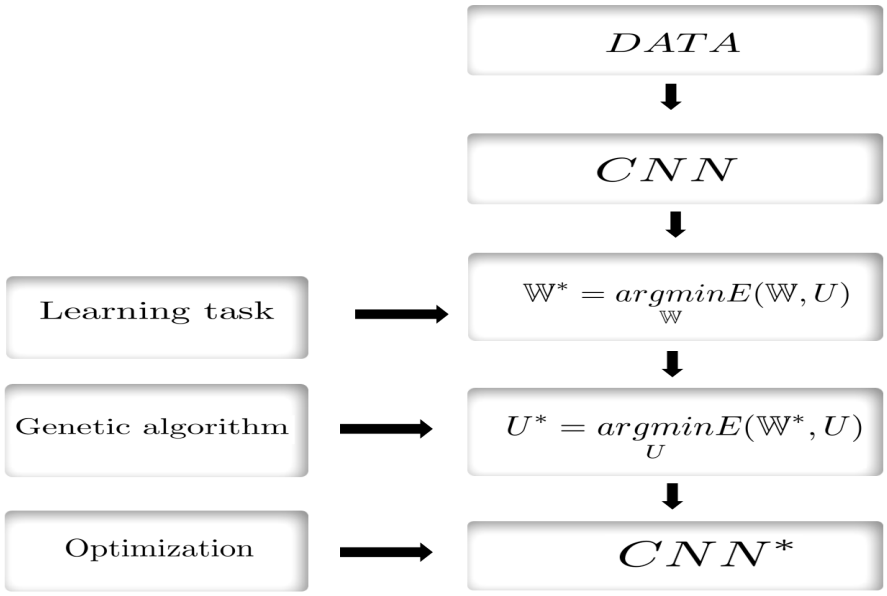

Figure 4.2 – Flow chart of the proposed model

4.3.2 Modelling the localization of redundancy in CNNs

Let consider the set $I = \{1, 2, .., N_{conv}\}$ that denotes the indexes of convolutional layers in a CNN model, the ones vector $e^i = \{1, 1, .., 1\}$ that indicates the ones vector of the same size of U_i wich is $N_{i-1} \times N_i$. We define the redundancy rate in each convolutional layer of CNN as follows:

$$R_i = \frac{||e^i - U_i^*||_1}{||e^i||_1} \qquad \text{for } i = 1, 2, ..., N_{conv} \tag{4.5}$$

Then, the redundancy proportion of each convolutional layer is defined as:

$$\alpha_i = \frac{R_i}{\sum_{j=1}^{N_{conv}} R_j} \qquad \text{for } i = 1, 2, ..., N_{conv} \tag{4.6}$$

Lets consider the one to one application defined as follows

$$\begin{array}{rl} \sigma: & \{1, 2, .., N_{conv}\} \longrightarrow \{1, 2, .., N_{conv}\} \\ & i \longrightarrow \sigma(i) \end{array}$$

verifying $\alpha_{\sigma(i)} \geq \alpha_{\sigma(j)}$ for $i < j$.

We obtain then, the set

$$\alpha_{\sigma(I)} = \{\alpha_{\sigma(1)}, \alpha_{\sigma(2)}, ..., \alpha_{\sigma(N_{conv})}\} \tag{4.7}$$

which represents an ordred set of the redundancy proportions.
Thus, the condensation of some given proportion η of redundancy can be formulated as the following problem:

$$s = \underset{m \in I}{Min}\{m \ / \ \sum_{j=1}^{m} \alpha_{\sigma(j)} > \eta\} \tag{4.8}$$

which means that η-proportion of redundancy is distributed over the first $s\ conv - layer_{\sigma}$.

4.3.3 Adaptation of GA for solving the obtained model

Genetic algorithm is a stochastic search method inspired by the development process of biological systems based on Darwin's evolution theory [**20**]. It is one of the most extensively used algorithms in combinatorial problems with respect to other heuristics [**88**]. GA operates as a population-based adaptive search process that makes it one of the best solutions approaches among other metaheuristics when large-scale problems are considered. Based on a set of random solutions named individuals, two parents are chosen through the selection to reproduce a new generation. The reproduction task consists of crossover, and mutation, the first one increases the diversity of the population and the second enriches the intensification. The use of genetic algorithm for solving the proposed model is described below.

4.3.3.1 Representation and initialisation

The population consists of N chromosomes generated by a binary row except the first row that contains only ones. Indeed, this individual allows preserving all the CNN components that will contribute to the solution search.
Fitness:
To evaluate the quality of individuals, a numerical value called fitness is assigned to each individual. It depends essentially on the value of the objective function and the constraints corresponding to this individual. An individual with a maximal fitness is the most adapted to the problem. In our work, we suggest the following fitness function:

$$fit(U) = \frac{1}{1 + E_p(\mathbb{W}^*, U)} \tag{4.9}$$

where $E_p(\mathbb{W}^*, U)$ is the penalty function associated with the problem (P). Consequently, an individual that minimizes $E(\mathbb{W}^*, U)$, maximizes the Fitness.
Selection:
During the algorithm, we select the best individuals having the greatest fitness to be reproduced. This selection is conducted by roulette-wheel selection [**70**]. This type of selection assigns to each individual a spot in a wheel. The spot size depends on the fitness of the individual. Precisely, the size of the spot increases with the fitness of the individual. Then a ball is thrown in the wheel. More fit is the individual more the probability that

the ball throws in its spot is great. The probability is defined by $P_i = \frac{fit(U^i)}{\sum_{i=1}^{N} fit(U^i)}$ and the selection point is chosen randomly from the interval $[0, 1]$.

4.3.3.2 Reproduction: Mutation and Crossover

Reproduction guarantees diversification and intensification of the search space thanks to its two operators: crossover and mutation. In the first operator, we have applied the one-point crossover which consists in cutting at the same point (chosen randomly) two parents and then, swapping the right portions between themselves thus constituting two new strings. While the crossover can only explore the combinations of the current parents, the mutation operation creates new individuals by changing the value of a predefined number of an arbitrary location in the chromosome **[69]**. In our work, we use the bit string mutation which takes a random genome and invert the bit (i.e. if the genome bit is 1, it is changed to 0 and vice versa).

At the end of the resolution process, the obtained solution $(\mathbb{W}^*, U^*)$ forms an optimal CNN^* with a reduced kernels number (Fig.4.2).
The main steps of the proposed approach is presented in the fallowing pseudo-code:

Localization-Optimization of redundancy in CNNs

- **Input:**

  -CNN, Dataset

- **Output:**

  - Optimal parameters $\mathbb{W}^*$ and U^* and the ordered redundancy distribution $\alpha_{\sigma(I)}$

— **Define:**

  — The decision variables U using Eq. 4.1

  — The energy function $E(\mathbb{W}, U)$ using Eq. 4.3

— **Step 1: CNN Training**

  — $\mathbb{W}^* = \underset{\mathbb{W}}{Argmin} \quad E(\mathbb{W}, U)$.

— **Step 2 : CNN Redundancy optimization**

  — $U^* = \underset{U}{Argmin} \quad E(\mathbb{W}^*, U)$ using GA.

  — Calculate $\alpha_{\sigma(I)}$ using Eq. 4.5, 4.6, 4.7.

— **Return**

  — $CNN^* = (\mathbb{W}^*, U^*)$

  — $\alpha_{\sigma(I)}$

4.4 Experiments

In this section, we aim to demonstrate the effectiveness of the proposed model. In this context, we perform three experiments described as follows: in the first one, we begin by a numerical study of the hyperparamter λ. In fact, we evaluate the optimization-classification throughout three synthetic models with different values of λ. The main advantage behind this experiment is extracting a prior estimation of λ that allows us to turn on the next experiments. The purpose of the second experiment is proving the performance of the proposed approach in term of the classification and the optimization, this task is realized with four evaluations metrics described in the the subsection 4.4.2. Finally, the variation of the local redundancy according to the convolution layers is utilized to visualize the redundancy condensation over the CNN models.

4.4.1 Description of datasets

In the first experiment, we opt for MNIST dataset to evaluate our model.
MNIST. Since the proposed model addresses the CNN architectures issue, it is natural to begin with one the naivest datasets which is the MNIST database [**71**]. This set of handwritten digits has a training set of 60,000 examples, and a test set of 10,000 examples.
In the second and the third experiments, we use the CIFAR dataset described as follows:
CIFAR. The CIFAR-10 dataset consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. CIFAR-10 consists of 10 categories of images, while CIFAR-100 consists of 100 categories of images. There are 50000 training images and 10000 test images [**81**].

4.4.2 Evaluation metrics

Four different performance measures have been utilized to evaluate the classification performance; accuracy, precision, recall, and F1-score. Accuracy is the proportion of all instances which were classified correctly. It is expressed as follows:

$$Acc = \frac{N_c}{N_d}$$

where N_c and N_d denote, respectively, the number of correctly classified instances and the total number of instances in a dataset.
Precision is the proportion of correctly classified instances among all instances in a class. It is defined as:

$$P_c = \frac{N_c^c}{N_d^c}$$

where N_c^c, N_d^c are the restrictions of N_c and N_d on each class.

The recall is the ratio of correctly classified instances among all instances classified in the corresponding class. Recall value is given by:

$$R_c = \frac{N_c^c}{K_d^c}$$

where K_d^c is the total number of the instances classified as belonging to class c.

F1-Score is a combination of precision and recall measure. F1-Score of a class c is defined as:

$$F_1 - Score = \frac{2R_cP_c}{P_c + R_c}$$

Regarding the optimization performance, we use the reduction rate R_r defined as:

$$R_r = \frac{\text{\#of Optimized Kernels}}{\text{\#of Kernels in CNN}}$$

4.4.3 Numerical study of the hyperparameter λ

In order to demonstrate how the optimization and the classification performances vary according to λ values. Three CNNs with two convolutional layers have been proposed to classify the MNIST dataset. The suggested models are successively 16×32 (Fig. 4.3), 32×64 (Fig. 4.4) and 64×128 (Fig. 4.5). These models have been applied using a kernel size 3×3, pooling with 2×2. Dropout is utilized with the probability 0.5 and the learning algorithm is Adam optimizer.

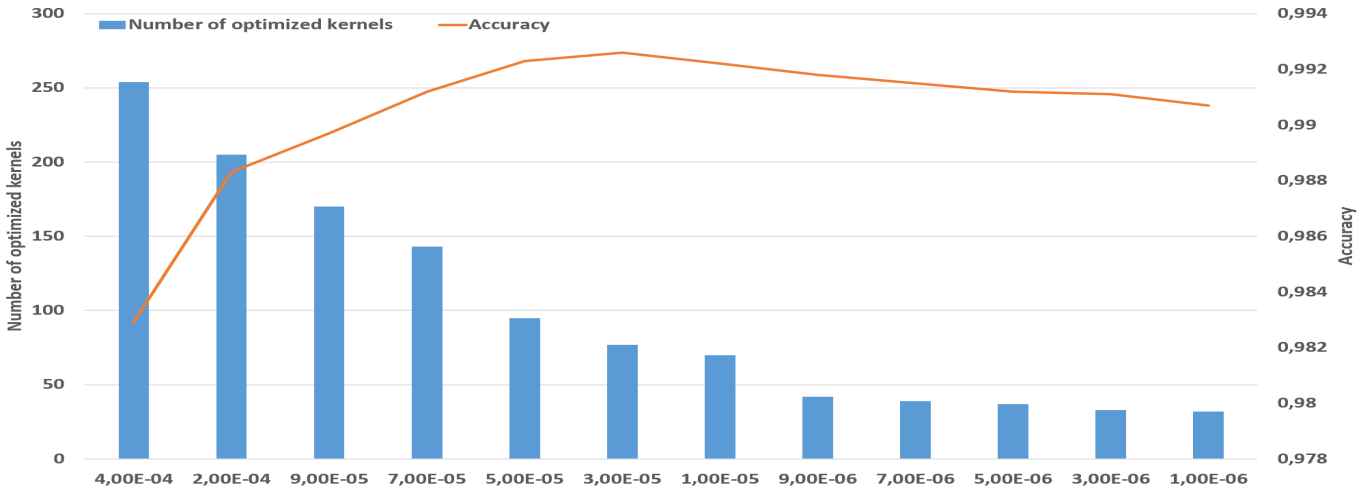

Figure 4.3 – Variation of optimized number of kernels and accuracy according to λ: model CNN:16×32

As shown form these Figures, the number of the optimized kernels is decreasing with the decrease of λ. Indeed, this behavior is justified as follows: when the hyperparameter λ is too small, the L_1 sparsity induction has a low impact, which means in other word that the model possesses a small proportion of zeros. On the contrary, the obtained accuracy gradually increases when λ decreases, and this until $\lambda = 3.10^{-5}$ for $CNN : 64 \times 128$ and $CNN : 16 \times 32$ and up to $\lambda = 2.10^{-5}$ for $CNN : 32 \times 64$ from which the accuracy becomes more stable or has a weak variations. Regarding the traedoff between the kernels optimization and the classification, a specific choice must be chosen in order to guarantee a considerable optimization number of kernels and a satisfactory accuracy. For instance,

