Deep Convolutional Networks without Backpropagation

Mubarakah Alotaibi

Doctor of Philosophy

University of York
Computer Science

November 2022

Declaration of Authorship

I, Mubarakah Alotaibi, declare that this thesis titled "Deep Convolutional Networks without Backpropagation" and the work presented in it are my own. I confirm that:

- This work was wholly completed while I was a student at the University of York pursuing a doctoral degree.
- Wherever I have studied the published work of others, I have always cited it properly.
- The source is always given when I have quoted from others' works. Except for such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- All of the codes are my own work, and I take full responsibility for the results.
- The work in Chapter 3 of this thesis has been previously published in [1]. In addition, the works in Chapters 4 and 6 were submitted to the International Joint Conference on Neural Networks (IJCNN 2023) and have been accepted for presentation (Chapter 4 as a supplementary material).

Signed: [illegible]

Date: November 2022

Abstract

This thesis attempts to develop networks trained without gradient descent or backpropagation designed specifically for classification tasks. The emergence of issues with gradient-based neural networks, such as long training time, vanishing or exploding gradients and high computational costs, has led to the development of such alternatives. In fact, the works presented in this thesis extend PCANet, with the fundamental objective being the development of networks capable of providing both good performance and significant improvements in network depth. Chapter 1 of this thesis formulates the problem, describes the challenges, outlines the research questions and summarises the contributions. In Chapter 2, gradient-based and non-gradient-based networks are reviewed. Chapter 3 presents the Multi-Layer PCANet, whose design is inspired by that of PCANet. However, using second-order pooling and CNN-like filters, the evaluation experiments indicate that the proposed network provides a considerable reduction in the number of features and, consequently, a gain in performance. The networks in Chapters 4 and 5 share the same design as the Multi-Layer PCANet but generate their filter banks using different supervised learning approaches. The experimental results on four databases (CIFAR-10, CIFAR-100, MNIST and TinyImageNet) show that semi-supervised Stacked-LDA filters are sufficient for providing good data representation in the convolutional layers. These filters are produced by combining 50% PCA filters (Chapter 3) with 50% Stacked-LDA filters (Chapter 4). Chapter 6 introduces deep residual compensation convolutional networks for image classification. The design of this network comprises several convolutional layers, each post-processed and trained with new labels learned from the residual information of all preceding layers. The evaluation experiments indicate that the proposed network is competitive with standard gradient-based networks not only in terms of accuracy but also in the number of FLOPs required for training. Chapter 7 summarises the findings and discusses the field's potential future directions.

Acknowledgements

To begin, I would like to express my gratitude to the Saudi Arabian Cultural Bureau in the United Kingdom for sponsoring and funding my studies while I have been abroad. I appreciate their thoughtful assistance, suggestions and generosity throughout my journey. I also wish to extend my gratitude to Taif University, where I work, for providing me with this scholarship. I would never have made it without your generous support, so thank you!

To my PhD supervisor, Prof. Richard Wilson, at the University of York, I am sincerely grateful for all of the time, effort and patience he has invested in guiding and supporting me during my PhD, especially during the first stages of it, when I barely knew anything about computer vision. My thanks to him for constantly reading my work and the papers I sent, for listening patiently to me explaining things and for providing valuable feedback. I also appreciate his responses to my emails even during busy periods of the academic year. I would say that I would never have made it without his guidance, so thanks very much!

I would also like to thank my family for their constant support and encouragement, particularly my sweet mother, who would check on me daily. I would never have been able to complete my research without their tremendous understanding and encouragement. In addition, I would like to thank all the friends I have made in York, with whom I have shared both happy and sad moments. Finally, I would like to extend my gratitude to everybody I have ever met, whether I know them or not, whose pleasant attitude would make my day. In my opinion, each and every one of you makes a difference. So, thank you for easing my way along my PhD journey!

Contents

Declaration of Authorship i

Abstract ii

Acknowledgements iii

1 Introduction 1
- 1.1 Motivation and Context . . . 1
- 1.2 Challenges . . . 3
- 1.3 Objectives and Research Questions . . . 4
- 1.4 Contributions . . . 5
- 1.5 Outline of the Thesis . . . 7

2 Background and Related Works 8
- 2.1 Introduction . . . 8
- 2.2 Gradient-Based Networks . . . 11
  - 2.2.1 Convolutional Neural Networks . . . 11
    - 2.2.1.1 LeNet-5 . . . 11
    - 2.2.1.2 AlexNet . . . 12
    - 2.2.1.3 VGGNet . . . 14
    - 2.2.1.4 Network in Network (NiN) . . . 18
    - 2.2.1.5 Maxout Network . . . 20
    - 2.2.1.6 Stochastic Network . . . 23
    - 2.2.1.7 Residual Network . . . 25
    - 2.2.1.8 The All Convolutional Net . . . 28
    - 2.2.1.9 Densely Connected Convolutional Neural Network . . . 30
    - 2.2.1.10 FractalNet . . . 32
  - 2.2.2 Deep Generative Models . . . 37
    - 2.2.2.1 Boltzmann Machine . . . 37
    - 2.2.2.2 Deep Directed Network . . . 39
    - 2.2.2.3 Deep Boltzmann Machine . . . 40

2.2.2.4 Deep Belief Network 40
2.3 Non Gradient-Based Networks 43
2.3.1 Deep-SVM 43
2.3.2 deep Forest 45
2.3.3 PCANet 46
2.3.4 PCANet+ 48
2.3.5 PCANet-II 51
2.3.6 ScatNet 52
2.4 Conclusions and Discussion 54
2.4.1 Gradient-Based Models and Their Limitations 55
2.4.2 Non-Gradient-Based Models And Their Advantages . . . 56
2.4.3 PCANet As Baseline Model 56

3 Multi-Layer PCANet for Image Classification 58

3.1 Introduction 58
3.2 Multi-Layer PCANet Structure 59
3.2.1 PCA Convolutional Layers 60
3.2.2 Second-Order Pooling 62
3.2.2.1 Covariance Computation 63
3.2.3 Spatial Pyramid Pooling 65
3.2.4 Late Fusion 66
3.3 Experiments and Results 69
3.3.1 Databases 69
3.3.2 Ablation Study on Multi-Layer PCANet Structure 71
3.3.2.1 The Impact of Using Second-Order Pooling and Z-score Normalisation 71
3.3.2.2 Second-Order Pooling for All Feature Maps and Late Fusion 73
3.3.3 The Network Hyper-parameters 75
3.3.3.1 The Impact of the Filter Size on the Classification Task Performance 75
3.3.3.2 The Impact of the Second-Order Pooling Block Size on the Recognition Rate 76
3.3.4 Multi-Layer PCANet for Image Classification Task 76
3.3.4.1 Evalutaion of CIFAR-10 Database 77
3.3.4.2 Evaluation on CIFAR-100 Database 80
3.3.4.3 Evaluation on the MNIST Database 81
3.3.4.4 Evaluation on TinyImageNet Database 82

3.4 Conclusion 82

4 Stacked Linear Discriminant Analysis (Stacked-LDA) 84

4.1 Introduction 84
4.2 General Stacked-LDA Model 86
4.2.1 Description and Algorithm 86
4.2.2 Experiments 90
4.2.2.1 Multi-Class Classifier 90
4.2.2.2 Digit Recognition on MNIST Dataset 94
4.2.3 Section Summary 96
4.3 New Relabelling Technique 96
4.3.1 Description of Relabelling-III 97
4.3.2 Experiments 99
4.3.3 Digits Classification on the MNIST Database 100
4.3.4 Pattern Recognition on CIFAR-10 Database 101
4.3.5 Section Summary 102
4.4 Convolutional Stacked-LDA model 102
4.4.1 Network Structure 102
4.4.1.1 Stacked-LDA convolutional layer 103
4.4.2 Experiments 104
4.4.2.1 Network Hyper Parameters 104
4.4.2.2 Image Classification on four Databases 107
4.5 Conclusions 112

5 Class-Embedding Networks 114

5.1 Introduction 114
5.2 Deep Supervised Networks 116
5.2.1 Preliminary Principles 119
5.2.1.1 Spectral Clustering 119
5.2.1.2 Extreme Learning Machine 120
5.2.1.3 Supervised Laplacian Eigenmaps 123
5.2.1.4 Hilbert–Schmidt Independence Criterion 124
5.2.2 Problem Settings 127
5.2.3 Producing Networks Filter Banks 127
5.2.3.1 Clustering Network 127
5.2.3.2 Supervised Extreme Learning Machine 128
5.2.3.3 Supervised Laplacian Eigenmaps 129
5.2.3.4 Hilbert–Schmidt Independent Criteria Network . 130
5.3 Experiments and Results 131

5.3.1 Selecting Hyperparameters for Proposed Architectures . . 132
5.3.2 Networks Complexity 132
5.3.3 Single-Layer Networks 133
5.3.3.1 Parameter Settings 133
5.3.3.2 Performance Analysis 135
5.3.4 Two-layer Networks 138
5.3.4.1 Parameter Settings 138
5.3.4.2 Performance Analysis 138
5.3.5 Combination of Filters 140
5.3.5.1 Motivation and Parameter Settings 140
5.3.5.2 Performance Analysis 141
5.4 Conclusions 144

6 Deep Residual Compensation Convolutional Network 147
6.1 Introduction 147
6.2 Network Architecture 149
6.3 Experiments and Results 157
6.3.1 Network Parameters 157
6.3.1.1 Experiment 1 157
6.3.1.2 Experiment 2 160
6.3.2 Image Classification over Four Benchmarks with No Data Augmentation 165
6.3.2.1 Handwritten Digit Recognition on the MNIST Database 165
6.3.2.2 Pattern Recognition on the CIFAR-10 Database . 168
6.3.2.3 Image Classification on the CIFAR-100 Database 170
6.3.2.4 Classification on the TinyImageNet Database . . 172
6.3.3 Image Classification on Three Benchmarks with Data Augmentation 174
6.3.3.1 Parameter Settings 174
6.3.3.2 Performance Analysis 175
6.4 Conclusions and Future Work 176

7 Conclusions 179
7.1 Summary of Contributions 179
7.2 Future Research Directions 180

References 190

A Solution to HSIC Optimisation Problem **191**

List of Figures

2.1 LeNet-5 architecture: a five-layer convolutional neural network for image classification on the MNIST database. 11

2.2 AlexNet architecture: eight-Layer CNN for large-scale image classification. More details about the network structure are described in Table 2.2. 13

2.3 The main architecture of Network in Network: a CNN model with three mlpconv layers and one global average pooling layer. Captured from [31]. 19

2.4 Comparison between a) linear convolution layer and b) mlpconv layer. The linear convolution layer slides a linear filter along the image, while the mlpconv layer slides a multi-layer perceptron over the image. 20

2.5 A single Maxout unit: a linear function that returns the maximum of the inputs (z_1 to z_n in the figure). 21

2.6 Graphical depiction of how the Maxout activation function can implement the rectified linear, absolute value rectifier, and approximate the quadratic activation function. Captured from [7]. . 21

2.7 An example of m Maxout units needed to map $x \in \mathbb{R}^d$ to m dimension outputs $h(x) \in \mathbb{R}^m$. $k \geq 2$ in the figure is a user-defined parameter. 22

2.8 Comparasion between ReLU and Maxout with multiple layers on the MNIST database. This figure is captured from [7]. 23

2.9 An example explaining the stochastic pooling idea. The activations inside a region is normalised to obtain the probabilities p_i. The pooled activation (1.6 in this example) is selected using multinomial distribution applied to the computed probabilities p_i. Captured from [8]. 24

2.10 Training error rate (left) and test error rate (right) for 20-layer and 56-layer plain networks on CIFAR-10. This picture is captured from [5]. 25

2.11 Residual learning: building block. This picture is captured from [5]. 26
2.12 ResNet architectures on the ImageNet database. 27
2.13 Training on the ImageNet database using 18 and 34 layer structure of (left) plain networks and (right) residual networks. This picture is captured from [5]. 28
2.14 DenseNet architecture: a deep learning model consists of several dense blocks separated by transition blocks. This picture is captured from [12]. 32
2.15 The architecture of FractalNet: simple Fractal expansion architecture (left), Fractal expansion stacked recursively as a single block (centre), and five blocks cascaded as FractalNet (right). Captured from [11]. 34
2.16 Drop-path: a fractal network block works with certain layer connections deactivated. Captured from [11]. 35
2.17 Two types of Boltzmann machine: restricted and unrestricted. In the unrestricted Boltzmann machine every node is connected to all other nodes. On the other hand, connection between nodes in the same layer is prohibited in the restricted version of Boltzmann machine. 37
2.18 Deep directed network architecture 40
2.19 An example of the explaining-away problem. The nodes "earthquick" and "truck hits the house" end to be dependent on each other. 41
2.20 Deep Boltzmann machine structure. 42
2.21 Deep belief net architecture. 42
2.22 Deep belief net on the MNIST database. 42
2.23 a) SVM maximises margin. b) Kernel trick of SVM. 43
2.24 Structure of deep SVM. Captured from [34]. 44
2.25 Two phases deep forest architecture: multi-grained scanning and cascade forest. This picture is taken from [13]. 45
2.26 The main architecture of PCANet. 47
2.27 The main architecture of PCANet+. 50
2.28 Accuracy of PCANet+ and PCANet on FERET DUP I and DUP II using different layers architectures. The figure is captured from [24]. 50

2.29 Accuracy of PCANet+ and PCANet on FERET DUP I and DUP II with and without mean pooling layers. The figure is captured from [24]. 51
2.30 The main architecture of PCANet-II. 52
2.31 Iterating scattering propagator $\tilde{W}$ defines the convolution network. Captured from [16]. 53

3.1 Multi-Layer PCANet structure 60
3.2 Example of extracting image patches using a filter of size $k_L \times K_L$. 62
3.3 Example of extracting the second-order features from a tensor X^L. 63
3.4 Example of using SPP to reduce the number of patches from 64 to 21 using 3-level SPP with 16, 4 and 1 bin respectively. 67
3.5 Late fusion methods versus early fusion methods. Early fusion methods work on the feature space while late fusion ones split the problem into multiple classification problems and combine the classifiers' decisions. 68
3.6 Mixed examples from the MNIST database. 69
3.7 Some examples from CIFAR-10 database. 70
3.8 A) The accuracy of the models $(A - G)$ using histogram descriptors and second-order features. We computed the second-order features after applying either z-score patch normalization (sop z-score) or the Heaviside function (sop step). B) The number of dimensions generated using the models $(A - G)$ after either using the histogram pooling (histogram in the figure) or the second-order pooling (sop). In both sub figures A and B, we used the CIFAR-10 database. 73
3.9 An experiment on the early fusion method with second-order pooling and z-score normalisation for models $A - G$ described by Table 3.3 using the CIFAR-10 database, where features of all layers are concatenated and processed every 8's, 16's, 24's, and all feature maps before being sent to a linear discriminant analysis classifier for final prediction. 74

3.10 An experiment with the late fusion method using second-order pooling and z-score normalisation techniques for models $A - G$, as outlined in Table 3.3, using the CIFAR-10 database. The experiment involves running a linear discriminant analysis (LDA) classifier on each layer, followed by concatenating the resulting posteriors from all layers before sending them to a final LDA classifier for prediction. Second-order pooling is calculated every 8, 16, 24, and all feature maps. 75
3.11 Accuracy of the models $(A - G)$ (Table 3.3) using different filter sizes on the CIFAR-10 database. 76
3.12 Accuracy of the models $(A - G)$ (Table 3.3) using different SOP block-sizes on the CIFAR-10 database. 77

4.1 An example of a non-linearly separable data set. 86
4.2 Simple feed-forward neural network. 87
4.3 A single LDA to distinguish the two classes. 89
4.4 The new classes generated in the first iteration. 89
4.5 An example of dividing class 5 from the MNIST database into subclasses for two iterations. 98
4.6 Convolutional Stacked-LDA model. 103
4.7 Accuracy of the CIFAR-10 database using different types of filters and different numbers of layers. 106

5.1 Conceptual illustration of the proposed shallow depth networks. The architecture consists of several convolutional layers that learn filter banks through supervised methods. The post-processing step involves applying a non-linear function to convert the data into a non-linear space, followed by second-order pooling. To reduce the high dimensionality of the features, the second-order pooling can be followed by a spatial pyramid pooling. Then we apply a classifier to the output of each layer and combine the classifier’s posteriors to make the final decision. 118
5.2 Structure of single-layer feedforward neural network. 121
5.3 Structure of single-layer ELM-AE. 123
5.4 Two-layer semi-supervised network. 141

6.1 Deep residual compensation convolutional network. 154

6.2 The performance (error rate %) of a deep residual compensation convolutional network with 30 filters throughout all of its layers on the CIFAR-10 database. 159
6.3 The performance (error rate%) of a deep residual compensation convolutional network with 50 filters throughout all of its layers on the CIFAR-10 database. 159
6.4 The performance (error rate %) of a deep residual compensation convolutional network with 50 filters throughout all of its layers on the CIFAR-10 database. The experiment was evaluated by setting α (in equation 6.17) to different values ranging from 0.2 to 1, and the number of layers was set to 30. 162
6.5 The performance (error rate %) of a deep residual compensation convolutional network with 50 filters throughout all of its layers on the CIFAR-10 database, where α (in equation 6.17) was set to 1. 163
6.6 The performance (error rate %) of a deep residual compensation convolutional network with 50 filters throughout all of its layers on the CIFAR-10 database, where α (in equation 6.17) was set to 0.4. 164
6.7 The training and testing error rates using the deep residual compensation network on the MNIST database. 167
6.8 The training and testing error rates (%) for the deep residual compensation convolutional network trained on the CIFAR-100 database. 171
6.9 The training and testing error rates (%) for the deep residual compensation convolutional network trained on the TinyImageNet database without data augmentation. 173

List of Tables

2.1 LeNet-5 architecture. 12
2.2 Explanation of the AlexNet architecture in detail. 13
2.3 VGG Configurations. Each column represents an architecture. . . 16
2.4 The number of parameters in millions in each configuration of VGGNet. 16
2.5 The performance of VGG configurations using the single-scale method. These results were reported in [4]. 17
2.6 The performance of VGG configurations using the multi-scale method. These results were reported in [4]. 18
2.7 Stochastic pooling network compared with max and average pooling using four databases: MNIST, CIFAR-10, CIFAR-100 and SVHN. 25
2.8 ResNet performance on ImageNet, compared with state-of-the-art results. These results were all reported by [5]. 28
2.9 Three baseline architectures for the CIFAR-10 and CIFAR-100 databases. 29
2.10 Three variants derived from model C described in Table 2.9. . . . 30
2.11 Error rate of baseline models described in Table 2.9 and their variants (2.10) on the CIFAR-10 database with no data augmentation. 30
2.12 ALL-CNN error rate (%) compared with some networks on the CIFAR-10 and CIFAR-100 databases with and without data augmentation. These results were reported in [9]. 31
2.13 DenseNet error rate (%) compared with other networks on the CIFAR-10, CIFAR-100 and SVHN databases. These results were all reported by [12]. The "+" symbol in the table refers to the use of data augmentation. 33

2.14 The error rate of FractalNet compared with other methods on three databases, CIFAR-10, CIFAR-100 and SVHN. These results were reported by [11]. The sign "+" in the table indicates the use of data augmentation, while "++" indicates the use of heavy data augmentations. 36
2.15 FractalNet performance compared with ResNet [4] and VGG [5] on the ImageNet database. 36
2.16 The performance of deep-SVM compared to the standard support vector machine using the Breast cancer database. 44
2.17 The performance of gcForest on the MNIST database. All of these results were reported by [13]. 46
2.18 The performance of gcForest on the CIFAR-10 database. These results were reported by [13]. 46

3.1 Fusion methodology 67
3.2 Classes and super-classes of the CIFAR-100 database. 71
3.3 Simple configurations to evaluate the Multi-Layer PCANet on the CIFAR-10 database. 72
3.4 Configurations for CIFAR-10/100, MNIST and TinyImageNet . . 78
3.5 Comparison of the accuracy (%) of some methods on the CIFAR-10/CIFAR-100 database with no data augmentation. We re-implemented the networks marked with (*). 80
3.6 Comparison of the accuracy and number of features of some PCANet methods on the MNIST database with no data augmentation. 81
3.7 Comparison of the accuracy of some methods on the TinyImageNet database with no data augmentation 82

4.1 Evaluation metrics 91
4.2 Different decomposition methods on the MNIST database 93
4.3 Accuracy (%) after applying a single LDA classifier to the MNIST database 94
4.4 Stacked-LDA model's accuracy (%) on the MNIST database using the first labelling method 95
4.5 Stacked-LDA model's accuracy (%) on the MNIST database using the second labelling method 95
4.6 Accuracy (%) with different hidden units using NN on the MNIST database [88] 96
4.7 Accuracy (%) on MNIST database using three relabelling methods.100

4.8 Accuracy (%) after applying LDA on CIFAR-10 101
4.9 Accuracy (%) on the CIFAR-10 database using relabelling methods I and III 101
4.10 Configurations for CIFAR-10/100, MNIST and TinyImageNet. . . 108
4.11 Accuracy of the Stacked-LDA network compared with different methods on the MNIST database with no data augmentation. . . 109
4.12 Accuracy of the Stacked-LDA network compared to some methods on the CIFAR-10 and CIFAR-100 databases with no data augmentation. 110
4.13 Comparison of the accuracy of some methods on the TinyImageNet database with no data augmentation. 112

5.1 The estimated computational complexity of the filters generated by the four proposed networks: SLE, HSIC, S-ELM, and clustering, in comparison to those associated with generating PCA and LDA filters (Chapters 3 and 4). 133
5.2 Accuracy (%) using different networks on the CIFAR-10, CIFAR-100, MNIST and TinyImageNet databases. PCANet in the table refers to Multi-Layer PCANet (Chapter 3). 135

6.1 Accuracy (%) on the CIFAR-10 database test set using ResCNet-50–1 and ResCNet-50–0.4 with different numbers of layers. . . . 163
6.2 Network architectures using the MNIST, CIFAR-100 and Tiny-ImageNet databases. 166
6.3 Accuracy of the deep residual compensation network compared with different methods on the MNIST database, with no data augmentation. 168
6.4 Accuracy of the deep residual compensation network compared with different methods on the CIFAR-10 and CIFAR-100 databases with no data augmentation. 169
6.5 Accuracy and number of FLOPs of the deep residual compensation network compared with different residual networks on the CIFAR-10 database with no data augmentation. 170
6.6 Accuracy (%) of the deep residual compensation convolutional network compared with other methods on the TinyImageNet database, without data augmentation. 174
6.7 Network architectures for the MNIST, CIFAR-100 and TinyImageNet databases, with data augmentation. 175

6.8 Accuracy (%) of deep residual compensation convolutional network on the CIFAR-10, CIFAR-100 and TinyImageNet databases, with and without data augmentation. 176

Chapter 1

Introduction

1.1 Motivation and Context

Deep learning is a machine learning technique that relies on representation learning, by which the system automatically learns the necessary representations from the raw data provided. The representations are learned using a multi-layer framework in which each layer passes the processed data to a higher abstraction layer. With enough transformations and parameters, complex non-linear functions can, therefore, be learned [2].

Convolutional neural networks (CNNs) are deep learning models that extract meaningful abstract representations from raw data. The basic CNN architecture is structured as a set of layers. The first few layers are convolutional layers responsible for detecting useful features using kernels that slide across different training data positions. Some of the convolutional layers are followed by pooling layers, which combine semantically similar features and reduce the amount of information. Between the layers of any CNN design, the data are processed using non-linear activations. At the end of the network, fully-connected layers and a softmax layer are employed. CNN networks are trained in an end-to-end learning fashion using the backpropagation method, which uses chain rules of derivatives to update the network's parameters with regard to its output's errors.

Since the early 2000s, CNNs have been successfully applied to different fields, including face recognition, image segmentation and detection [2]. However, CNNs started to become popular in computer vision and machine learning in 2012, when AlexNet [3] won the ImageNet challenge with a top-five error rate of 15%, effectively halving the error rates of the best competing methods. Since then, many CNN architectures have been investigated. VGGNet [4], for instance, increased the CNN's depth to 19 layers and placed second in the 2014

ImageNet challenge with a top-5 error rate of 7.3%. Using short connections between at least two convolutional layers, ResNet [5] has exceeded the 100-layer barrier and won the ILSVRC competition with an error rate of 3.75%. Other deep CNN examples include Fractional Net [6], Maxout [7], stochastic pooling [8], ALL-CNN [9], binary network [10], FractalNet [11] and DenseNet [12], among many others.

Along with the development of computer hardware and software, CNN architectures have achieved human-level performance and become the dominant approach in several domains, including image classification tasks. Since then, deep learning and CNNs have been used interchangeably, with deep learning now explicitly referring to the use of CNNs with several layers.

Before deep learning, the limitations of neural networks were well-known. For example, they are not explainable, slow to train and require large amounts of training samples. The evolution of CNNs has also led to complex models with hundreds of layers and billions of parameters. Proper tuning of these parameters is crucial for optimal learning performance. This makes CNNs' training not only challenging but also something closer to an art than a scientific or technical discipline [13]. In addition, the network error is minimised using an iterative method called stochastic gradient descent. This method aims to adjust the weights so that the network error reaches its global minimum. Most of the time, however, it sticks at its local minimum. Backpropagation also uses chain rules of derivatives, while not all properties in the world can be differentiated [13]. Some additional issues that have emerged with deeper CNNs include vanishing gradient, slow convergence rate and high computing cost [14]. These limitations remain in the era of deep learning.

The main message from the last five years is to use effective deep architectures to represent real-world patterns, but this does not rely specifically on the use of neural networks. Therefore, efforts have been made to investigate non-differentiable-style networks, which are trained without gradient descent or backpropagation. Among these networks are deep forest [13], PCANet [15], scattering network [16] and extreme learning machine networks [14, 17, 18, 19]. The PCANet architecture, with basic processing steps, including two cascaded principal component analysis (PCA) layers, binary hashing and histogramming, has reached state-of-the-art performance on many databases, including

MNIST, extended Yale B, AR and FERET, confirming its generalisation capability across different tasks. As a result of PCANet's success, a variety of similar strategies have been developed. DCT-Net [20], LBP-Net [21] and ICANet [22], for instance, shared the same architecture as PCANet but generated different features using different filter types. PCANet-II [23] replaced the original PCANet's histogram pooling with second-order pooling to reduce the number of features. PCANet+ [24] presented a filter ensemble learning approach to adhere to CNN architectures. While these networks have not provided high classification accuracy on more challenging datasets such as CIFAR-10, they have shown that network training without gradient descent or backpropagation is possible. In addition, these networks offer less training time than gradient-based style networks since they are trained sequentially in a single pass without requiring iterations to update the weights. Moreover, the vanishing or exploding gradient problem is not an issue since these networks are trained without gradient descent. However, further study in this area is required, which is the purpose of this research. The primary objective is to determine ways to build more complicated structures without using backpropagation for classification tasks.

1.2 Challenges

The main objective of this research is to investigate deep, non-differentiable-style networks that are trained without backpropagation for classification tasks. In this research, the PCANet structure [15] serves as the baseline for designing our non-differentiable networks. Training networks without gradient descent or backpropagation is not a trivial task, and in the following, we highlight some challenges we encountered while working with non-differentiable networks.

The first challenge was to reduce the number of features and, therefore, memory consumption. The current works suffer from the feature explosion problem. Taking PCANet [15] as an example, images are convolved with filters per channel; as each filter works on a single channel to create a new grayscale image, the number of channels at the next layer equals channels $\times$ filters and grows rapidly. Local histogramming in PCANet is another factor contributing to the feature explosion problem. While several studies (e.g., [24] and [23]) have attempted to tackle this issue, doing so effectively remains an open challenge.

The second challenge was to improve the feature representation process either by changing the filter type or the post-processing procedures. PCANet [15], for example, used PCA-based filters, whereas the unsupervised PCA method is applied to local patches extracted from images. Other studies used different filter types to generate different features. DCTNet [20], CCANet [25] and CFR-ELM [26] are all examples of networks that create filter banks using unsupervised approaches. The LDANet [15], DCCNet [27], OSNet [28] and CKNet [29] networks are examples of those that use supervised approaches to produce their filter banks. DFSNet [30] is a good example of a network that uses semi-supervised filters. Even though several of these studies used more complicated procedures to create their filters, the networks' accuracy results were similar to those of the original PCANet. The primary difficulty is in producing filter banks that improve data representations while still offering noticeable improvements over the baseline PCANet design. To further enhance the representations, post-processing the feature maps after convolving them with the filter banks is required. All variants of PCANet involve a binarization step that reduces the filter responses to zeros and ones using the signum function, resulting in a big loss of information. Among the challenges was finding a suitable alternative to the signum function that would maintain essential information and provide better representations.

The final challenge was to considerably increase the network depth while maintaining a good level of accuracy. Existing non-differentiable networks do not have more than three layers, despite the importance of network depth in achieving good classification results. This limitation, we believe, is due to the subsampling nature of these shallow-depth networks. As we add more layers, we lose more information about the original data, and unlike gradient-based networks, we have no mechanism to update the network parameters concerning the network's error. Hence, the network's errors are propagated as additional layers are added.

1.3 Objectives and Research Questions

This research again focuses on the classification tasks. Our main goal is to design deep, non-differentiable networks that can be trained in a single pass without backpropagation. Specifically, the layers are added sequentially and trained without iterations or extensive parameter updates. More precisely, this research aims to answer the following questions based on the main challenges described in Section 1.2.

- What modifications may be made to the current PCANet model to reduce the dimensionality of the data and, hence, the computational cost without compromising classification accuracy?
- What type of convolutional filters are to be used to extract good representations that lead to a noticeable gain in the model's accuracy?
- How can we simultaneously increase the network's depth and ensure that classification errors are corrected while traversing it?

1.4 Contributions

This thesis makes the following contributions to the area of non-differentiable networks by addressing the research questions listed in Section 1.3:

- In Chapter 3, the Multi-Layer PCANet architecture is presented as a hybrid model that combines filter ensemble learning from [24] with second-order pooling from [23]. In contrast to other variants of PCANet, which employ the Heaviside activation function, we post-process the data using the z-score method, which maintains more information and provides better representations. In addition, we use a late fusion technique to reduce the dimensionality of the data further. Due to these modifications, we were able to enhance the original PCANet's performance while increasing the number of layers to a maximum of nine. Our experiments in the chapter demonstrated that the Multi-Layer PCANet outperformed PCANet not only in terms of accuracy but also in the number of floating operations (FLOPs) required for the training. The model also showed a significant improvement in the number of FLOPs compared to many gradient-based models.
- In Chapter 4, we introduce the Stacked-LDA network, which replaces the unsupervised PCA filters in the Multi-Layer network with supervised Stacked-LDA filters. The computation of these filters involves selecting a subset of image-based patches and then applying a linear discriminant analysis classifier to train the selected patches with their classes. The technique searches iteratively for patches with separable classes. When such

classes are found, their weights are accumulated and used as Stacked-LDA filters. In all classification tasks, the Stacked-LDA network outperformed the Multi-Layer PCANet structure, demonstrating the significance of the filter type on the model's accuracy. In addition, our experimental results demonstrated that using only one layer of the Stacked-LDA model produced a comparable performance to that of using multiple layers of the Multi-Layer PCANet, but with a significant reduction of around 50% in the number of trainable parameters.