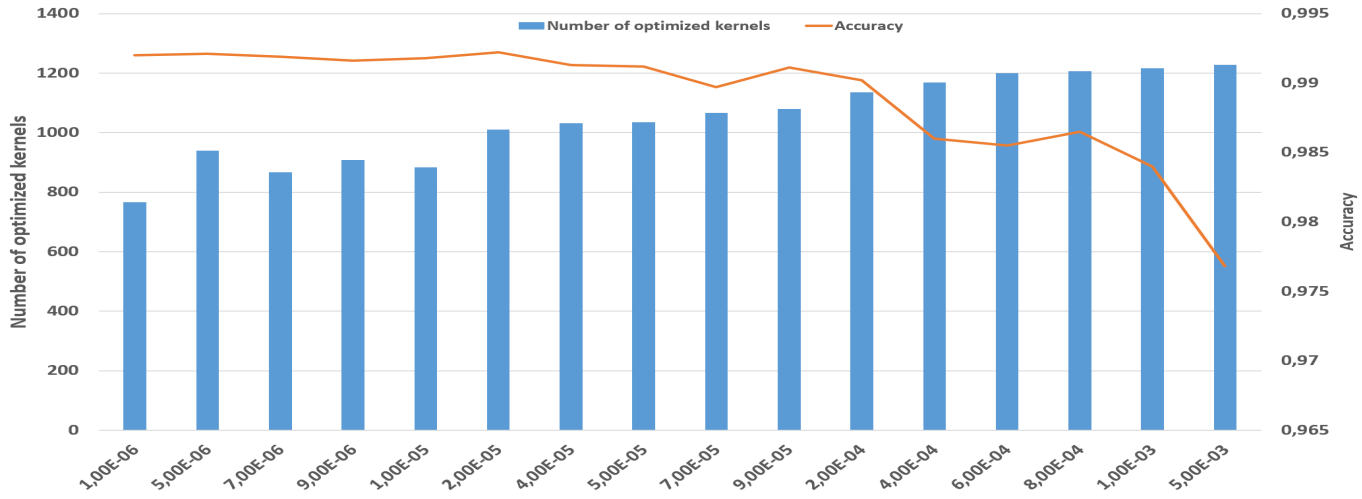

Figure 4.4 – Variation of optimized number of kernels and accuracy according to λ: model CNN:32×64

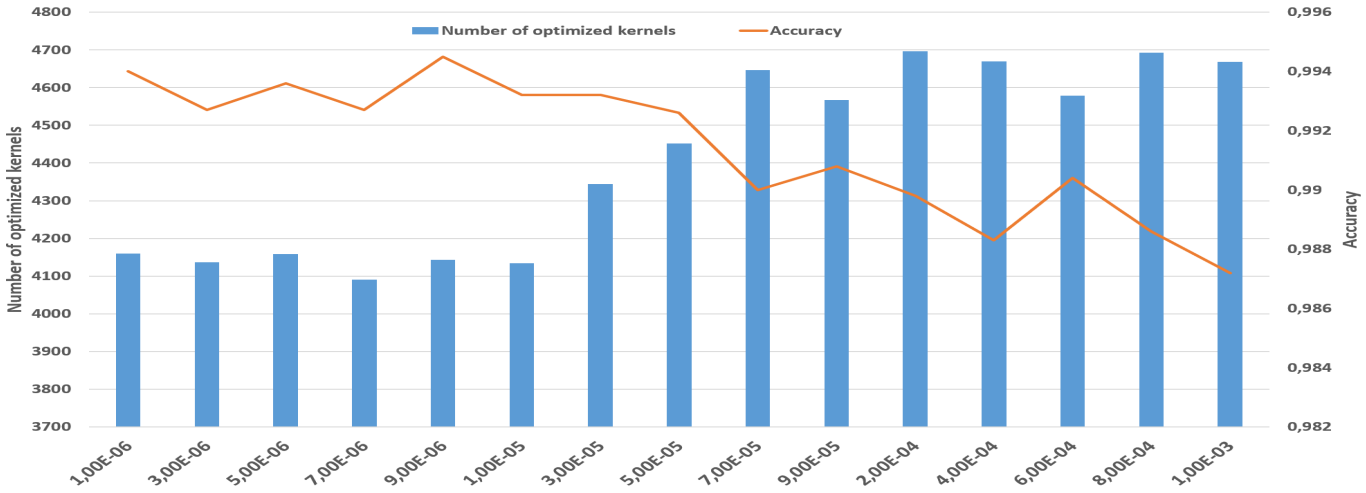

Figure 4.5 – Variation of optimized number of kernels and accuracy according to λ: model CNN:64×128

we may choose $\lambda \in [3.10^{-5}, 5.10^{-5}]$ for $CNN : 16 \times 32$ and $\lambda \in [2.10^{-5}, 5.10^{-5}]$ for $CNN : 32 \times 64$.

4.4.4 Performance of proposed approach

In the second experiments, we evaluated the classification-optimization of CIFAR dataset using ResNet [**53**], NetinNet [**90**] and VGGNet [**132**]. In this context, CNN was trained using stochastic gradient descent optimizer (SGD) with momentum 0.9, batches of size 128 and the number of epochs was 200. In Net in net, the learning rate was initialized to 0.01 in the first 80 epochs, 0.005 from the epoch 81 to 140 and 0.001 in the last 60 epochs. In ResNet, the learning rate started with 0.1 in the first 80 epochs, 0.01 in the epoch 81

until 121 and 0.001 in the last 79 epochs. In VGG19, the learning rate was 0.1 in the first 80 epochs and 0.01 from the epoch 81 to the epoch 160 and 0.001 in the last 40 epochs. This experiment is also performed with $\lambda = 10^{-5}$ and the real-time data augmentation for the purpose of reducing the overfitting.
Regarding the genetic algorithm, we have used the elitist version of GA [**8**]. It consists in eliminating the regression in the performance from one generation to the next by introducing directly one or many of the best individuals of the parent generation into the offspring generation. The GA is turned on with 100 individuals under 100 iterations. The crossover is performed with the probability 0.5 while the mutation is used with a low probability that equals 0.1.

Table 4.2 – Evaluation metrics obtained using ResNet and the optimized ResNet* in the classification of Cifar10 and Cifar100

Evaluation metrics	Cifar10		Cifar100	
	ResNet	ResNet*	ResNet	ResNet*
Accuracy	0.9226	0.9365	0.6784	0.6876
F1 score	0.9231	0.9368	0.6880	0.6994
Precision	0.9257	0.9390	0.7204	0.7325
Recall	0.9206	0.9348	0.6593	0.6702
$R_r(\%)$		12.68		11.37

Table 4.3 – Evaluation metrics obtained using VGG19 and the optimized VGG19* in the classification of Cifar10 and Cifar100

Evaluation metrics	Cifar10		Cifar100	
	VGG19	VGG19*	VGG19	VGG19*
Accuracy	0.9333	0.9419	0.7207	0.7315
F1 score	0.9333	0.9429	0.7284	0.7322
Precision	0.9341	0.9450	0.7475	0.7342
Recall	0.9326	0.9409	0.7109	0.7303
$R_r(\%)$		3.14		2.25

In Tables 4.2,4.3 and 4.4, the classification-optimization of Cifar 10 and Cifar 100 has been evaluated. Indeed, the optimized CNN* associated with Net in Net, ResNet and VGG19 models have led to significant evaluation metrics in comparison with CNNs. For instance, the model VGG19 has generated the larger metrics in the case of Cifar 10 with a considerable number of the optimized kernels ($R_r = 3.14\%$). These results mean that the

Table 4.4 – Evaluation metrics obtained using Net-in-Net and the optimized Net-in-Net* in the classification of Cifar10 and Cifar100

Evaluation metrics	Cifar10		Cifar100	
	Net-in-Net	Net-in-Net*	Net-in-Net	Net-in-Net*
Accuracy	0.8787	0.8823	0.6137	0.6175
F1 score	0.8791	0.8848	0.6261	0.6356
Precision	0.8882	0.9031	0.6714	0.7265
Recall	0.8705	0.8677	0.5877	0.5673
$R_r(\%)$		6.61		5.78

suggested optimization model can improve the performance of the convolutional neural networks regardless its architecture. In other word, the suggested method can be considered as a plug and play (PnP) that can be adapted to the class of convolutional neural networks.

4.4.5 Localization of redundancy in CNNs

In the last experiment, we aim to show the distribution of the global redundancy over the convolutional neural networks. In this context and based on the previous experiment, the redundancy proportion α in these models is represented according to $conv\_layer_\sigma$.
In Figure 4.6, the local redundancy in ResNet is presented. As shown, the values of α decrease gradually with small differences well observed in the first four $conv\_layers_\sigma$. But in general, we can assume that the redundancy is relatively weak and almost distributed with the same manner overs the network.

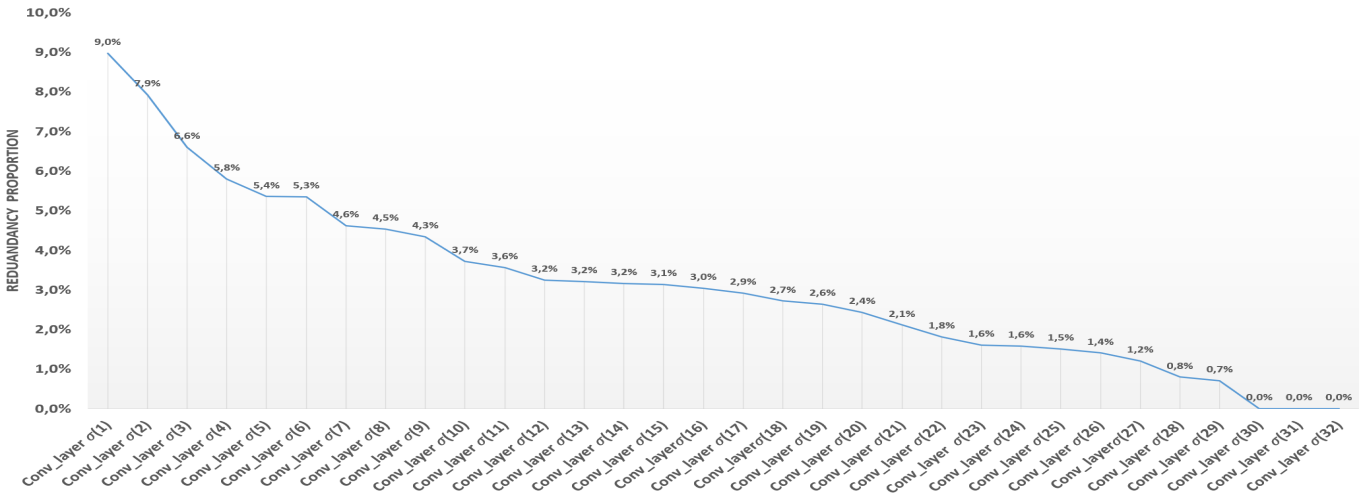

Figure 4.6 – Local redundancy distribution within ResNet

In the case of VGG19, Figure 4.7 shows that 100% of the redundancy is condensed uniformly in the first five $conv\_layer_\sigma$. Therefore, the reduction of the redundancy is performed at the level of these five $conv\_layer_\sigma$.

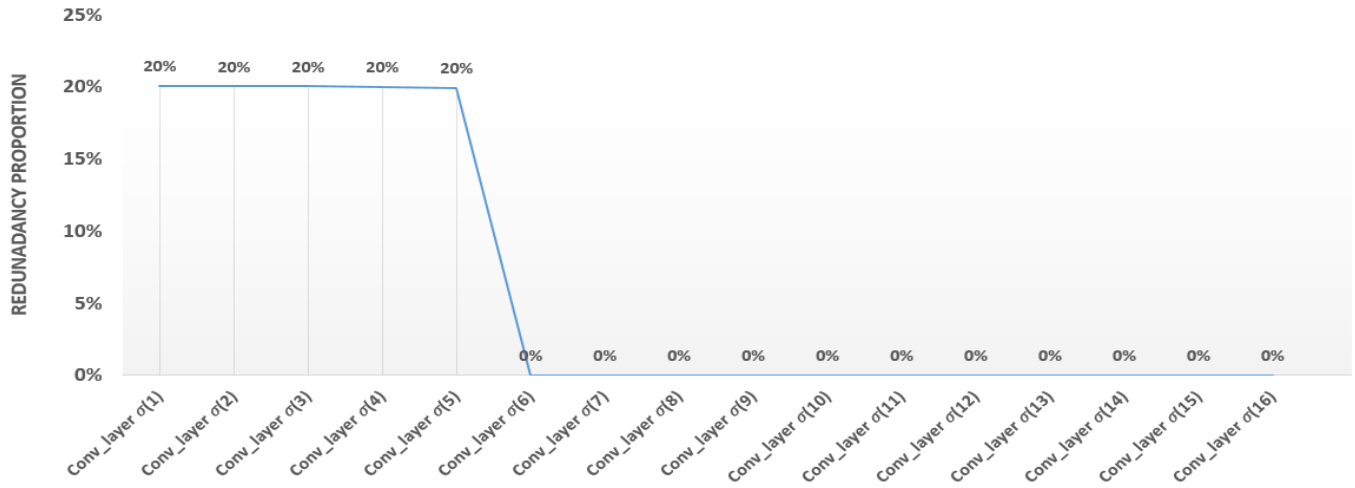

Figure 4.7 – Local redundancy distribution within VGG19

Regarding the model Net in Net, Figure 4.8 reports that 56% of the redundancy is situated on the first $conv\_layer_\sigma$ while 44% is uniformly divided on the next four $conv\_layer_\sigma$.

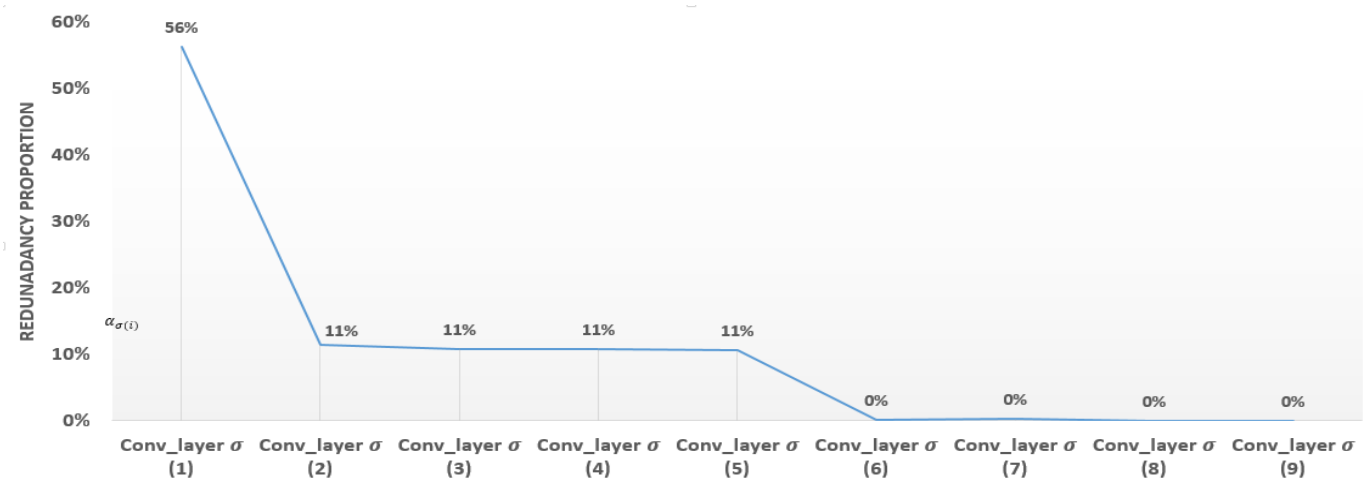

Figure 4.8 – Local redundancy distribution within Net in Net.

4.4.6 Comparison with other methods

To further justify the superiority of our approach on the existing methods, we have conduct several comparisons with some sparsity related works like l_1 and transformed l_1 (Tl_1) regularization **[129]** in terms of classification performance. As shown from Table 4.5,

Table 4.5 – Performance in term of accuracy of different methods on the datasets.

datasets	Basline	$l1$	$Tl1$	Our proposal
MNIST	0.9861±0.0003	0.9889±0.0007	0.9850±0.0002	0.9890±0.0005
Fashion-MNIST	0.8862±0.0004	0.8947 ± 0.0009	0.8873±0.0026	0.8874±0.0007
CIFAR-10	0.7513±0.0002	0.7716±0.0052	0.7804±0.0025	0.7823±0.0003
CIFAR-100	0.6893 ± 0.0019	0.7006 ± 0.0004	0.7048 ± 0.0031	0.7052 ± 0.0005

the obtained results are almost similar with a little superiority of our approach. Moreover, it should be noted that the most redundancy reducing methods belong to the regularization techniques that generate sparsity in models which is different from our approach in which we identify and localize the redundant kernels making the usage of CNN more efficient and less correlated.

4.5 Conclusion

In this chapter, the L_1-sparsity optimization was introduced for localizing and reducing the redundancy in convolutional networks. In the proposed model, the learning loss function has been enriched by a set of decision variables and an additive L_1-sparsity term. The main idea is controlling the contribution of the kernels in the network training using a combinatorial optimization approach. In order to solve the modeled problem, the genetic algorithm was technically adapted to the optimization part of the model. Our approach has led to a significant redundancy detection. Indeed, the introduced L_1-sparsity optimization allowed to determine the unwanted kernels, as well as their distribution over the CNNs. Experiments show that our method can improve the performance in terms of the optimization and the classification using different public datasets and several CNN models including ResNet, Net-in-Net, and VGG19. In the future, we will extend the optimization to the number of convolutional layers.

Chapter 5

Optimization and regularization of deep neural networks for time series classification

5.1 Abstract

Time Series Classification (TSC) started getting a lot of attention recently, mostly due to the important real-world applications of time series such as in the financial industry. In this context, various approaches have been proposed to treat this important and challenging problem in data mining. These approaches include deep learning models that are outperformed the traditional classification techniques. However, the choice of an optimal and efficient deep neural network for TSC is still a major problem. Motivated by this challenge, we propose in this chapter, an optimized deep learning framework for TSC tasks, using normal cloud representation and convolutional neural networks optimization (NCR-CNNO). Our approach consists of two phases: In the first one, we convert the raw time series to a matrix of characteristics using two-dimensional normal cloud representation. In the second phase, we suggest a convolutional neural network to handle the matrix generated in the first phase, as well as, an optimization model is proposed to optimize the unnecessary and redundant parameters. Experiments conducted on extensive time series datasets demonstrate that NCR-CNNO yields significant improvement in the performance of time series classification compared to the state of the art.

Keywords: Deep learning. Convolutional neural networks. binary optimization. Time series classification. cloud representation

5.2 Introduction

A time series is a collection of numerical values sorted in chronological order. Time series have gained great attention because of their relevant applications in various fields of real life such as financial industry [**64**], ecological environment [**28**] and renewable energy [**115**].
Time series classification (TSC) is one of the important Time series data mining tasks, which has attracted great attention during the two past decades. Various approaches have been developed for TSC. These approaches can be divided into two categories: the classical approaches and the deep learning based approaches.
The classical approaches including Bag of SFA Symbols (BOSS) [**122**] which is a competitive structure-based similarity measure based on applying noise optimization to the raw time-series. Bagnall et al. [**128**] proposed an approach, based on transformation and ensembling, named Collective of Transformation Ensembles (COTE). In [**1**], the authors developed an approach based on a bag-of-features representation for TSC. The main idea behind this latter is the selection of multiple subsequences randomly and partitioned they into shorter intervals to extract the local relevant features. Furthermore, by combining a distance measure and the entropy gain, Deng et al. introduced a tree-ensemble method for TSC named time series forest (TSF) [**17**]. Recently, the authors in [**125**] proposed a competitive method for TSC based on ensemble learning and multi-feature dictionary representation. Most of mentioned methods achieved promising results than others on the UCR datasets. However, due to the complex characteristics of time series and the fact that is usually non-stationary and non-linear, the classical methods cannot achieve accurate time series classification results.
Recently, deep learning models, especially deep neural networks have a great success in solving engineering problems including computer vision, natural language processing, speech recognition, and many others [**52**, **51**]. They also show their high capability for solving time series data mining problems. Therefore, they may offer clues to enhance the performance of time series data mining methods. Obviously, for improving and developing TSC approaches [**47**].
In recent years, as a result of the extraordinary development of more machine learning fields, deep neural networks becoming more and more important in the classification of time series. In this context, various based deep learning approaches have been developed for TSC. Among these approaches, the deep time-growing neural network (DTGNN) proposed by Gharehbaghi et al. [**6**]. This methods allows learning the cyclic contents of stochastic time-series by combining the unsupervised and supervised methods using a multi-scale learning structure. In [**38**], the authors developed a deep learning model for the classification of time series generated by continuous and discrete dynamical systems. Khan et al. introduced a deep learning framework for TSC, based on hybridization of Bidirectional LSTM and recurrent neural networks [**36**]. Deep learning approach has been suggested also to classify the multivariate time series [**62**].
Convolutional neural network is one of the most successful deep learning models which has made significant improvements in numerous machine learning challenging problems. Motivated by these improvements, various CNN based approaches for TSC have been proposed. These approaches include that proposed by Hatami et al. [**39**] in which the au-

thors proposed combining Recurrence Plots (RP) and CNN for the classification of time series. RP was used to transform a raw time series to a 2D image and than the obtained image is treated and classified using a CNN. In [**7**], the authors proposed a new 2D representation method named relative position matrix to convert a raw time series to a 2D image and use the same TSC procedure as in [**39**]. Cui et al. proposed a CNN model for TSC called Multi-Scale CNN (MCNN) [**89**]. This model incorporates feature extraction and classification in a single framework. CNNs have been also applied to classify the cyclo-stationary time-series [**77**]. This latter achieved with two steps: the first allows increasing size and diversity of time series datasets, the second consists to retrieve knowledge from the obtained model.
The CNN-based approaches for TSC achieved competitive results compared with the other existing methods. However, the use of an arbitrary representation method may lead to a loss of information which in turn affects the classification procedure. Moreover, the use of CNN in this context still with no a priory preparation due to the lack of the theoretical study related to the CNNs architectures [**78**]. Consequently, this may provide some undesirables phenomenon such as the overfitting which prevents the CNN generalization ability [**61**]. Motivated by these challenges, we propose in this chapter, a new approach for TSC using Normal Cloud Representation and Convolutional Neural Network Optimization (NCR-CNNO). The suggested method consists of two tasks (Representation and classification). In the representation phase, the normal cloud representation is proposed to transform the raw time series to a two dimensional representation generating at the end a set of relevant characteristics presented to the CNN. In the classification part, a CNN is suggested and reinforced by an optimization study related to its architecture, notably its parameters redundancy.
The main contributions of this paper are summarized as follows:

- Using Piecewise two-dimensional normal cloud representation to represent a time series with its relevant characteristics.
- Using CNN to extract more features and to classify the time series.
- Eliminate unnecessary and redundant parameters in CNN using an optimization model.

The remainder of this chapter is organized as follows: in the next Section, we briefly describe the necessary background for TSC in two parts: one to introduce time series representation background and the second for introducing the essential elements of deep convolutional neural networks. In Section 5.4, the proposed optimized deep learning framework for TSC is explicated. Section 5.5 details the experiments and presents the numerical results. Section 5.6 provides a brief conclusion and directions for future work.

5.3 Preliminaries

To clarify the proposed approach and makes the paper more compact, we propose, in this section, some preliminaries related to normal cloud representation and convolutional neural networks.

5.3.1 Time series representation background

During the last two decades, time series representation and classification have been considered as two of the most challenging problems in data mining. Firstly, we provide the following definitions

Definition 7: A

ime series X is a sequence of real values $X = (x_1, x_2, ..., x_n)$; where n is the length of X.

Definition 8: L

t X be a time series of length n. A time series representation of X is defined as a model $\bar{X}$ of reduced dimension m $(m \ll n)$, where $\bar{X}$ contain the relevant features of X or closely approximates X.

Various time series representation approaches have been proposed in the literature [**47**]. These approaches include the cloud representation as one of the most efficient and challenging representation models. This latter was introduced by Li [**40**] as a new cognition model of uncertainty which is based on fuzzy set and probability theory. One of the most important cloud models is the normal cloud model, in which phenomenons satisfy the normal distribution [**12**]. Normal cloud representation is defined as follows

Definition 9: L

t U be the universe of discourse and $\tilde{A}$ be a qualitative concept in U. If $x \in U$ is a random realization of $\widetilde{A}$ that satisfies $x \sim N\left(Ex, En'^2\right)$, where $En' \sim N\left(En, He^2\right)$, and the certainty degree y of x belonging to $\widetilde{A}$ satisfies

$$y = e^{-\frac{(x-Ex)^2}{2\left(En'\right)^2}}$$

then the distribution of x in universe U is called a normal cloud model and x is a cloud droplet. A cloud consists of many droplets that can reflect the important characteristics of the qualitative concepts, which can be understood as the quantitative descriptions of qualitative concepts.

5.3.2 Convolutional neural networks

Unlike the classical machine learning models, CNN has the particularity of the features extraction part that is responsible to extract relevant characteristics within the data. this part makes CNN very powerful by integrating the data pretreatment in the network training. Generally, the features extraction is achieved using a sequence of two layer types: convolutional and pooling. The features extracted in the first part are then classified using a multiplayer perceptron. this latter represents the second part in CNN. In this section, we describe each CNN component as well as, the functioning mechanisms.

5.3.2.1 Convolution layer

The convolution is one of the principal mechanisms in a CNN. It is responsible to extract relevant features within input feature maps. This extraction is accomplished using the following expression:

$$C_l^k = f\left(\sum_{j=1}^{N_{k-1}} K_{j,l}^k * X_j^{k-1}\right) \tag{5.1}$$

where f represents the used activation function in the convolutional layer and $*$ refers to the discrete convolution product.
Among competitive activation function widely used to imports non linearity to deep neural networks, we can highlight The ReLU function which its definition is given as follow:

$$f(x) = ReLU(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases}$$

5.3.2.2 Pooling layer

The pooling (subsampling) mechanism applies to the feature maps generated from the previous convolution layer. It is another relevant component of CNN based on dimension reduction. In fact, It allows transforming inputs to a reduced dimension feature maps by the use of a pooling operation. These operations include max-pooling [**117**] and average pooling [**72**]. We next give the definition of these two pooling methods.

- Max Pooling [**117**] is a successful pooling method, which allows reducing feature maps dimension by taking the largest activation in each pooling region as expressed in Eq.5.2.

$$s_k = \max_{a_i \in R_k} a_i \quad for \quad k = 1, ..., K \tag{5.2}$$

  This method is a very powerful subsampling technique achieved better performance especially in extracting local characteristics.

- Average Pooling [**72**] is another reducing dimension method that produces the average of activations in each pooling region. This is accomplished using the following

expression (Eq.5.3).

$$s_k = \frac{1}{\#R_k} \sum_{a_i \in R_k} a_i \quad for \quad k = 1, ..., K \tag{5.3}$$

The use of average operation allows to smooth subsampling regions, as a result, this method is useful for extracting global characteristics.

where $\{a_1, a_2, .., a_{\#R_k}\}$ denotes the kth subsampling region.