- In Chapter 5, more filter types are introduced to generate different features. Clustering networks, for instance, use spectral clustering to divide each class into sub-classes, which are then used to train an LDA on batch-based image data to produce the network filter banks. The SLE network generates non-linear filters using supervised laplacian eigenmaps and an extreme learning machine. The supervised laplacian eigenmaps method is based on the principle of employing separation criteria that maximises the distance between samples of different classes while minimising it within a class. As an alternative to the SLE network, the HSIC network generates its filter banks based on the information criterion (Hilbert–Schmidt independence criterion), which reduces the dependency on the inputs and increases it on the labels. In addition, the S-ELM network employs the supervised extreme learning machine auto-encoder to produce filter banks, which in turn give a compact representation that is informed by both input and target data. The experiments presented in Chapter 5 demonstrated that the Stacked-LDA (Chapter 4) provided the highest accuracy among the networks under consideration, despite the robust competition from the S-ELM network. In addition, regarding computational complexity, our experiments demonstrated that the Stacked-LDA filters showed better computational efficiency compared to several other proposed filters, including HSIC, SLE and clustering network filters.

- In Chapter 6, we present a novel deep residual compensation convolutional network (ResCNet). This network's structure comprises multiple convolutional layers, each followed by some post-processing steps and a classifier. Each layer is trained with classes generated using the residual information of all of its preceding layers. In terms of accuracy, this network structure outperformed all previous non-differentiable networks. Unlike neural networks, the ResCNet structure does not need to be known in advance. Moreover, to the best of our knowledge, ResCNet is the first non-differentiable network to be implemented with over 950 layers and achieve outstanding performance. This performance was comparable to standard gradient-based models such as Network in network [31], stochastic pooling [8] and several residual networks. In addition, regarding the number of FLOPs required for training, ResCNet used fewer FLOPs than some residual networks. While ResCNet is implemented with hundreds of layers, these layers are trained sequentially, one layer at a time, without any iterations or extensive updates to the network's parameters. Additionally, we trained each layer with a relatively small number of filters, which never exceeded 60. Therefore, although ResCNet comprises hundreds of layers, it remains computationally efficient and does not suffer from the same complexity issues that arise with deep gradient-based models.

1.5 Outline of the Thesis

This thesis is organised as follows:

- In Chapter 2, gradient-based and non-gradient-based works are discussed.
- In Chapter 3, the Multi-Layer network is presented, and its performance is compared with that of the original PCANet.
- The primary purpose of Chapters 4 and 5 is to generate features using different convolution filters.
- Chapter 6 describes the deep residual compensation convolutional network, which increases the network's depth to more than 950 layers while simultaneously correcting classification errors.
- The works discussed in this thesis are summarised in Chapter 7, and some directions for future research are presented.

In addition, it is worth noting that the work presented in Chapter 3 has been published in [1], whereas the work described in Chapter 6 has been submitted and accepted for presentation at the International Joint Conference on Neural Networks (IJCNN 2023). Additionally, the Stacked-LDA filters discussed in Chapter 4 were submitted to IJCNN2023 as supplementary material.

Chapter 2

Background and Related Works

2.1 Introduction

The artificial neural network is a sub-area of machine learning that has produced deep learning. Since its inception, deep learning has led to ever greater disruption and achieved extraordinary success in practically all areas of applications. In deep learning, a class of machine learning that was introduced in 2006, ’learning’ refers to the procedure of estimating a model’s parameters to perform a specific task. It consists of many layers that separate the input and the output, providing multiple steps of nonlinear data processing units with hierarchical constructions that are used to learn features and classify patterns. In other words, deep learning is a universal approach composed of multiple layers that learn representations from data and is not task-specific [32].

The key difference between deep learning and traditional methods is in features are derived from the data. Traditional machine learning methods rely on hand-crafted features using feature extraction methods such as scale-invariant feature transform (SIFT), the speed-up robust feature (SURF) and many more. The extracted features are then classified using a classifier such as the support vector machine (SVM). Deep learning, on the other hand, automatically learns functions from the raw data using a hierarchical, multi-level structure. This makes deep learning a hot topic for many researchers.

Image classification is one area in which deep learning has shown effectiveness. This deep learning success story began in 2012 when AlexNet [3], with an eight-layer architecture, obtained a top-5 error rate of 15.3% on the ImageNet database. AlexNet architecture, which consists of five convolutional layers, three pooling layers and three fully connected layers, has served as the basis for the development of a wide variety of other networks. For example, VGGNet [4] is a classic deep CNN with the same design as AlexNet. However, unlike AlexNet, and with the help of small, 3×3 receptive fields, VGGNet increases the network depth to 19 layers. VGGNet placed second in the 2014 ImageNet challenge, with a top-5 error rate of 7.3%, which was reduced to 6.8% after the submission. Network in network (NiN) [31] is a CNN architecture that set out to modify the design of AlexNet and VGGNet by replacing their fully connected layers with a global average pooling. This method has proven to be effective in reducing the total number of parameters, and it has since been adopted by many other well-known networks, such as ResNet [5]. NiN also incorporated fully connected networks such as the multi-layer perceptron to be utilised between layers to extract the feature maps of the convolutional layers. This network was tested on several benchmarks and set new records in the CIFAR-10 and CIFAR-100 databases. ResNet [5] is an example of CNN architecture that follows the same design as VGGNet and AlexNet but replaces the fully connected layers with global average pooling, as in NiN. ResNet added short connections between at least two convolutional layers and increased the network depth to over 100 layers. This network placed first in the ILSVRC ImageNet challenge in 2015 with an error rate of 3.75% and won first place in the COCO database competition. Other examples of deep convolutional neural networks include Maxout [7], ALL-CNN [9], stochastic pooling [8], binary network [10], DenseNet [12], FractalNet [11], Fractional Net [6] and many others. Some of these networks have modified one aspect of the VGGNet architecture while still adhering to the network's fundamental design. For instance, fractional [6] and stochastic pooling [8] networks work with the pooling layers, Maxout [7] uses a new activation function, and EffNet-L2 [33] improves the optimisation loss function of the network. Other networks have changed the network's basic design, such as Fractal Net [11] and DenseNet [12].

With the development of deeper networks, CNNs have shown success in classification tasks and reached human-level performance in many benchmarks. Despite their success, several problems have emerged, such as vanishing gradient descent, in which the gradients become very small or vanish by the time they reach the network end. CNNs also involve a high computational cost, which negatively affects training and inference times. Other issues associated with CNNs include a slow convergence rate, intensive human intervention in the training phase and local minima [14]. Consequently, considerable efforts have been devoted to examining different strategies that might extract deep feature representations while simultaneously reducing computational costs. Examples of such networks include deep forest [13], deep SVM [34], ScatNet [16], PCANet [15], and extreme learning machines (ELM) [14, 17, 18, 19]. All of these networks are non-differentiable models that have been trained without gradient descent or backpropagation. PCANet [15] is a well-known network that facilitates faster training convergence and does not need iterations to adjust the weights of the hidden layers. Instead, network training is accomplished by applying unsupervised principal component analysis (PCA) to local image patches. PCANet architecture is simple and consists only of basic processing steps, including two cascaded PCA stages, binary hashing and histogramming. PCANet, with its simple structure, has achieved state-of-the-art performance on many databases, including MNIST, extended Yale B, AR and FERET, confirming its generalisation capability across various tasks. The success of this network has resulted in the development of a family of similar networks. For instance, while maintaining the same structure, DCT-Net [20], LBP-Net [21], and ICANet [22] generated different features by modifying the filters used by PCANet [15]. PCANet-II [23] shared the same design and filters as the original PCANet; however, it used second-order pooling instead of histogramming to reduce the number of features. PCANet+ [24] changed the topology of PCANet filters by proposing PCA filters ensemble learning that constrains to CNNs' structure. In addition, multiple kernel approaches have been investigated in an effort to learn a non-linear representation of the PCA filters (see, for example, [35]). Despite not achieving state-of-the-art performance on more challenging datasets such as CIFAR-10 or Image-Net, these networks have demonstrated the importance of convolution layers for obtaining good performance. However, there is a need for additional research in this area, and the objective of this study is to satisfy that need. The research was carried out to investigate the design of PCANet-like networks, which are optimised for classification tasks and are trained without backpropagation or gradient descent.

This chapter is organised as follows. The whole chapter is divided into two main sub-sections. The first one (Section 2.2) describes some examples of gradient-based models and is divided into the following two subsections: 2.2.1 explains some CNNs such as LeNet-5 [36], AlexNet [3], VGGNet [4], ResNet [5], FractalNet [11], DenseNet [12], ALL-CNN [9], stochastic pooling [8] and Maxout [7], while 2.2.2 details some deep probabilistic models such as the Boltzmann machine [37], deep belief and deep direct networks [38]. The second section in this chapter (Section 2.3) reviews some models designed without relying on gradient descent or backpropagation. Among these models are deep-SVM [34], deep forest [13], ScatNet [16], PCANet [15], PCANet+ [24] and

PCANet-II [23].

2.2 Gradient-Based Networks

In this section, we begin by discussing different basic CNN architectures, including AlexNet [3], VGGNet [4] and ResNet [5]. We then describe gradient-based probabilistic models, such as deep belief nets [39] and deep Boltzmann machines [38].

2.2.1 Convolutional Neural Networks

This section provides an overview of different fundamental CNN structures. All these networks use a mathematical operation known as convolution to capture the different local patterns of images. The convolutional layers in CNNs use shared-weight convolution filters that slide along input features and create translation-equivariant feature maps.

2.2.1.1 LeNet-5 [36]

LeNet-5, which was introduced in [36] in 1998, is one of the earliest deep learning networks designed to recognise hand digits from the MNIST database. The structure of the network is simple and is described in figure 2.1. The architecture comprises seven layers, including three convolutional layers, two down-sampling layers, and two fully connected layers. It derives its name from the fact that five of its layers consist of trainable parameters.

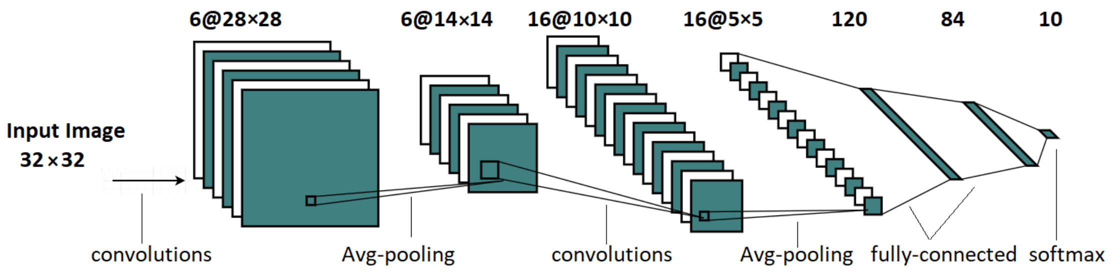


FIGURE 2.1: LeNet-5 architecture: a five-layer convolutional neural network for image classification on the MNIST database.

Table 2.1 provides more details about the network structure. The network's first layer, the input layer, is designed to receive 32×32 grayscale images. In article [36], the MNIST images of 28×28 size were zero-padded and normalised to values of between -0.1 and 1.175 to ensure that the image patches had a mean of about 0 and a standard deviation of roughly 1. The 32×32-pixel images were then passed to the first convolutional layer (C1), which used a 5×5-pixel kernel to generate six feature maps. The downsampling layer (S2) reduced the size of the dimensions by a factor of two, bringing them down to 14×14. In the paper, the downsampling layer averaged the pixel values inside a 2×2 window, multiplied the average by a coefficient and added a bias term before passing them to the activation function. The second convolutional layer (C3) used a kernel size of 5×5 to generate 16 feature maps, which were then pooled by the second downsampling layer with a filter size of 2×2 and a stride of 2, to get 16 feature maps of size 5×5 each. The final convolutional layer (C5) employed a 5×5 filter size to generate 120 feature maps, which were then flattened and sent to a fully connected layer containing 84 neurons. The last layer of the structure was the softmax layer, which consisted of 10 neurons representing the MNIST digits (0–9). The network was trained to minimise the mean square error loss using around $60,000$ parameters. The testing accuracy reported in the paper was 99.05% on the MNIST database.

TABLE 2.1: LeNet-5 architecture.

Layer name	**Layer type**	**Filter size**	**Image size**	**Activation**
Input	Image	–	$32 \times 32 \times 1$	–
C1	Convolution	$5 \times 5 \times 1 \times 6$	$28 \times 28 \times 6$	Tanh
S2	Sub-sampling	2×2, stride $= 2$	$14 \times 14 \times 6$	Sigmoid
C3	Convolution	$5 \times 5 \times 6 \times 16$	$10 \times 10 \times 16$	Tanh
S4	Sub-sampling	2×2, stride $= 2$	$5 \times 5 \times 16$	Sigmoid
C5	Convolution	$5 \times 5 \times 16 \times 120$	$1 \times 1 \times 120$	Tanh
F6	fully connected	–	84	Tanh
Output	fully connected	–	10	Softmax

2.2.1.2 AlexNet [3]

AlexNet, described in [3], has a structure similar to LeNet-5 but is much deeper. The network was trained to classify the 1.2 million high-resolution images from the ImageNet LSVRC-2010 competition into 1000 distinct categories. On the test data, AlexNet obtained top-1 and top-5 error rates of 37.5% and 17.0%, respectively, which was an improvement of more than 8% over the previous state-of-the-art results. The network also won the ILSVRC-2012 challenge, with a top-5 test error rate of 15.3%.

Figure 2.2 describes the network architecture. The architecture consists of eight layers; five of these are convolutional layers, while the remaining three are fully connected. Some of the convolutional layers are followed by max-pooling layers. The first convolutional layer applies 96 kernels of size $11 \times 11 \times 3$ with a stride of 4 pixels to the $227 \times 227 \times 3$ input images. The second employs 256 kernels of size $5 \times 5 \times 96$ to convolve the normalised and pooled output of the first convolutional layer. The third, fourth and fifth are linked with no pooling or normalising layers in between. The third layer employs 384 $3 \times 3 \times 256$ filters to convolve the outputs of the second convolutional layer after normalising and pooling them. While the fourth layer uses 384 $3 \times 3 \times 384$ filters, the fifth layer uses only 256. Moreover, each fully connected layer has a total of 4096 neurons. Table 2.2 provides more details about the network architecture, including the activation functions used at each layer.

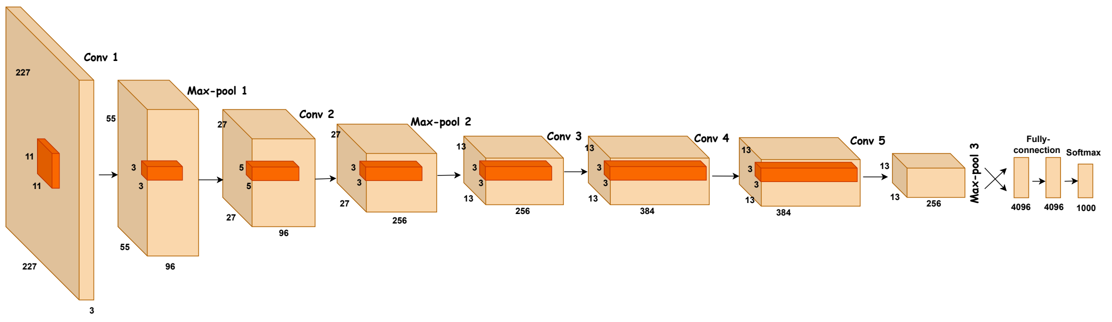


FIGURE 2.2: AlexNet architecture: eight-Layer CNN for large-scale image classification. More details about the network structure are described in Table 2.2.

TABLE 2.2: Explanation of the AlexNet architecture in detail.

Layer	Filter size	Stride	Padding	Image size	Activation
Input	–	–	–	$227 \times 227 \times 3$	–
Conv 1	$11 \times 11 \times 3 \times 96$	4	–	$55 \times 55 \times 96$	ReLU
Max-pool 1	3×3	2	–	$27 \times 27 \times 96$	–
Conv 2	$5 \times 5 \times 96 \times 256$	1	2	$27 \times 27 \times 256$	ReLU
Max-pool 2	3×3	2	–	$13 \times 13 \times 256$	–
Conv 3	$3 \times 3 \times 256 \times 384$	1	1	$13 \times 13 \times 384$	ReLU
Conv 4	$3 \times 3 \times 384 \times 384$	1	1	$13 \times 13 \times 384$	ReLU
Conv 5	$3 \times 3 \times 384 \times 256$	1	1	$13 \times 13 \times 256$	ReLU
Max-pool 3	3×3	2	–	$6 \times 6 \times 256$	–
fully connected 1	–	–	–	4069	ReLU
fully connected 1	–	–	–	4069	ReLU
Output	–	–	–	1000	Softmax

In [3], AlexNet was trained with 60 million parameters on GTX 580 3GB GPUs for six days. So that the network would be trained faster, the authors used the rectified linear unit (ReLU), defined as $\max(0, x)$, instead of the tanh or the sigmoid activation functions. The ReLU activation was used after each convolutional layer, as shown in Table 2.2. In addition, two data augmentation methods and the dropout technique were used to address the overfitting problem. Translation and horizontal flipping were the first forms of data augmentation adopted. The original training images of size $256 \times 256 \times 3$ and their horizontally flipped copies were randomly cropped to size $227 \times 227 \times 3$. At the testing time, the testing images were cropped into ten patches of size 227×227, five representing the four corners and the centre of the original images and the remaining representing the four corners and the centre of the horizontally flipped images. The prediction was then made as the average probabilities of these ten images. The second type of data augmentation used in the article was the PCA-based transformation of RGB channel intensities. With a probability of 50%, the dropout strategy deactivated specific neurons during training. This method enabled neurons to learn representative features independent of the values of neighbouring neurons.

Despite its success, AlexNet suffers from a hyperparameters problem. The variability in the configuration that is external to the model and cannot be inferred from the data is referred to as the hyperparameters [40]. There are two types of hyperparameter: those responsible for determining the network's structure and those associated with its training. Among the hyperparameters that may have an effect on the structure of the network are the kernel size, stride value, padding value, number of hidden layers and activation functions that are utilised. The learning rate, momentum, number of epochs and patch size are examples of the hyperparameters defined during the network's training [40]. In Alex-Net, the number of user-defined hyper-parameters is relatively high; this is what we refer to as the hyperparameters problem.

2.2.1.3 VGGNet [4]

The main contribution made by VGGNet [4], where VGG stands for the visual geometry group, was to highlight the importance of network depth on the accuracy of CNNs. Utilising a very small convolutional filter size of 3×3, the authors of [4] were able to increase the network depth to 19 layers. It is essential to mention that GoogLeNet [41] is also a deep CNN that consists of 22 layers. It uses very small convolutional filter sizes, much as VGGNet does; however, its structure is more complicated than VGGNet.

The authors of the VGGNet paper used six configurations, all of which are described in Table 2.3, whereby each column represents a single structure of the VGGNet. The configurations follow the same design but differ in the number of layers, which range from 11 to 19 in configurations A through E. The input size is fixed to $224 \times 224 \times 3$ for all VGGNet configurations. Every convolutional layer uses relatively small receptive fields of size 3×3 with a stride and padding of 1 pixel each. In some configurations, 1×1 convolution filters are also employed to add non-linearity without changing the layer's receptive field. In every configuration, five max-pooling layers are used; however, not every convolutional layer is followed by a max pooling layer. Every max pooling layer uses a 2×2 window with a two-pixel stride. The number of filters in each network starts with 64 and increases by a factor of 2 after each max-pooling layer until it reaches 512. All VGGNet configurations use three fully connected layers, with 4096 neurons in the first two and 1000 neurons in the third. Similar to AlexNet, ReLU is the activation function utilised between the layers. Table 2.4 displays the total number of parameters for each VGGNet configuration.

The main difference between VGGNet [4] and AlexNet [3] is that the former has more layers. In contrast to AlexNet, which employs a single huge respective field of size 11×11, VGGNet consists of multiple layers, each of which uses a very small respective field of size 3×3. In addition to increasing nonlinearity and producing more discriminative data, using small receptive fields reduces the number of parameters. For instance, a single layer with a filter size of 5×5 can be represented by two layers of 3×3 convolutional filters. The number of parameters required for the 5×5 filter is 25. Using two layers of 3×3 filters, however, generates $(3 \times 3) + (3 \times 3) = 18$ parameters. The concept of small-size convolution filters allows VGGNet to have a larger number of layers, resulting in a performance gain.

In [4], to train VGGNet, the input to each architecture was cropped to a fixed size of $224 \times 224 \times 3$ using rescaled images. The paper referred to the scale parameter for the training images as $\mathcal{S}$ and the one for the test set as $\mathcal{Q}$. It investigated two types of scaling parameters, namely the single-scale parameter and the multi-scale parameter. In the single-scale approach, images were scaled using a fixed value, whereas in the multi-scale approach, each image was scaled by randomly sampling $\mathcal{S}$ from a given range $[\mathcal{S}_{min}, \mathcal{S}_{max}]$. If multiple scales were used during testing, the class scores were averaged.

TABLE 2.3: VGG Configurations. Each column represents an architecture.

Convolution network configurations					
A	**A-LRN**	**B**	**C**	**D**	**E**
11 weights layers	11 weights layers	13 weights layers	16 weights layers	16 weights layers	19 weights layers
Input (224 $\times$ 224 **RGB image)**					
Conv3-64	Conv3-64LRN	Conv3-64 Conv3-64	Conv3-64 Conv3-64	Conv3-64 Conv3-64	Conv3-64 Conv3-64
MAX-pooling					
Conv3-128	Conv3-128	Conv3-128 Conv3-128	Conv3-128 Conv3-128	Conv3-128 Conv3-128	Conv3-128 Conv3-128
MAX-pooling					
Conv3-256 Conv3-256	Conv3-256 Conv3-256	Conv3-256 Conv3-256	Conv3-256 Conv3-256 conv1-256	Conv3-256 Conv3-256 Conv3-256	Conv3-256 Conv3-256 Conv3-256 Conv3-256
MAX-pooling					
Conv3-512 Conv3-512	Conv3-512 Conv3-512	Conv3-512 Conv3-512	Conv3-512 Conv3-512 conv1-512	Conv3-512 Conv3-512 Conv3-512	Conv3-512 Conv3-512 Conv3-512 Conv3-512
MAX-pooling					
Conv3-512 Conv3-512	Conv3-512 Conv3-512	Conv3-512 Conv3-512	Conv3-512 Conv3-512 conv1-512	Conv3-512 Conv3-512 Conv3-512	Conv3-512 Conv3-512 Conv3-512 Conv3-512
MAX-pooling					
FC-4096					
FC-4096					
FC-1000					
Softmax					

TABLE 2.4: The number of parameters in millions in each configuration of VGGNet.

Network	A/A-LRN	B	C	D	E
Number of parameters	133	133	134	138	144

Table 2.5 displays the top-1 and top-5 error rates of the six configurations using the single-scale method, and Table 2.6 displays the performance of VGGNets using the multi-scale method. These results are based on the submission by the authors of [4] to ImageNet Challenge 2014. As indicated in Table 2.5, configuration E provided the best performance for VGG on a single test scale, with a top-1 error of 25.5% and a top-5 error of 8.0%. Configuration D, with 16 layers, achieved comparable performance. According to Table 2.6, VGGNet obtained a top-1 error of 24.8% and a top-5 error of 7.5% in multiple test scales when using configurations E and D with 16 and 19 layers. In addition, VGGNet placed second in the 2014 ImageNet competition with a top-5 error rate of 7.3%, which was obtained by training seven networks and averaging their class scores. The error rate, as indicated in the paper, was decreased to 6.8% after the submission by ensembling two architectures, D and E.

TABLE 2.5: The performance of VGG configurations using the single-scale method. These results were reported in [4].

Network	**Smallest image side**		**Top-1 error (%)**	**Top-5 error (%)**
	Train(S)	**Test(Q)**		
A	256	256	29.6	10.4
A-LRN	256	256	29.7	10.5
B	256	256	28.7	9.9
C	256	256	28.1	9.4
	384	384	28.1	9.3
	[256; 512]	384	27.3	8.8
D	256	256	27	8.8
	384	384	26.8	8.7
	[256; 512]	384	**25.6**	**8.1**
E	256	256	27.3	9
	384	384	26.9	8.7
	[256; 512]	384	**25.5**	**8**

TABLE 2.6: The performance of VGG configurations using the multi-scale method. These results were reported in [4].

Network	Smallest image side		Top-1 error (%)	Top-5 error (%)
	Train ($\mathcal{S}$)	Test ($\mathcal{Q}$)		
B	256	224,256,288	28.2	9.6
C	256	224,256,288	27.7	9.2
	384	352,384,416	27.8	9.2
	[256; 512]	256,384,512	26.3	8.2
D	256	224,256,288	26.6	8.6
	384	352,384,416	26.5	8.6
	[256; 512]	256,384,512	**24.8**	**7.5**
E	256	224,256,288	26.9	8.7
	384	352,384,416	26.7	8.6
	[256; 512]	256,384,512	**24.8**	**7.5**

2.2.1.4 Network in Network (NiN) [31]

LeNet-5 [36], AlexNet [3] and VGGNet [4] all share the same basic structure, with multiple convolutional layers, some of which are followed by a max pooling layer, and fully connected layers plugged in at the end of the network. The difference between VGGNet and ALexNet is that VGGNet uses small convolution filters of size 3×3 to increase the CNN's depth. In general, this design faces two primary challenges. First, the fully connected layers use an extremely large number of parameters. Using VGGNet as an example (configurations in Table 2.3), in order to add a fully connected layer with 4069 dimensions to the last convolutional layer of size $7 \times 7 \times 512$, we need about $7 \times 7 \times 512 \times 4096$ parameters, which amounts to more than 102 million. The second issue is that adding fully connected layers earlier in the network to increase non-linearity is impossible. Network in network (NiN) [31] provides an alternative solution capable of resolving both issues by using a straightforward technique. Instead of using linear filters followed by non-linear activation functions, as in conventional CNNs, NiN uses a micro-network comprised of multiple non-linear layers to generate feature maps, which results in more abstract features for local patches. By using global average pooling instead of fully connected layers, the network is able to achieve better interpretable results while also eliminating the need to optimise additional parameters.

Figure 2.3 depicts the primary architecture of NiN, which consists of stacked multi-layer perceptron convolutional layers (MLPconv layers) followed by global average pooling. The difference between the standard convolutional layers and the MLPconv layers is explained in Figure 2.4, in which a shared multi-layer perceptron is applied by the latter to different image patches. Similar to CNN, the feature maps are generated by sliding the multi-layer perceptron over the input before being passed to the next layer. The MLPconv layer performs the following calculation:

$$\begin{aligned} f^1_{i,j,k_1} &= \max({w^1_{k_1}}^T x_{i,j} + b_{k_1}, 0) \\ &\vdots \\ f^n_{i,j,k_n} &= \max({w^n_{k_n}}^T f^{n-1}_{i,j} + b_{k_n}, 0), \end{aligned} \tag{2.1}$$

where n denotes the number of layers in the multi-layer perceptron, (i, j) represents the pixel index of the feature map, k is the index of the feature map channel, $x_{i,j}$ is the initial input, and $f_{i,j}$ represents the output of the previous layer in the multi-layer perceptron.

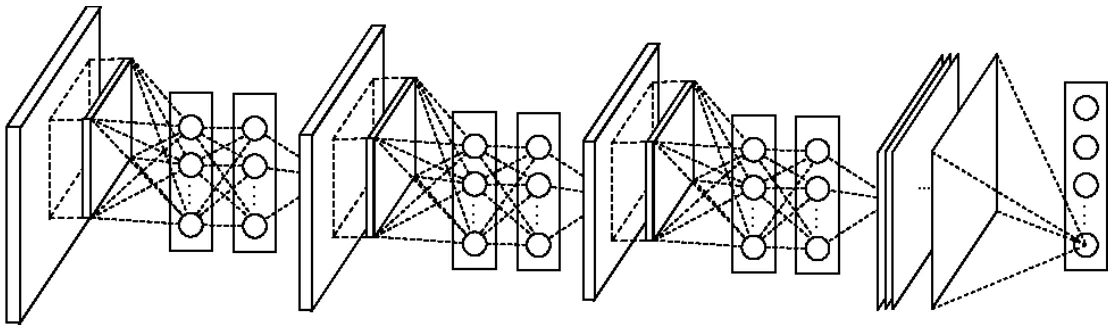


FIGURE 2.3: The main architecture of Network in Network: a CNN model with three mlpconv layers and one global average pooling layer. Captured from [31].

The global average pooling plug-in at the end of the network works by averaging each feature map and feeding the resulting vector straight into the softmax layer. One benefit of employing global average pooling is that it forces correspondences between feature maps and categories. Additionally, there are no parameters to optimise in global average pooling; therefore, there is no overfitting at this layer. Moreover, global average pooling works by adding all spatial information in each channel, making the output more invariant to spatial translations.

In [31], NiN was evaluated across four benchmarks, CIFAR-10, CIFAR-100, MNIST and the Street View House Numbers database (SVHN). The network presented state-of-the-art performance on the CIFAR-10 and CIFAR-100 databases, with a CIFAR-10 accuracy of 89.59% and a CIFAR-100 accuracy of 64.32%. In

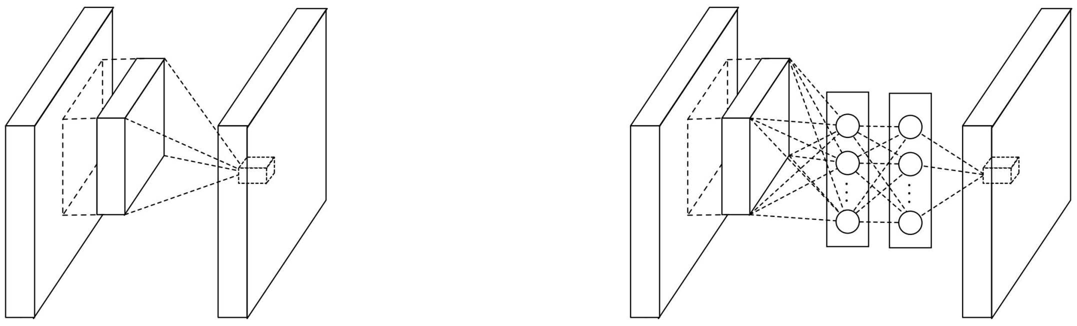


FIGURE 2.4: Comparison between a) linear convolution layer and b) mlpconv layer. The linear convolution layer slides a linear filter along the image, while the mlpconv layer slides a multi-layer perceptron over the image.
Captured from [31].

MNIST (99.52%) and SVHN (97.65%), the network achieved excellent but not state-of-the-art performance.

2.2.1.5 Maxout Network [7]

The Maxout network [7] is nothing more than a feed-forward architecture, such as a multi-layer perceptron or deep convolutional neural network, although it uses a new activation function that is referred to as the Maxout unit. Figure 2.5 provides a simple example of the Maxout unit. As shown in the figure, the Maxout unit finds the maximum of weighted sum units. Given $x \in \mathbb{R}^d$, which represents the input or the output of any hidden layer, the mathematical expression of the Maxout unit can be described as follows:

$$\begin{aligned} h(x) &= \max(z_1, z_2, \dots, z_n) \\ &= \max(x^T W_1 + b_1, x^T W_2 + b_2, \dots, x^T W_n + b_n), \end{aligned} \tag{2.2}$$

where $[W_1, W_2, \dots, W_n]$ and $[b_1, b_2, \dots, b_n]$ are all trainable parameters. As indicated by equation 2.2, the Maxout unit takes the maximum value among the values from n linear functions. Figure 2.6 illustrates how it can approximate several activation functions, including ReLU, absolute value and quadratic function. The figure shows that the Maxout can represent the ReLU and absolute value activation functions using two linear functions while approximating the quadratic function with four linear functions. In other words, the ReLU function is a special case of the Maxout activation function.