5.3.2.3 Batch normalization layer

Data normalization is a successful process widely used during the learning. This process allows data to be comparable with respect to its characteristics. However, during training, changing of network parameters generate a change in the network activations distribution. For this reason, various approaches have been developed in order to improving the convergence during training. Among these approaches, we highlight batch normalization which is a successful method that greatly solved the mentioned issue. It consists of normalizing on mean and in variance the layers outputs of the network; Mathematically, lets consider $B = \{X_1, X_2, \ldots, X_d\}$ a mini batch of size d, and $(y_1, y_2, \ \ldots, y_d)$ their linear transformations correspondents. Batch normalization $BN_{\gamma,\beta}$ is presented by the following expression:

$$BN_{\gamma,\beta}: \quad \{X_1, X_2 \ldots, X_d\} \longrightarrow \{Y_1, Y_2 \ldots, Y_d\}$$

$$X_k \longrightarrow Y_k = \gamma \cdot \hat{X}_k + \beta$$

where $\hat{X}_k = \frac{X_k - \mu_B}{\sqrt{\sigma_B^2 + \varepsilon}}$ is the normalized value of X_k and

$$(\mu_B, \sigma_B^2) = (\frac{1}{d} \sum_{k=1}^{d} X_k, \frac{1}{d} \sum_{k=1}^{d} (X_k - \mu_B)^2)$$

present the (mean, variance) of X.

5.3.2.4 Classification layers

Once we achieve the last layer in the features extraction, all the feature maps are converted to a one dimension (1D) array which, in turn, is presented as an input to a classifier composed of fully-connected layers where the outputs are given as follows:

$$Z^k = \begin{pmatrix} Z_1^k \\ Z_2^k \\ \vdots \\ Z_{n_{FC_k}}^k \end{pmatrix} = \begin{pmatrix} f\Big(\sum_{i=1}^{n_{k-1}} W_{1i}^k \times Z_i^{k-1}\Big) \\ f\Big(\sum_{i=1}^{n_{k-1}} W_{2i}^k \times Z_i^{k-1}\Big) \\ \vdots \\ f\Big(\sum_{i=1}^{n_{k-1}} W_{n_{FC_k}i}^k \times Z_i^{k-1}\Big) \end{pmatrix}$$

The Softmax function is then used to calculate the network output,

$$Y = softmax(Z_{FC}^N)$$

where N_{FC} is the number of hidden layers.

5.4 Proposed framework

As mentioned previously, the suggested method consists of representation and optimization-classification. These tasks are summarized in Figure. 5.1.

5.4.1 Representation phase

Encouraged by the effectiveness of cloud representation, we suggest using a two dimensional representation based on cloud model named two dimension normal cloud representation **[99]**, to transform a raw time series to a 2D representation. This representation can be divided into four fundamental steps as illustrated in Fig. 5.2.
Let consider a raw time series $X = (x_1, x_2, ..., x_n)$ and m is the reduced dimension. Based on **[99]**, the four steps of the suggested representation method are described as follow:

- Step 1: Calculate the first order difference ΔX

  $$\Delta X = (d_1, d_2, ..., d_n)$$

  where

  $$d_j = \begin{cases} 0 \quad \text{if} \quad j = 1 \\ x_j - x_{j-1} \quad \text{otherwise} \end{cases}$$

- Step 2: Segmentation of X and ΔX into m segments simultaneously.

$$\begin{cases} (X_1, \Delta X_1) = \big((x_1, d_1), (x_2, d_2), ..., (x_{w_1}, d_{w_1})\big) \\ (X_2, \Delta X_2) = \big((x_{w_1+1}, d_{w_1+1}), ..., (x_{w_2}, d_{w_2})\big) \\ \quad \vdots \\ (X_m, \Delta X_m) = \big((x_{w_{m-1}+1}, d_{w_{m-1}+1}), ..., (x_n, d_n)\big) \end{cases}$$

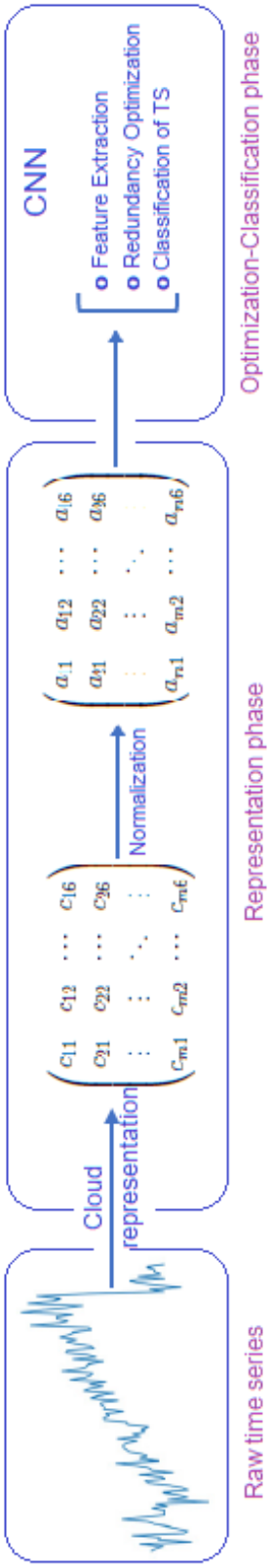

Figure 5.1 – Proposed framework for time series classification

- Step 3: Perform 2D normal cloud representation:
  Transform each sub-sequence pair $(X_j, \Delta X_j)$ into a 2D normal cloud C_j using T-BNCT algorithm described below (Algorithm 1).
  Then the matrix of characteristics C is obtained by considering each 2D normal cloud C_j as the jth row in the matrix C.
- Step 4: Normalization
  The final step of the representation phase allows converting the characteristics matrix C to the gray value matrix. For this, we apply the min-max normalization to obtain the final matrix A as expressed follow:

$$A = \frac{C - \min(C)}{\max(C) - \min(C)} \times 255$$

The T-BNCT algorithm used in the third step is summarized in the following pseudocode as presented in [**100**]:

Algorithm 1: T-BNCT

- **Input:**
  Samples (y_j, z_j) $(j = 1, 2, \ldots, m)$.
- **Output:**
  Numerical features $(Ey, Ez, Emy, Enm, Hey, Hez)$.

Step 1: Compute the average and variance for each sample dimension using the following formulas:

$$\bar{Y} = \frac{1}{m}\sum_{j=1}^{m} y_j, \quad Sy^2 = \frac{1}{m-1}\sum_{j=1}^{m}\left(y_j - \bar{Y}\right)$$

$$\bar{Z} = \frac{1}{m}\sum_{j=1}^{m} z_j, \quad Sz^2 = \frac{1}{m-1}\sum_{j=1}^{m}\left(z_j - \bar{Z}\right)$$

Step 2: Compute expectation Ey and Ez using the expressions:

$$Ey = \bar{Y}, Ez = \bar{Z}$$

Step 3: Compute entropy Emy and Emz via the equations:

$$Emy = \sqrt{\frac{\pi}{2}} \times \frac{1}{m}\sum_{j=1}^{m}|y_j - Ey|, \quad Emz = \sqrt{\frac{\pi}{2}} \times \frac{1}{m}\sum_{j=1}^{m}|z_j - Ez|$$

Step 4: Compute hyper-entropy Hey and Hez:

$$Hey = \sqrt{Sy^2 - Emy^2}, \quad Hez = \sqrt{Sz^2 - Emz^2}$$

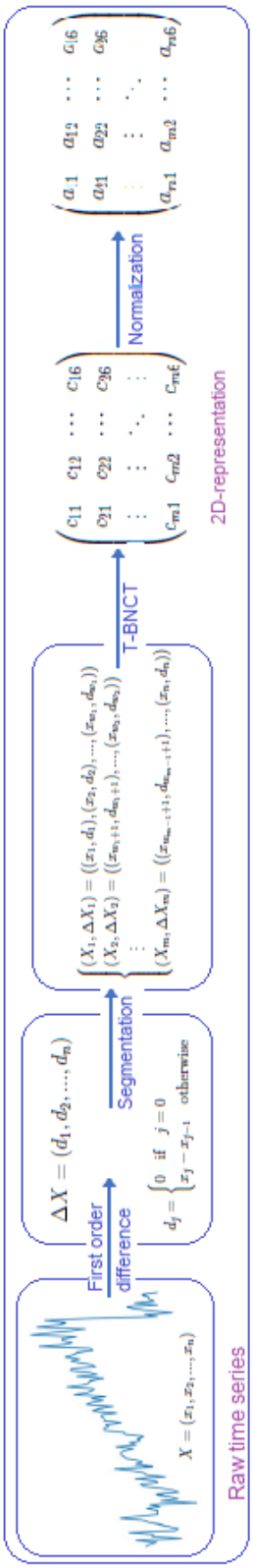

Figure 5.2 – Time series representation phase

5.4.2 Classification-Optimization phase

In this phase, we begin first by describing the optimization model of the redundancy reduction in CNN, and next we adapt it to TSC using the normal cloud representation.

5.4.2.1 CNN redundancy optimization

The use of CNN for classification task is generally applied using default settings, i.e without out a study of the redundancy in these models. Moreover, the distribution of redundancy in CNNs is not clearly described [**78**]. In this paper, the use of CNNs for TSC is reinforced by a combinatorial optimization approach. The suggested model presents a new vision of the redundancy localisation and reduction by defining a set of decision variables. In the features extraction part of CNN, we associate to each kernel K_{ij}^k a binary variable u_{ij}^k defined as follows:

$$u_{ij}^k = \begin{cases} 0 & \text{if the kernel } K_{ij}^k \text{ is redundant} \\ 1 & \text{else} \end{cases}$$

In the classification part, we associate to each connection W_{ij}^k a binary variable v_{ij}^k defined as follows:

$$v_{ij}^k = \begin{cases} 0 & \text{if the weight } W_{ij}^k \text{ does not contribute} \\ 1 & \text{else} \end{cases}$$

Thus, integrating these binary variables allows controlling the influence of each parameter in the network. Since, the process of propagation in the convolutional layers becomes:

$$C_l^k = f\left(\sum_{j=1}^{N_{k-1}} (u_{j,l}^k . K_{j,l}^k) * X_j^{k-1} \right) \tag{5.4}$$

Regarding the classification part, the process of propagation becomes

$$Z_j^k = f\Big(\sum_{i=1}^{n_{k-1}} (v_{ji}^k \times W_{ji}^k) \times Z_i^{k-1} \Big) \tag{5.5}$$

Taking into consideration the variables U and V, The parameters redundancy reducing problem can be formalised as:

$$\underset{\mathbb{W},U,V}{\operatorname{argmin}} \{ l(\mathbb{W}, U, V) \}$$

where $l(\mathbb{W}, U, V)$ is the accuracy-related loss function (e.g., the cross-entropy loss for classification tasks).

Constraints:

Integrating binary variables U and V in the network necessitates a specific consideration. Indeed, in the convolution process, generating an output feature maps necessitates at least one convolutional kernel. Mathematically, this can be expressed as follows

$$\sum_{i=1}^{N_{h-1}} u_{i,j}^h \geq 1$$

for $j = 1, ..., N_l$ and $h = 1, ..., N_{conv}$.
Moreover, to prevent the total elimination of connections between neurons in the classification layers, it is necessary to keep at least one connection for each neuron. Thus, this constraint can be formulated as follow:

$$\sum_{i=1}^{n_{h-1}} v_{i,j}^{h} \geq 1$$

for $j = 1, ..., n_h$ and $h = 1, ..., N_{FC}$.

The CNN parameters $\mathbb{W}$ represent an union of two tensors K and W. Such that K presents the CNN variables within the features extraction part.

$$K = \left((K_{i,j}^{l})_{p,q} \right)_{\substack{l=1,...,N_{conv} \\ i=1,...,N_k \\ j=1,...,N_{k-1} \\ p,q=1,...,D}}$$

Each element of this tensor is a real variable $(K_{i,j}^{k})_{p,q} \in \mathbb{R}$

The second parameters tensor W composed of CNN variables within fully connected layers:

$$W = \left(W_{i,j}^{k} \right)_{\substack{k=1,...,N_{FC} \\ i=1,...,n_k \\ j=1,...,n_{k-1}}}$$

such that $W_{i,j}^{k} \in \mathbb{R}$
Obtained optimization model:
The resulting optimization model is a non linear problem with constraints and mixed

variables. It is expressed as:

$$
(P)\begin{cases}
\underset{\mathbb{W},U,V}{Min} \quad E(\mathbb{W},U,V) = l\left(\mathbb{W},U,V\right) \\
Subject\ to: \\
\quad \sum_{i=1}^{N_{h-1}} u_{i,j}^{h} \geq 1 \quad for\ \ j = 1,...,N_h \quad \text{and} \ \ h = 1,...,N_{conv}. \\
\quad \sum_{i=1}^{n_{h-1}} v_{i,j}^{h} \geq 1 \quad for\ \ j = 1,...,n_h \quad \text{and} \ \ h = 1,...,N_{FC}. \\
\quad \mathbb{W} = K \cup W \\
\quad K = \left((K_{i,j}^{h})_{p,q} \right)_{\substack{h=1,...,N_{conv} \\ i=1,...,N_{h-1} \\ j=1,...,N_h \\ p,q=1,...,D}} \quad \text{where} \ \ (K_{i,j}^{h})_{p,q} \in \mathbb{R} \\
\quad W = \left(W_{i,j}^{h} \right)_{\substack{h=1,...,N_{FC} \\ i=1,...,n_{h-1} \\ j=1,...,n_h}} \quad \text{where} \ \ W_{i,j}^{h} \in \mathbb{R} \\
\quad U = \left(u_{i,j}^{h} \right)_{\substack{h=1,...,N_{conv} \\ i=1,...,N_{h-1} \\ j=1,...,N_h}} \quad \text{where} \ \ u_{i,j}^{h} \in \{0,1\} \\
\quad V = \left(v_{i,j}^{h} \right)_{\substack{h=1,...,N_{FC} \\ i=1,...,n_{h-1} \\ j=1,...,n_h}} \quad \text{where} \ \ v_{i,j}^{h} \in \{0,1\}
\end{cases}
$$

In fact, Our suggested model consists in associating a binary variable with each parameters (kernels in convolutional layers and weights in fully connected layers) in order to localized the unnecessary parameters and controlling the redundancy reduction. The obtained model belongs to the mixed integer non linear programming problem (MINLP) with the CNN parameters as continuous variables and the binary optimization variables. Resolution of the proposed model is performed using minimization and optimization tasks. In the first one, a specific learning optimizer, such as stochastic gradient descent, is used for the parameters set. In the second, we opt for the evolutionary genetic algorithm **[20]**.

Adaptation of GA for solving the obtained model

The genetic algorithm is a stochastic search method inspired by the process of developing a biological system based on Darwin's theory of evolution. **[20]**. This is one of the most common algorithms for combinatorial problems related to other heuristics **[88]**. GA acts as a population-based adaptive search process and it presents one of the best approaches among other metaheuristics when facing complex issues. Based on a group of random solutions called individuals, two parents are specifically selected to recreate a new generation. The reproductive task consists of crossovers and mutations, the first one increases the population diversity, and the second riches the intensification. The use of genetic algorithms to solve the proposed model is described below.

• **Representation and initialisation**

The population is composed of N chromosomes formed with binary rows except the first one which contains only ones. In fact, this particular individual allows the contribution of all the kernels in the optimization process, which means a rapid convergence in the case of weak redundancy.

— **Fitness:**
In order to compare the quality of solutions in GA process, a numerical value called fitness is assigned to each individual. This function is related to the principal problem including the objective and constraints functions. Individual with a maximal fitness presents a very good solution of the problem. In our paper, we propose the following fitness function:

$$fit(U, V) = \frac{1}{1 + E_p(\mathbb{W}^*, U, V)} \tag{5.6}$$

where $E_p(\mathbb{W}^*, U, V)$ denotes the penalty function associated with the problem (5.4.2.1). Consequently, an individual that minimizes $E(\mathbb{W}^*, U, V)$, maximizes the Fitness.

— **Selection:**

During the functioning of the algorithm, we select the best individual to reproduce. This selection is made by the roulette wheel selection [**70**]. This type of selection assigns each individual a location on the wheel. The spot size, which plays an important role in this process, depends on individual fitness. In other words, the size of the spot increases with the increasing of fitness. Thereafter, the ball is thrown into the wheel. Individuals that are considerably fit are much more likely to have the ball thrown in place. This operation is defined by the following probability $P_i = \frac{fit(U^i, V^i)}{\sum_{i=1}^{N} fit(U^i, V^i)}$ and the selection point is chosen randomly from the interval $[0, 1]$.

• **Reproduction: Mutation and Crossover**

Reproduction ensures diversification and intensification of the solutions space thanks to its operators: crossover and mutation. In this paper, we opt for the one-point crossover, in which we cut at the same point two selected parents and next swapping the right portions between themselves thus producing two new strings. While the crossover allows discovering the mixtures of the parents, the mutation ensures the creation of new solutions by changing one or more elements of the individuals [**69**]. In our work, we use the bit string mutation which takes a random genome and invert the bit (i.e. if the genome bit is 1, it is changed to 0 and vice versa).

5.4.2.2 Adaptation of CNNO to TSR-NCR

The obtained CNN^* is then used to classify the data received from the normal cloud representation. In fact, based on a raw time series, the normal cloud representation allows to

build a 2D data that forms the inputs of CNN. These inputs that represent the time series features are then reinforced using the CNN integrated extractor at the same time with the redundancy reduction within CNNs. Times series classification approach is shown explicitly in Figure 5.1.

5.5 Experiments

In this section, we aim to evaluate and demonstrate the performance of our proposed NCR-CNNO. For this purpose, we conduct several experiments on time series classification tasks using seventeen benchmark time series datasets. The experiments are implemented at three levels: In the first one, we show the efficiency of the normal cloud representation by studying its reduced dimension m. Secondly, we provide surprising results regarding the redundancy reduction over the CNNs. Finally, the proposed method is compared against several methods from the literature. Datasets, evaluation metrics as well as the experiments are explicitly described below.

5.5.1 Description of datasets

We evaluate and approve the performance of our approach using 20 time series datasets from the UCR time series classification archive [**124**], which is derived from a broad range of applications including yoga poses (Yoga), monitoring of fish migration (Fish), the shapes of Swedish leaves (Swedish Leaf), and so on. A summary of the datasets is shown in Table 5.1.

5.5.2 Evaluation metrics

In addition to the error rate, five different performance measures have been proposed to evaluate the efficiency of all TSC approaches. These measures are described as follow:

- **Wins:** represents the number of datasets on which achieve the lowest error rate.
- On each dataset, we also rank all methods and report the average ranking results **Mean Rank (MR)** and **Mean Error (ME)**.
- **Mean Per-Class Error (MPCE):** represents the arithmetic mean of the Peer-Class Error (PCE). This can be expressed as:

$$MPCE = \frac{1}{N}\sum_{j=1}^{N} PCE_j$$

and

$$PCE_j = \frac{e_j}{c_j}$$

where N denotes the number of datasets, e_j and c_j are respectively the error rate and the number of classes in the jth dataset.

- Regarding the optimization performance, we use the reduction rate R_r defined as:

$$R_r = \frac{\text{\#of Optimized parameters}}{\text{\#of parameters in CNN}}$$

5.5.3 Description of experiments

As mentioned in the above, three important experiments are conducted to show the efficiency of the normal cloud representation, the redundancy reduction in CNN and both of them in Time series classification.
In the first experiment, we aim to discuss the normal cloud representation by studying the reduced dimension choice m. In fact, such hyperparameter plays an important role in improving the representation performance. Thus, it is essentially to study the optimal choice of m taking into the consideration the information within raw time series. In this context, we use a synthetic CNN with one convolution layer, batch normalisation, one pooling layer and two fully connected layers with specific characteristics described in Fig.5.3. The suggested CNN is used to classify three time series datasets that are differents in term of the number of classes and the length of time series. For each dataset, we utilize a different value of m and we calculate its corresponding error rate generating at the end the variation of m according to the error rate.
In the second experiment, we fix the reduced dimension m based on the first experiment, and we show the redundancy reduction in CNNs using the optimization model. Indeed, we provide exactly the redundancy reduction rate by applying the proposed CNN (Fig.5.3) to the seventeen datasets presented in Table. 5.1. Moreover, we compare the performance of optimal and the standard CNNs in TSC.
In the last experiment, the optimal CNN and the normal cloud representation with the optimal choice of m are applied to the classification of the proposed datasets and compared with different TSC methods frequently used in the literature.
The suggested CNN has been turned on using a kernel size 3×3, a pooling with 2×2 and the learning algorithm is Adam optimizer.

5.5.4 Results and discussion

5.5.4.1 Influence of the reduced dimensions m

In the first experiment, we evaluated the error rate using a set
$\{6, 8, 10, 12, 14, 16, 18, 20, 22, 24\}$. As shown from Figures 5.4,5.5 and 5.6, the error rate decrease with the reduced dimension increasing. Indeed, this seems to be logical because of the pertinent information provided by a large number of segments in the used representation. These Figures report also that starting at certain reduced dimension, the error rate decreasing becomes very slow and tend to be a stationary. For instance, in Figure 5.4 , form $m = 14$, no change occurs. These results indicate that for the proposed time series representation, there is a specific reduced dimension which is a very sufficient to represent the relevant information within a raw time series. For example, $m = 12$ $m = \{12, 14\}$ and $\{14\}$ are the optimal reduced dimension for classification of

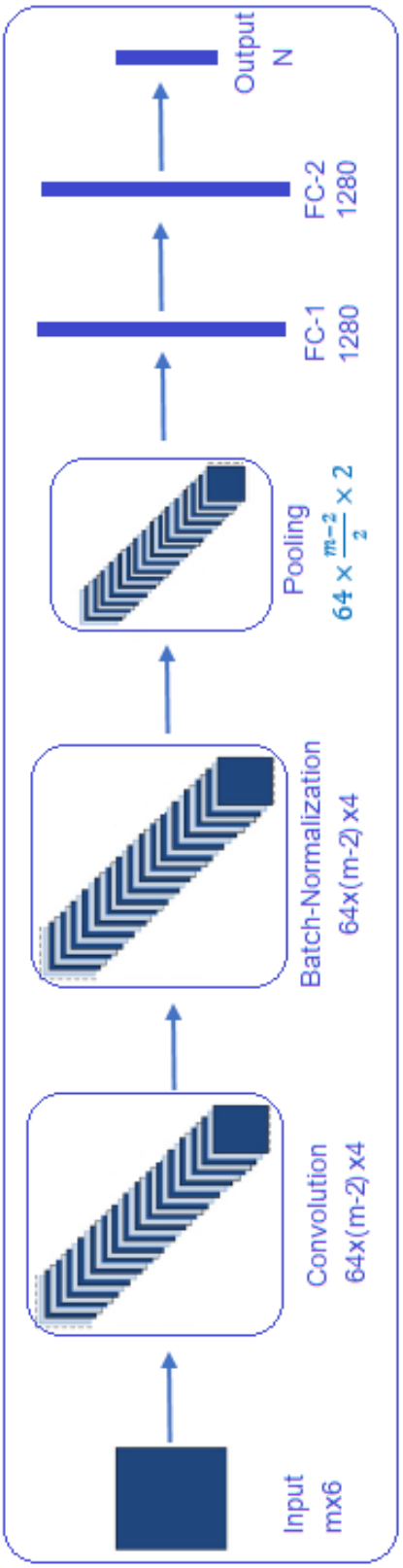

Figure 5.3 – Suggested CNN architecture for time series classification

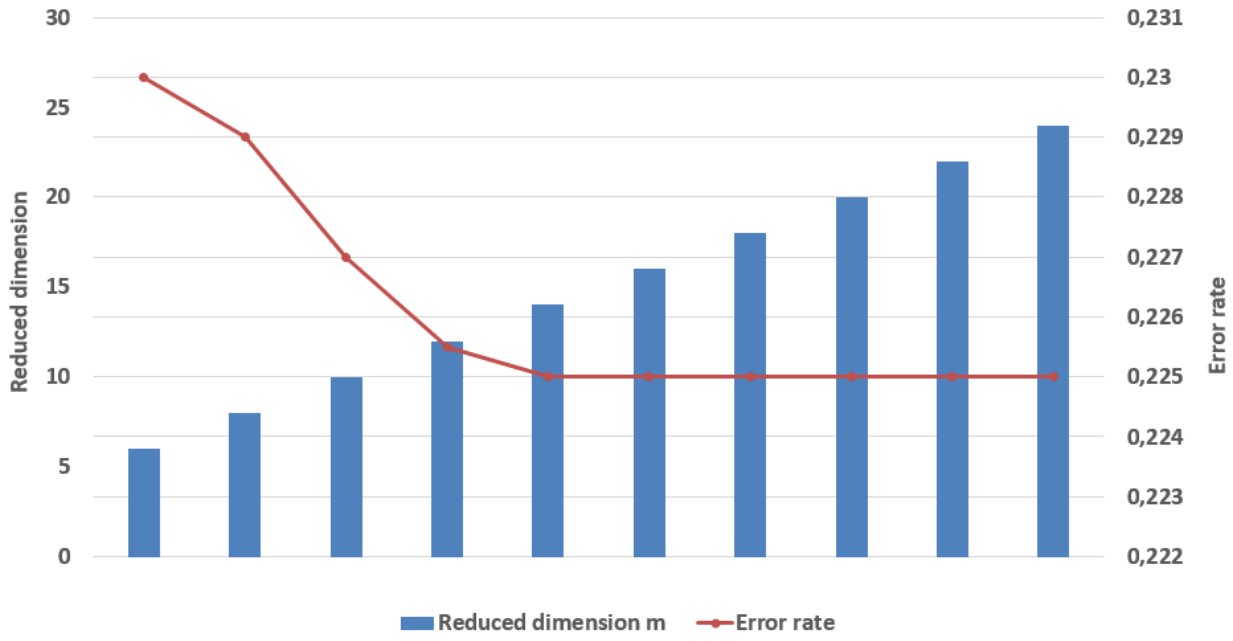

Figure 5.4 – Variation of the error rate according to the reduced dimensions m: case of *Lightning* – 7 dataset

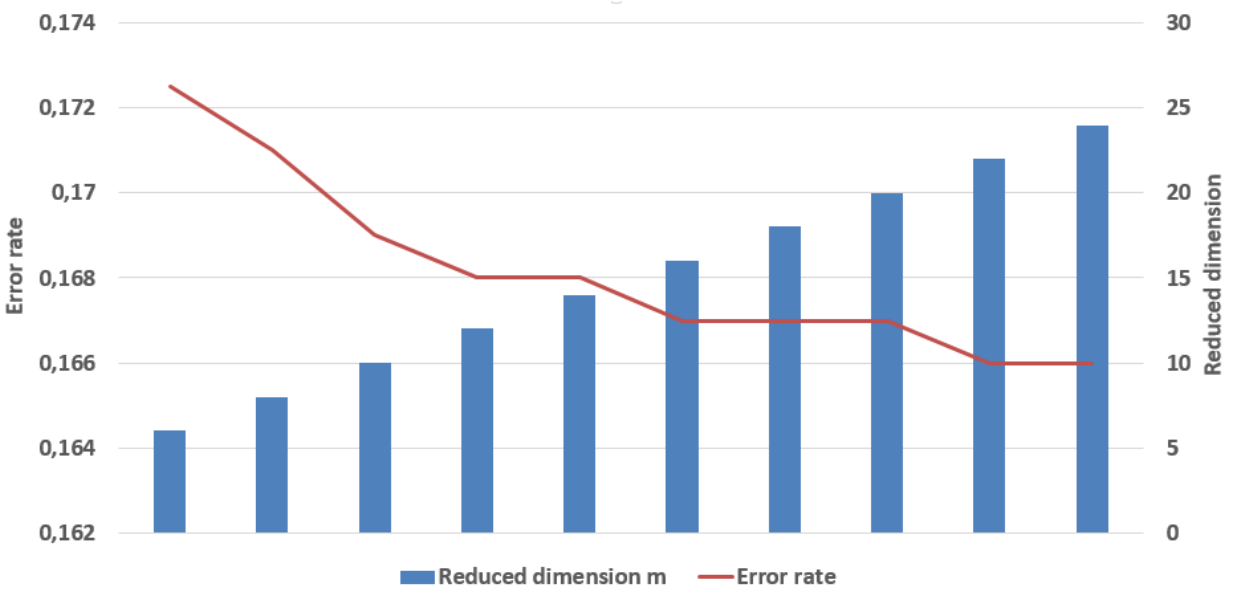

Figure 5.5 – Variation of the error rate according to the reduced dimensions m: case of *Beef* dataset