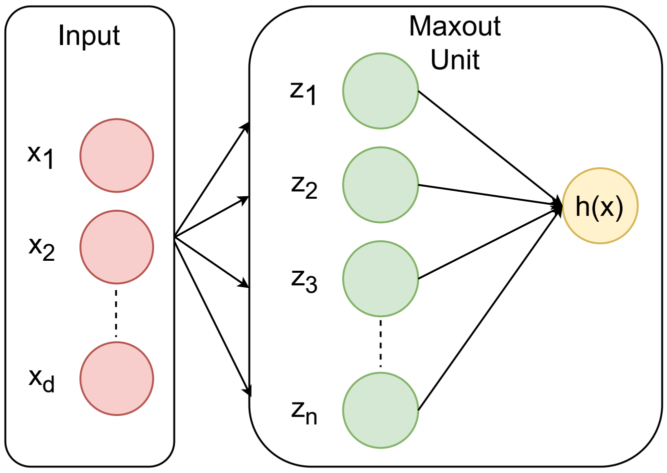


FIGURE 2.5: A single Maxout unit: a linear function that returns the maximum of the inputs (z_1 to z_n in the figure).

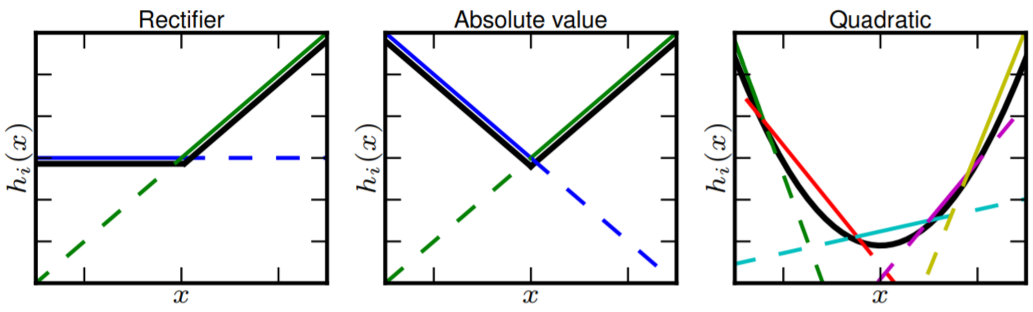


FIGURE 2.6: Graphical depiction of how the Maxout activation function can implement the rectified linear, absolute value rectifier, and approximate the quadratic activation function. Captured from [7].

To generalise the previous idea, in each layer of a Maxout network, instead of computing a single Maxout unit of the input, several Maxout units are computed per layer. Given an input $x \in \mathbb{R}^d$ that needs to be transformed into m dimension outputs $h(x) \in \mathbb{R}^m$ using the Maxout layer, m Maxout units are needed, each consisting of k neurons, where k is a user-defined parameter. Obviously, each Maxout unit must have at least two neurons (that is, $k \geq 2$). In other words, the number of output neurons is first increased by a factor of k, which is at least two, and then the weighted sum of the hidden layer with the new number of neurons ($k \times m$) is computed using the following formula (see Figure 2.7):

$$z_{ij} = x^T W_{ij} + b_{ij}, \tag{2.3}$$

where $W \in \mathbb{R}^{d\times(m\times k)}$ and $b \in \mathbb{R}^{1\times(m\times k)}$ are learnable parameters. In order to get m outputs $h(x) \in \mathbb{R}^m$, the largest value among each of the m units is chosen, as shown below:

$$h_i(x) = \max_{j \in [1,k]} z_{ij},\ i = [1, 2, \ldots, m]. \tag{2.4}$$

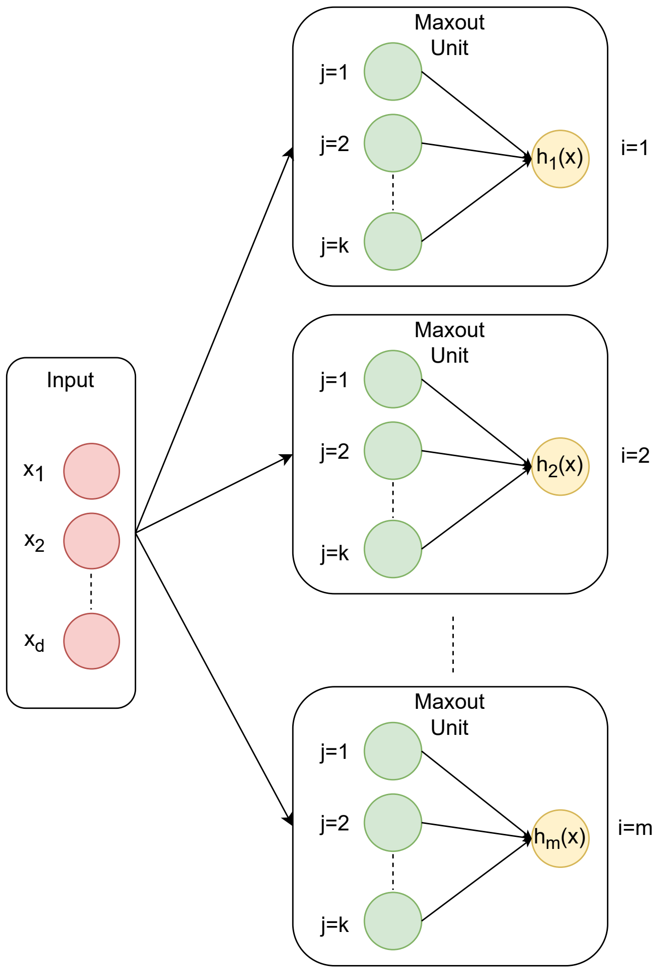


FIGURE 2.7: An example of m Maxout units needed to map $x \in \mathbb{R}^d$ to m dimension outputs $h(x) \in \mathbb{R}^m$. $k \geq 2$ in the figure is a user-defined parameter.

In [7], the performance of Maxout networks was evaluated using four different datasets, CIFAR-10, CIFAR-100, MNIST and SVHN. The network achieved state-of-the-art performance across all datasets, reaching an accuracy of 88.32% on CIFAR-10, 61.43% on CIFAR-100, 99.55% on MNIST and 97.53 on SVHN. The Maxout activation function performed much better than the ReLU activation function when applying the same preprocessing on the CIFAR-10 database. The performance of the Maxout algorithm was also demonstrated to be superior to that of the ReLU algorithm when increasing the network depth using the MNIST database, as shown in Figure 2.8. However, Maxout requires at least twice as many parameters as other activation functions, making the network more susceptible to the overfitting problem.

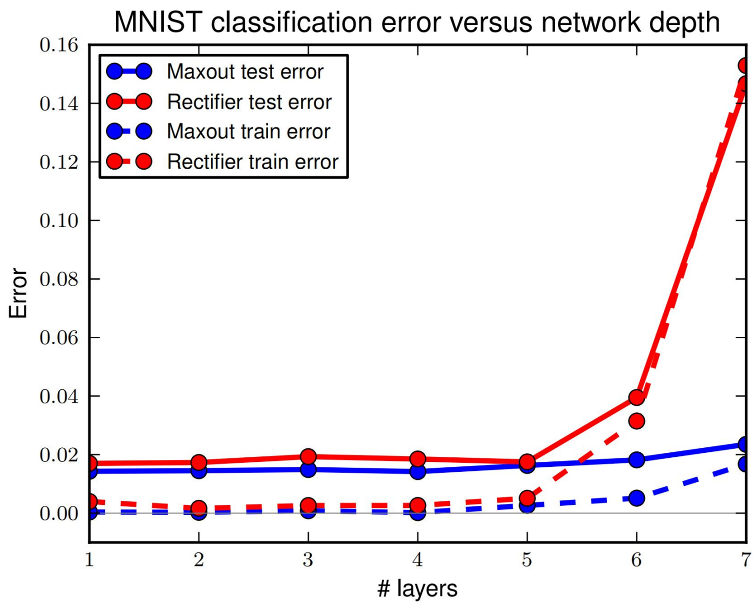


FIGURE 2.8: Comparasion between ReLU and Maxout with multiple layers on the MNIST database. This figure is captured from [7].

2.2.1.6 Stochastic Network [8]

CNNs consist of several convolutional layers, some of which are followed by a pooling layer, which is either max pooling or average pooling. Average pooling considers all elements in a region, regardless of their magnitude [8]. For example, combining average pooling with the ReLU activation function causes down-weighting of strong activations, since many elements with zero value are included in the average calculation. On the other hand, max pooling does not suffer from such a problem but easily overfits the training samples. Stochastic pooling was proposed as a scheme by [8] to take advantage of max pooling while avoiding overfitting. This technique relies on sampling from activations in each region based on multimodal distribution. The larger the value of the activation, the more likely it is to be picked. To be more specific, the technique begins with calculating probabilities p within a window j by normalising the activations inside the window, as follows:

$$p_i = \frac{a_i}{\sum_{k \in R_j} a_k}. \tag{2.5}$$

The pooled activation (a_l) is then selected by sampling from the multinomial distribution based on p in order to select a position l inside the region, as follows:

$$s_j = a_l \text{ where } \sim P(p_1, p_2, \ldots, p_{|R_j|}). \tag{2.6}$$

Figure 2.9 illustrates the concept of stochastic pooling. It indicates that 1.6 has a 40% chance of being selected, whereas 2.4 has a 60% chance; nevertheless, the sampling procedure selects 1.6. In contrast to max pooling, which captures only the highest activation within a region, stochastic pooling guarantees that non-maximal activations, which may contain additional information, are also considered. Through testing, the authors of [8] demonstrated that stochastic pooling introduces noise to the network predictions, resulting in performance degradation. Instead, a probabilistic form of averaging was used, which may be formally represented as follows:

$$s_j = \sum_{i \in R_j} p_i a_i. \tag{2.7}$$

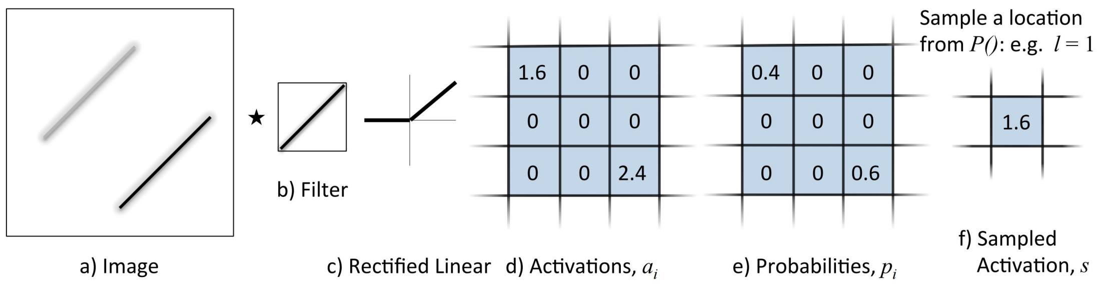


FIGURE 2.9: An example explaining the stochastic pooling idea. The activations inside a region is normalised to obtain the probabilities p_i. The pooled activation (1.6 in this example) is selected using multinomial distribution applied to the computed probabilities p_i. Captured from [8].

The stochastic network in [8] was evaluated across many datasets, including MNIST, CIFAR-10, CIFAR-100 and SVHN, and stochastic pooling was compared with max and average pooling. The model's accuracy on each of the four datasets is summarised in Table 2.7. According to Table 2.7, the performance of stochastic pooling was significantly better than that of maximum or average pooling. In addition, the accuracy attained was state-of-the-art across all databases. Using a small subset of each database, the authors of [8] compared max pooling, average pooling, and stochastic pooling and demonstrated that stochastic pooling outperformed all the other pooling techniques in terms of accuracy.

TABLE 2.7: Stochastic pooling network compared with max and average pooling using four databases: MNIST, CIFAR-10, CIFAR-100 and SVHN.

Method	Test Accuracy%			
	MNIST	**CIFAR-10**	**CIFAR-100**	**SVHN**
Avg pooling	99.17	80.76	52.23	96.28
Max pooling	99.45	80.6	49.1	9619
Stochastic pooling	**99.53**	**84.87**	**57.49**	**97.2**

2.2.1.7 Residual Network [5]

Deep CNNs have led to tremendous success in image classification tasks. With a top-5 test error rate of 15.3%, AlexNet [3] won the ILSVRC-2012 competition. VGG [4], which came second in the 2014 ImageNet competition, with a top-5 error rate of 7.3%, emphasises the importance of network depth for achieving high recognition rates. Despite the importance of network depth, the performance of plain CNNs degrades when the network depth increases. Figure 2.10 depicts a specific illustration of this problem. The figure indicates that the CNN with 56 layers had a higher error rate than the CNN with 20 layers. The authors of [5] stated that this issue could not be a result of overfitting or vanishing gradient descent, since the network had been trained using batch normalisation (BN), which guarantees that the signals travelling in the forward direction have non-zero values. Moreover, according to [5], the fundamental cause of this phenomenon is unclear.

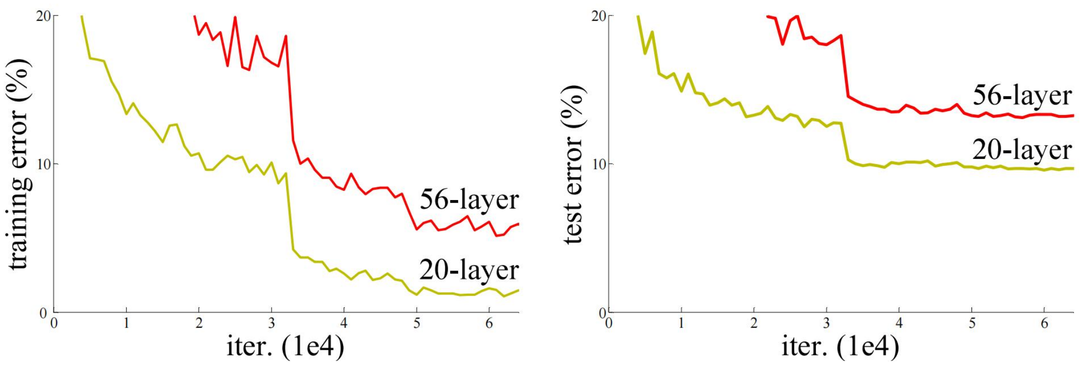


FIGURE 2.10: Training error rate (left) and test error rate (right) for 20-layer and 56-layer plain networks on CIFAR-10. This picture is captured from [5].

The authors of [5] proposed the deep residual network as a solution to the degradation problem. The basic idea of their solution was based on considering the deeper layers as identity mapping, and the other layers were copied from a shallow depth network. In this new construction, the deeper model should not produce a higher error rate than the shallower model. In order to put this idea into practice, the authors of [5] used shortcut connections, shown in Figure 2.11, to skip one or more layers. The outputs of the shortcut connections were added to those of the stacked layers, serving as identity mapping. The residual learning building block shown in Figure 2.11 is technically described as follows:

$$y = \mathcal{F}(x, \{W_i\}) + x, \tag{2.8}$$

where x and y represent the layer's input and output, respectively. The resid-

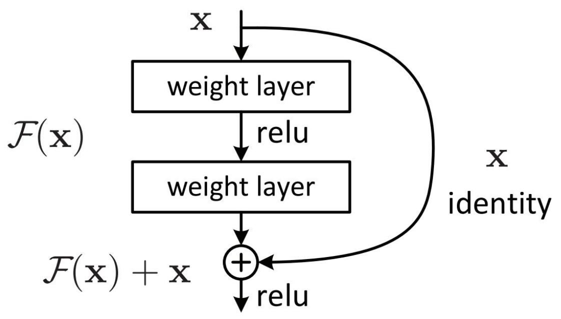

x
weight layer
relu
weight layer
$\mathcal{F}(\mathbf{x})$
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

FIGURE 2.11: Residual learning: building block. This picture is captured from [5].

ual learning function $\mathcal{F}$ in Figure 2.11 consists of two layers and can be expressed as follows:

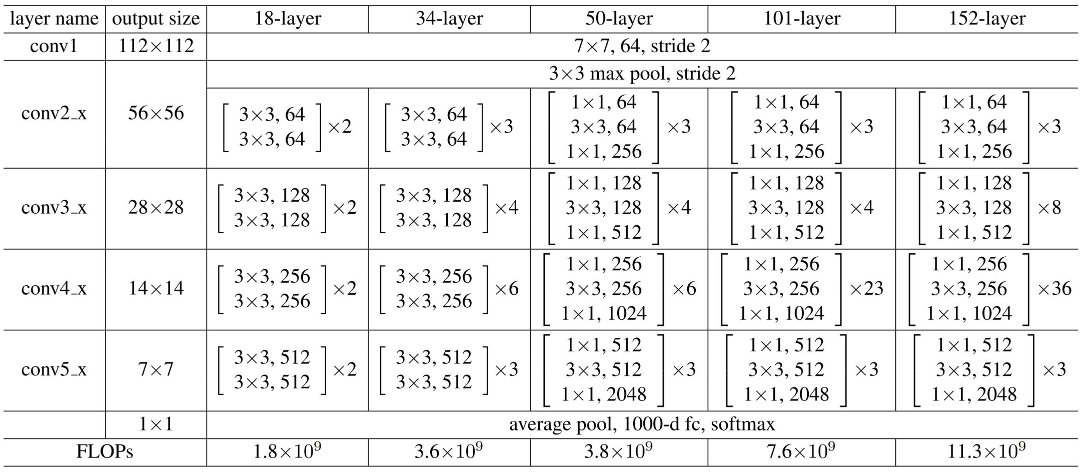
$$\mathcal{F} = W_2\sigma(W_1 x), \tag{2.9}$$

where σ is the ReLU activation function, and W_1 and W_2 are learnable parameters. The dimensions of x and $\mathcal{F}$ should be the same to perform element-wise addition between them. If this is not the case, the following two approaches could be used to increase the dimension of x to match $\mathcal{F}$:

- A: increase the size of x using zero-padding shortcuts; therefore, no additional learning parameters are required.
- B: Apply the following linear projection transformation to x to increase its size:

$$y = \mathcal{F}(x, \{W_i\}) + W_s x. \tag{2.10}$$

The residual network was evaluated in [5] using different architectures on many benchmarks, including ImageNet, CIFAR-10 and the COCO object detection database. Figure 2.12 shows the configurations used for the ImageNet database. All of the networks followed the same design as VGGNet, albeit with the addition of skip connections between two or more layers. Additionally, the fully connected layers were replaced with the global average pooling that was described in [31].



layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix}3\times3, 64\\3\times3, 64\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 64\\3\times3, 64\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix}3\times3, 128\\3\times3, 128\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 128\\3\times3, 128\end{bmatrix}\times4$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times4$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times4$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix}3\times3, 256\\3\times3, 256\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 256\\3\times3, 256\end{bmatrix}\times6$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times6$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times23$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix}3\times3, 512\\3\times3, 512\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 512\\3\times3, 512\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

FIGURE 2.12: ResNet architectures on the ImageNet database. This picture is captured from [5].

Figure 2.13 compares the error rates of ResNet with its plain counterpart on the ImageNet database using 18 and 34 network architectures. ResNet showed significant improvement in the error rate compared with its plain network counterpart, demonstrating its capability to address the degradation problem. Table 2.8 shows the performance of ResNet compared with state-of-the-art models on the ImageNet database. The results show that ResNet with 152 layers generated the lowest top-1 and top-5 error rates compared with the other models. By combining six models with different depths, ResNet won 1st place in ILSVRC 2015 with an error rate of 3.57%. ResNet was also tested on CIFAR-10 using different architectures with a number of layers ranging from 20 to 1202. The best error rate achieved was 6.43% using 110 layers, and, as stated in [5], the 1202 layers introduced an overfitting problem to the system. Finally, ResNet was evaluated using the COCO database and won first place at the COCO 2015 competition.

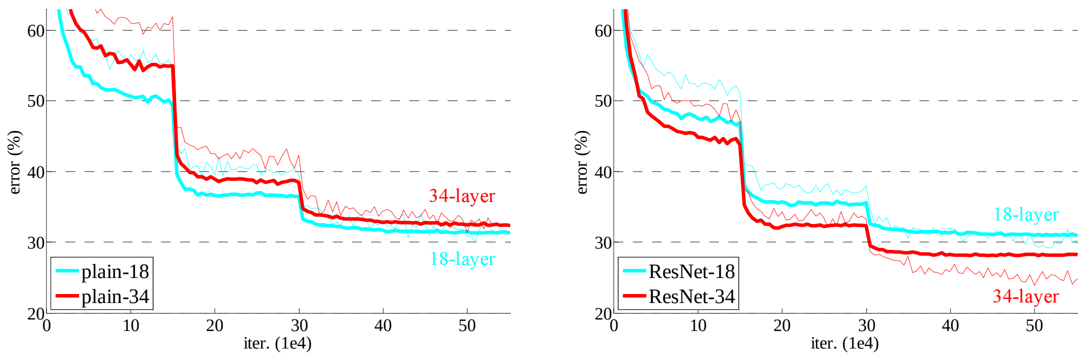


FIGURE 2.13: Training on the ImageNet database using 18 and 34 layer structure of (left) plain networks and (right) residual networks. This picture is captured from [5].

TABLE 2.8: ResNet performance on ImageNet, compared with state-of-the-art results. These results were all reported by [5].

Method	**top-1 error rate**	**top-5 error rate**
VGG (v5) [4]	24.4	7.1
PReLU-net [42]	21.59	5.71
BN-inception [43]	21.99	5.81
ResNet-34 [5]	21.84	5.71
ResNet-50 [5]	20.74	5.25
ResNet-101 [5]	19.87	4.60
ResNet-152 [5]	**19.38**	**4.49**

2.2.1.8 The All Convolutional Net [9]

Deep CNNs such as LeNet-5 [36], AlexNet [3] and VGGNet[4] have the same architecture, comprising a convolutional layer followed by max-pooling layers, fully connected layers and a softmax layer at the end of the network. The all convolutional network [9] (ALL-CNN), as its name suggests, consists only of convolutional layers followed by fully connected layers and a classifier. In other words, ALL-CNN replaces the max pooling layer with a convolutional layer that has a stride of more than 1. For example, a 3×3 max pooling layer with a stride of 2 is represented as a 3×3 convolutional layer with a stride of 2. Without pooling layers, complex activation functions and response normalisation, this basic design has achieved state-of-the-art results across many benchmarks, including CIFAR-10 and CIFAR-100.

In [9], ALL-CNN was evaluated across three databases, CIFAR-10, CIFAR-100 and ImageNet. The basic architectures used for CIFAR-10 and CIFAR-100 are described in Table 2.9. For each of these basic designs, three variants were examined in [9]. An example of these three variants for architecture (C) is described in Table 2.10. In general, each basic architecture has the following additional models:

- A model in which max-pooling layers are eliminated and the stride of the convolution layers that come before the max-pooling layers is increased by one (Strided-CNN-C in Table 2.10);
- A model replaces the max-pooling layers with convolutional layers (ALL-CNN-C in Table 2.10);
- A model that increases the number of convolutional layers while retaining the maximum pooling layers (ConvPool-CNN-C in Table 2.10). The advantage of this model is that it ensures that ALL-CNN's accuracy is not affected by the increase in learnable parameters generated by the replacement of max-pooling with convolutional layers.

TABLE 2.9: Three baseline architectures for the CIFAR-10 and CIFAR-100 databases.

Models		
A	**B**	**C**
32×32 RGB image		
5×5 conv. 96 ReLU	5×5 conv. 96 ReLU 1×1 conv. 96 ReLU	3×3 conv. 96 ReLU 3×3 conv. 96 ReLU
3×3 max-pooling stride $= 2$		
5×5 conv. 192 ReLU	5×5 conv. 192 ReLU 1×1 conv. 96 ReLU	3×3 conv. 192 ReLU 3×3 conv. 192 ReLU
3×3 max-pooling stride $= 2$		
3×3 conv. 192 ReLU		
1×1 conv. 192 ReLU		
1×1 conv. 10 ReLU		
Global average pooling over 6×6 spatial dimensions		
10 or 100 softmax		

Table 2.11 compares the error rate of the baseline models with the additional variants on the CIFAR-10 database with no data augmentation. The results show that ALL-CNN-C achieved the best performance among the other architecture, with an error rate of 9.08%. Table 2.12 compares the best accuracy achieved by ALL-CNN architecture with some state-of-the-art results on CIFAR-10 and CIFAR-100 databases [9]. According to the table, ALL-CNN-C achieved good results in both databases, indicating the effectiveness of the network. ALL-CNN achieved state-of-the-art performance on the CIFAR-10

TABLE 2.10: Three variants derived from model C described in Table 2.9.

Models		
Strided-CNN-C	**ConvPool-CNN-C**	**ALL-CNN-C**
32×32 RGB image		
3×3 conv. 96 ReLU 3×3 conv. 96 ReLU with stride $r = 2$	3×3 conv. 96 ReLU 3×3 conv. 96 ReLU 3×3 conv. 96 ReLU	3×3 conv. 96 ReLU 3×3 conv. 96 ReLU
	3×3 max-pooling stride $= 2$	3×3 conv. 96 ReLU with stride $r = 2$
3×3 conv. 192 ReLU 3×3 conv. 192 ReLU with stride $r = 2$	3×3 conv. 192 ReLU 3×3 conv. 192 ReLU 3×3 conv. 192 ReLU	3×3 conv. 192 ReLU 3×3 conv. 192 ReLU
	3×3 max-pooling stride$= 2$	3×3 conv. 192 ReLU with stride $r = 2$

$\vdots$

database and competitive results on the CIFAR-100 database. On ImageNet, ALL-CNN obtained results comparable with those produced by AlexNet, albeit with six times fewer parameters.

TABLE 2.11: Error rate of baseline models described in Table 2.9 and their variants (2.10) on the CIFAR-10 database with no data augmentation.

Model	**Error rate (%)**
Model A	12.47
Strided-CNN-A	13.46
ConvPool-CNN-A	10.21
ALL-CNN-A	10.30
Model B	10.20
Strided-CNN-B	10.98
ConvPool-CNN-B	9.33
ALL-CNN-B	9.10
Model C	9.74
Strided-CNN-C	10.19
ConvPool-CNN-C	9.31
ALL-CNN-C	**9.08**

2.2.1.9 Densely Connected Convolutional Neural Network [12]

To handle the vanishing gradient descent problem, the densely connected convolutional layer (DenseNet) [12] was introduced. The main idea of DenseNet is that each layer receives as input the feature maps of all preceding levels. This

TABLE 2.12: ALL-CNN error rate (%) compared with some networks on the CIFAR-10 and CIFAR-100 databases with and without data augmentation. These results were reported in [9].

Model	**C10**	**C10+**	**C100**	**C100+**
Maxout [7]	11.68	9.38	34.57	–
NiN [31]	10.41	8.81	35.68	–
Fractional pooling [6] [1 test]	–	–	**31.45**	–
Fractional pooling [12 tests]	–	–	**26.39**	–
Fractional pooling [100 tests]	–	–	–	**3.43**
ALL-CNN [9]	**9.08**	**7.25**	33.71	4.41

indicates that a neural network with L layers has $L\frac{(L-1)}{2}$ connections. Figure 2.14 shows the architecture of DenseNet, which consists mainly of dense blocks separated by transition layers. Before entering the first dense block, DenseNet initially applies a 3×3 convolution with a batch normalisation operation. The first convolutional operation is applied with a stride of 1 and a padding of 1. Every dense block consists of several dense layers, with the l^{th} layer receiving the outputs of all its predecessor layers $(x_0, x_1, \ldots, x_{l-1})$, as follows:

$$x_l = H_l([x_0, x_1, \ldots, x_{l-1}]), \tag{2.11}$$

where:

- $[x_0, x_1, \ldots, x_{l-1}]$ is the concatenation of feature maps generated by layers $[0, 1, \ldots, l-1]$;
- H_l is a composite function of three operations: batch normalization (BN), a rectified linear unit (ReLU) and a 3×3 convolution. If dense layers are defined as bottleneck layers (DenseNet-B in the paper), H_l is a function comprised of the following consecutive operations: BN, ReLU, 1×1 convolution, BN, ReLU and 3×3 convolution.

If each layer in a dense block generates k features, the l^{th} layer receives $k_0 + k(l-1)$ feature maps, where k_0 represents the number of channels in the input layer. The term 'growth rate' is used throughout the paper to refer to the parameter k. According to [12], with a specific growth rate of k, each 3×3 convolution operation generates k feature maps, but any 1×1 convolution operation generates $4k$ feature maps.

The transition blocks that exist between the DenseNet blocks work as a downsampling step and consist of a 1×1 convolutional layer followed by a

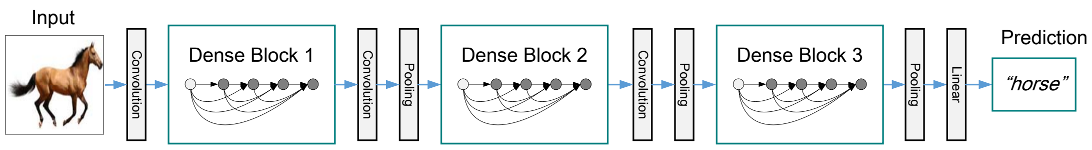


FIGURE 2.14: DenseNet architecture: a deep learning model consists of several dense blocks separated by transition blocks. This picture is captured from [12].

2×2 max-pooling layer with a stride of 2. The 1×1 convolutional layer generates the same number of feature maps unless specifying a compression factor (C); in such case (DenseNet-C in the paper), the convolution reduces the size of the feature maps by C, which was specified as 0.5 in [12]. The last transition block is followed by a global average pooling and a softmax layer.

DenseNet was evaluated using different architectures over four databases, CIFAR-10, CIFAR-100, SVHN and ImageNet [12]. Table 2.13 shows the performance of DenseNet compared with the models achieving state-of-the-art results across the CIFAR-10, CIFAR-100 and SVHN databases. According to Table 2.13, DenseNet achieved state-of-the-art performance in the three databases with a significant improvement over the existing models, achieving a 30% lower error rate on the CIFAR-10 and CIFAR-100 databases than FractalNet with drop-path regularisation. On the ImageNet database, DenseNet was evaluated using different architectures and compared with ResNet. The results on this database were comparable to those obtained by ResNet, with fewer parameters.

2.2.1.10 FractalNet [11]

The success of ResNet [11] has motivated other researchers to develop similar networks based on it. Their research includes studies on pre-activation ResNet [47], WRN [46], RiR [48], RoR [49] and stochastic depth [45]. The authors of [11] presented an alternative non-residual network named FractalNet. This network produced results competitive with ResNet's findings on classification tasks, including on the CIFAR and ImageNet databases. FractalNet has therefore demonstrated that residual representations are not strictly required for the success of deep convolutional neural networks.

The structure of FractalNet is illustrated in Figure 2.15. Let C represent the index of the truncated fractal; the structure of the network, including its connections and layer types, is specified by the function $f_C(.)$. In Figure 2.15 (the

TABLE 2.13: DenseNet error rate (%) compared with other networks on the CIFAR-10, CIFAR-100 and SVHN databases. These results were all reported by [12]. The "+" symbol in the table refers to the use of data augmentation.