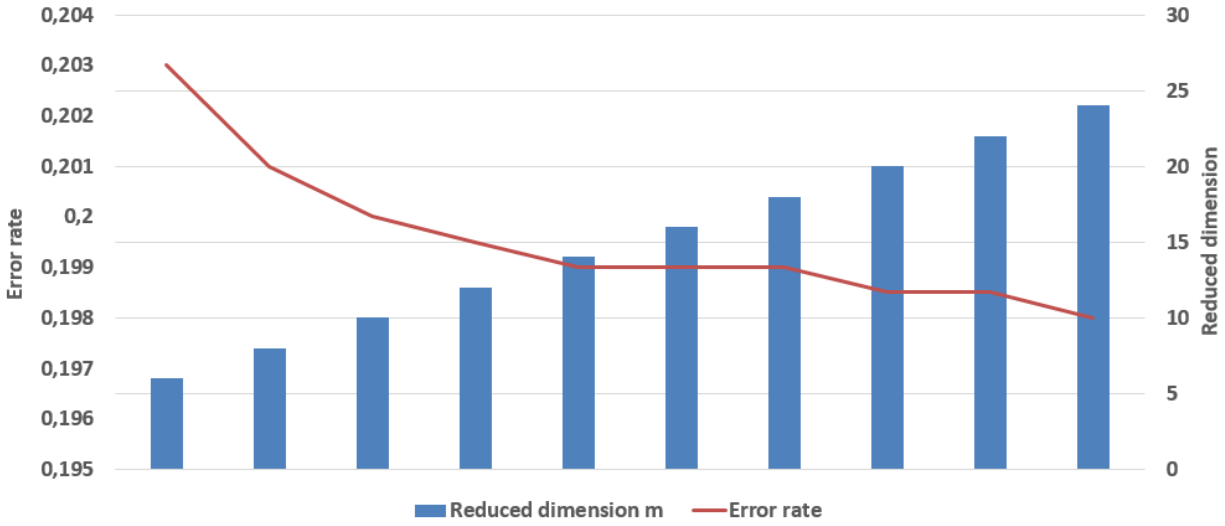

Figure 5.6 – Variation of the error rate according to the reduced dimensions m: case of *Adiac* dataset

respectively $Lightning-7$, $Beef$ and $Adiac$. These result report also that we can choose a common optimal choice of the hyperparameter m that still available for the other time series datasets. Indeed, for the three datasets $Lightning-7$, $Beef$ and $Adiac$, $m = 12$ is a very good candidate to be a segments number taking into the consideration that for $Beef$ dataset, the error rate is constant for $m = 12$ and $m = 14$ and in the last one, between $m = 12$ and $m = 14$, the error rate is equals to 0.0005 which is negligible with respect to the dimension reduction. Thus, according to these experiments, we can choose $m = 12$ as an optimal and sufficient segment number to represent these time series. It is important to note that the three datasets were used in this experiment with different number of classes ($5, 7$ and 37) and with different time series length ($470, 319$ and 176) which is an encouraged result to use the obtained reduced dimension m to classify the other datasets.

5.5.4.2 Performance of CNN optimization

In the second experiment, we evaluated the performance in term of the redundancy reduction in CNNs. In this sense, for each dataset of Table 5.1, we used the proposed CNN with the optimal hyperparameter m obtained from the previous experiment. Furthermore, we adopt the optimization model to reduce the redundant and unnecessary parameters in CNN. Fig. 5.7 illustrates the total and the needed parameters in CNN for each dataset. As shown from this Figure, the number of parameters in CNN is considerably reduced for each dataset. That means that the redundant parameters, defined as the difference between the total and necessary parameters, are well identified. For instance, in the case of $Lightning-7$ and $50Words$, the parameters numbers are respectively reduced by 47851 and 55749, which is an important reduction. Thus, the optimization process provided a significant reduction in the computational cost as well as in the model complexity. In terms of the time series classification performance, we compared the error rate of our

model using both of CNN and optimized CNN. For this purpose, the synthetic CNN (Fig. 5.3) and its optimized model were applied to the suggested datasets (Table 5.1). Result of this experiment is numerically presented in Table 5.3. As shown from this Table, in most cases, the error rates generated by the optimized model is less that those obtained from the standard model. In fact, fourteen cases among the seventeen datasets confirmed the superiority of the optimized CNN.

5.5.4.3 Comparison with state-of-the-art methods

As mentioned above, the third experiment aims to demonstrate the outperforming of the suggested TSC approach (NCR-CNNO) compared to the existing methods. For this purpose, the optimal reduced dimension m obtained from the first experiment is used in the time series representation, while the optimized CNN architecture obtained from the optimization model is utilized to classify the seventeen time series datasets. Moreover, to confirm the robustness of our approach, we compared our results with five most efficient TSC methods recently proposed. These methods include two classical TSC approaches represented by BOSS [**122**] and COTE [**128**] and three deep learning based approaches that are MCNN [**89**], RPCNN [**39**], RPMCNN [**7**]. Table 5.3 reports the error rates obtained using each method. As stated form this Table, our obtained results are encouraged and satisfactory with respect to the other methods with reference to RPMCNN and PRCNN results that are close to NCR-CNNO results. These superiorities are explained by the use of the CNN as well as its optimization. This experiment also reported the evaluation using other metrics previously described in paragraph 5.5.2. Table 5.4 showed the comparaison of NCR-CNNO with the other TSC methods. This table confirmed the efficiency of the suggested approach as shown in the case of the Wins metric in which our approach has generated 11 against 6 for RPMCNN and 5 for RPCNN. This table also approved the NCR-CNNO performance using MPCE metric. Indeed, our proposal provided 0.0063 against 0.0124 for RPMCNN and 0.0127 for RPCNN.

5.6 Conclusion

In this paper, a new optimized deep learning framework for time series classification was proposed. The suggested approach consisted of two stages: In the first one, a raw time series was converted to a two-dimensional representation using normal cloud representation. In the second part, we have exploited the normalized matrix generated in the representation phase as an input of a convolutional neural network. This latter is reinforced and improved by an optimization model related to its parameter redundancy. Our approach has led to a significant improvement in the performance of time series classification and important redundancy reduction. Experiments show that the proposed framework can improve the performance in terms of redundancy optimization and classification using different time series datasets. In the future, we will extend the optimization model to reduce the redundancy in the first phase and explore more analysis for a better time series classification.

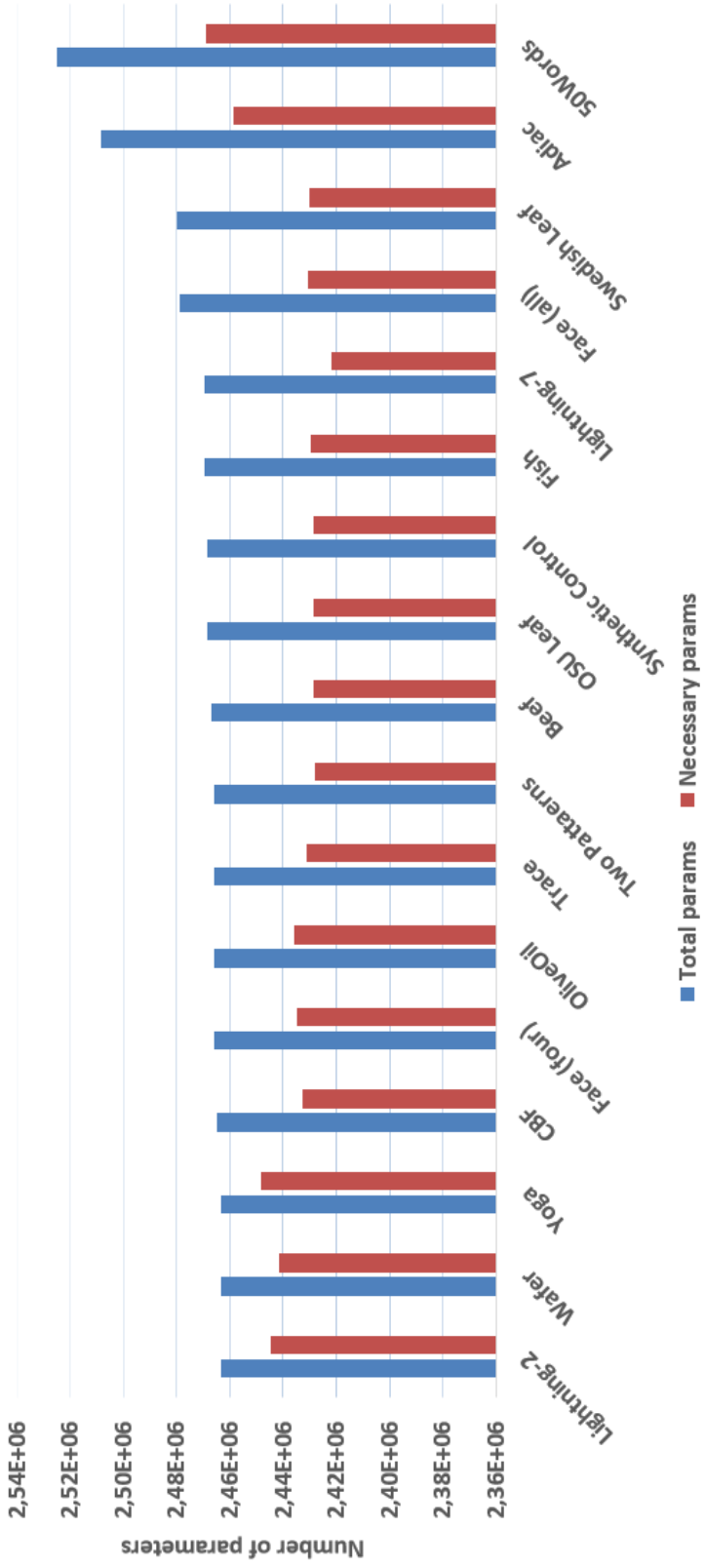

Figure 5.7 – Parameters number in CNN and in the corresponding optimized CNN* for each time series dataset

Table 5.1 – Description of the used time series datasets

Name	Number of classes	Size of training set	Size of testing set	Time series length	Type
Lightning-2	2	60	61	637	Real
Wafer	2	1000	6174	152	Real
Yoga	2	300	3000	426	Shape
CBF	3	30	900	128	Synthetic
Face (four)	4	24	88	350	Shape
OliveOil	4	30	30	570	Real
Trace	4	100	100	275	Synthetic
Two Patterns	4	1000	4000	128	Synthetic
Beef	5	30	30	470	Real
OSU Leaf	6	200	242	427	Shape
Synthetic Control	6	300	300	60	Synthetic
Fish	7	175	175	463	Shape
Lightning-7	7	70	73	319	Real
Face (all)	14	560	1690	131	Shape
Swedish Leaf	15	500	625	128	Shape
Adiac	37	390	391	176	Shape
50Words	50	450	455	270	Real

Table 5.2 – Comparison of NCR-CNN with and without using redundancy optimization on different datasets

Dataset	NCR-CNN(without optimization)	NCR-CNNO(with optimization)
Lightning-2	0.001	0
Wafer	0	0
Yoga	0.095	0.021
CBF	0	0
Face (four)	0.034	0.002
OliveOil	0.102	0.083
Trace	0.023	0.001
Two Patterns	0.002	0
Beef	0.167	0.121
OSU Leaf	0.048	0.011
Synthetic Control	0	0
Fish	0.072	0.015
Lightning-7	0.225	0.213
Face (all)	0.103	0.098
Swedish Leaf	0.079	0.072
Adiac	0.199	0.189
50Words	0.143	0.127

Table 5.3 – Error rates of different approaches on 17 UCR time series datasets

Dataset	BOSS	COTE	MCNN	RPCNN	RPMCNN	NCR-CNNO
Lightning-2	0.148	0.164	0.164	0.001	0	0
Wafer	0.001	0.001	0.002	0	0.001	0
Yoga	0.081	0.113	0.112	0	0.11	0.021
CBF	0.001	0.001	0.002	0.005	0	0
Face (four)	0.001	0.091	0	0	0.034	0.002
OliveOil	0.1	0.1	0.133	0.11	0.1	0.083
Trace	0	0.01	0	0.001	0	0.001
Two Patterns	0.016	0	0.002	0.17	0	0
Beef	0.2	0.133	0.367	0.08	0.133	0.121
OSU Leaf	0.012	0.145	0.271	0.29	0.186	0.011
Synthetic Control	0.03	0	0.003	0	0.003	0
Fish	0.011	0.029	0.051	0.085	0.034	0.015
Lightning-7	0.342	0.247	0.219	0.26	0.247	0.213
Face (all)	0.21	0.105	0.235	0.19	0.189	0.078
Swedish Leaf	0.072	0.046	0.066	0.06	0.034	0.072
Adiac	0.22	0.233	0.231	0.28	0.182	0.189
50Words	0.301	0.191	0.19	0.26	0.193	0.127

Table 5.4 – Evaluation metrics of different approaches using 17 UCR time series datasets

Metrics	BOSS	COTE	MCNN	RPCNN	RPMCNN	NCR-CNNO
Wins	3	2	2	5	6	11
Mean Rank	3.647	3.176	4.176	3.764	2.647	1.823
Mean Error	0.102	0.094	0.12	0.105	0.088	0.059
MPCE	0.0161	0.0176	0.0213	0.0127	0.0124	0.0063

Conclusions and Future work

A deep neural network (DNN) has obtained a great attention for solving engineering problems including computer vision, natural language processing, speech recognition, and many others. A deep neural network is an artificial neural network (ANN) with several hidden layers and deeper combinations, which is responsible for learning from data with impressive capability. However, the biggest issue faced by machine learning experts is the overfitting problem. Indeed, it refers to the phenomenon where a neural network learns from data but also from patterns, considered as noise, that are not related to the problem. Thus, overfitting describes a complex model with a very high variance type error.