Method	**C10**	**C10+**	**C100**	**C100+**	**SVHN**
Network in Network [31]	10.41	8.81	35.68	–	2.35
All-CNN [9]	9.08	7.25	–	33.71	–
Highway Network [44]	–	7.72	–	32.39	–
FractalNet [11]	10.18	5.22	35.34	23.30	2.01
with Dropout/Drop-path	7.33	4.6	28.20	23.73	1.81
ResNet [5]	–	6.61	–	–	–
Resnet reported by [45]	13.63	6.41	44.74	27.22	2.01
ResNet with stochastic depth [45]	11.66	5.23	37.80	24.58	1.75
Wide ResNet [46]	–	4.81	–	22.01	–
with Dropout	–	–	–	–	1.64
DenseNet ($k = 12$, $L = 40$) [12]	7	5.24	27.55	24.42	1.79
DenseNet ($k = 12$, $L = 100$)	5.77	4.10	23.79	20.20	1.67
DenseNet ($k = 24$, $L = 100$)	5.83	3.74	23.42	19.25	**1.59**
DenseNet-BC ($k = 12$, $L = 100$)	5.92	4.51	24.15	22.27	1.76
DenseNet-BC ($k = 24$, $L = 100$)	**5.19**	3.62	**19.64**	17.60	1.74
DenseNet-BC ($k = 40$, $L = 190$)	–	**3.46**	–	**17.18**	–

middle block), C denotes the total number of columns ($C = 4$ in the figure). To begin, consider the base case $f_1(z)$ as a convolutional layer defined as follows:

$$f_1(z) = \text{conv}(z). \tag{2.12}$$

The recursive fractals (columns) are added as follows:

$$f_{C+1} = [(f_C \circ f_C)(z)] \oplus [\text{conv}(z)], \tag{2.13}$$

where $\circ$ is the decomposition operation and $\oplus$ is the element-wise mean (the green layers in Figure 2.15). The network depth within a block consists of $2^{(C-1)}$ layers. In Figure 2.15, where the middle image represents a block and $C = 4$, the number of convolutional layers within the block is $2^3 = 8$ layers. By stacking five blocks ($B = 5$), as in FractalNet on the right side of Figure 2.15, the number of convolutional layers within the entire network is $B \times 2^{(C-1)}$, which is $5 \times 2^3 = 40$ layers. Moreover, 2×2 max pooling is added between every two blocks in the network to minimise the size of feature maps. After each convolution, Batch Norm and ReLU are also used.

To reduce overfitting, the authors of [11] proposed a regularisation approach

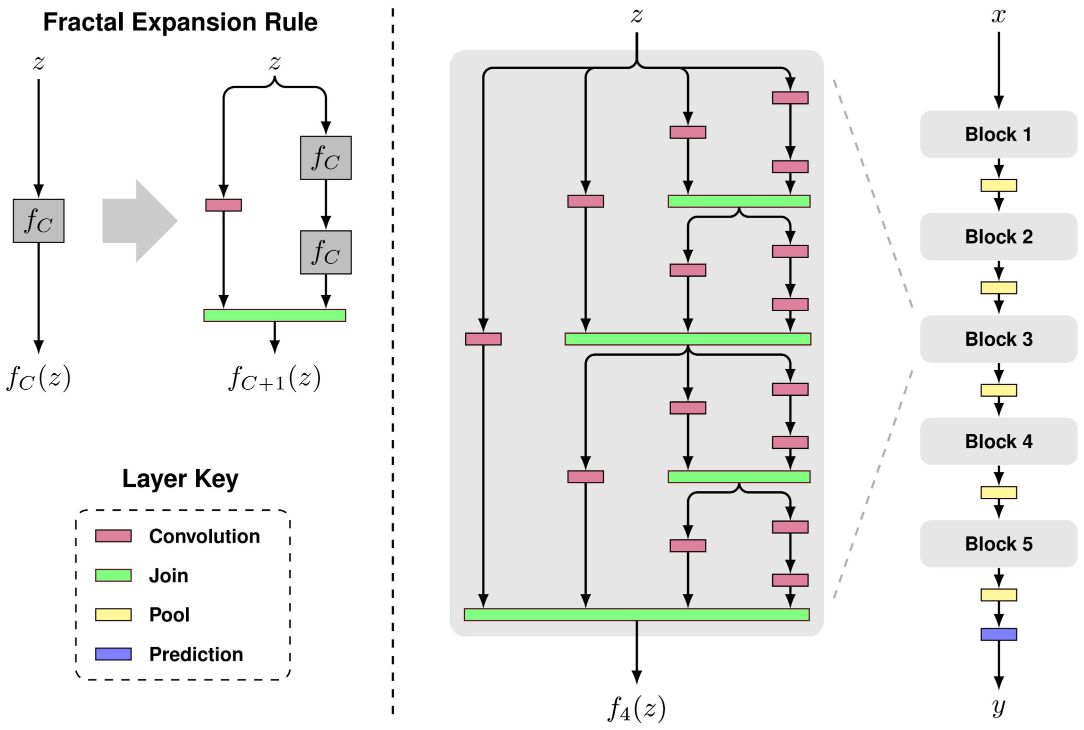


FIGURE 2.15: The architecture of FractalNet: simple Fractal expansion architecture (left), Fractal expansion stacked recursively as a single block (centre), and five blocks cascaded as FractalNet (right). Captured from [11].

named drop-path, which is represented in Figure 2.16. By randomly removing the operands of the join layers (the green layers in the figure), the drop-path algorithm prevents the co-adaptation of parallel paths. The following two types of sampling strategies were considered in the paper:

- **Local**, where the join layer deactivates some inputs with a fixed probability and ensures at least one input is activated.
- **Global**, where a single path or column is selected for the whole network.

The network was evaluated across four classification benchmarks, CIFAR-10, CIFAR-100, SVHN and ImageNet, both with and without data augmentation. Table 2.14 displays the error rates of the test set of FractalNet on three datasets, CIFAR-10, CIFAR-100 and SVHN, compared with several ResNet variants. The results show that FractalNet achieved the lowest error rate compared with ResNet and all of its variants except DenseNet in all databases without data augmentation. For example, the error rate obtained by FractalNet on CIFAR-100 with no data augmentation and no regularisation was 35.34%, which was 9% lower than the 44.76% achieved by the original ResNet. With

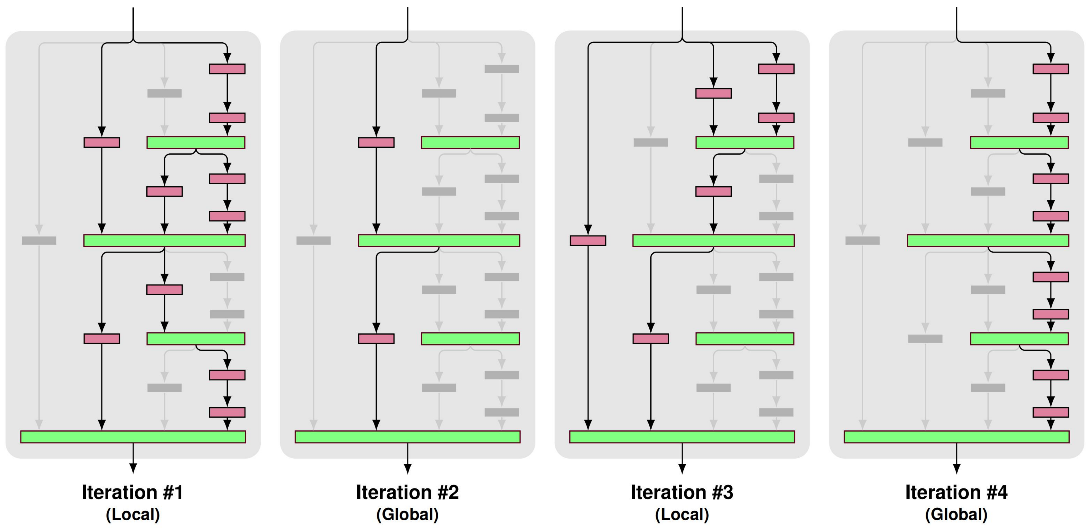


FIGURE 2.16: Drop-path: a fractal network block works with certain layer connections deactivated. Captured from [11].

data augmentation, FractalNet outperformed almost all ResNet variants except for the wide residual network (WRN), CIFAR-10 and CIFAR-100. With heavy data augmentations, FractalNet did not always achieve the best performance compared with other networks on CIFAR-10 and CIFAR-100. Table 2.15 compares the performance of FractalNet with that of ResNet-34 and VGG-16 on the ImageNet database. The results show that FractalNet provided comparable results with those of ResNet-34, which is evidence of the network efficiency [11].

TABLE 2.14: The error rate of FractalNet compared with other methods on three databases, CIFAR-10, CIFAR-100 and SVHN. These results were reported by [11]. The sign "+" in the table indicates the use of data augmentation, while "++" indicates the use of heavy data augmentations.

Method	**C100**	**C100+**	**C100++**	**C10**	**C10+**	**C10++**	**SVHN**
NiN [31]	35.68	–	–	10.41	8.81	–	2.35
Generalised pooling [50]	32.37	–	–	7.62	6.05	–	**1.69**
Recurrent CNN [51]	31.75	–	–	8.69	7.06	–	1.77
Multi-scale [52]	27.56	–	–	6.87	–	–	1.76
FitNet [53]	–	35.04	–	–	8.39	–	2.42
Deeply supervised [12]	–	34.57	–	9.69	7.97	–	1.92
All-CNN [9]	–	33.71	–	9.08	7.25	4.41	–
Highway Net [44]	–	32.39	–	–	7.72	–	–
ELU [54]	–	24.28	–	–	6.55	–	–
Scalable BO [55]	–	–	27.04	–	–	6.37	1.77
Fractional max-pool [6]	–	–	26.32	–	–	**3.47**	–
FitResNet [56]	–	27.66	–	–	5.84	–	–
ResNet [57]	–	–	–	–	6.61	–	–
ResNet [45]	44.76	27.22	–	13.63	6.41	–	2.01
Stochastic depth [45]	37.80	24.58	–	11.66	5.23	–	1.75
Identity mapping [47]	–	22.68	–	–	4.69	–	–
ResNet in ResNet [48]	–	22.90	–	–	5.01	–	–
Wide [46]	–	20.50	–	–	4.17	–	–
DenseNet [12]	**19.64**	**17.60**	–	**5.19**	**3.62**	–	1.74
FractalNet [11] (20 layers, 38.6 M params)	35.34	23.30	22.85	10.18	5.22	5.11	2.01
+ drop-path + dropout	28.20	23.73	23.36	7.33	4.60	4.59	1.87
$\rightarrow$ deepest column alone	29.05	24.32	23.60	7.27	4.68	4.63	1.89
FractalNet [11] (40 layers, 22.9 M params)	–	22.49	**21.49**	–	5.24	5.21	–

TABLE 2.15: FractalNet performance compared with ResNet [4] and VGG [5] on the ImageNet database.

Method	**Top-1 (%)**	**Top-5 (%)**
VGG-16	28.07	9.33
ResNet-34	24.19	7.40
FractalNet-34	**24.12**	**7.39**

2.2.2 Deep Generative Models

Deep generative models aim to create an explicit probability distribution for the data. The next sub-sections describe three generative models, the deep directed network, the deep Boltzmann machine and the deep belief network. The Boltzmann machine is also explained, as it is the basis of many models, such as deep Boltzmann machine.

2.2.2.1 Boltzmann Machine [37]

This is a probability model that is represented by the undirected graph. The primary purpose of the Boltzmann machine is 'to determine the probability distribution of the input' or to reconstruct the original data. There are two types of Boltzmann machine: restricted and unrestricted (Figure 2.17). In the unrestricted Boltzmann machine, every node is connected to all other nodes; however connection between the nodes in the same layer is not permitted in the restricted version. Each node in the Boltzmann machine has a binary state of 0 or 1, and the hidden units contain the representations derived from the model.

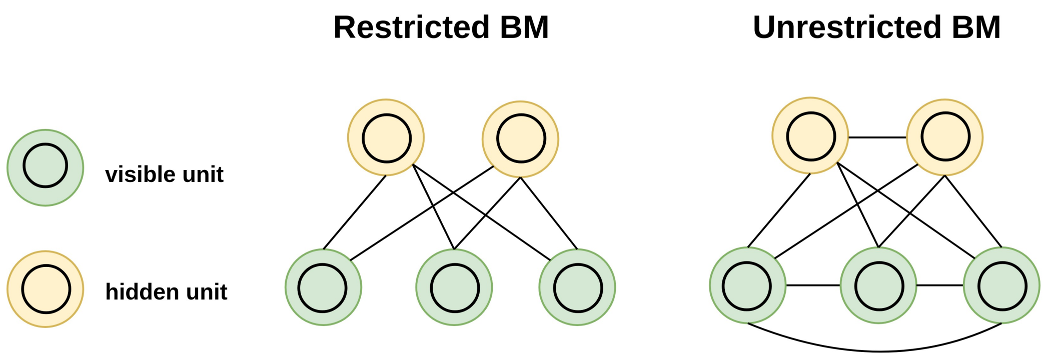


FIGURE 2.17: Two types of Boltzmann machine: restricted and unrestricted. In the unrestricted Boltzmann machine every node is connected to all other nodes. On the other hand, connection between nodes in the same layer is prohibited in the restricted version of Boltzmann machine.

The restricted Boltzmann machine is an energy-based function that focuses on minimising the energy function (equation 2.14) to increase probability (equation 2.15).

$$E(v,h;\theta) = -\sum_{i,j} v_i w_i h_j - \sum_j b_j h_j - \sum_i c_i v_i = -v^T wh - b^T h - c^T v \tag{2.14}$$

$$E(v, h; \theta) = \frac{1}{z(\theta)} \exp(-E(v, h; \theta)), \tag{2.15}$$

where

- v is the visible unit and h is the hidden unit;
- W denotes the weight connecting the visible and the hidden units;
- b is the bias term related to the hidden units and c is the bias term of the visible units;
- z is the partition function and the normalisation factor ($z = \sum_{v,h} e^{-E(v,h)}$), which is an intractable term.

The conditional probabilities of the nodes are independent of one another in the restricted Boltzmann machine. In addition, the following equation calculates the probability that a hidden node or a visible node is active (equal to one):

$$p(h = 1|v; \theta) = \sigma(w^T v + b) \tag{2.16}$$

$$p(v = 1|h; \theta) = \sigma(w^T h + c), \tag{2.17}$$

where

- σ is the sigmoid function;
- W is the weight connecting the visible unit with the hidden one.;
- h is the hidden unit and v is the visible one;
- b is the bias term for the hidden unit and c is the bias term for the visible unit.

The learning rule in the Boltzmann machine involves the process of learning the parameter θ in Equation 2.15, which involves the bias terms that relate to both the visible and hidden units and the weights that connect the two layers. This learning process is accomplished by applying stochastic gradient descent to minimise the negative log loss between the input and the reconstructed image, resulting in two terms (Equation 2.18). The first term (Equation 2.18) refers to the positive term, where it is easy to calculate, while the second term is intractable. There are many algorithms to approximate the second term, such as contrastive divergence with k-steps of Gibbs sampling. After computing the second term, we can update the weights and proceed with the learning process.

$$\sigma\frac{-\log p(v^t)}{\sigma\theta} = E_h\left[\frac{\sigma E(v^{(t)},h)}{\sigma\theta}|v^{(t)}\right] - E_{v,h}\left[\frac{\sigma E(v,h)}{\sigma\theta}\right], \tag{2.18}$$

where $\sigma\frac{-\log p(v^t)}{\sigma\theta}$ is the derivative of the log probability of the training vector, v is the visible unit, h is the hidden unit and θ refers to the model parameters, including the bias terms related to the hidden and the visible units and the weight between them.

2.2.2.2 Deep Directed Network [38]

Figure 2.18 defines the structure of this network, where the lowest level contains the observed data and the remaining layers are hidden. It is also a probabilistic model in which the nodes are binary. If the conditional probability function is a logistic function, the directed network is called a sigmoid belief network. The following equation defines the joint probability of the model:

$$\begin{aligned} p(h_1,h_2,h_3,v|\theta) = \prod_i \mathrm{Ber}(v_i|sign(h_1^T w_{0i})) \prod_j \mathrm{Ber}(h_{1j}|sign(h_2^T w_{1j})) \\ \prod_k \mathrm{Ber}(h_{2k}|sign(h_3^T w_{2k})) \prod_L \mathrm{Ber}(h_{3L}|w_{3L}), \end{aligned} \tag{2.19}$$

where Ber is the Bernoulli distribution, h_1, h_2 and h_3 are the hidden units, v is the symbol for the visible nodes and w is the weight connecting the visible and the hidden units.

The main problem with the directed network is the explaining-away problem. Figure 2.19 shows an example of this problem, where 'the earthquake' and 'the truck hits the house' cause the house to jump, and these nodes are independent of each other. However, if there is an 'earth-quick', we know that the house will jump, and so we do not need to check if there is a truck that has hit the house. Thus, we can say the earthquick node explains away the truck hits the house one. In other words, the inference in the directed networks is intractable because the nodes are correlated.

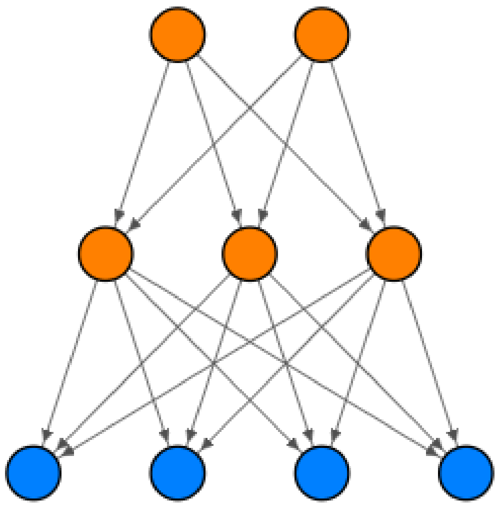


FIGURE 2.18: Deep directed network architecture

2.2.2.3 Deep Boltzmann Machine [39]

The Deep Boltzmann machine is a deep undirected model composed of a stack of Boltzmann machines (Figure 2.20). The following equation shows the joint probability of the model:

$$p(h_1, h_2, h_3, v|\theta) = \frac{1}{z(\theta)} \exp\left(\sum_{i,j} v_i h_{1j} w_{1ij} + \sum_{jk} h_{1j} h_{2j} w_{2jk} + \sum_{kL} h_{2k} h_{2L} w_{3kL} \right), \tag{2.20}$$

where, h_1, h_2, h_3 are the set of the hidden units, w is the model parameter representing the interactions between the visible and the hidden units and v is the visible unit.

The main advantage of using this model over the directed one is its ability to perform Gibbs sampling or block mean-field, since all of its nodes are conditionally independent of each other. However, the undirected graph is hard to train because of the intractable term Z.

2.2.2.4 Deep Belief Network [38]

This is a mixed model between the directed and undirected models (Figure 2.21). The following equation defines the model:

$$p(h_1, h_2, h_3, v|\theta) = \prod_i \text{Ber}\left(v_i | sigm(h_1^T w_1)\right) \prod_j \text{Ber}\left(h_{ij} | sigm(h_2^T w_{2i})\right) \frac{1}{z(\theta)} \exp\left(\sum_{kL} h_{2k} h_{3L} w3k \right), \tag{2.21}$$

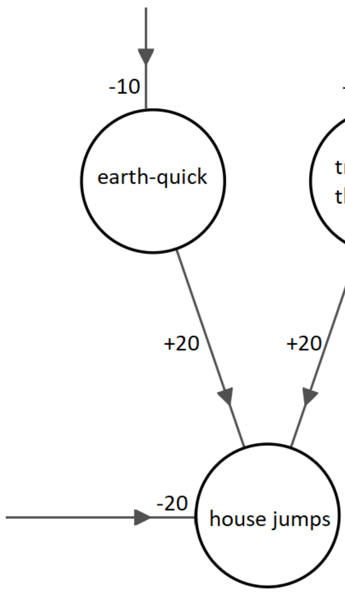


FIGURE 2.19: An example of the explaining-away problem. The nodes "earth-quick" and "truck hits the house" end to be dependent on each other.

where h is the hidden unit, v is the visible unit and w is the model parameter representing the interactions between the visible and the hidden units.

The two upper levels serve as allocation memories, while the other remaining levels generate the output. The model uses greedy layer-by-layer learning and top-down procedures to fine-tune the weights. The authors of [38] tested the model on the MNIST database (Figure 2.22) by linking the top restricted Boltzmann machine with a softmax layer of 10 neurons. The number of parameters was $28 \times 28 \times 500 + 500 \times 500 + 510 \times 500 = 1,662,000$; the top-down process took a week, and the error rate reached was 1.25%.

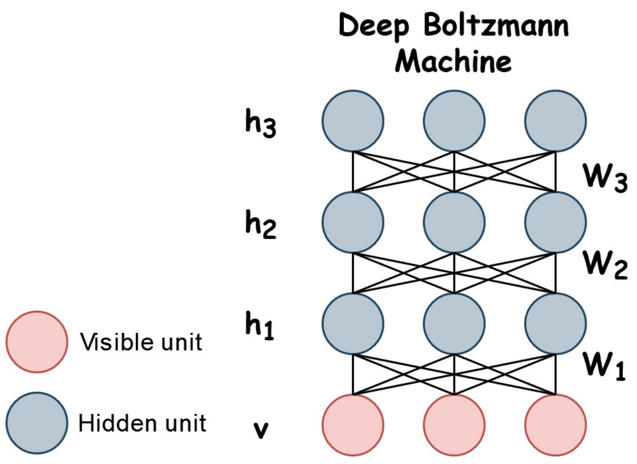


FIGURE 2.20: Deep Boltzmann machine structure.

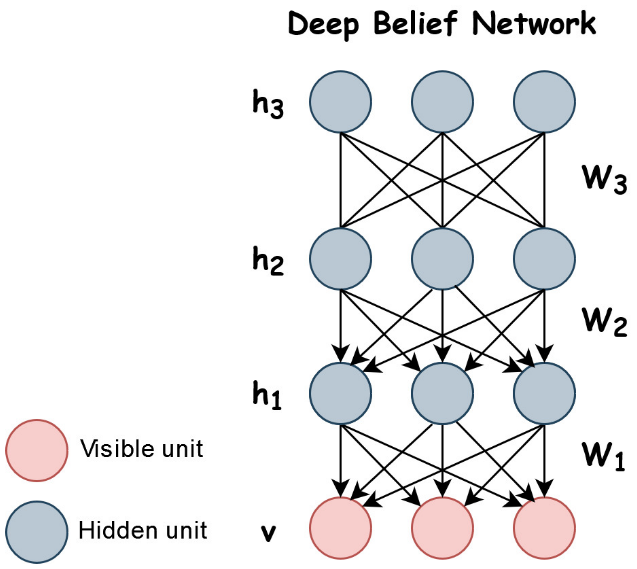


FIGURE 2.21: Deep belief net architecture.

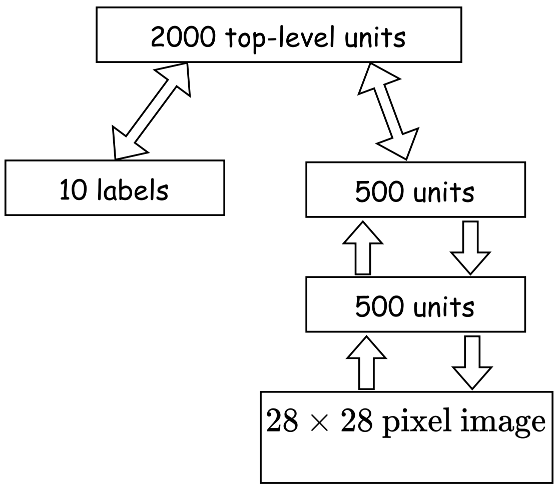


FIGURE 2.22: Deep belief net on the MNIST database.

2.3 Non Gradient-Based Networks

This section summarises some non-differentiable networks. A non-differentiable network is any network that is trained without gradient descent or backpropagation. Examples of such networks include PCANet [15], ScatNet [16] and deep forest [13].

2.3.1 Deep-SVM [34]

The SVM is a powerful binary classifier that aims to select the best hyperplane that divides two sets (Figure 2.23 (a)). This is accomplished by maximising the distance between the points close to the hyperplane, known as the support vectors, and the hyperplane itself. Moreover, the support vector machine can convert the linear space into a non-linear one using the kernel trick (Figure 2.23 (b)), by which we do not need to know the mapping itself as long as we know the kernel function. There are many kernel functions, such as linear, polynomial, RBF and sigmoid functions. Therefore, choosing the right kernel function that fits our data is not a trivial task. For this reason, the authors of [34] introduced deep SVM.

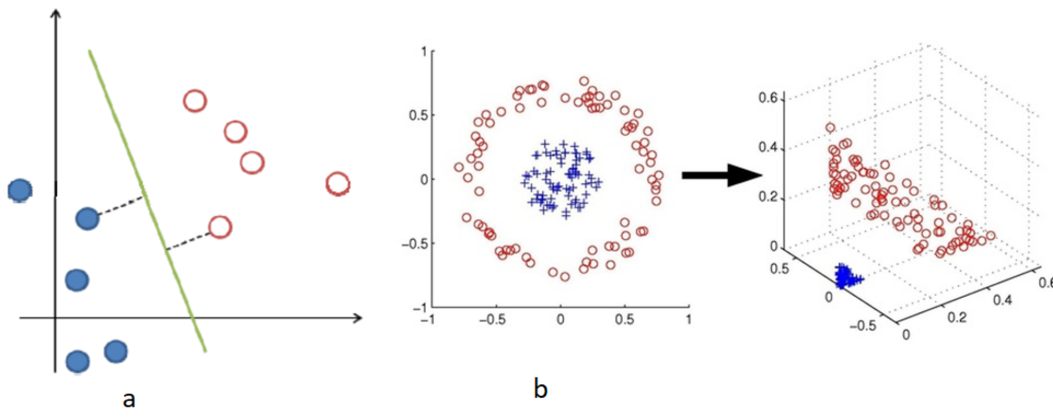


FIGURE 2.23: a) SVM maximises margin. b) Kernel trick of SVM.

The structure of deep-SVM is simple and displayed in Figure 2.24. Algorithm 1 presents the procedures for training deep SVM. The first layer's training is similar to the support vector machine's regular training, with the RBF kernel applied to the input data. The output signals of this layer are P support vectors (S_i) and P Lagrange multipliers (α_i). The RBF kernel is then applied for each support vector (S_i) with each input X, and the result is multiplied by the corresponding target (t_i) and the Lagrange multiplier at this point (α_i). The

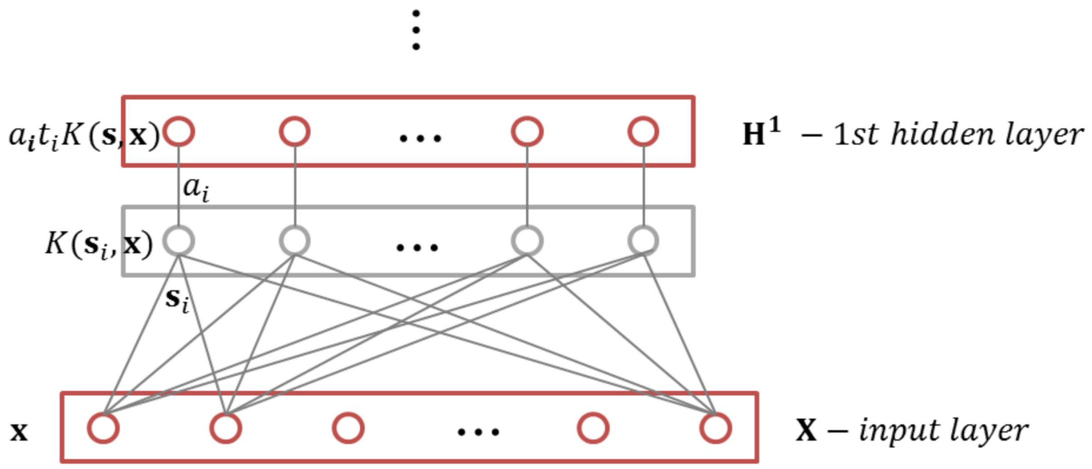

$a_i t_i K(\mathbf{s}, \mathbf{x})$
a_i
$K(\mathbf{s}_i, \mathbf{x})$
$\mathbf{s}_i$
$\mathbf{x}$
$\mathbf{H}^1$ – 1st hidden layer
$\mathbf{X}$ – input layer

FIGURE 2.24: Structure of deep SVM. Captured from [34].

same process is repeated for the subsequent layers until the desired number of layers is reached.

Algorithm 1 Training deep-SVM

```
Input: Training set: $X_1 = [x_1, x_2, \dots, x_N]$, target: $t = [t_1, t_2, \dots, t_N]$, RBF kernel
    function: $K()$ and number of Layers: $n_{layers}$
Output: $ErrorRate$
1: for $i \leftarrow 1$ to $n_{layers}$ do
2:     $[S_i, a_i, bi] \leftarrow \text{train}_{svm}(X_i, t)$
3:     $t_{si} \leftarrow$ target labels corresponding to $S_i$
4:     $p \leftarrow$ number of support vectors corresponding to $S_i$
5:     for $j \leftarrow 1$ to $N$ do
6:         $x_j^{i+1} \leftarrow t_{s1}^i \times a_1^i \times K(S_1^i, X_j^i), \dots, t_{sp}^i \times a_p^i \times K(S_p^i, X_j^i)$
7:     end for
8: end for
```

In [34], the network was tested on the Breast Cancer Database, which is a small database with only 384 instances with eight attributes. The authors compared their network to the ordinary support vector machine (Table 2.16) and demonstrated that their approach produced better results.

TABLE 2.16: The performance of deep-SVM compared to the standard support vector machine using the Breast cancer database.

	SVM	**Deep-SVM**
Training accuracy	100%	100%
Test accuracy	91.2%	95.6%

2.3.2 Deep Forest [13]

The idea of the deep forest [13] was inspired by the training of neural networks. The model consists of the cascade forest, Instead of the deep layers concept in CNNs, and the multi-grain scan level in the deep forest corresponds to the convolution layers in the CNN. After expanding the model to a new level, the authors of [13] validated it using a validation set. If there is no improvement, the algorithm ends, and the number of layers is determined.

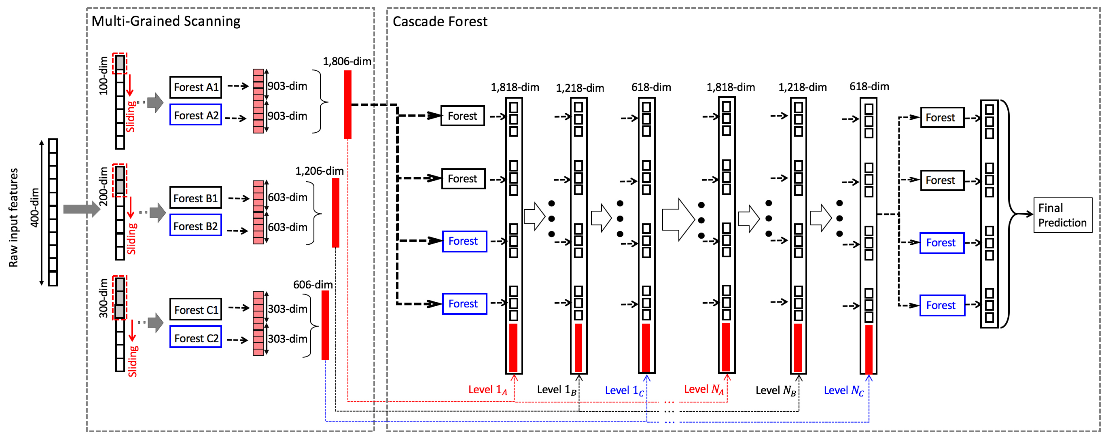


FIGURE 2.25: Two phases deep forest architecture: multi-grained scanning and cascade forest. This picture is taken from [13].

The deep forest model (Figure 2.25) consists of two independent phases, namely multi-grained scanning and cascade forest. In multi-grained scanning, different sliding windows are used to convert the input (either image or text) into many instances. These instances are then sent to two forests, each with 500 trees. The output of each forest is the averaged class distribution, which is then chained to other outputs to get the final level 1 output. The second stage takes the input (either the output of the first stage or the row data) and sends it to four forests, two of which are completely random forests and random forests, each with 500 trees. Similar to the first phase, the four random forests estimate the averaged class distributions, concatenate them and pass them to the next level. Before expanding to a new level, the entire model is validated in the validation set. If there is no improvement, the algorithm is stopped, and the output is defined.

Zhi-Hua and Ji [13] tested the deep-forest model across various areas, including image classification, face recognition, music classification, sentiment classification and low-dimensional data. They showed that their method delivers competitive results, especially for small datasets. They also emphasised the importance of multi-grained scanning for features representation. Tables 2.17 and 2.18 show the accuracy results of the deep forest model on the MNIST and CIFAR-10 databases, respectively. The performance of the model on challenging database such as CIFAR-10 was not as good as that of CNNs, but it was still better than other machine learning algorithms.

TABLE 2.17: The performance of gcForest on the MNIST database. All of these results were reported by [13].

Method	**Accuracy**%
gcForest	**99.26**%
SVM (RBF) kernel	98.75% [38]
Deep belief network	98.75%
Random forest	96.80%

TABLE 2.18: The performance of gcForest on the CIFAR-10 database. These results were reported by [13].

Method	**Accuracy**%
ResNet	**93.57**%
AlexNet	83.00%
gcForest (gbdt)	69.00%
gcForest (5 grains)	63.37%
Deep belief network	62.20%
gcForest (default)	61.78%
Random forest	50.17%
MLP	42.20%
Logistic regression	37.32%
SVM (linear kernel)	16.32%

2.3.3 PCANet

PCANet [15] is a simple deep learning network that does not rely on stochastic gradient descent in its learning process. Instead, the features are extracted using cascaded principal component analysis (PCA), which is applied to localised patches of the images. Figure 2.26 shows the architecture of PCANet. As shown in the figure, PCANet is comprised of three primary processing steps, namely cascaded principal component analysis, binary hashing and block-wise histogram, and the entire architecture is used as a feature extractor. After extracting the features using PCANet, the authors of [15] classified the extracted features using the linear SVM classifier.

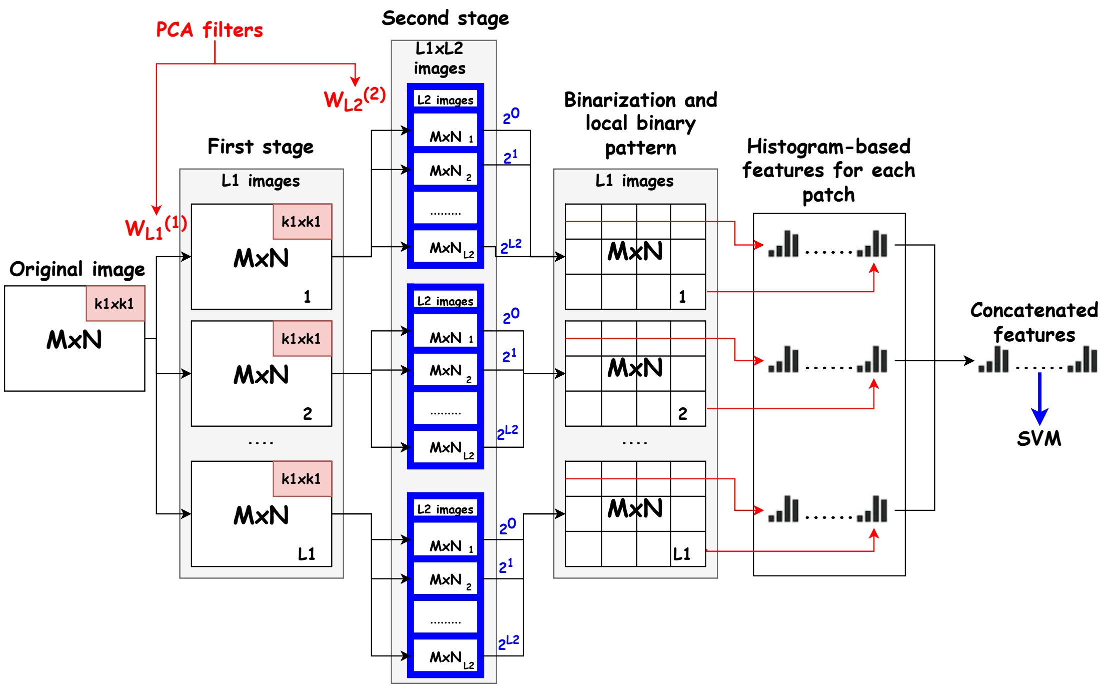


FIGURE 2.26: The main architecture of PCANet.

PCANet was evaluated using a maximum of two cascaded PCA stages [15]. Figure 2.26 is an example of a two-stage PCANet. The first stage consists mainly of two steps. The first step is patch-mean removal, in which the images are divided into patches of a specific size defined by the user. Then the patches of each image are centralised to their local mean. In the second step of the first stage, PCA is applied to the centralised patches to produce $L1$ principal components as the PCA filters for this stage, where $L1$ is the number of filters for the first stage and a user-defined parameter. After zero-padding the original images, the output of the first stage is computed by convolving the $L1$ principal components with the original images. The spatial dimensions of the resulting images are the same size as the original ones.

The second stage is similar to the first stage, but the output of the first convolution layer is used instead of the original dataset. In fact, the PCA filters are applied on a one-channel basis. For example, each filter operates on one channel and produces one new greyscale image. As a result, the number of images in this stage is equivalent to $L_1 \times L_2$, where L_2 is the number of filters for the second stage.

After the filter learning stages, the output of the last convolutional layer is binarized using the Heaviside function. The local binary pattern is then used to combine the $L2$ binary images into a single image, producing images with $L1$

channels. Then each of the L_1 images is divided into B blocks, the histogram of the decimal values is computed in each block and all of the B-histograms are concatenated into a single vector. The resulting L_1 vectors are combined to represent the features extracted from the corresponding original images. Finally, the SVM classifier is applied to classify the images. PCANet's user-defined hyper-parameters include the size of the patches, the number of filters within each level and the total number of levels.

PCANet shares some similarities with CNNs. For example, patch-mean removal in PCANet corresponds with local contrast normalisation in CNNs. However, in contrast to CNNs and ScatNet, PCANet does not have any non-linearity between its convolutional layers. Therefore, the whole process before the binarisation step is linear. The authors of [15] showed that the use of quantisation and local histograms in the last stage causes sufficient invariance for the final features. They also introduced two other networks, R-NET and LDA-NET. The difference between those and PCANet is that the cascaded filters are selected either randomly in R-Net or by learning a linear discriminant analysis in LDA-Net. The findings of the experiments in [15] showed that LDA-Net and PCANet are equivalent, although R-Net produced results with the highest error rate.

PCANet was designed mainly for classification tasks and tested on different databases, including LFW for face verification, MultiPIE, Extended Yale B, AR and FERET for face recognition, MNIST for hand-written digit recognition and CIFAR-10 for pattern recognition. Comparing PCANet's performance with the state-of-the-art results and hand-crafted features showed that the network's results were comparable [15]. On the MNIST database, PCANet reported an error rate of 0.6%, which was considered to be state-of-the-art at that time. In addition, the network set new records on other databases, including Extended Yale B, AR and FERET, confirming its exceptional capabilities across various tasks. Despite its success, PCANet may not be able to deal with extreme variability in some challenging databases due to its simple architecture, which relies on extracting unsupervised PCA features. Such databases include PASCAL VOC, which contains roughly aligned scales and a variety of poses.

2.3.4 PCANet+ [24]

PCANet is a shallow, deep structure that shows excellent results across different benchmarks, including image classification and face verification. However, this simple stacked deep architecture suffers from the feature's explosion problem, which limits its depth to only two layers. The feature explosion problem refers to the fact that the number of features extracted by the PCA layers in the network grows rapidly as the depth of the network increases, which makes it difficult for the classifier to classify the extracted features after adding a few layers. The problem is caused by the fact that PCANet learns the filter weights on a per-channel basis. For instance, when given an input of size $h \times w \times c$ and a filter of size $k \times k$, PCANet learns $k \times k$ weights for each channel c in the input. In other words, the number of feature maps in one layer depends on the number of feature maps in its previous layers. As the depth of the network increases, the size of the feature maps also increases, leading to high computational cost, high memory consumption and an increased number of parameters required for training subsequent layers. PCANet+, proposed by Low, Teoh, and Toh [24], provided an alternate solution to the feature explosion problem by conforming to CNN's constraints. The proposed network employed a filter ensembling mechanism to directly determine the number of feature maps for each layer. This mechanism aggregates the feature maps across all channels to extract $k \times k \times c$ patches, similar to CNNs. In this context, k denotes the filter size, and c represents the number of channels in the previous layer.

Figure 2.27 illustrates the structure of PCANet+ as an entirely unsupervised network. The designers of the network [24] used a mean-pooling layer between any two convolution layers as a replacement for the non-linear functions in CNNs. Moreover, they changed the topology of the PCANet filters by proposing an ensemble learning technique that conforms to CNN's structure. Given a layer $L-1$ with d_{L-1} images of size $M \times N$, all overlapping patches are extracted from each channel and concatenated, so the result is of size $k_L \times k_L \times d_{L-1}$, where k_L is the size of the filters in layer L. Next, the extracted patches are standardised to their unit variance using the z-score method before the PCA of the normalised patches is computed and d_L principal components are chosen to be the PCA filters, where d_L is the number of the filters for layer L. The output is calculated by convolving the d_L principal components with the input images after zero-padding them. After several PCA learning stages, the output of each layer is binarised and the histogram-based features are found in the way described in PCANet [15]. Then all extracted features from all layers are concatenated, and PCA whitening is applied to them to reduce their dimensions. To classify the data, the cosine similarity between the training and the test samples is used.

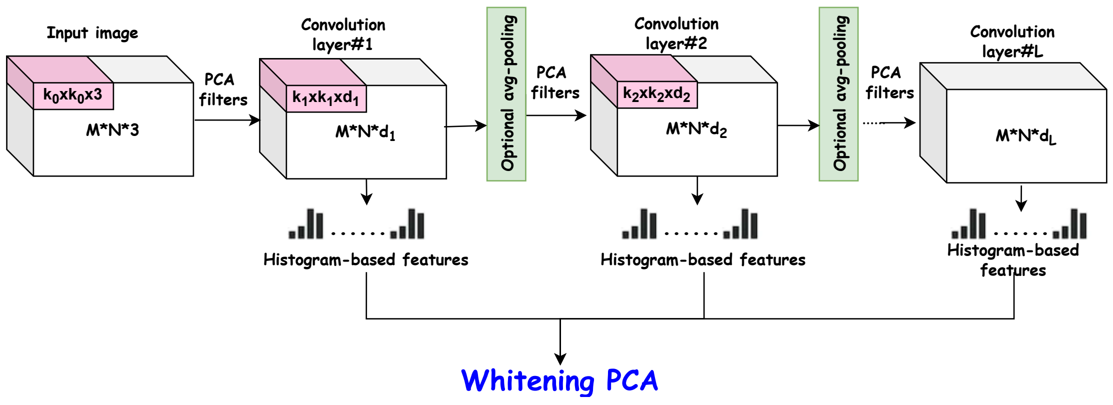


FIGURE 2.27: The main architecture of PCANet+.

In [24], PCANet+ was evaluated across three face recognition databases, including face recognition technology (FERET), LFW and YouTube faces (YTF). In all experiments, PCANet produced better accuracy results than PCANet. Figure 2.28 compares the performance of PCANet+ with that of the original PCANet using a different number of layers on the FERET DUP I and II probe sets. According to the results, PCANet+ outperformed the original PCANet in both one- and two-layer architectures. In addition, the average performance of PCANet+ increased from 70.09% in layer 1 to 89.73% in layer 4, indicating the importance of the network depth to achieve better accuracy. Figure 2.29 illustrates the significance of mean-pooling layers to the accuracy of the PCANet+ model. As shown in the figure, the performance of PCANet+ degraded somewhat without the mean pooling layers, with roughly 1% less accuracy in all topologies.

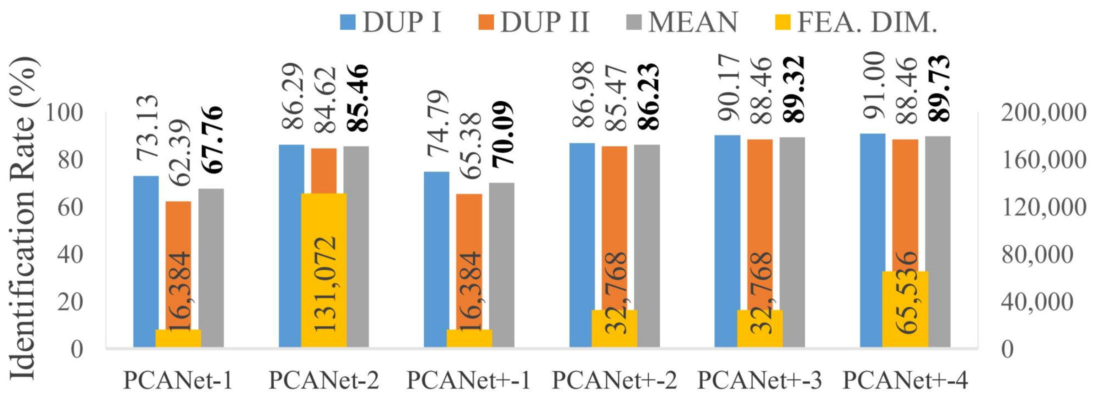


FIGURE 2.28: Accuracy of PCANet+ and PCANet on FERET DUP I and DUP II using different layers architectures. The figure is captured from [24].

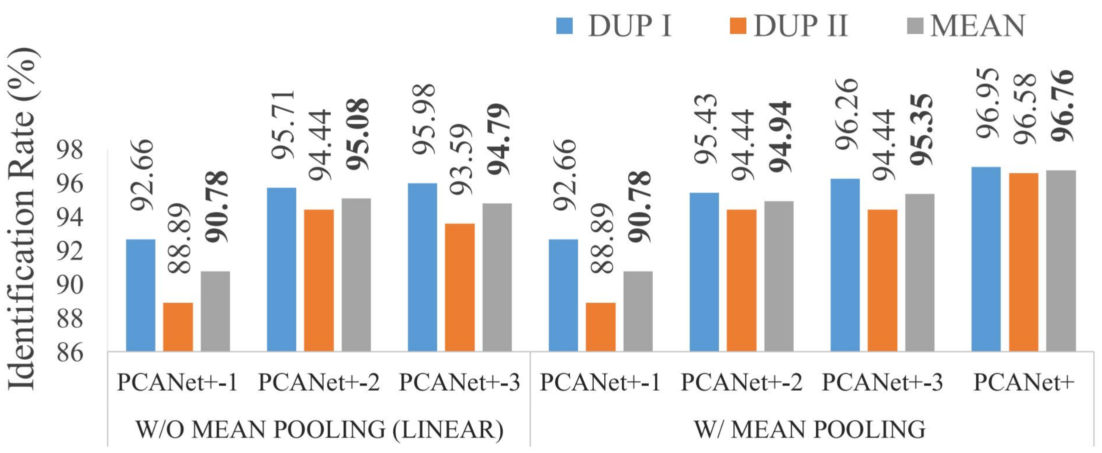


FIGURE 2.29: Accuracy of PCANet+ and PCANet on FERET DUP I and DUP II with and without mean pooling layers. The figure is captured from [24].

2.3.5 PCANet-II [23]

PCANet, as mentioned in Section 2.3.4, suffers from the feature explosion problem, which limits its depth to two layers. Histogram pooling, which results in exponential growth in the total number of features, is one of the factors that contribute to this issue. PCANet-II [23] replaced the histogram pooling in PCANet with second-order pooling, which is popular in CNNs. Examples of CNNs using second-order pooling include [58, 59, 60, 61] and [62]. Figure 2.30 illustrates PCANet-II architecture in detail. PCANet-II starts by computing the PCA filters, as in the original PCANet [15]. However, Tian, Hong et al. [23] calculated a pixel-wise binary feature difference (BDF) in each layer and considered it to be a new feature map added to that layer. The main purpose of adding a BFD image is to preserve more discriminant information because the binarisation step in PCANet [15] cuts the filters' responses into only 0's and 1's, leading to a large discriminant information loss. After the filter learning process, the resulting images are divided into blocks of size $r \times c$, where r and c are user-defined parameters representing the width and height of each block. Then each block is reshaped into $r \times c$ rows and d_L columns, where d_L is the number of filters in layer L. After that, the covariance matrix is calculated for each block and the resulting features are concatenated. Finally, the second-order features are sent to a classifier to classify the data.

PCANet-II has been tested on different benchmarks for classification and progression tasks. Among these databases are CAS-PEAL-R1 for classification and the UNBC-McMaster pain dataset for the regression task. The results of

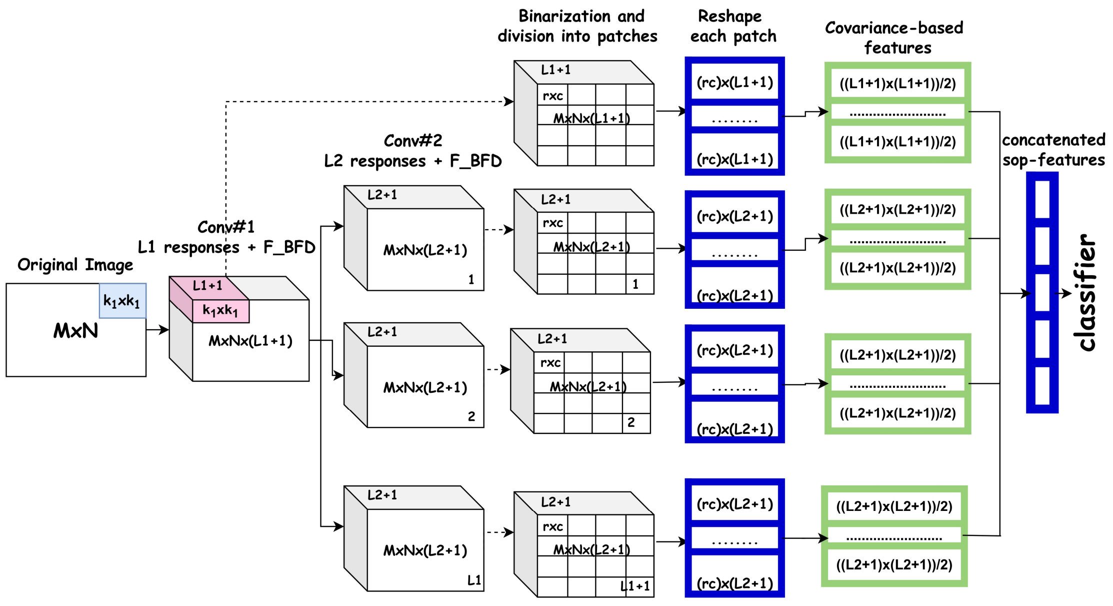


FIGURE 2.30: The main architecture of PCANet-II.

the experiments proved the effectiveness of using both second-order pooling and the BFD scheme, with improved running time and fewer features.

2.3.6 ScatNet [16]

Rigid and non-rigid transformations of the images inside a class make the classification task difficult, and the Euclidean distance cannot measure the similarity within a class due to this variability [16]. As a result, the authors of [16] concentrated on creating an invariant of each class and maintaining useful information from the images included inside this class.

The researchers initially focused on the translation transformation and proved that the Fourier transform is not stable for small deformations. This instability of low distortion becomes large at high frequencies, not at low rates. As a result, some researchers have used the Fourier transform to kill the high-frequency signals, resulting in the loss of such information. In contrast, the wavelet transform is stable for deformations. Thus, if we have a signal x, we can compute the wavelet operator by multiplying the Fourier transform of the signal by that of the wavelet and applying a band pass filter to cover the low frequencies that are not covered by the wavelet, such as described in the following equation:

$$pUx = x * \varnothing(t), x * \psi_\gamma \; with \; \psi_\gamma(t) = 2^{-jQ}\psi(2^{-jQ(t)}), \; and \; \gamma = 2^{-jQ}, \tag{2.22}$$

where Q is the wavelet bandwidth (number of wavelets per octave), γ is the location in the frequency and ψ_γ is the band-pass filter.

After applying the wavelet to the signal, the authors of [16] found that higher frequencies to be localised in time, but not in the frequency domain. They removed this sensitivity by averaging and restoring the lost information by adding more layers (deep network). They also found the use of a non-linear map to be mandatory, since the integration of a linear activation function results in zero at all times ($\int x * \psi(t)\ dt = 0$). They also proved the module to be the best nonlinear function for their work. Hence, the scattering transformation is calculated by scattering the signal information along different paths $\gamma_1, \gamma_2, \ldots, \gamma_m$ with a cascade of wavelet module operators implemented in a deep network (Figure 2.31).

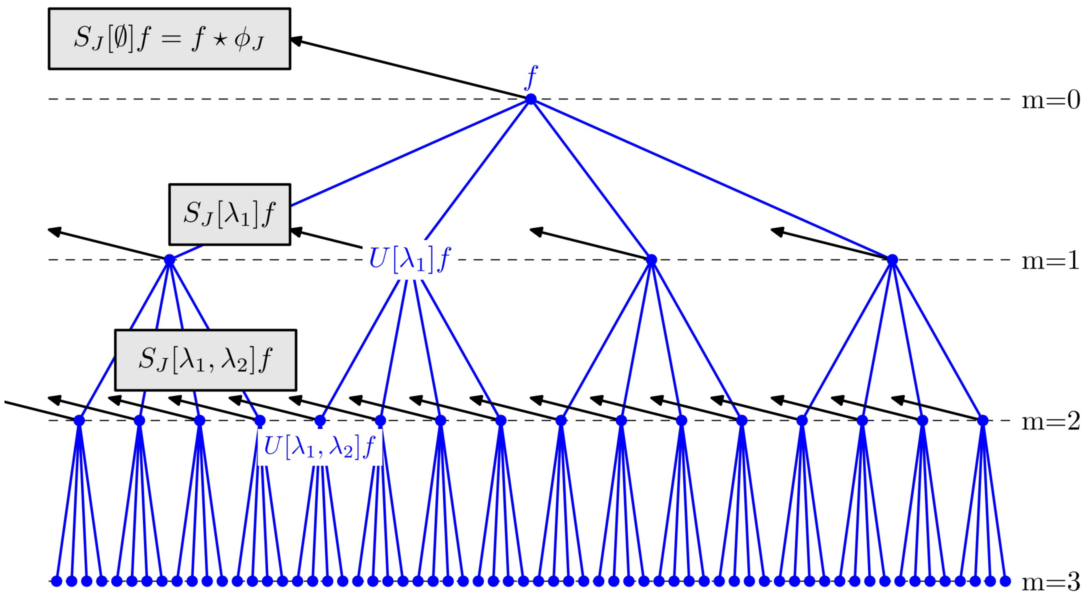

$S_J[\emptyset]f = f \star \phi_J$
f
m=0
$S_J[\lambda_1]f$
$U[\lambda_1]f$
m=1
$S_J[\lambda_1, \lambda_2]f$
$U[\lambda_1, \lambda_2]f$
m=2
m=3

FIGURE 2.31: Iterating scattering propagator $\tilde{W}$ defines the convolution network. Captured from [16].

Both ScatNet [16] and CNNs use nonlinearity across layers, a pooling mechanism (max pooling or averaging) and the convolution concept (Wavelet in ScatNet). The primary difference between the two networks is that only the last layer of the CNN produces outputs, while each layer of ScatNet generates coefficients. In addition, in ScatNet, the entire structure is known in advance as we bring the high frequencies back to low rates, when the depth of the network increases and the energy at the end of the structure becomes zero.

The scattering network was designed for classification tasks. The authors of [16] first tested their network using the MNIST dataset, with the only transformation of the images being translation. The size of the invariant used to test the MNIST was of the order of eight pixels $2^j = 8$, and so the number of the windows was 16 in each layer, and the architecture consisted only of two layers. After applying the scattering network to the entire database, the result was a vector that combined all the first- and second-order coefficients. The researchers ended by applying PCA to the output of the network to transfer it to the PCA space. Therefore, to test a new sample, the scattering network is used to extract the features, move the output to the PCA space and measure the distance between it and the training examples to find the closest class, which is the nominated one. In [16], the scattering network achieved a state-of-the-art result in the MNIST database with an error rate of 0.6%. The authors also tested ScatNet on the UIUC database, which they referred to as a type of texture classification, and three types of transformation were also considered, including translation, scaling and rotation. They achieved a state-of-the-art result in the UIUC dataset, with an error rate of 1%. The authors also used the SVM to test their network and found a slight improvement in the results.

2.4 Conclusions and Discussion

The ability of a system to learn complex non-linear functions that map the raw input directly to the output, known as "representation learning", is a crucial capability in deep learning techniques. Having the ability to learn complex functions automatically is becoming increasingly important as the amount of data and potential applications for machine learning techniques increase [34]. This chapter provided an overview of deep learning models that can automatically learn representations from raw data using gradient-based or non-gradient-based techniques. The content of this section is divided into three subsections: a summary of gradient-based models with their limitations (Section 2.4.1), an overview of non-gradient-based style models (Section 2.4.2) and a justification for selecting PCANet as the baseline for the upcoming chapters (Section 2.4.3).

2.4.1 Gradient-Based Models and Their Limitations

In Section 2.2.1, we provided a comprehensive overview of standard gradient-based convolutional architectures that are specifically designed for classification tasks. We started the section by describing LeNet-5, as this network served as the baseline architecture for all CNNs designs. Next, we discussed AlexNet, a convolutional neural network that has influenced many others and contributed significantly to deep learning for computer vision. Then we reviewed other standard state-of-the-art convolutional architectures, such as VGGNet, network in network, ResNet, Maxout, stochastic pooling, fractal network and many others.

Despite the remarkable achievements of deep gradient-based models in image classification, these models suffer from some problems. The vanishing or exploding problem is one example of these problems. Although various techniques, such as normalisation layers or ReLU activations, have been developed to mitigate this problem, it remains a relevant challenge in the deep learning era. Another problem with deep gradient-based models is the large number of parameters these models require for the training. Transfer learning [63] may be used to reduce the number of parameters needed for training a deep learning model and improve performance on a particular task. However, it is essential to carefully consider the choice of the pre-trained model, the fine-tuning dataset, and any additional techniques needed to mitigate potential issues such as overfitting.

The backpropagation is also a problem related to the gradient-based models since prior knowledge of the forward architecture is required [64], and not all properties in the world are differentiable. Deep generative models such as those discussed in Section 2.2.2 are probabilistic models that do not require backpropagation during their pre-training phase, although they can still be fine-tuned using backpropagation. During pre-training of the deep belief network, for example, each layer is trained as a Restricted Boltzmann Machine (RBM) using the Contrastive Divergence (CD) loss, which involves sampling from the distribution of the RBM and then using the difference between the positive and negative phase distributions to compute the gradient of the weights. The CD loss is computationally expensive as it involves repeatedly sampling from the model's distribution to update the network weights. In general, deep generative models offer an alternative to deep gradient-based models with backpropagation, but they also suffer from computational challenges that should be carefully considered.

2.4.2 Non-Gradient-Based Models And Their Advantages

In Section 2.3 of this chapter, we reviewed non-gradient-based models that are trained without backpropagation or gradient descent. Examples of such networks include deep SVM, deep forest, scattering network and PCANet. Although non-gradient-based models may not have the same level of computational capability as gradient-based models, they present some advantages over them. For example, these networks involve no gradient calculations, which eliminates the need to deal with vanishing or exploding gradient problems. Moreover, these networks are trained sequentially, one layer after another, which provides a faster training time advantage to these models over traditional gradient-based models.

Deep non-gradient-based models have achieved state-of-the-art performance in some databases, such as UIUC and MNIST. However, their overall performance is not comparable to that of standard deep gradient-based models. Therefore, this thesis aims to explore the limitations of non-gradient-based models and develop techniques to improve their performance.

2.4.3 PCANet As Baseline Model

PCANet is chosen as the baseline for our works for the subsequent chapters for the following reasons:

- PCANet has achieved state-of-the-art performance in many benchmarks, including MNIST for hand-written recognition and many other face recognition and verification databases, including Extended Yale, AR and FERET, which confirms its generalisation abilities across different classification tasks;

- Compared to other non-gradient-based models, such as scattering network, PCANet architecture is simple and easy to implement, which makes this network an ideal candidate for a baseline model.

PCANet has proven successful in image classification tasks, but it suffers from some limitations. Firstly, the filters' learning process involves using the PCA method, which is not designed specifically to perceive relationships between classes in the context of image classification. Secondly, despite the importance of network depth in achieving good classification accuracy, PCANet depth is limited to only two layers. Thirdly, PCANet did not provide good accuracy performance with databases that contains variability in the images, such as in the CIFAR-10 database. This thesis focuses on the development of networks that are robust against these types of weaknesses. In the following chapters, we discuss these networks and their abilities to address the previous limitations.

Chapter 3

Multi-Layer PCANet for Image Classification

3.1 Introduction

In the context of networks trained without backpropagation, PCANet [15] offers remarkably simple architecture for image classification. The network's topology relies on three main stages, which are 1) filter learning, 2) binarisation and encoding and 3) histogram pooling. The filter bank is generated using an unsupervised PCA method applied to stacking patches of images. The filters are applied to the images on a single-channel basis, meaning that each filter is applied to one channel to produce one new greyscale image. Consequently, the number of channels in the subsequent layer grows rapidly to channels$\times$filters. The binarisation step in PCANet cuts the filters' responses into only zeros and ones using the Heaviside function. The local binary pattern method then encodes the binary feature maps into a single representation that is sent to the histogram pooling to generate the features required for the classification.

PCANet has shown good performance across several benchmarks and state-of-the-art results in the MNIST database. However, some potential problems associated with its structure include:

- A shallow network structure that is limited to two layers, which may cause losses in performance, especially with complex databases;
- A feature maps binarisation step that restricts the filters' responses to only ones or zeros, leading to a significant information loss;
- Histogram-pooling and one-channel basis convolution mechanisms that lead to features explosion problem;

- An unsupervised filter learning technique in which it is possible for the subsequent layers to lose discriminative information of the previous levels.

In this chapter, we present the Multi-Layer PCANet network, which aims to tackle the main problems with PCANet. First, we employ late fusion to integrate class posteriors, incorporating supervised learning into PCANet while maintaining its single-pass structure. Second, we reduce the total number of features using second-order pooling, which was recently adopted by CNNs and PCANet-II [23]. We also refine the approach by replacing the binarisation step in PCANet with z-score normalisation, which provides more informative features. Finally, we use the standard PCANet+ [24] convolutional layers, which were inspired by CNN.

The chapter is organised as follows: Section 3.2 introduces the network architecture with its main components, namely convolutional layers (Section 3.2.1), second-order pooling (Section 3.2.2), spatial pyramid pooling (Section 3.2.3) and late fusion (Section 3.2.4). Section 3.3 then summarises the results obtained by using our network, including a description of the databases (Section 3.3.1), ablation study to show the importance of each component in the network (Section 3.3.2), the effect of the network hyper-parameters (Section 3.3.3) and classification performance across four benchmarks (Section 3.3.4). Finally, the conclusion of the chapter is presented in Section 3.4.

3.2 Multi-Layer PCANet Structure

The overall structure of Multi-Layer PCANet, as described in Figure 3.1, comprises a stack of PCA convolutional layers; on top of each one lies a second-order pooling and a classifier. The probabilities generated by all layers are combined using a final classifier, which is referred to as late fusion. Spatial pyramid pooling [65] can be added between the second-order pooling and the classifier to reduce the number of dimensions. The following subsections highlight the key components of the network: PCA convolutional layers, second-order pooling, spatial pyramid pooling and late fusion.