To deal effectively with this issue, regularization is a set of techniques that can prevent overfitting in deep neural networks by controlling the variance type error, and thus improves the accuracy of a deep learning model when facing completely new data from the problem domain. The main idea behind this is to make the model more simple by imposing some constraints. Generally speaking, it reduces the variance type error and improves the generalization of the solution. There are many forms of regularization, including the Euclidean-type regularization L_2 which favors low coefficients while a lasso-type regularization L_1 (used when the number of input variables is high) will promote the "sparsity" of the representation by pushing the algorithm to take into account only a small part of the data, ignoring the others.

In this thesis, our study focuses on regularization of deep neural networks using different approaches. In fact, motivating by the success of DropConnect method, a theoretical analysis is developed in order to estimate its hyperparameters. This estimation is based on optimizing the generalization gap using Rademacher complexity. This generated at the end a new adaptive regularization method. The second approach consists of regularizing CNNs by reducing the parameters redundancy. In this context, an optimization model is proposed to control and localize the redundancy over the networks. In addition, the time series classification is treated using deep learning as the last contribution of this work. For each problem studied in this work, a number of numerical results are presented and compared against several approaches from the literature.

Future work

The research findings made out of this thesis has opened several research directions, which have a scope for further investigations. The estimation of DropConnect hyperparameters utilizes risk bound analysis to optimize the generalization gap, which provides a significant train of thought for follow-on research. Our planned future work includes the derivation of more compact bound estimation of the generalization gap and development of more stable and accurate regularization approaches. Furthermore, it is observed that the redundancy optimization model proposed focuses on reducing the redundant kernels. In this context, we will investigate the optimization models to reduce more undesirable phenomena such as overfitting and parameters redundancy. Finally, the application treated in the last contribution generate several suggestions about future research in time series classification and deep neural networks.

Bibliography

[1] Baydogan M G, Runger G, Tuv E (2013) A bag-of-features framework to classify time series. IEEE transactions on pattern analysis and machine intelligence, 35(11), 2796-2802.

[2] Ding H, Chen K, Yuan Y, et al (2017) A compact CNN-DBLSTM based character model for offline handwriting recognition with Tucker decomposition. In: 2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR). IEEE, pp 507–512

[3] Somu N, MR GR, Ramamritham K (2021) A deep learning framework for building energy consumption forecast. Renewable and Sustainable Energy Reviews 137:110591

[4] Hssayni E H, Joudar N, Ettaouil M (2021) A deep learning framework for time series classification using normal cloud representation and CNN optimization.

[5] Chen W, Shi K (2019) A deep learning framework for time series classification using Relative Position Matrix and Convolutional Neural Network. Neurocomputing 359:384–394

[6] Gharehbaghi A, Lindén M (2017) A deep machine learning method for classifying cyclic time series of biological signals using time-growing neural network. IEEE transactions on neural networks and learning systems, 29(9), 4102-4115.

[7] Zhang Q, Yang L T, Chen Z, Li P, (2018) A dropconnect deep computation model for highly heterogeneous data feature learning in mobile sensing networks. IEEE Network 32(4) pp. 22-27

[8] Deb K, Pratap A, Agarwal S, Meyarivan T (2002) A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE transactions on evolutionary computation 6:182–197

[9] Hinton GE, Osindero S, Teh Y-W (2006) A fast learning algorithm for deep belief nets. Neural computation 18:1527–1554

[10] Khan S, Rahmani H, Shah S A A, Bennamoun M (2018) A guide to convolutional neural networks for computer vision. Synthesis Lectures on Computer Vision, 8(1), 1-207

[11] Zhang Y, Zhu F (2021) A kernel-based weight decorrelation for regularizing CNNs. Neurocomputing 429:47–59

[12] Li D, Liu C, Gan W (2009) A new cognitive model: Cloud model. International journal of intelligent systems, 24(3), 357-375.

[13] Hssayni E H, Ettaouil M (2021) A Novel Pooling Method for Regularization of Deep Convolutional Neural networks and Application to Image Classification. In : International Conference on Advanced Intelligent Systems for Sustainable Development. Springer, Cham, 2020. p. 721-732.

[14] Hssayni E H, Ettaouil M (2020) A Novel Pooling Method for Regularization of Deep Neural networks. 2020 International Conference on Intelligent Systems and Computer Vision (ISCV), IEEE, pp 1-6

[15] Shen Z, Zhang Y, Lu J, Xu J, Xiao G (2020) A novel time series forecasting model with deep learning. Neurocomputing, 396, 302-313.

[16] Liu W, Wang Z, Liu X, et al (2017) A survey of deep neural network architectures and their applications. Neurocomputing 234:11–26

[17] Deng H, Runger G, Tuv E, Vladimir M (2013) A time series forest for classification and feature extraction. Information Sciences, 239, 142-153.

[18] Polyak B T, Juditsky A B (1992) Acceleration of stochastic approximation by averaging. SIAM journal on control and optimization, 30(4), 838-855.

[19] Hoseini F, Shahbahrami A, Bayat P (2019) AdaptAhead optimization algorithm for learning deep CNN applied to MRI segmentation. Journal of digital imaging 32:105–115

[20] Holland JH (1992) Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence. MIT press

[21] Kingma D P, Ba J (2014) Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980

[22] Duchi J, Hazan E, Singer Y (2011) Adaptive subgradient methods for online learning and stochastic optimization. Journal of machine learning research, 12(7).

[23] Hssayni E H, Joudar N, Ettaouil M (2022) An adaptive Drop method for deep neural networks regularization: Estimation of DropConnect hyperparameter using generalization gap.

[24] Polap D, (2020) An adaptive genetic algorithm as a supporting mechanism for microscopy image analysis in a cascade of convolution neural networks. Applied Soft Computing, 97, 106824.

[25] Warde-Farley D, Goodfellow I J, Courville A, Bengio Y, (2013) An empirical analysis of dropout in piecewise linear networks, arXiv preprint arXiv:1312.6197

[26] Yadav N, Yadav A, Kumar M (2015) An introduction to neural network methods for differential equations (pp. 13-15). Berlin: Springer

[27] Zhang T (2010) Analysis of multi-stage convex relaxation for sparse regularization. Journal of Machine Learning Research, 11(3)

[28] Jiang L, Liu Y, Wu S, Yang C (2021) Analyzing ecological environment change and associated driving factors in China based on NDVI time series data. Ecological Indicators, 129, 107933.

[29] Mahdavifar S, Ghorbani AA (2019) Application of deep learning to cybersecurity: A survey. Neurocomputing 347:149–176

[30] Amini M R (2015) Apprentissage machine: de la théorie à la pratique. Editions Eyrolles

[31] Li D, Du Y (2017) Artificial intelligence with uncertainty. CRC press

[32] Polap D, Wlodarczyk-Sielicka M, Wawrzyniak N, (2021) Automatic ship classification for a riverside monitoring system using a cascade of artificial intelligence techniques including penalties and rewards. ISA transactions.

[33] Wu W, Fan Q, Zurada J M, Wang J, Yang D, Liu Y (2014) Batch gradient method with smoothing L1/2 regularization for training of feedforward neural networks. Neural Networks, 50, 72-78

[34] Ioffe S, Szegedy C (2015) Batch normalization: Accelerating deep network training by reducing internal covariate shift. In International conference on machine learning (pp. 448-456). PMLR

[35] Poernomo A, Kang D K, (2018) Biased dropout and crossmap dropout: learning towards effective dropout regularization in convolutional neural network. Neural networks 104:60-67

[36] Khan M, Wang H, Riaz A, Elfatyany A, Karim S (2021) Bidirectional lstm-rnn-based hybrid deep learning frameworks for univariate time series classification. The Journal of Supercomputing, 77(7), 7021-7045.

[37] Jia Y, Shelhamer E, Donahue J, Karayev S, Long J, Girshick R, et al. (2014) Caffe: Convolutional architecture for fast feature embedding. In Proceedings of the 22nd ACM international conference on Multimedia (pp. 675-678)

[38] Boullé N, Dallas V, Nakatsukasa Y, Samaddar D (2020) Classification of chaotic time series with deep learning. Physica D: Nonlinear Phenomena, 403, 132261.

[39] Hatami N, Gavet Y, Debayle J (2018, April) Classification of time-series images using deep convolutional neural networks. In Tenth international conference on machine vision (ICMV 2017) (Vol. 10696, p. 106960Y). International Society for Optics and Photonics.

[40] Li Z, Dong M, Wen S, et al (2019) CLU-CNNs: Object detection for medical images. Neurocomputing 350:53–59

[41] Xu D, Zhang S, Zhang H, Mandic D P (2021) Convergence of the rmsprop deep learning method with penalty for nonconvex optimization. Neural Networks, 139, 17-23

[42] Sermanet P, Chintala S, LeCun Y (2012) Convolutional neural networks applied to house numbers digit classification. In Proceedings of the 21st International Conference on Pattern Recognition (ICPR2012) (pp. 3288-3291). IEEE

[43] Tai C, Xiao T, Zhang Y, Wang X (2015) Convolutional neural networks with low-rank regularization. arXiv preprint arXiv:151106067

[44] Louati H, Bechikh S, Louati A, et al (2021) Deep convolutional neural network architecture design as a bi-level optimization problem. Neurocomputing 439:44–62

[45] LeCun Y, Bengio Y, Hinton G (2015) Deep learning. nature, 521(7553), 436-444

[46] Goodfellow I, Bengio Y, Courville A (2016) Deep learning. MIT press.

[47] Fawaz H I, Forestier G, Weber J, Idoumghar L, Muller P A (2019) Deep learning for time series classification: a review. Data mining and knowledge discovery, 33(4), 917-963

[48] Hernandez KAL, Rienmüller T, Baumgartner D, Baumgartner C (2020) Deep learning in spatiotemporal cardiac imaging: A review of methodologies and clinical usability. Computers in Biology and Medicine 104200

[49] Patterson J, Gibson A (2017) Deep learning: A practitioner's approach. " O'Reilly Media, Inc.".

[50] Calin O (2020) Deep Learning Architectures: A Mathematical Approach. Springer.

[51] Li G, Liang S, Nie S, Liu W, Yang Z, (2021) Deep neural network-based generalized sidelobe canceller for dual-channel far-field speech recognition. Neural Networks 141, 225-237.

[52] Wang Y L, Jahanshahi H, Bekiros S, Bezzina F, Chu Y M, Aly A A (2021) Deep recurrent neural networks with finite-time terminal sliding mode control for a chaotic fractional-order financial system with market confidence. Chaos, Solitons and Fractals, 146, 110881.

[53] He K, Zhang X, Ren S, Sun J (2016) Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp 770–778

[54] Santra B, Paul A, Mukherjee D P, (2020) Deterministic dropout for deep neural networks using composite random forest. Pattern Recognition Letters 131:205-212

[55] Mobiny A, Yuan P, Moulik S K, Garg N, Wu C C, Van Nguyen H, (2021) Dropconnect is effective in modeling uncertainty of bayesian deep networks. Scientific reports, nature, 11(1) pp. 1-14.

[56] Iosifidis A, Tefas A, Pitas I, (2015) Dropelm: Fast neural network regularization with dropout and dropconnect. Neurocomputing 162:57-66

[57] Gal Y, Ghahramani Z, (2016) Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In: international conference on machine learning, PMLR, pp. 1050-1059

[58] Gao W, Zhou Z H, Dropout rademacher complexity of deep neural networks. Science China Information Sciences 59(7) pp. 1-12

[59] Wager S, Wang S, Liang P S, (2013) Dropout training as adaptive regularization. Advances in neural information processing systems 26:351-359.

[60] Mou M, Zhou Y, Gao J, Wang L, (2018) Dropout training, data-dependent regularization, and generalization bounds. In: International conference on machine learning, PMLR, pp. 3645-3653

[61] Srivastava N, Hinton G, Krizhevsky A, Sutskever I, Salakhutdinov R, (2014) Dropout: a simple way to prevent neural networks from overfitting. The journal of machine learning research 15(1) pp. 1929-1958.

[62] Khan M, Wang H, Ngueilbaye A, Elfatyany A (2020) End-to-end multivariate time series classification via hybrid deep learning architectures. Personal and Ubiquitous Computing, 1-15.

[63] Xiao H, Rasul K, Vollgraf R (2017) Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. arXiv preprint arXiv:170807747

[64] Sezer O B, Gudelek M U, Ozbayoglu A M (2020) Financial time series forecasting with deep learning: A systematic literature review: 2005–2019. Applied Soft Computing, 90, 106181.