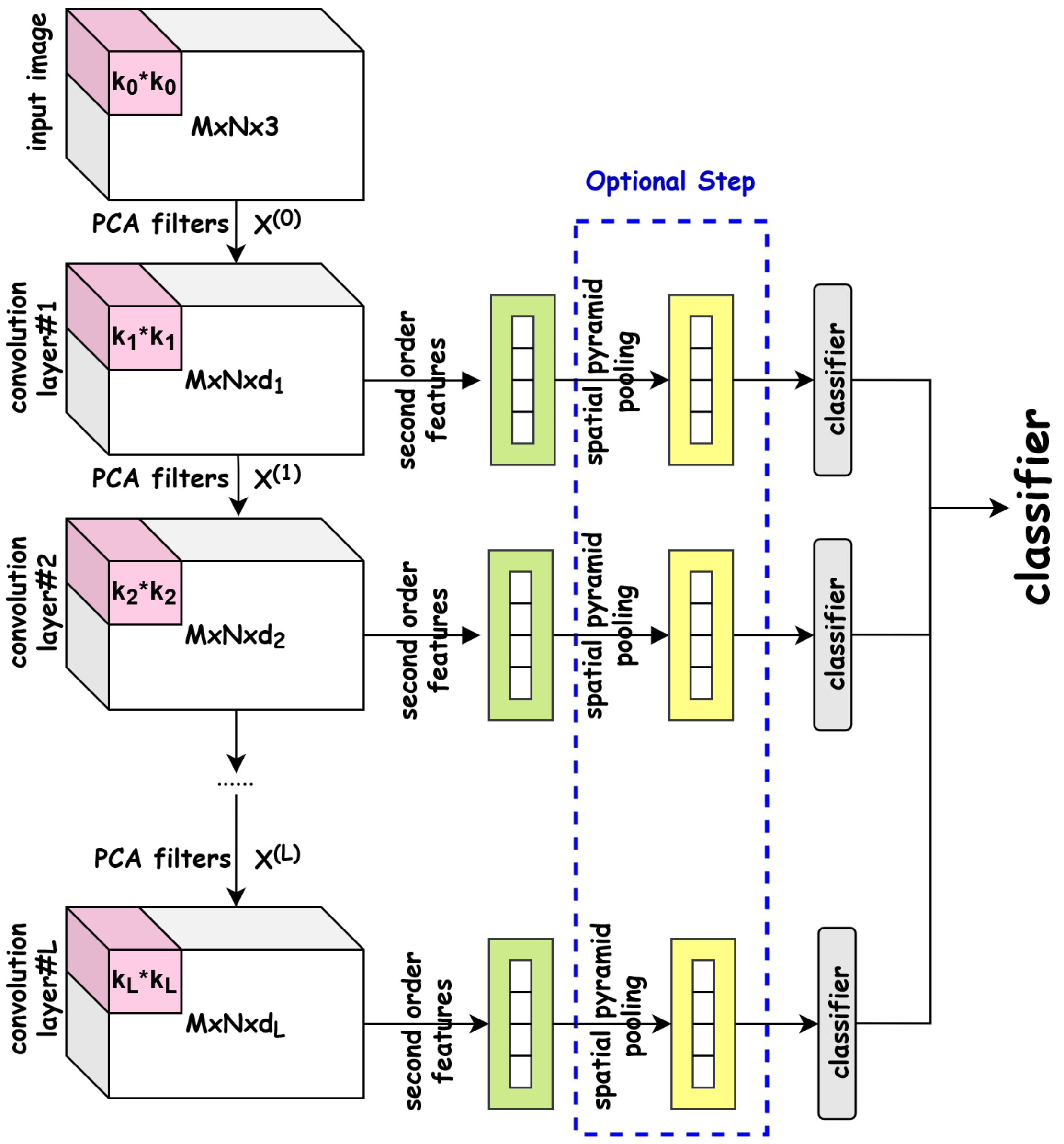


FIGURE 3.1: Multi-Layer PCANet structure

3.2.1 PCA Convolutional Layers

Unlike CNNs, PCA convolutional layers rely on learning $W^{(L)}$ from $X^{(L-1)}$ with no backpropagation, where X^{L-1} represents the previous layer's output. For N training images, i.e. $X^{(L-1)} = \{\{X_i\}\}_{i=1}^{N} : X_i \in \mathbb{R}^{m \times n \times d_{L-1}}\}$, where d_{L-1} represents the number of filters in layer $(L-1)$, the PCA filters can be calculated as follows:

1. Given an input image $X^{(L-1)}$ and a filter size $k_L \times k_L$, we first scan every possible location in the input feature maps to extract local windows.

Figure 3.2 gives an example of such a process, whereby a vector is constructed from each local window and then the vectors are combined to form a matrix.

2. We subtract the mean from the matrix generated by the previous step, resulting in a matrix $\bar{X}_i \in \mathbb{R}^{(k_L^2 d_{L-1}) \times \tilde{m}\tilde{n}}$, where $\tilde{m} = (m - k_L) + 1$, $\tilde{n} = (n - k_L) + 1$ and m and n are the width and the height of the image, respectively;

3. We repeat the previous operation for all images in the dataset. We then concatenate the vectorised version of all zero-mean patches to form a matrix $\bar{X} \in \mathbb{R}^{(k_L^2 d_{L-1}) \times N\tilde{m}\tilde{n}}$;

4. We solve the following equation to find d_L principle components of $(\bar{X}^{L-1}\bar{X}^{L-1^T})$:

$$\min_{V \in R^{(k_L^2) \times d_{L-1}}} ||\bar{X}^{L-1} - VV^T\bar{X}^{L-1}||_F^2,\ V^TV = I_{d_L-1}, \tag{3.1}$$

where $I_{d_{L-1}}$ is the identity matrix of size $d_{L-1} \times d_{L-1}$;

5. We can express the PCA filters as the following:

$$W_s^L = \underset{k_L \times k_L \times d_{L-1}}{\text{mat}} q_s,\ s = 1, 2, \ldots, d_L, \tag{3.2}$$

where $\underset{k_L \times k_L \times d_{L-1}}{\text{mat}} (v)$ is the function that maps the vector $v \in \mathbb{R}^{k_L^2 d_{L-1}}$ to tensor $W \in \mathbb{R}^{k_L \times k_L \times d_{L-1}}$, q_s is the s^{th} principal eigenvector of $\bar{X}^{L-1}\bar{X}^{L-1^T}$ and d_L is the number of filters chosen for the layer (L);

6. We obtain the output of each convolution layer by convolving each filter with the sample images.

$$X_i^L = X_i^{L-1} * W_s^L \in \mathbb{R}^{m \times n \times d_L}, \tag{3.3}$$

where $s = 1, 2, \ldots, d_L$ and X_i^{L-1} is zero-padded to get the same image size.
In fact, the convolution operation could be transformed into a matrix multiplication [66]. Thus, after dividing the image into zero-mean patches as described in step 2, we multiply the resulting matrix by the vectorized filter and reshape the image.

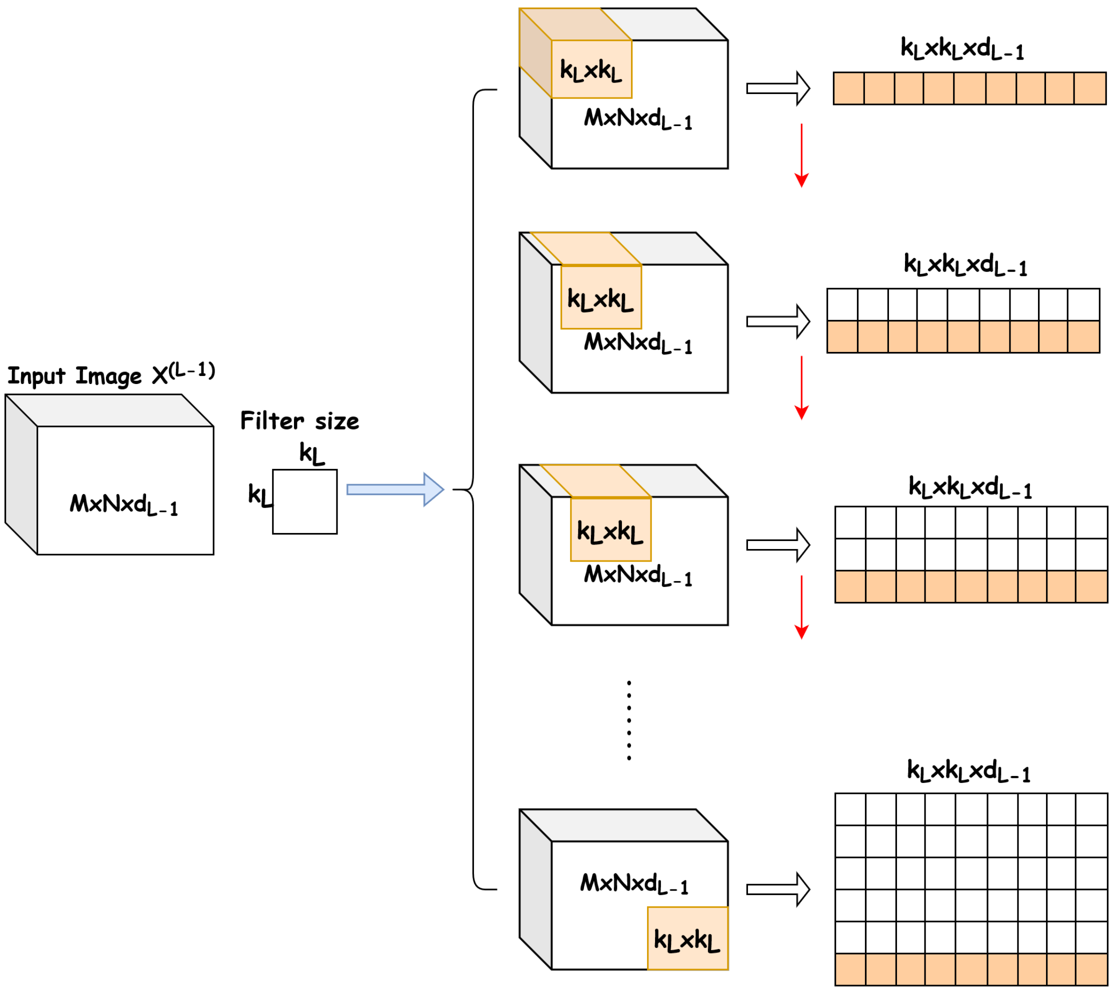


FIGURE 3.2: Example of extracting image patches using a filter of size $k_L \times K_L$.

3.2.2 Second-Order Pooling

Second-order pooling, as shown in Figure 3.1, is attached to the output of each convolutional layer. The computation of the second-order features is done locally for every single image in the database. Therefore, for N training samples $X_i^L \in \mathbb{R}^{m \times n \times d_L}$, where d_L is the number of filters in layer L, we first divide each tensor into patches of the same size, e.g. $(r \times c)$. These patches could be overlapped. Then, unlike PCANet [3] and its variant, we normalise each patch using the z-score normalisation method, which is defined as follows:

$$z = \frac{x - \mu}{\sigma}, \tag{3.4}$$

where μ is the mean of the patch data and σ is the standard deviation of the patch observed values. Finally, we compute the covariance matrix of the channels using the samples from the patch. This could be done by reshaping every single patch into $(rc \times d_L)$ before calculating the covariance metrics. The covariance matrix is positive semidefinite and has a symmetric property. As a result, we only require the upper or lower triangle of the covariance matrix to capture the second-order features of each patch rather than the whole matrix. Figure 3.3 summarises the procedure of computing the second-order features from the tensor X_i^L. More details about computing the covariance matrix are illustrated in the following subsection.

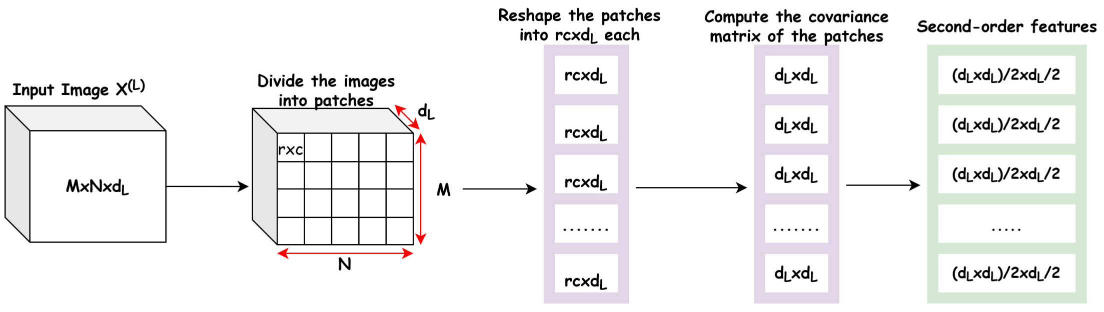


FIGURE 3.3: Example of extracting the second-order features from a tensor X^L.

3.2.2.1 Covariance Computation

For M data points $X \in \mathbb{R}^{M \times d}$, where d is the number of the features and X is drawn from the Gaussian distribution, the covariance matrix of X is defined as follows:

$$\Sigma = \frac{1}{M} \sum_{k=1}^{M} (x_k - \mu)^T (x_k - \mu), \tag{3.5}$$

where μ is the sample mean defined as follows:

$$\mu = \frac{1}{M} \sum_{i=1}^{M} X_i. \tag{3.6}$$

One of the fundamental properties of the covariance matrix is that it is a symmetric non-negative semi-definitive matrix that lies on a Riemannian manifold. Most of the classifiers, including SVM, are developed in the Euclidean space. Therefore, using these classifiers directly to the covariance features is inappropriate. In order to consider the characteristic of the Riemannian manifold, the covariance features should be encoded to the Euclidean space before using the classifiers, e.g. SVM. The mapping to the Euclidean space could be done using the logarithmic matrix [67] or the root square matrix $(\Sigma^{\frac{1}{2}})$ [59, 67, 68]. In general, the square root matrix tends to produce good results compared to the logarithmic matrix [67].

To calculate the square root matrix of (Σ), we first factorise (Σ) using eigen decomposition or singular value decomposition, as in the following:

$$\Sigma = U \Lambda U^T, \tag{3.7}$$

where $\Lambda = \text{diag}(\lambda_i : i = 1, 2, 3, \ldots d)$ is the diagonal matrix with eigenvalues $[\lambda_i : i = 1, 2, 3, \ldots d]$. U represents the orthogonal matrix whose column U_i is the eigenvector that corresponds to λ_i eigenvalue. The square root matrix is then defined as the following:

$$\Sigma^{\frac{1}{2}} = U \, \text{diag}(\lambda_1^{\frac{1}{2}}, \lambda_2^{\frac{1}{2}}, \ldots \lambda_d^{\frac{1}{2}}) U^T. \tag{3.8}$$

The covariance matrix in our method, as mentioned previously, is computed locally in a patch-wise, channel-wise manner. Since a covariance matrix is created for each patch, the number of samples, which corresponds to the number of pixels in a patch, may be smaller than the number of filters. Therefore, the square root matrix cannot be estimated accurately using equation 3.8. To tackle this problem, we use the following equation, as described in [69], to estimate the covariance matrix, whereby $\widetilde{\Sigma}$ is a regularized estimate of $\Sigma^{1/2}$:

$$\widetilde{\Sigma} = U \text{diag}(\delta_i : i = 1, 2, \ldots, d) U^T \ , \ \delta_i = \sqrt{\left(\frac{1-\alpha}{2\alpha}\right)^2 + \frac{\lambda_i}{2\alpha}} - \frac{1-\alpha}{2\alpha}, \tag{3.9}$$

where U is the orthogonal matrix with the eigenvectors as its columns, $(\lambda_i, i = 1 \ldots d)$ are the eigenvalues in decreasing order and α is a regularising parameter that sets to $\frac{1}{2}$ in all experiments, as in [59].

To gain more discriminative information, we combined the first-order statistic (mean) with the second-order statistic, as in [59], using the following symmetric positive definitive matrix:

$$\mathcal{N}(\mu, \widetilde{\Sigma}) \sim \begin{pmatrix} \widetilde{\Sigma} + \mu\mu^T & \mu \\ \mu^T & 1, \end{pmatrix} \tag{3.10}$$

where $\widetilde{\Sigma}$ and μ are the estimated covariance and the sample mean, respectively.

In summary, for any PCA features, we first use equations 3.5 and 3.6 to calculate the mean and covariance of the samples, respectively. We then compute the estimated covariance through equation 3.9. Finally, we combine the mean and the covariance using equation 3.10. The number of features for each convolution layer is $(d_L + 1)(d_L + 2)/2 - 1\times$ the number of patches.

3.2.3 Spatial Pyramid Pooling

Spatial pyramid pooling (SPP) was introduced in [65] to eliminate the requirement for training CNN with a fixed size image input (e.g. 244 $\times$ 244) and therefore enable the CNN to be trained with images of different sizes. The basic idea of SPP is to generate a fixed representation by pooling features in arbitrary regions or sub-images called bins.

In our network, adding SPP is optional. The images have fixed size (e.g. 32 $\times$ 32 $\times$ 3), and the SPP is connected optionally to the second-order pooling (SOP) to reduce the number of features and extract information invariant to complex backgrounds and large poses. As described in Section 3.2.2, the number of features depends not only on the covariance matrix dimension but also on the number of blocks (patches) each tensor is divided into. Attaching SPP to SOP helps to reduce the number of blocks (patches) and, in general, the SOP features.

Figure 3.4 provides an example of using SPP to reduce the number of patch features from 64 to 21 where the features f_i, $i = [1, 2, 3, \dots, 64]$ are the second-order features $f_i \in \mathbb{R}^{\frac{d_L \times d_L}{2} + \frac{d_L}{2}}$ and d_L is the number of responses (filters) in layer (L).

SPP utilises fixed multi-level spatial bins. For instance, in Figure 3.4, we have three-level spatial pyramid pooling with a number of bins equal to 16, 4 and 1, respectively. SPP can consistently maintain spatial information by pooling data in those locally-defined bins whose sizes scale with the number of patches. Thus, for every single bin, we pool the data using max-pooling to get a feature P_i, where $i = [1,\ 2, \dots,$ number of bins$]$. As a result, the number of the features in the example is reduced from $f \in \mathbb{R}^{(\frac{d_L \times d_L}{2} + \frac{d_L}{2}) \times 64}$, where $f = [\{f_i\}_{i=1}^{64}, f_i \in \mathbb{R}^{\frac{d_L \times d_L}{2} + \frac{d_L}{2}}]$ to $P \in \mathbb{R}^{(\frac{d_L \times d_L}{2} + \frac{d_L}{2}) \times 21}$, where $P = [\{P_i\}_{i=1}^{21}, P_i \in \mathbb{R}^{\frac{d_L \times d_L}{2} + \frac{d_L}{2}}]$, and (21) is the total number of bins.

3.2.4 Late Fusion

The output of the intermediate convolutional layers could provide interesting information about the images, e.g. local parts, edges and low-level features. Therefore, fusion strategies are essential for improving the recognition rate and offering precise results [70]. The data fusion aims to combine multiple convolutional layers' outputs for a better performing recognition model, as opposed to using a single layer's output. We can distinguish two fusion approaches, as shown in Figure 3.5: early fusion, where the features extracted by each layer are concatenated before classification, and late fusion, where the layers-wise classification results are merged.

Early fusion, also known as feature-level fusion, relies on fusing the second-order features we obtain from the convolutional layers to create a new representation that is more expressive than the separate representation generated by a single layer. In this kind of fusion, we combine the features first and send the fused features to a classifier to predict the final recognition rate. Table 3.1 describes possible fusion methods to incorporate the layers' features, including concatenation, max pooling and the statistical mean. One disadvantage of using early fusion is that the number of features grows faster if we concatenate the features, and when we use the max/mean fusion method, we are restricted to using the same number of filters for all layers. Applying a selection method to the fused features is sometimes preferable for achieving better recognition results.

Late fusion, also called decision-level fusion, combines the posteriors' probabilities (the decisions) from multiple convolutional layers' classifiers to make a more precise and reliable decision. The methods explained in Table 3.1 can be used to combine the posteriors of the convolutional layers. The late fusion method significantly reduces the number of features compared to the early fusion one. Moreover, researchers [70, 71] have shown that the late fusion strategy could produce a comparable and even better recognition rate than the early fusion method.

In our experiments, we use the late fusion method. In particular, our procedure starts with extracting the second-order features from each convolutional layer separately. Then we average the classifiers' scores and send the fused scores to the support vector machine (SVM) classifier for the final prediction.

TABLE 3.1: Fusion methodology

Fusion method	Definition
Concatenation	$f = [f_1, f_2, \dots, f_N]$, where $f_i \in \mathbb{R}^{d_i}$ $f \in \mathbb{R}^{\sum_{k=1}^{N} d_k}$
Max	$f_k = \max_{i=1}^{d}(f_k^i)$, where $k = 1, \dots, N$ and $f \in \mathbb{R}^N$
Sum	$f_k = \sum_{i=1}^{d}(f_k^i)$, where $k = 1, \dots, N$ and $f \in \mathbb{R}^N$

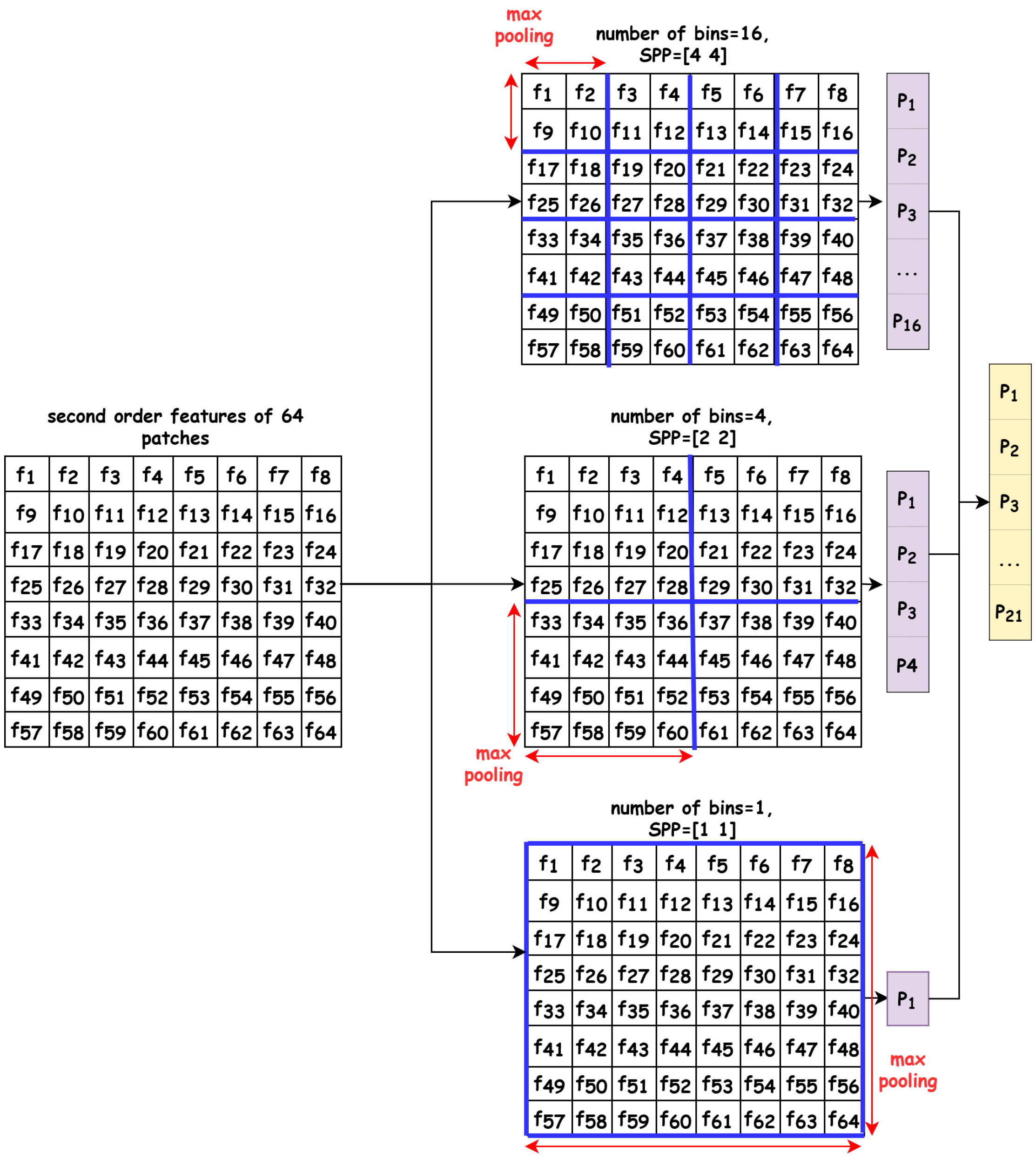

max pooling
number of bins=16, SPP=[4 4]
second order features of 64 patches
number of bins=4, SPP=[2 2]
number of bins=1, SPP=[1 1]
f1 … f64
P1 P2 P3 … P16
P1 P2 P3 P4
P1
P1 P2 P3 … P21

FIGURE 3.4: Example of using SPP to reduce the number of patches from 64 to 21 using 3-level SPP with 16, 4 and 1 bin respectively.

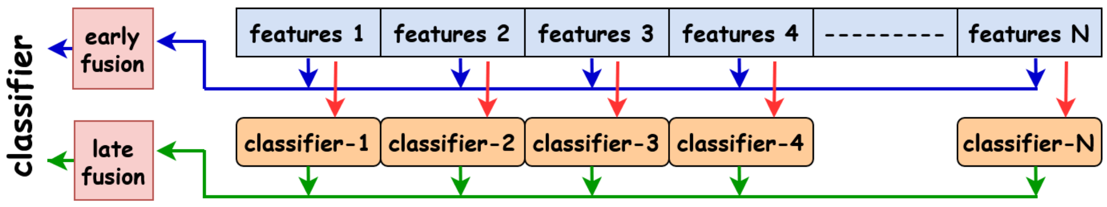


FIGURE 3.5: Late fusion methods versus early fusion methods. Early fusion methods work on the feature space while late fusion ones split the problem into multiple classification problems and combine the classifiers' decisions.

3.3 Experiments and Results

In this section, we start by introducing the four primary datasets we use to evaluate this network (Section 3.3.1). Then, we study the importance of second-order pooling, z-score normalisation and late fusion on classification accuracy (Section 3.3.2). The choice of the network hyper-parameters is also presented in Section 3.3.3, indicating the significant impact of the second-order pooling block size on the recognition rate. Finally, the classification performance of our network using MNIST, CIFAR-10, CIFAR-100 and TinyImageNet is revealed in Section 3.3.4, showing that our network achieves good results, improves on PCANet [15] and is competitive with old neural networks–based methods, but not with recent ones.

3.3.1 Databases

We used four standard benchmarks in our experiments: CIFAR-10 [72], CIFAR-100 [72], MNIST [73] and TinyImageNet [74].

The modified National Institute of Standards and Technology (MNIST) [73] is an open resource dataset consisting of handwritten digits images to test machine learning models while saving effort in terms of preprocessing the images. The database is composed of 60000 training examples and 10000 test images. All images are drawn from the same distribution, normalised and centered in a fixed-size image. All images have a size of 28 × 28 pixels, and their centroid is in the centre of the image. Figure 3.6 shows some examples from this database.

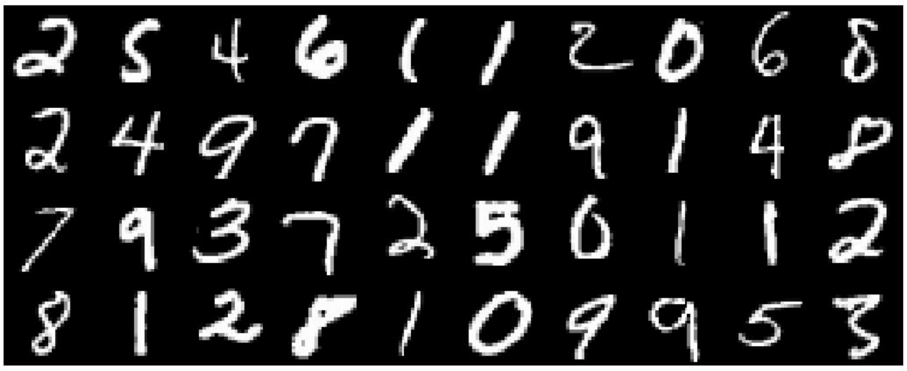


FIGURE 3.6: Mixed examples from the MNIST database.

The CIFAR-10 database [72], named after the Canadian Institute for Advanced Research, which funded the project, consists of 10 classes with 50000 images for training and 10000 test images. The classes include aeroplane, ship, automobile, truck, bird, cat, dog, deer, frog and horse. The database is a balanced set, with 5000 training examples per class. All of the images are coloured and have a size of $32 \times 32 \times 3$. As shown in Figure 3.7, the images are low-resolution with different angles and poses.

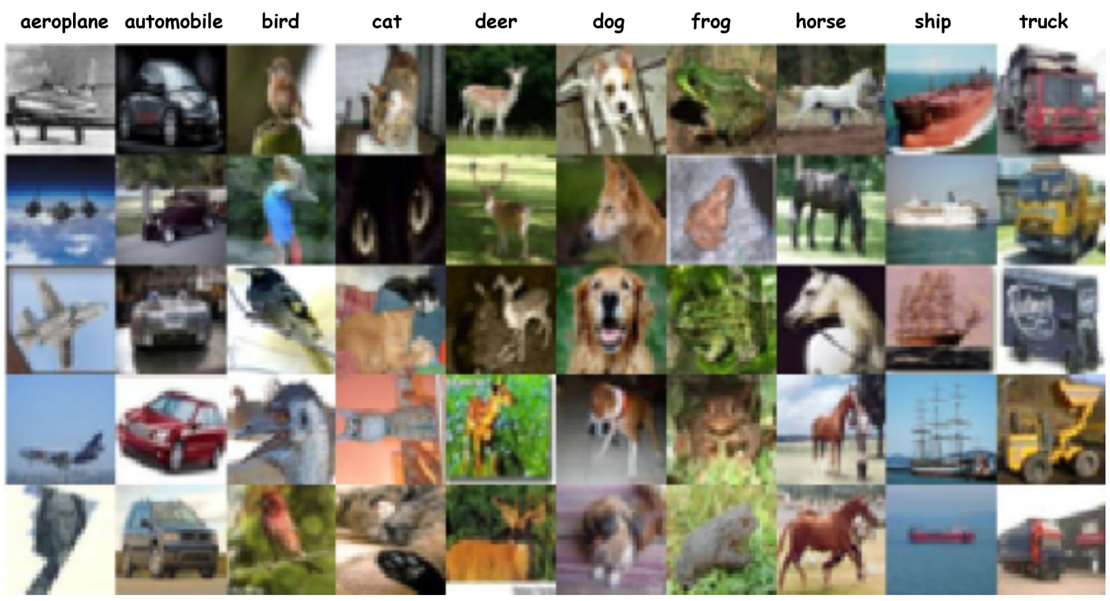


FIGURE 3.7: Some examples from CIFAR-10 database.

CIFAR-100 [72] is similar to CIFAR-10, but with 100 classes. Each class contains 500 images for the training and 100 examples for the testing. The total number of training samples is 50000, while the test set consists of 10000 samples. The 100 classes are grouped into 20 super-classes. Table 3.2 describes the names of the classes and their super-classes. Every super-class consists of five classes, and it is hard to distinguish the classes belonging to the same super-class.

The TinyImageNet database [74] consists of 100000 training images of size $64 \times 64 \times 3$. The images are divided into 200 categories, with 500 images each. The validation and the test sets contain 10000 images each, with 50 images per class. The test set is not labelled, and our experiments' performance is reported on the validation set.