[65] Liang Z, Wang H (2021) Efficient class-specific shapelets learning for interpretable time series classification. Information Sciences, 570, 428-450

[66] Yao P, Wu H, Gao B, et al. (2020) Fully hardware-implemented memristor convolutional neural network. Nature, 577(7792), 641-646

[67] Hssayni E H, Ettaouil M (2021) Generalization ability augmentation and regularization of deep convolutional neural networks using $l^{1/2}$ pooling. International Journal on Technical and Physical Problems of Engineering (IJTPE), Issue 48, Vol. 13, No. 3, pp. 1-6.

[68] Kawaguchi K, Kaelbling L P, Bengio Y, (2017) Generalization in deep learning. arXiv preprint arXiv:1710.05468

[69] Goldberg DE (2006) Genetic algorithms. Pearson Education India

[70] Gen M, Cheng R (1999) Genetic algorithms and engineering optimization. John Wiley and Sons

[71] LeCun Y, Bottou L, Bengio Y, Haffner P (1998) Gradient-based learning applied to document recognition. Proceedings of the IEEE 86:2278–2324

[72] LeCun Y, Boser B, Denker JS, et al (1989) Handwritten digit recognition with a backpropagation network. In: Proceedings of the 2nd International Conference on Neural Information Processing Systems. pp 396–404

[73] Tong T, Tanaka G, (2019) Hybrid pooling for enhancement of generalization ability in deep convolutional neural networks. Neurocomputing 333:76-85

[74] Krizhevsky A, Sutskever I, Hinton GE (2012) Imagenet classification with deep convolutional neural networks. Advances in neural information processing systems 25:1097–1105

[75] Wang Z, Oates T (2015, June) Imaging time-series to improve classification and imputation. In Twenty-Fourth International Joint Conference on Artificial Intelligence.

[76] Chen Z, Zuo W, Hu Q, Lin L (2015) Kernel sparse representation for time series classification. Information Sciences, 292, 15-26.

[77] Cabrera D, Sancho F, Cerrada M, Sánchez R V, Li C (2020) Knowledge extraction from deep convolutional neural networks applied to cyclo-stationary time-series classification. Information Sciences, 524, 1-14.

[78] Hssayni E H, Joudar N, Ettaouil M (2022) KRR-CNN: kernels redundancy reduction in convolutional neural networks. Neural Computing and Applications, 34(3) pp. 2443–2454

[79] Bottou L (2010) Large-scale machine learning with stochastic gradient descent. In Proceedings of COMPSTAT'2010 (pp. 177-186). Physica-Verlag HD.

[80] LeCun Y, Huang F J, Bottou L, (2004) Learning methods for generic object recognition with invariance to pose and lighting. In: Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004. CVPR 2004, Vol. 2, IEEE, 2004, pp. II-104.

[81] Krizhevsky A, Hinton G (2009) Learning multiple layers of features from tiny images

[82] Sun M, Song Z, Jiang X, Pan J, Pang Y (2017) Learning pooling for convolutional neural network. Neurocomputing, 224, 96-104

[83] Berthelier A, Yan Y, Chateau T, Blanc C, Duffner S, Garcia C (2021) Learning Sparse Filters In Deep Convolutional Neural Networks With A l1/l2 Pseudo-Norm. In CADL 2020: Workshop on Computational Aspects of Deep Learning-ICPR 2020.

[84] Tieleman T, Hinton G (2012) Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural networks for machine learning, 4(2), 26-31

[85] Hssayni E H, Joudar N, Ettaouil M (2022) Localization and reduction of redundancy in CNN using L1-sparsity induction. Ambient Intelligence and Humanized Computing,

[86] Sainath TN, Kingsbury B, Sindhwani V, et al (2013) Low-rank matrix factorization for deep neural network training with high-dimensional output targets. In: 2013 IEEE international conference on acoustics, speech and signal processing. IEEE, pp 6655–6659

[87] Gomes L (2014) Machine-learning maestro michael jordan on the delusions of big data and other huge engineering efforts. IEEE spectrum 20:

[88] Joudar N, Ettaouil M (2019) Mathematical mixed-integer programming for solving a new optimization model of selective image restoration: modelling and resolution by CHN and GA. Circuits, Systems, and Signal Processing 38:2072–2096

[89] Cui Z, Chen W, Chen Y (2016) Multi-scale convolutional neural networks for time series classification. arXiv preprint arXiv:1603.06995.

[90] Lin M, Chen Q, Yan S (2013) Network in network. arXiv preprint arXiv:13124400

[91] Aggarwal C C (2018) Neural networks and deep learning. Springer, 10, 978-3.

[92] Nielsen M A (2015) Neural networks and deep learning (Vol. 25). San Francisco, CA: Determination press.

[93] Hssayni E H, Ettaouil M (2020) New approach to face recognition using co-occurrence matrix and Bayesian neural networks. In: 2020 IEEE 6th International Conference on Optimization and Applications (ICOA). pp. 1-5

[94] Xie J, Hu K, Guo Y, Zhu Q, Yu J (2021) On loss functions and CNNs for improved bioacoustic signal classification. Ecological Informatics, 101331.

[95] Zhang H, Tang Y (2017) Online gradient method with smoothing ℓ_0 regularization for feedforward neural networks. Neurocomputing, 224, 1-8

[96] Bai C, Huang L, Pan X, et al (2018) Optimization of deep convolutional neural network for large scale image retrieval. Neurocomputing 303:60–67

[97] Xu Q, Zhang M, Gu Z, Pan G (2019) Overfitting remedy by sparsifying regularization on fully-connected layers of CNNs. Neurocomputing, 328, 69-74

[98] Junior FEF, Yen GG (2019) Particle swarm optimization of deep neural networks architectures for image classification. Swarm and Evolutionary Computation 49:62–74

[99] Deng W, Wang G, Xu J (2016) Piecewise two-dimensional normal cloud representation for time-series data mining. Information Sciences, 374, 32-50

[100] Yang Z H, Li D (1998) Planar model and its application in prediction. Chinese Journal of computers, 21(11), 961-969.

[101] Denil M, Shakibi B, Dinh L, et al (2013) Predicting parameters in deep learning. arXiv preprint arXiv:13060543

[102] Passricha V, Aggarwal RK (2019) PSO-based optimized CNN for Hindi ASR. International Journal of Speech Technology 22:1123–1133

[103] Bartlett P L, Mendelson S, (2002) Rademacher and gaussian complexities: Risk bounds and structural results. Journal of Machine Learning Research 3 (Nov) 463-482.

[104] Wang H, Yang W, Zhao Z, Luo T, Wang J, Tang Y, (2019) Rademacher dropout: An adaptive dropout for deep neural network via optimizing generalization gap. Neurocomputing 357:177-187

[105] Koltchinskii V, Panchenko D, (2000) Rademacher processes and bounding the risk of function learning. In: High dimensional probability II, Springer, pp. 443-457.

[106] Netzer Y, Wang T, Coates A, Bissacco A, Wu B, Ng A Y, (2011) Reading digits in natural images with unsupervised feature learning.

[107] Zhang Q, Zhang M, Chen T, et al (2019) Recent advances in convolutional neural network acceleration. Neurocomputing 323:37–51

[108] Tibshirani Robert (2011) Regression shrinkage and selection via the lasso: a retrospective. Journal of the Royal Statistical Society: Series B (Statistical Methodology),73(3):273–282

[109] Hssayni E H, Ettaouil M (2020) Regularization of deep neural networks with average pooling dropout. Journal of Advanced Research in Dynamical and Control Systems 12(04-sp), pp. 1720-1726

[110] Khan S H, Hayat M, Porikli F (2019) Regularization of deep neural networks with spectral dropout. Neural Networks, 110, 82-90

[111] Wan L, Zeiler M, Zhang S, LeCun Y, Fergus R, (2013) Regularization of neural networks using dropconnect. In: International conference on machine learning, pp. 1058-1066

[112] Ide H, Kobayashi T, Watanabe K, Kurita T (2020) Robust pruning for efficient CNNs. Pattern Recognition Letters 135:90–98

[113] Ramachandran P, Zoph B, Le Q V (2017) Searching for activation functions. arXiv preprint arXiv:1710.05941

[114] Ye Y, Jiang J, Ge B, Dou Y, Yang K (2019) Similarity measures for time series data classification using grid representation and matrix distance. Knowledge and Information Systems, 60(2), 1105-1134

[115] Ramírez A F, Valencia C F, Cabrales S, Ramírez C G (2021) Simulation of photovoltaic power generation using copula autoregressive models for solar irradiance and air temperature time series. Renewable Energy, 175, 44-67.

[116] Ciuparu A, Nagy-Dăbâcan A, Mureşan R C (2020) Soft++, a multi-parametric non-saturating non-linearity that improves convergence in deep neural architectures. Neurocomputing, 384, 376-388

[117] Ranzato M, Boureau Y-L, LeCun Y (2007) Sparse feature learning for deep belief networks. Advances in neural information processing systems 20:1185–1192

[118] Jaderberg M, Vedaldi A, Zisserman A (2014) Speeding up convolutional neural networks with low rank expansions. arXiv preprint arXiv:14053866

[119] Lebedev V, Ganin Y, Rakhuba M, et al (2014) Speeding-up convolutional neural networks using fine-tuned cp-decomposition. arXiv preprint arXiv:14126553

[120] Zeiler M D, Fergus R (2013) Stochastic pooling for regularization of deep convolutional neural networks. arXiv preprint arXiv:1301.3557

[121] Singh A, Rajan P, Bhavsar A (2020) SVD-based redundancy removal in 1-D CNNs for acoustic scene classification. Pattern Recognition Letters 131:383–389

[122] Schäfer P (2015) The BOSS is concerned with time series classification in the presence of noise. Data Mining and Knowledge Discovery, 29(6), 1505-1530.

[123] Baldi P, Sadowski P, (2014) The dropout learning algorithm. Artificial intelligence 210:78-122.

[124] Chen Y, Keogh E, Hu B, Begum N, Bagnall A, Mueen A, Batista G (2015) The UCR Time Series Classification Archive.

[125] Bai B, Li G, Wang S, Wu Z, Yan W (2021) Time series classification based on multi-feature dictionary representation and ensemble learning. Expert Systems with Applications, 169, 114162.

[126] Jiang W (2020) Time series classification: Nearest neighbor versus deep learning models. SN Applied Sciences, 2(4), 1-17

[127] De Maio C, Fenza G, Gallo M, Loia V, Parente M, (2019) Time-aware adaptive tweets ranking through deep learning. Future Generation Computer Systems, 93, 924-932.

[128] Bagnall A, Lines J, Hills J, Bostrom A (2015) Time-series classification with COTE: the collective of transformation-based ensembles. IEEE Transactions on Knowledge and Data Engineering, 27(9), 2522-2535.

[129] Ma R, Miao J, Niu L, Zhang P (2019) Transformed l^1 regularization for learning sparse deep neural networks. Neural Networks 119:286–298

[130] Hahn S, Choi H (2020) Understanding dropout as an optimization trick. Neurocomputing, 398, 64-70

[131] Ravichandran J, Kaden M, Saralajew S, Villmann T, (2020) Variants of dropconnect in learning vector quantization networks for evaluation of classification stability. Neurocomputing 403:121-132

[132] Simonyan K, Zisserman A (2014) Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:14091556

[133] McDiarmid C (1989) On the method of bounded differences. Surveys in combinatorics, 141(1), 148-188.

[134] Vapnik V N, Chervonenkis A Y (2015) On the uniform convergence of relative frequencies of events to their probabilities. In Measures of complexity (pp. 11-30). Springer, Cham.

[135] Ledoux M, Talagrand M (1991) Probability in Banach Spaces: isoperimetry and processes (Vol. 23). Springer Science and Business Media.

[136] Ettaouil, M., Lazaar, M., Elmoutaouakil, K., and Haddouch, K. (2013). A new algorithm for optimization of the kohonen network architectures using the continuous hopfield networks. wseas transactions on computers, 12(4), 155-163.

[137] Ettaouil, M., Ghanou, Y., Elmoutaouakil, K., and Lazaar, M. (2011). A new architecture optimization model for the Kohonen networks and clustering. Journal of Advanced Research in Computer Science (JARCS), 3(1), 14-32.

[138] Ramchoun, H., and Ettaouil, M. (2019). Hamiltonian Monte Carlo based on evidence framework for Bayesian learning to neural network. Soft Computing, 23(13), 4815-4825.

[139] Ettaouil, M., Ghanou, Y., El Moutaouakil, K., and Lazaar, M. (2011). Image medical compression by a new architecture optimization model for the Kohonen networks. International Journal of Computer Theory and Engineering, 3(2), 204.

[140] Ettaouil, M., Abdelatifi, E., Belhabib, F., and Moutaouakil, K. E. (2012). Learning algorithm of kohonen network with selection phase. WSEAS Transactions on Computers Issue, 11.

[141] Joudar, N. E., En-Naimani, Z., and Ettaouil, M. (2019). Using continuous Hopfield neural network for solving a new optimization architecture model of probabilistic self organizing map. Neurocomputing, 344, 82-91.

[142] Lazaar Mohamed (2013) Contributions à la Sélection des modèles d'architectures des réseaux de neurones artificiels et Apprentissage- Applications aux problèmes réels. Thèse de doctorat.

[143] Ghanou Youssef (2010) Optimisation des architectures dans des réseaux de neurones artificiels et apprentissage Application à l'imagerie médicale. Thèse de doctorat.

[144] Haddouch Khalid (2013) Programmation par contraintes et réseaux de neurones artificiels- Application aux problèmes réels. Thèse de doctorat.