Super-class	**Classes**
aquatic mammals	beaver, dolphin, otter, seal, whale
fish	aquarium fish, flatfish, ray, shark, trout
flowers	orchids, poppies, roses, sunflowers, tulips
food containers	bottles, bowls, cans, cups, plates
fruit and vegetables	c apples, mushrooms, oranges, pears, sweet peppers
household electrical devices	clock, computer keyboard, lamp, telephone, television
household furniture	bed, chair, couch, table, wardrobe
insects	bee, beetle, butterfly, caterpillar, cockroach
large carnivores	bear, leopard, lion, tiger, wolf
large man-made outdoor things	bridge, castle, house, road, skyscraper
large natural outdoor scenes	cloud, forest, mountain, plain, sea
large omnivores and herbivores	camel, cattle, chimpanzee, elephant, kangaroo
medium-sized mammals	fox, porcupine, possum, raccoon, skunk
non-insect invertebrates	crab, lobster, snail, spider, worm
people	baby, boy, girl, man, woman
reptiles	crocodile, dinosaur, lizard, snake, turtle
small mammals	hamster, mouse, rabbit, shrew, squirrel
trees	maple, oak, palm, pine, willow
vehicles 1	bicycle, bus, motorcycle, pickup truck, train
vehicles 2	lawn-mower, rocket, streetcar, tank, tractor

TABLE 3.2: Classes and super-classes of the CIFAR-100 database.

3.3.2 Ablation Study on Multi-Layer PCANet Structure

In this section, we use simple configurations (Table 3.3) on the CIFAR-10 database. These configurations are chosen carefully with different layers, different filter sizes and a different number of filters. So, we can show the effect of these parameters on the accuracy of the proposed network, as discussed in Section 3.3.3. In this section, we use the architectures in Table 3.3 to conduct ablation study on some network components to show their importance. We start by studying the impact of replacing the histogram descriptor (used in PCANet [15]) with second-order pooling. We then analyse the effectiveness of using the late fusion methods over the early fusion ones. More details are described in the following subsections.

3.3.2.1 The Impact of Using Second-Order Pooling and Z-score Normalisation

To show the effectiveness of using SOP features over the histogram descriptor, we tested both by using the configurations described in Table 3.3. Each entry gives the receptive field size and the number of output filters for that layer. For example, configuration A has two layers each with 32 filters generated from a 5×5 window.

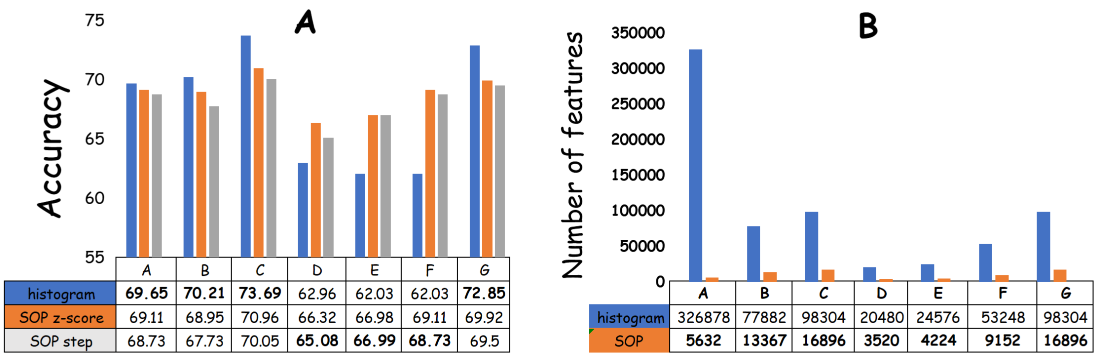
TABLE 3.3: Simple configurations to evaluate the Multi-Layer PCANet on the CIFAR-10 database.

A	**B**	**C**	**D**	**E**	**F**	**G**
Input image $32 \times 32 \times 3$						
5×5 conv-32	3×3 conv-24	3×3 conv-24	5×5 conv-40	5×5 conv-40	5×5 conv-40	5×5 conv-64
5×5 conv-32	3×3 conv-128	3×3 conv-128		5×5 conv-8	5×5 conv-64	5×5 conv-128
		3×3 conv-40				

The method for calculating the histogram descriptor is described in [24]. First, we converted the output of each convolution layer into binary using the Heaviside function. We then combined every eight responses to form a single image using local binary patterns. Next, we divided the resulting images into patches and computed the histogram for each of them. Finally, we concatenated the output histograms of all convolution layers and sent it to a classifier, which is SVM in this experiment. To extract the second-order features, we again converted the filter's responses to binary using the Heaviside function. We then calculated the covariance matrix for the filter's responses as groups of eight after dividing them into patches.

We studied the effectiveness of replacing the Heaviside function by z-score normalisation for the second-order pooling. Therefore, assuming that the output of a convolution layer (L) is $Y \in \mathbb{R}^{m \times n \times d_L}$, where d_L is the number of filters in layer (L) and m and n are the width and the height of the output images, respectively, we first divide each $m \times n$ image into P patches using a specific block size and stride, resulting in $Y = [p_i \in \mathbb{R}^{r \times c}, i = [1, 2, 3, \ldots, P]]_1^{d_L}$. We then vectorise the features of every single patch (p_i) and normalise them using the z-score method. In this experiment, we divided each image into 16 patches with block size = 8 and stride = 8 before calculating the features (e.g., histogram or SOP).

Figure 3.8 shows the accuracy and number of dimensions of the models $(A - G)$ using the three described methods. In terms of accuracy, the results show a mixed picture. The z-score normalisation was always better than the step function and confirmed that more discriminative information was retained. In some configurations, the histogram was better, although the SOP was better in others. The best result was obtained by histograms in configuration C, with a 3% advantage over SOP. However, the histogram method generated approximately six times as many features. Our justification for using SOP is therefore that, while it degrades performance in a specific configuration, the huge reduction in number of features allows us to apply more complex architectures.



	A	B	C	D	E	F	G
histogram	**69.65**	**70.21**	**73.69**	62.96	62.03	62.03	**72.85**
SOP z-score	69.11	68.95	70.96	66.32	66.98	69.11	69.92
SOP step	68.73	67.73	70.05	**65.08**	**66.99**	**68.73**	69.5

	A	B	C	D	E	F	G
histogram	326878	77882	98304	20480	24576	53248	98304
SOP	**5632**	**13367**	**16896**	**3520**	**4224**	**9152**	**16896**

FIGURE 3.8: A) The accuracy of the models $(A - G)$ using histogram descriptors and second-order features. We computed the second-order features after applying either z-score patch normalization (sop z-score) or the Heaviside function (sop step). B) The number of dimensions generated using the models $(A - G)$ after either using the histogram pooling (histogram in the figure) or the second-order pooling (sop). In both sub figures A and B, we used the CIFAR-10 database.

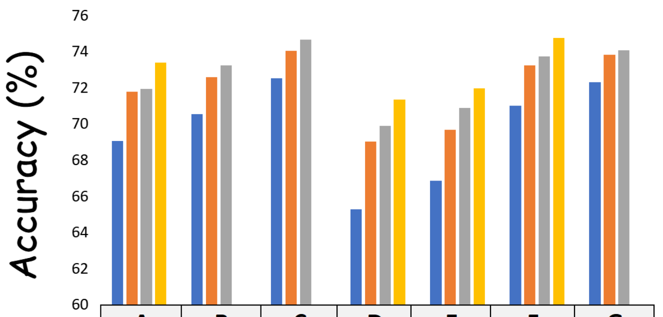
3.3.2.2 Second-Order Pooling for All Feature Maps and Late Fusion

In [15], the number of histogram bins were assigned to 2^{L2}, where $L2$ represents the number of filters in the second stage. This implies that the number of features increases exponentially with the number of feature maps in the second layer ($L2$), limiting the number of feature maps in the second layer to eight. In [24] and [23], the histogram-based pooling and second-order pooling were calculated after every eight filters' responses, and the features were concatenated. In [60, 61, 62], all feature maps were combined to compute the covariance features for the hand-crafted and neural network–based methods.

In this experiment, we computed the second-order pooling for the models in Table 3.3 as combinations of 8, 16 and 24 filter responses. This experiment aimed to show how accuracy would be affected by combining more feature maps. Therefore, we first divided the feature maps of every convolution layer into 16 patches with block size $= 8$ and stride $= 8$. We then normalised each patch using the z-score method. Finally, we calculated the covariance features for each patch of images by dividing them into sets of 8s, 16s or 24s and concatenating the results. The second-order features grew quadratically with the size of the feature maps, and so the number of features extracted from each convolution layer was reduced by PCA while keeping 95% of the data. Linear discriminant analysis was used as the classifier in this experiment.

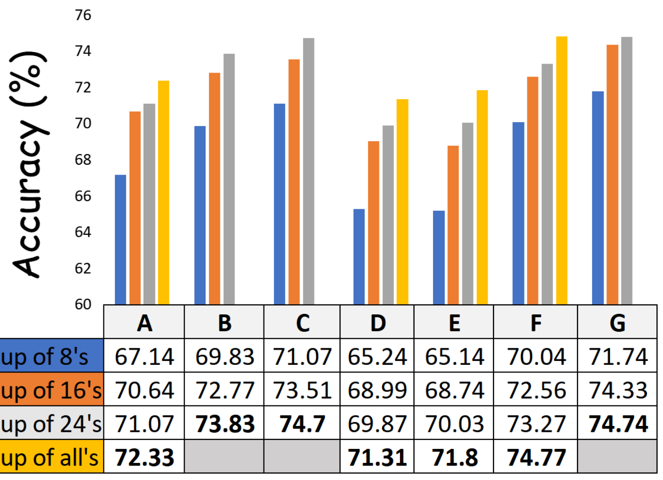
Figure 3.9 presents the accuracy of calculating the covariance features for each 8, 16 or 24 feature maps on the models described in Table 3.3. The figure also displays the accuracy of some configurations (A, D, E and F) when using all filter responses to compute the covariance features. With more feature maps involved in computing the covariance features, the accuracy improves. As a result, we decided to use all filters' responses to compute the covariance features.

Finally, we added the late fusion method, whereby we ran an LDA classifier for each layer and combined the posteriors by concatenating them. This, again, radically reduced the number of features we needed to use at each stage. Figure 3.10 shows the accuracy of using the late fusion method for computing the covariance features for the same models described in Table 3.3. The late fusion method generated comparable or even better results than the early fusion one, with fewer features for each layer.



	A	B	C	D	E	F	G
Group of 8's	69.01	70.52	72.49	65.24	66.83	70.99	72.28
Group of 16's	71.74	72.56	74.02	68.99	69.65	73.21	73.8
Group of 24's	71.9	**73.21**	**74.64**	69.87	70.85	73.71	**74.04**
Group of all's	**73.36**			**71.31**	**71.94**	**74.72**	

FIGURE 3.9: An experiment on the early fusion method with second-order pooling and z-score normalisation for models $A - G$ described by Table 3.3 using the CIFAR-10 database, where features of all layers are concatenated and processed every 8's, 16's, 24's, and all feature maps before being sent to a linear discriminant analysis classifier for final prediction.



	A	B	C	D	E	F	G
Group of 8's	67.14	69.83	71.07	65.24	65.14	70.04	71.74
Group of 16's	70.64	72.77	73.51	68.99	68.74	72.56	74.33
Group of 24's	71.07	**73.83**	**74.7**	69.87	70.03	73.27	**74.74**
Group of all's	**72.33**			**71.31**	**71.8**	**74.77**	

FIGURE 3.10: An experiment with the late fusion method using second-order pooling and z-score normalisation techniques for models $A - G$, as outlined in Table 3.3, using the CIFAR-10 database. The experiment involves running a linear discriminant analysis (LDA) classifier on each layer, followed by concatenating the resulting posteriors from all layers before sending them to a final LDA classifier for prediction. Second-order pooling is calculated every 8, 16, 24, and all feature maps.

3.3.3 The Network Hyper-parameters

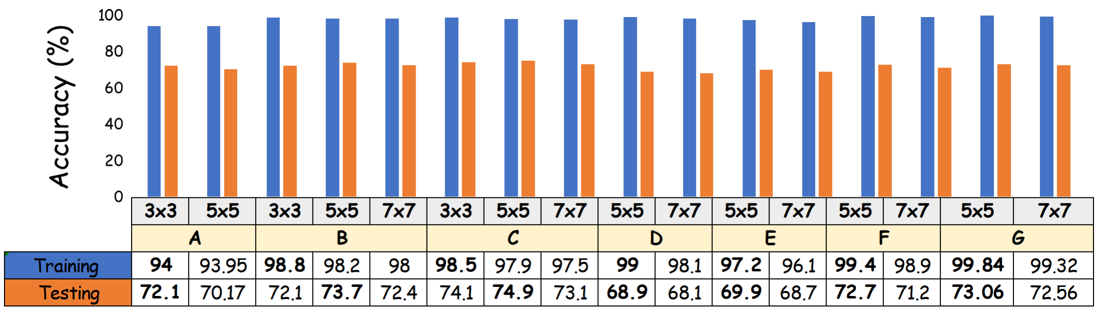
Hyper-parameters in this section refers to the parameters set by the user and related to the network structure. These parameters include the filter size, number of layers, number of filters in each layer and the second-order pooling block size. The network is trained in a layer-wise manner, and so, the number of layers is easy to determine; e.g., if the recognition accuracy does not improve, the training terminates. The following subsections study the effect of the filter size and second-order pooling block size on the classification task accuracy.

3.3.3.1 The Impact of the Filter Size on the Classification Task Performance

We used the CIFAR-10 database to study the impact of the filter size on the models $(A - G)$ described in Table 3.3. The filter sizes used in this experiment were 3×3, 5×5 and 7×7. The min-max normalisation (equation 3.11) was used between the layers, and the second-order pooling block size was fixed to 8×8 with stride=8. SVM was the classifier used for each layer, and the probabilities of the layers were averaged and sent to another SVM.

$$X = \frac{X - \min(X)}{\max(X) - \min(X)}. \tag{3.11}$$

Figure 3.11 shows the accuracy of using different filter sizes on the structures $(A-G)$ defined in Table 3.3. The results stated that the filter size slightly impacts the recognition rate. The accuracy when using filter size 7×7 was the worst in all models. However, this degradation was not significant, e.g., the difference between the best and the worst accuracy in configuration A was around 1.93%. Hence, filter size does not seem to be a significant factor.



	3x3	5x5	3x3	5x5	7x7	3x3	5x5	7x7	5x5	7x7	5x5	7x7	5x5	7x7	5x5	7x7
	A		B			C			D		E		F		G	
🟦 Training	**94**	93.95	**98.8**	98.2	98	**98.5**	97.9	97.5	**99**	98.1	**97.2**	96.1	**99.4**	98.9	**99.84**	99.32
🟧 Testing	**72.1**	70.17	72.1	**73.7**	72.4	74.1	**74.9**	73.1	**68.9**	68.1	**69.9**	68.7	**72.7**	71.2	**73.06**	72.56

FIGURE 3.11: Accuracy of the models $(A-G)$ (Table 3.3) using different filter sizes on the CIFAR-10 database.

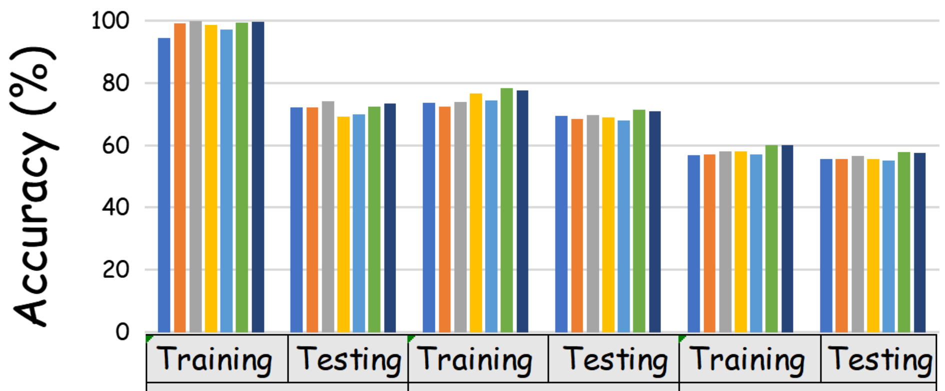
3.3.3.2 The Impact of the Second-Order Pooling Block Size on the Recognition Rate

We used the CIFAR-10 database to test the configurations $(A-G)$ with different block sizes to study the impact of the second-order pooling block size. The experiment was conducted with 8×8, 16×16 and 32×32 block sizes. The stride of the chosen block sizes was equivalent to the size of the block sizes themselves. The classifier used was the SVM.

Figure 3.12 shows that the second-order pooling block size dramatically affects the recognition rate. We found that the smaller the SOP block sizes, the better the model's accuracy was. On the other hand, using large block sizes like 32×32 caused a significant drop in the performance. This indicates that the smaller block sizes can capture more local information than the full image size while increasing the number of features.

3.3.4 Multi-Layer PCANet for Image Classification Task

We investigated the performance of different architectures using the four databases, including CIFAR-10, CIFAR-100, MNIST and TinyImageNet (described in Section 3.3.1). We found that the best performances were achieved using the architectures displayed in Table 3.4. Each entry in Table 3.4 represents the receptive



	8x8		16x16		32x32	
	Training	Testing	Training	Testing	Training	Testing
A	**94.4**	**72.22**	73.62	69.47	56.74	55.6
B	**99.09**	**72.22**	72.4	68.34	57.16	55.56
C	**99.99**	**74.12**	73.86	69.59	58.05	56.5
D	**98.7**	**69.07**	76.55	68.86	58.07	55.67
E	**97.12**	**69.91**	74.46	67.93	57.11	55.14
F	**99.37**	**72.38**	78.38	71.45	60.07	57.85
G	**99.74**	**73.44**	77.67	70.94	59.96	57.66

FIGURE 3.12: Accuracy of the models $(A - G)$ (Table 3.3) using different SOP block-sizes on the CIFAR-10 database.

field size and number of output filters for that layer. The databases were used without data augmentation. The only preprocessing used was the min-max normalisation of the input images (equation 3.11). We set the filter size to 3×3 in all of the experiments, and the classifier used in every layer was the linear discriminant analysis (LDA). The LDA classifier was chosen because of its efficiency, as it generates decision boundaries quicker than other classifiers. The posteriors of all layers were averaged and classified by SVM classifier. Min-max normalisation was also applied between the convolutional layers. The following subsections provide more details about the architectures and their accuracies.

3.3.4.1 Evalutaion of CIFAR-10 Database

The architecture used to evaluate our network on the CIFAR-10 database is described in Table 3.4. The network consisted of eight convolutional layers with 27 filters for the first layer and 50 for the remaining layers. In every layer, the second-order pooling block size was set to 8×8 with stride$=1$. SPP had been attached to every layer with 4×4, 2×2 and 1×1 subregions. The first layer

TABLE 3.4: Configurations for CIFAR-10/100, MNIST and Tiny-ImageNet

MNIST			
Layer Number	Input Size	Filter Size	Output Size
1	$28 \times 28 \times 1$	$3 \times 3 \times 1 \times 9$	$28 \times 28 \times 9$
2	$28 \times 28 \times 9$	$3 \times 3 \times 9 \times 40$	$28 \times 28 \times 40$
3	$28 \times 28 \times 40$	$3 \times 3 \times 40 \times 40$	$28 \times 28 \times 40$
4	$28 \times 28 \times 40$	$3 \times 3 \times 40 \times 40$	$28 \times 28 \times 40$
5	$28 \times 28 \times 40$	$3 \times 3 \times 40 \times 40$	$28 \times 28 \times 40$
6	$28 \times 28 \times 40$	$3 \times 3 \times 40 \times 40$	$28 \times 28 \times 40$
7	$28 \times 28 \times 40$	$3 \times 3 \times 40 \times 40$	$28 \times 28 \times 40$
8	$28 \times 28 \times 40$	$3 \times 3 \times 40 \times 40$	$28 \times 28 \times 40$
9	$28 \times 28 \times 40$	$3 \times 3 \times 40 \times 40$	$28 \times 28 \times 40$
CIFAR-10			
Layer Number	Input Size	Filter Size	Output Size
1	$32 \times 32 \times 3$	$3 \times 3 \times 3 \times 27$	$32 \times 32 \times 27$
2	$32 \times 32 \times 27$	$3 \times 3 \times 27 \times 50$	$32 \times 32 \times 50$
3	$32 \times 32 \times 50$	$3 \times 3 \times 50 \times 50$	$32 \times 32 \times 50$
4	$32 \times 32 \times 50$	$3 \times 3 \times 50 \times 50$	$32 \times 32 \times 50$
5	$32 \times 32 \times 50$	$3 \times 3 \times 50 \times 50$	$32 \times 32 \times 50$
6	$32 \times 32 \times 50$	$3 \times 3 \times 50 \times 50$	$32 \times 32 \times 50$
7	$32 \times 32 \times 50$	$3 \times 3 \times 50 \times 50$	$32 \times 32 \times 50$
8	$32 \times 32 \times 50$	$3 \times 3 \times 50 \times 50$	$32 \times 32 \times 50$
CIFAR-100			
Layer Number	Input Size	Filter Size	Output Size
1	$32 \times 32 \times 3$	$3 \times 3 \times 3 \times 27$	$32 \times 32 \times 27$
2	$32 \times 32 \times 27$	$3 \times 3 \times 27 \times 50$	$32 \times 32 \times 50$
3	$32 \times 32 \times 50$	$3 \times 3 \times 50 \times 50$	$32 \times 32 \times 50$
4	$32 \times 32 \times 50$	$3 \times 3 \times 50 \times 50$	$32 \times 32 \times 50$
5	$32 \times 32 \times 50$	$3 \times 3 \times 50 \times 50$	$32 \times 32 \times 50$
TinyImageNet			
Layer Number	Input Size	Filter Size	Output Size
1	$64 \times 64 \times 3$	$3 \times 3 \times 3 \times 27$	$64 \times 64 \times 27$
2	$64 \times 64 \times 27$	2×2 Max Pooling, stride$= 2$	$32 \times 32 \times 27$
3	$32 \times 32 \times 27$	$3 \times 3 \times 27 \times 70$	$32 \times 32 \times 70$
4	$32 \times 32 \times 70$	$3 \times 3 \times 70 \times 70$	$32 \times 32 \times 70$
5	$32 \times 32 \times 70$	$3 \times 3 \times 70 \times 70$	$32 \times 32 \times 70$

generated 8508 features, while subsequent layers yielded 27,825 dimensions.

Table 3.5 compares the performance achieved by our model with the performance obtained using PCANet-2 and current state-of-the-art models with no data augmentation. Our eight-layers model gained 81.72% accuracy, which is 4.58% better than PCANet with only two layers. However, the current state-of-the-art architectures obtained better accuracy than the one we achieved. We also compared our model to its CNN counterpart model, which has the same structure as our network but is trained using backpropagation. The model was trained by attaching a fully connected layer with ten neurons to the output of the last convolutional layer. When the training had finished, we disconnected the fully connected layer and added second-order pooling, spatial pyramid pooling and an LDA classifier to each convolutional layer. The neural network counterpart of our model had trained for 100 epochs with Adam optimiser and a start learning rate of 0.001. The CNN's counterpart of our model achieved an accuracy of 85.1%, which is 3.39% better than our network. However, our model learns the features faster and is competitive with some simpler deep learning methods.

Our architecture is a simple yet improved version of PCANet. To further illustrate the efficacy of our network, we compared the number of floating point operations (FLOPs [1]) required by our model to those of PCANet and other existing methods during the training phase of these networks. Such comparisons are described in Table 3.5. We re-implemented four residual networks, including ResNet-20, ResNet-32, ResNet-44 and ResNet-56, using the same parameters described by [5], and the reported results are the average of five runs. Our findings demonstrated that our network produced the fewest FLOPs on the CIFAR-10 database without data augmentation. Although our model's accuracy was not comparable to that of the state-of-the-art gradient-based models, it improved more than 4% over the original PCANet and used fewer FLOPs. Overall, the accuracy achieved by our model is promising, considering that it was trained without complex non-linear functions or regularization techniques.

[1]Measured in quadrillions (P)

TABLE 3.5: Comparison of the accuracy (%) of some methods on the CIFAR-10/CIFAR-100 database with no data augmentation. We re-implemented the networks marked with (*).

Method	**CIFAR-10**	**CIFAR-100**	**#FLOPs (P)**
Stochastic pooling [8]	84.87	57.49	0.608
Maxout network [7]	88.32	61.43	–
Network in network [31]	89.59	64.32	–
ALL CNN [9]	90.2	-	14.0
Fractal network [11]	89.82	64.66	–
110 ResNet, reported by [45, 75]	86.82	55.26	7.14
ResNet stochastic depth [45]	-	62.20	–
164-ResNet (pre-activation), reported by [75]	-	64.42	–
Dense network(k=24) [75]	94.08	76.58	625
Dense network-BC (k=24) [75]	**94.81**	**80.36**	–
ResNet-20*	84.442	–	1.29
ResNet-32*	84.664	–	2.1444
ResNet-44*	83.17	–	2.8953
ResNet-56*	83.29	–	3.844
PCANet-2 [15]	77.14	–	0.0166
PCANet-2*	–	51.62	0.0176
PCANet-2 (combined) [15]	78.67	-	–
Multi-Layer PCANet-8	81.72	–	0.00824
Multi-Layer PCANet-5	–	57.86	0.00501
Multi-Layer PCANet-8 CNN counterpart	85.1	–	4.56
Multi-Layer PCANet-5 CNN counterpart	–	56.96	2.49

3.3.4.2 Evaluation on CIFAR-100 Database

We adopted the same network structure used for the CIFAR-10 database, albeit with the number of layers reduced to five. After the fifth layer, no performance gain was noticed; therefore, we stopped adding layers. Table 3.5 compares the accuracy and FLOPs obtained by our model with that of PCANet and previous neural networks-based works with no data augmentation. PCANet accuracy result was achieved by adopting the same model used in [15] for the CIFAR-10 database, albeit with the CIFAR-100 database. The CNN counterpart of our network is a neural network-based model trained with the same structure as our network but with backpropagation. The CNN counterpart of our model is trained using Adam optimiser with an initial learning rate of 0.001, and the number of epochs is 100.

As shown in Table 3.5, we obtained an accuracy of 57.86% on this dataset, which is an improvement of more than 6% compared to PCANet with two layers. Our five-layer model also showed a significant enhancement of 2.60% over ResNet with 110 layers with fewer FLOPs required for training. Moreover, the proposed network generated about the same error rate as the stochastic pooling method [8] and its CNN counterpart but showed a considerable reduction in the number of FLOPs compared to these networks. The dense network obtained the best performance among the other networks, with an error rate of 22% less than the one we achieved. Overall, our approach improved on PCANet-2 in terms of accuracy and number of FLOPs required for training and was competitive against some older CNN-based networks, but not as good as more recent ones.

3.3.4.3 Evaluation on the MNIST Database

In this experiment, we used nine convolutional layers with 9 filters for the first layer and 40 for succeeding layers. The last convolutional layer was connected to a second-order pooling with a 7×7 block size and a stride of 1. The features generated by the second-order pooling were reduced using a three-level SPP of 16,4 and 1 bins. On this database, the accuracy was not improved by using the late fusion technique. Hence the late fusion method was not used in this experiment; instead, the accuracy of the last layer is provided as the performance result.

Table 3.6 displays the accuracy of our model compared to that of PCANet with one and two layers. We obtained the same results as those obtained by PCANet, which indicates that adding more layers to the MNIST database does not necessarily improve accuracy. Table 3.6 also compares the number of features generated by our network to that generated by two layers PCANet and LDANet. Our network produced approximately six times fewer features than the original baseline. This indicates that despite achieving similar performance as PCANet, our network significantly reduces the number of features a classifier needs to handle.

TABLE 3.6: Comparison of the accuracy and number of features of some PCANet methods on the MNIST database with no data augmentation.

Method	Accuracy(%)	#Features
PCANet-1 [15]	99.06	–
PCANet-2 [15]	99.34	73728
LDANet-1 [15]	99.02	–
LDANet-2 [15]	99.38	73728
PCANet-1 ($k = 13$) [15]	99.38	–
Multi-Layer PCANet (ours)	**99.40**	18060

3.3.4.4 Evaluation on TinyImageNet Database

The network structure used in this experiment consisted of five convolutional layers and one max-pooling layer (Table 3.4). The first layer was a convolutional layer with 27 filters followed by a max-pooling layer to reduce the number of dimensions—finally, four convolutional layers with 70 filters each. The second-order pooling attached to each convolutional layer had a block size of 16×16 with a stride of 1. The covariance features of the first convolutional layer were reduced using three-level SPP with 16,4 and 1 bins. A two-level SPP with 2×2 and 1×1 subregions was attached to the subsequent convolutional layers' second-order features. We ran the LDA classifier for each layer, and the SVM classifier made the final prediction on the averaged posteriors. The accuracy reported was in the validation set, since the test set was not labelled. It is important to note that the validation set was utilised only as the test set and was not included in any training procedures. In addition, the first convolutional layer generated 8505 features, while the following convolutional layers produced 9450 dimensions.

Table 3.7 compares the accuracy of our model with those obtained by PCANet-2, ResNet-34 and ResNet-50, as reported by [76], with no data augmentation. The optimal parameters we found for PCANet-2 consisted of a filter size $k_1 = k_2 = 5$, number of filters $L_1 = 30$, $L_2 = 8$ and histogramming block size$=$ 16×16 with overlapping ratio$= 0.5$. We achieved the best error rate with no data augmentation to the best of our knowledge.

TABLE 3.7: Comparison of the accuracy of some methods on the TinyImageNet database with no data augmentation

Method	Accuracy%
ResNet-34 [76]	33.50
ResNet-50 [76]	26.20
PCANet-2	30.00
Multi-Layer PCANet (ours)	**40.87**

3.4 Conclusion

In this chapter, we have presented the Multi-Layer PCANet, which improves the performance of PCANet [15] and reduces the number of features. The network structure relies on three main components and one optional layer. The three main elements of the network include late fusion, multi-channel convolutional layers and second-order pooling. The spatial pyramid pooling is an optional layer applied to reduce the number of dimensions. The z-score normalisation has replaced the binarisation step in the original PCANet.

The experiments section indicates the importance of each component in the network. It shows that replacing the histogram pooling with the second-order pooling generated good performance while reducing the number of dimensions and allowing a deeper structure. The usage of z-score normalisation showed better performance than the Heaviside function. Moreover, we have shown that Multi-Layer PCANet improved performance over PCANet and achieved competitive results with some simpler CNN architectures.

The Multi-Layer PCANet could be presented as a hybrid model that combines the CNN-like filter from PCANet+ [24] with the second-order pooling used in PCANet-II [23], along with other refinements, such as z-score normalization and late fusion. The experimental section did not compare Multi-Layer PCANet to PCANet+ or PCANet-II, as these networks use histogram pooling or 1-D convolution, which generates many features that a classifier cannot handle. For example, using PCANet+ with one layer and 40 filters, we would require to compute 2^{40} histogram features, which is challenging for a classifier to handle.

In summary, the Multi-Layer PCANet has demonstrated better accuracy and fewer FLOPs than the original PCANet, as well as outperforming all gradient-based models in terms of the number of FLOPs required for training. The performance achieved by the Multi-Layer PCANet is promising for a method whereby the features are unsupervised, but this is also a weakness of the architecture because we cannot learn which features are essential for the classification task. The next chapter studies supervised convolutional layers whereby the filters are learned with straightforward closed-form solutions similar to PCANet.

Chapter 4

Stacked Linear Discriminant Analysis (Stacked-LDA)

4.1 Introduction

The work in [15] presented a two-stage network named PCANet for image classification tasks. The filters in this network are obtained with the principal component analysis (PCA) algorithm with convolutional layers to carry out the learning process. Post-processing procedures, such as binary hashing and block-wise histograms, are applied to the extracted features, which can then be used in the final classification step. While PCANet may not have the same capabilities as conventional CNNs, which often use backpropagation to extract features, its accuracy has been proven to be competitive in benchmark datasets. The authors of [15] reported good accuracy in many tasks, including hand digits and face recognition and face verification. Moreover, the network achieved state-of-the-art results with the MNIST dataset.

Inspired by PCANet [15], we presented Multi-Layer PCANet for image classification tasks in the previous chapter. The main contribution of this network is that it increases the depth of PCANet while reducing the computational cost. With the adoption of CNN-like filters [24] and usage of second-order pooling, it is possible to reduce the number of features and subsequently increase the number of convolutional layers. The late fusion method reduces the number of features that the final classifier can handle while providing competitive performance. Z-score normalisation replaces the binary-hashing operation in the original network and provides more stable information. In general, Multi-Layer PCANet achieved satisfactory results in terms of accuracy and computational cost. The network presented better accuracy than the original architecture and was competitive with old CNN-based networks, although not recent ones.

PCANet [15] and Multi-Layer PCANet (Chapter 3) are feature extractor methods that use unsupervised PCA to learn the filters bank without class information. This architecture's weakness prevents the network from learning essential features for the classification task. LDANet, described in [15], incorporates class information by replacing the PCA filters in the original architecture with filters learned using multi-class linear discriminant analysis (LDA). However, the authors of [15] stated that LDANet did not show improved classification performance over its PCANet counterpart. Therefore, in this chapter, we propose the Stacked-LDA model to incorporate class information while maintaining reasonable accuracy over the baseline architecture.

In contrast to PCANet and Multi-Layer PCANet, the supervised Stacked-LDA network is a suitable alternative to the single-layer neural network since it is faster to train and does not need gradient computations to update the network's weights. The proposed algorithm is composed by stacking two LDA classifiers. The first classifier converts the original samples into a higher dimensional space using a set of new labels. The second is then trained to classify the posteriors resulting from the previous layer into the actual classes. Section 4.2 provides a detailed description of the Stacked-LDA algorithm (4.2.1), two possible ways to generate the new labels by considering the classifier's error iteratively (4.2.1) and two experiments on digits recognition using the MNIST database (4.2.2.1 and 4.2.2.2). The results from the experiments show that our model presents good accuracy, reduces training time and compares well to the full-image architectures. The following section (Section 4.3) proposes a new relabelling technique to generate the desired number of classes in a non-iterative form. The proposed approach, which is evaluated on two databases (4.3.2), CIFAR-10 and MNIST, reduces training time and consequently gains performance. Section 4.4 of this chapter presents the convolutional version of the Stacked-LDA network. The network architecture, described in 4.4.1, consists mainly of several convolutional layers followed by a non-linear activation layer, second-order pooling and an optional spatial pyramid pooling layer. The convolutional filters are obtained by training the stacked-LDA algorithm in each layer. The proposed model has been evaluated on four databases (4.4.2): CIFAR-10, CIFAR-100, MNIST and TinyImageNet. The error results in comparison with PCANet and Multi-Layer PCANet show that the proposed architecture has a lower error rate in all of the evaluated datasets except the

MNIST database, as it provides the same performance as the baseline architectures. Finally, we conclude the work of this chapter in Section 4.5.

4.2 General Stacked-LDA Model

4.2.1 Description and Algorithm

Stacked-LDA is a single-layer network, and its design is inspired by how the traditional neural network works with non-linearly separable datasets. Figure 4.1 shows an example of a dataset consisting of two non-linearly separable classes. This binary classification problem involves 200 instances with two input features. A single-layer neural network like the one shown in Figure 4.2 is sufficient to distinguish the classes in this database. This neural network model transforms the two input features into a higher-dimensional space, for example four dimensions. Then the output of the hidden layer's nodes is processed and passed to the last layer, where the final classification decision is made. The number of hidden units can be four, eight or ten; however, the minimum number for representing this dataset is four. The backpropagation method is then used to update the weights concerning the output error. A non-linear activation function, such as the ReLU or sigmoid, is also used between the layers.

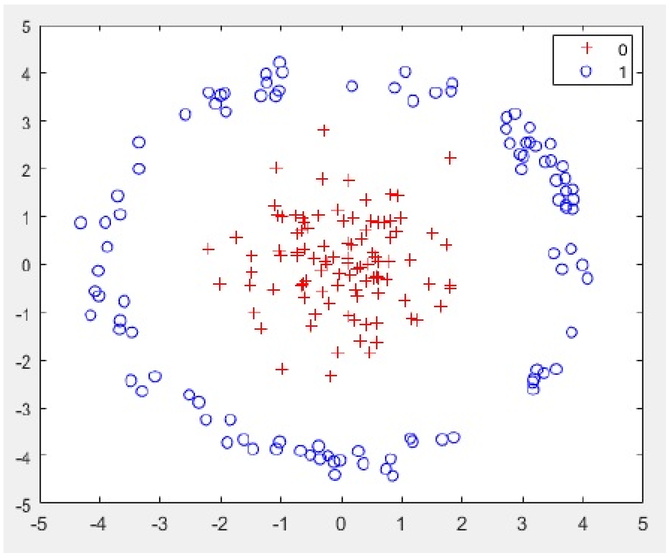


FIGURE 4.1: An example of a non-linearly separable data set.

In Stacked-LDA, we convert the data into a higher-dimensional space considering the output errors, albeit with no gradient descent or backpropagation,

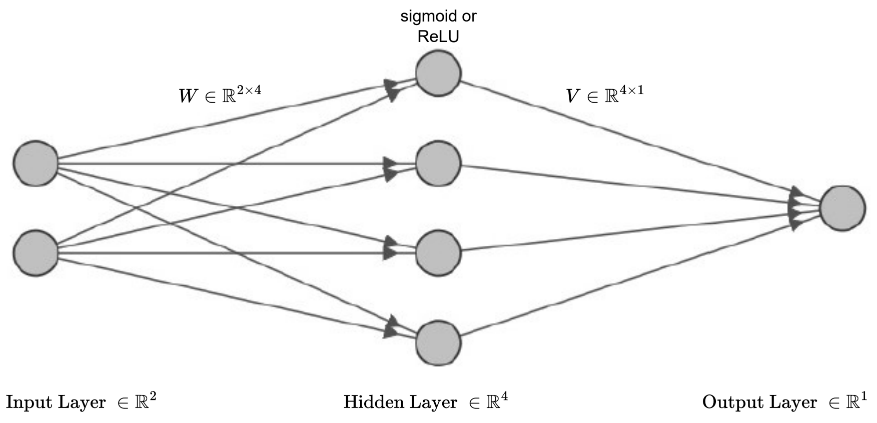

sigmoid or ReLU
$W \in \mathbb{R}^{2\times4}$
$V \in \mathbb{R}^{4\times1}$
Input Layer $\in \mathbb{R}^2$
Hidden Layer $\in \mathbb{R}^4$
Output Layer $\in \mathbb{R}^1$

FIGURE 4.2: Simple feed-forward neural network.

similar to the neural networks. Algorithm 2 describes the procedures for applying the Stacked-LDA model to a set of samples $X \in \mathbb{R}^{N\times M}$ and original classes $Target \in \mathbb{R}^{N\times1}$. We start by applying a regular LDA to the data. In our example, the hyperplane found by the initial LDA is shown in Figure 4.3. Then we assign a new label to each instance based on the actual class and the predicted one, giving up to $(c \times c)$ classes at the next level, where c is the number of the original classes. For example, with a binary classification task with two classes (0 or 1), we get at most four new labels at the next level, such as 0E, 0C, 1E and 1C (check if class 0 is correctly classified as 0 (0C), or incorrectly as 1 (0E)). Figure 4.4 shows the new predicted labels in our example (Figure 4.1). After generating the new labels, an LDA using the original samples and the new labels is trained, equivalent to training the neural network's first layer (Figure 4.2). We then train another LDA using the output of the last LDA as an input and the original classes (equivalent to the last layer of the NN in Figure 4.2). The sigmoid function is applied between the layers to represent the class probabilities. The model works because if the first layer is trained to discriminate the new labelled data, the last layer would recognise the sub-classes that belong to the same original class. For example, suppose we generate new classes '0E', '0C,' '1E' and '1C' from the first layer; in that case, the last layer would learn that all instances belong to '0E' (0 whilst classified incorrectly as 1) or '0C' (0 and classified correctly as 0) belong to class 0, and so on. We proceed by generating more classes or converting to more higher-dimensional space in the same way. However, we take the last generated labels (e.g. four classes) and create other new sub-classes (e.g. 16 classes) by comparing them to the previous labels generated and not to the original ones. Again, we train the last layer's LDA with the original target. This process is repeated, and we validate the model using a validation set. If the error rate is not enhanced, the algorithm terminates.

Algorithm 2 Stacked-LDA

```
Input: Training set: $X \in \mathbb{R}^{N \times M}$, where $\{x_i^N \ , \ x_i \in \mathbb{R}^M\}$, original classes:
    $Target \in \mathbb{R}^{N \times 1}$
Output: $ErrorRate$
 1: $T \leftarrow Target$.
 2: Find the linear discriminant analysis (LDA) between the input X and the
    true classes T: $L_1 = \text{LDA}(X, T)$.
 3: Find the predicted classes: $predicted_y = \text{prediction}(L_1, X)$.
 4: $current\_error\_rate \leftarrow the\ previous\ LDA's\ performance(L_1)$.
 5: $previous\_error\_rate \leftarrow 1$.
 6: while $current\_error\_rate < previous\_error\_rate$ do
 7:     Find the new classes based on both true and predicted labels:
        $new\_classes = label(predicted_y, T)$.
 8:     $T \leftarrow new\_classes$.
 9:     Find the LDA between the original samples and the new classes: $L1 =$
        $\text{LDA}(X, T)$.
    ▷ The name $L1$ is chosen to resemble the training of the first layer of the
    neural network (Figure 4.3).
10:     Find the predicted classes: $predicted_y = \text{prediction}(L_1, X)$.
11:     Find the LDA between the previous LDA's output and the original
        classes: $L2 = \text{LDA}(L_{1_{Outputs}}, Target)$.
12:     $previous\_error\_rate \leftarrow current\_error\_rate$.
13:     $current\_error\_rate \leftarrow the\ last\ LDA's\ performance(L_2)$.
14: end while
15: $ErrorRate \leftarrow previous\_error\_rate$.
```

Since the LDA is a linear classifier for the binary classification task, we extend it to the multi-class classification problem using the one-versus-all decomposition method [77]. This method is chosen because of its simplicity, parallelism and scalable properties [78]. The other methods may not be scalable like the one-vs-one [77], have difficulty finding a split point like hierarchical classifiers [79, 80, 81, 82, 83, 84, 85], need to average over several samples like the weighted one-versus-all [78] or struggle to find strings matrix like the ECOC [86]. Furthermore, we use the posteriors' results from the LDA classifier to determine the nominated classes.

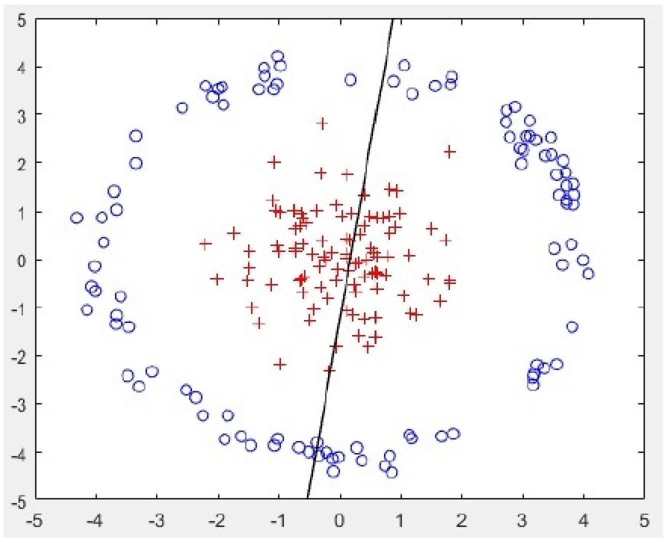


FIGURE 4.3: A single LDA to distinguish the two classes.

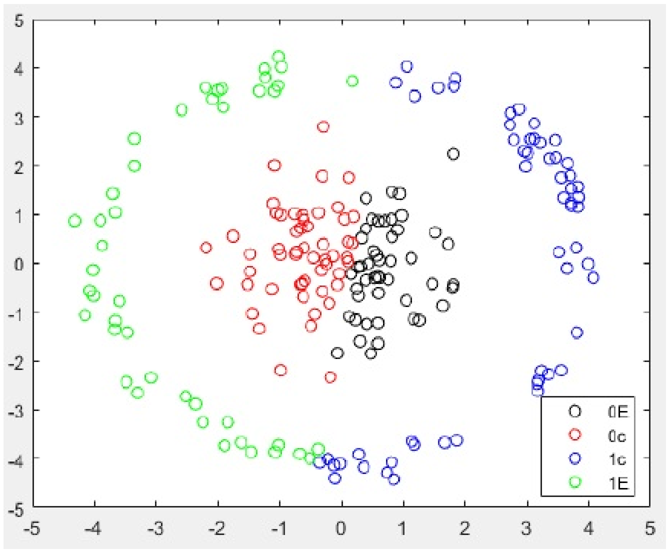


FIGURE 4.4: The new classes generated in the first iteration.

In step 7 of the Stacked-LDA algorithm (Algorithm 2), the labelling process tries to find new labels based on both the true and the predicted classes. In this section, we use two different approaches. The first method relies on assigning a new label to an instance based on the actual class and the predicted classes, giving us at most $c \times c$ new labels at the next level. For example, if we have three classes, such as 0, 1 and 2, we will have nine new classes in the next iteration, namely 00, 01, 02, 10, 11, 12, 20, 21, 22 (0 classified as 0, 1 or 2, and so on). The second method is to assign a new label to an example based on the real class and whether the class is correctly or mistakenly classified, giving up to $2 \times c$ new classes at the following level. For example, for a classification task with three classes, such as 0, 1 and 2, we will get at most six classes in the next iteration, which are 01, 00, 10, 11, 20, and 21 (class 0 correctly classified as 0 (01) or incorrectly classified as 1 or 2 (00)).

4.2.2 Experiments

In this section, we implement two experiments. The first one (Section 4.2.2.1) aims to choose the best decomposition method for allowing the LDA to be extended to the multi-class problem. In the second experiment (Section 4.2.2.2), we evaluate the Stacked-LDA model on the handwritten digits (MNIST) database. The results from the first experiment show that one-versus-all is the best decomposition method because of its scalability, simplicity and parallelism. The second experiment demonstrates the effectiveness of the Stacked-LDA model as a non-convolutional architecture compares well to other full-image methods. It also highlights the impact of the relabelling techniques on the model's performance. We achieved approximately the same performance using two different relabelling approaches. However, we reached the performance faster in the first method with a larger number of classes, while we needed more iterations with fewer classes in the second approach.

4.2.2.1 Multi-Class Classifier

The LDA is a binary classifier. To extend it to the multi-class problem, we need to convert the multi-class problem into multiple binary classification tasks and combine their results. Selecting the proper classifier that decomposes the multi-class problem into many binary classifiers is essential, and it affects the final performance of the Stacked-LDA model. In this section, we use the LDA with multiple decomposition methods, which include one-versus-one [77], one-versus-all [77], one-versus-all with sampling [87], weighing one-versus-all [78] and the hierarchical classification described in [79, 80, 81, 82, 83, 84, 85]. The regulation factor used to solve the singularity problem of the LDA was set to 0.001 for all methods. Moreover, Table 4.1 describes the evaluation metrics used in this experiment.

Table 4.2 shows the LDA's performance with different decomposition methods on the MNIST database. We noticed that the hierarchal classifiers (5 to 9 in Table 4.2) were the worst in terms of accuracy, and their performance varied with the different split points used. The other methods (1 to 4) achieved almost similar results. However, we averaged 100 samples' results to obtain the final

TABLE 4.1: Evaluation metrics

Metric	**Definition**	**Formula**
True positive	The model predicts the positive class correctly.	TP
True negative	The model predicts the negative class correctly.	TN
False positive	The model predicts the negative class incorrectly as a positive class.	FP
False negative	The model predicts the positive class incorrectly as a negative one.	FN
Accuracy	The ratio of the number of correct predictions to the total number of predictions.	$\frac{TP+TN}{TP+TN+FP+FN}$
Recall	The fraction of the true positives to all instances should be classified as positive.	$\frac{TP}{TP+FN}$
Precision	The fraction of true positives to all instances predicted as positive.	$\frac{TP}{TP+FP}$
F measure	The weighted measure between the recall and precision.	$\frac{2 \times recall \times precision}{recall + precision}$
Error rate	The ratio of incorrect predictions to the total number of the dataset.	$1 - Accuracy$
Log loss	The negative average of the log of corrected predicted probabilities for each sample.	$-\frac{1}{N}\sum_{i=1}^{N} log(P_i)$
RSME	The square root of the mean difference between the real and predicted values.	$\sqrt{\frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2}$

output in the weighted one-versus-all and one-versus-all with sample balancing. Moreover, the one-versus-one method is expensive and not scalable, since we need to run K(K-1)/2 binary classifiers, where K is the number of classes. As a result, we choose to use the one-versus-all method, which is more interpretable, scalable and simple.

TABLE 4.2: Different decomposition methods on the MNIST database

Method name	Evaluation metric	Validation set	Test set
1. One vs one	Accuracy	0.8648833	0.8735
	Log loss	0.6852399	0.6540918
	F measure	0.8649321	0.8732151
	Recall	0.8634923	0.8722464
	Precision	0.8663769	0.8741859
	RMSE	1.523091	1.456491
2. One vs all	Accuracy	0.8654001	0.87326
	Log loss	0.6774291	0.6429743
	F measure	0.8650675	0.8727013
	Recall	0.8637264	0.8717764
	Precision	0.8664128	0.873628
	RMSE	1.523114	1.459275
3. One vs all (sample balancing)	Accuracy	**0.8656667**	**0.87405**
	Log loss	**0.3813066**	**0.3650582**
	F measure	**0.865335**	**0.8734523**
	Recall	**0.8641324**	**0.8726922**
	Precision	**0.8665411**	**0.8742135**
	RMSE	**1.520169**	**1.450719**
4. Weighted one vs all reduced to binary classification using costing	Accuracy	0.8653501	0.87334
	Log loss	0.5852362	0.5565949
	F measure	0.8650177	0.8727835
	Recall	0.86367	0.871856
	Precision	0.8663651	0.8737131
	RMSE	1.522441	1.458412
5. Hierarchical classifier scatter parameter $S = \frac{\|\|m_1-m_2\|\|^2}{s_1^2+s_2^2}$	Accuracy	0.7929167	0.80066
	Error rate	0.2070833	0.19934
	F measure	0.7908518	0.798878
	Recall	0.786377	0.7939879
	Precision	0.7953807	0.8038287
	RMSE	1.752254	1.704249
6. Hierarchical classifier, Bhattacharyya distance.	Accuracy	0.8299834	0.8357099
	Error rate	0.1700167	0.16429
	F measure	0.8285081	0.83392
	Recall	0.8272302	0.8328396
	Precision	0.8297904	0.8350035
	RMSE	1.661152	1.632243
7. Hierarchical classifier (centre of gravity)	Accuracy	0.78085	0.78617
	Error rate	0.21915	0.21383
	F measure	0.779009	0.7846838
	Recall	0.7777537	0.7841641
	Precision	0.780	0.7852071
	RMSE	1.91367	1.828576
8. Hierarchical classifier scatter parameter $S = \frac{\|\|m_1-m_2\|\|^2}{cov_1^2+cov_2^2}$	Accuracy	0.82605	0.83098
	Error rate	0.17395	0.16902
	F measure	0.8253298	0.8297455
	Recall	0.822922	0.8281859
	Precision	0.8277534	0.8313118
	RMSE	1.670007	1.621838
9. Hierarchical classifier using LDA's confusion matrix	Accuracy	0.82605	0.83098
	Error rate	0.1801333	0.17358
	F measure	0.8180751	0.8243912
	Recall	0.8173777	0.8244576
	Precision	0.8187745	0.8243259
	RMSE	1.729855	1.708418

4.2.2.2 Digit Recognition on MNIST Dataset

In this experiment, we evaluated the Stacked-LDA model discussed in Section 4.2.1 on the MNIST dataset. We vectorised the images to represent our training samples. To solve the singularity problem of the LDA, we applied PCA to the data followed by an LDA classifier, keeping 95% of our data after applying the PCA to them. The sigmoid function is the non-linear activation used between the layers with a sigmoid scale parameter chosen to be 16, as follows:

$$\text{sigmoid}(x) = \frac{1}{1 + e^{-x/16}}. \tag{4.1}$$

We used the whole $60,000$ training samples with tenfold cross-validation to validate the model. Moreover, we tested our model using the entire $10,000$ test set.

Table 4.3 displays the training and testing accuracies after applying the standard LDA algorithm to the original MNIST database. Tables 4.4 and 4.5 show the results after implementing the Stacked-LDA model on the MNIST using the two different labelling techniques discussed in 4.2.1. Compared with the standard LDA classifier, the Stacked-LDA model saw significantly improved accuracy. The architecture reached accuracy of 96.812%, which was $\sim$ 9% higher than that obtained by the regular LDA classifier. The best performance achieved by each of the two relabelling methods was approximately the same; however, the first method required few iterations with more labels, while the second obtained the same performance with more iterations and fewer classes. Therefore, the relabelling technique is a key factor for obtaining good accuracy by the Stacked-LDA model.

TABLE 4.3: Accuracy (%) after applying a single LDA classifier to the MNIST database

Validation set	**Test set**
86.88	87.47

The Stacked-LDA model's accuracy was good, but it did not provide a state-of-the-art result. In [88], the authors designed a single-layer neural network for the MNIST database. The sigmoid function was the non-linear function used, and the network was trained to minimise the cross-entropy loss function. The researchers varied the number of hidden units from 10 to 50 (Table 4.6) and recorded the error rates. The number of hidden units was equivalent to the

TABLE 4.4: Stacked-LDA model's accuracy (%) on the MNIST database using the first labelling method

Iteration number	Validation set	Test set	Number of classes produced
1	88.672	89.246	98
2	95.3817	95.365	1396
3	96.7583	96.682	4779
4	**96.9183**	**96.812**	**9298**
5	96.6217	96.584	14184

TABLE 4.5: Stacked-LDA model's accuracy (%) on the MNIST database using the second labelling method

Iteration number	Validation set	Test set	Number of classes produced
1	86.78	87.446	20
2	86.948	87.96	40
3	87.845	88.378	80
4	89.978	90.4	160
5	92.42	92.689	299
6	93.6033	93.868	524
7	94.3667	94.56	798
8	94.7967	94.96	1130
9	95.1617	95.367	1526
10	95.4383	95.664	1944
11	95.6667	95.859	2376
12	95.8483	96.041	2839
13	95.975	96.17	3303
14	96.165	96.316	3757
15	96.2767	96.448	4217
16	96.425	96.501	4661
17	96.545	96.558	5096
18	96.5633	96.629	5475
19	96.66	96.68	5827
20	96.67	96.71	6109
21	**96.7383**	**96.739**	**6372**
22	96.7333	96.772	6603

number of classes in the Stacked-LDA model, which achieved better performance. The authors [88] also stated that the error rate could be enhanced with more units, and the best error rate was achieved by the use of the convolutional neural network. Therefore, Stacked-LDA compares well with the full-image neural network models, and to achieve better performance, we should consider the relabelling method and use of the convolution concept.

TABLE 4.6: Accuracy (%) with different hidden units using NN on the MNIST database [88]

Number of hidden units	**10**	**20**	**30**	**40**	**50**
Accuracy	92.1	93.8	94,4	95.1	**95.7**

4.2.3 Section Summary

In this section, we presented the general Stacked-LDA model as a single-layer network motivated by the design of the traditional neural network (Section 4.2.1). Since Stacked-LDA model considers network error without backpropagation or gradient descent, we used the LDA classifier with two relabelling methods that would generate the middle classes based on the incorrectly predicted values iteratively. The experiments section demonstrated the effectiveness of our model, as we improved on the standard LDA with an accuracy of $\sim$9% higher on the MNIST database (Section 4.2.2.2). The importance of the relabelling techniques in achieving good accuracy of the model was also indicated. Moreover, we think it is essential to use the convolutional layers to improve the model's performance.

4.3 New Relabelling Technique

In the previous section (Section 4.2.1), we discussed two relabelling approaches that depend on the LDA's errors. The first method gives a new label to an instance based on the true class and the predicted one, giving up to $c \times c$ new labels at the next level. The second approach assigns a new class to an example based on the actual class and whether the class is accurately or inaccurately classified, giving up to $2 \times c$ new classes at most in the next iteration. For simplicity, we shall name the first approach 'relabelling-I' and the second one 'relabelling-II' for the rest of this chapter .

The relabelling-I and relabelling-II approaches produce the same performance (Section 4.2.2.2) but with more classes in the first method and more iterations in the second one, indicating the significance of the relabelling techniques. The number of classes generated automatically grows exponentially with the relabelling-I method and doubles with each iteration in the relabelling-II method. Therefore, it becomes harder for the LDA to classify the data after a few iterations.

In this section, we describe a new relabelling approach, named 'relabelling-III' for simplicity (Section 4.3.1). We then compare its performance with the previous relabelling methods (Section 4.3.2).

4.3.1 Description of Relabelling-III

The Stacked-LDA (Algorithm 2) is a single-layer network that consists of an input layer, a hidden layer and an output layer. The main network structure relies on running two LDA classifiers. The first of these is applied between the original sample and the hidden layer's units. The second is applied between the previous LDA's output (after applying a non-linear activation) and the original classes. The hidden layer's units are the new labels generated iteratively concerning the previous iteration's errors. If we could immediately find the optimal new labels, the Stacked-LDA model's algorithm would be re-written as described in Algorithm 3.

The relabelling-III technique is inspired by how relabelling-I and II generate the final classes. Figure 4.5 describes an example of how class 5 in the MNIST database is divided using the relabelling-I approach for two iterations. In the first iteration, class 5 with 5421 instances is divided into ten new labels with a different number of instances. In the second iteration, we have shown how the previous new class, number 6, is divided into nine new classes. We notice that the number of instances per class becomes significantly smaller with more iterations. The minimum number of instances to represent a class is one. The relabelling-III technique aims to find such small classes that separate the data perfectly. Our method for finding these classes is simple and described in Algorithm 4.

Algorithm 4, which describes the procedures for applying the relabelling-III technique, starts by picking a random class c from the original classes. We then

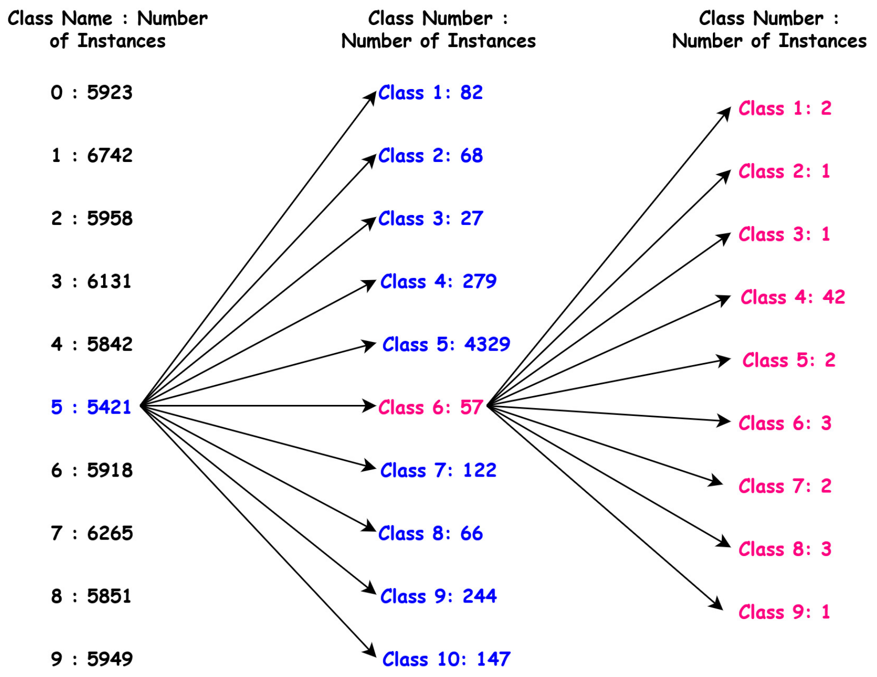


FIGURE 4.5: An example of dividing class 5 from the MNIST database into subclasses for two iterations.

Algorithm 3 Modified Stacked-LDA

```
Input: Training set: $X \in \mathbb{R}^{N \times M}$, where $\{x_i^N\ ,\ x_i \in \mathbb{R}^M\}$, original classes: $Target \in \mathbb{R}^{N\times 1}$, new labels: $new\_classes \in \mathbb{R}^{N\times 1}$
Output: $ErrorRate$
1: Find the LDA between the original samples and the new labels: $L1 = \text{LDA}(X, new\_classes)$.
2: Find the LDA between the previous LDA's output and the original classes: $L2 = \text{LDA}(L_{1_{Outputs}}, Target)$.
3: $ErrorRate \leftarrow the\ last\ LDA's\ performance(L_2)$.
```

choose $N_{positive}$ random instances that belong to class c and $N_{negative}$ random examples that are not in class c, where $N_{positive}$ and $N_{negative}$ are the number of positive and negative samples and are user-predefined parameters. Next, an LDA classifier discriminates between the positive and the negative samples. After that, we check if our chosen random samples are linearly separable, which can be done by comparing the error rate of the LDA classifier with a small value of nearly zero called tolerance (tol) and chosen by the user. If the LDA's error rate is lower than the tolerance, we consider the positive class to be a new class and collect the weights of the LDA. On the other hand, if the LDA's error rate is greater than the tolerance, the chosen samples are not similar and cannot be grouped. Therefore, the algorithm proceeds to find other classes that separate the data accurately in the same way until reaching the required number of classes ($N\_classes$). When algorithm 4 terminates, we can use the generated LDA's weights to find the output of the first LDA (step 1 in Algorithm 3). We then follow the steps in Algorithm 3 to find the performance of the Stacked-LDA model.

Algorithm 4 Relabelling-III method

```
Input: Training set: $X \in \mathbb{R}^{N\times M}$, where $\{x_i^N , x_i \in \mathbb{R}^M\}$, original classes:
    $Target \in \mathbb{R}^{N\times 1}$, number of classes user wants to generate: $N\_classes$, number of positive samples: $N_{positives}$, number of negative samples: $N_{negatives}$
    and tolerance of performance user can afford: $tol$
Output: LDA's weights: $weights \in \mathbb{R}^{N\times N\_classes}$ and LDA's bias or constant:
    $bias \in \mathbb{R}^{N\_classes}$
 1: $weights \leftarrow [\,]$.
 2: $bias \leftarrow [\,]$.
 3: $i \leftarrow 1$.
 4: while $i < N\_classes$ do
 5:     Pick a random class $c$ from the Target.
 6:     Pick random $N_{positives}$ samples from class c ($S_{positives}$).
 7:     Choose $N_{negatives}$ samples that are not in class c ($S_{negatives}$) randomly.
 8:     Combine the negative and positive samples: $S \leftarrow [S_{positives}, S_{negatives}]$.
 9:     $T \leftarrow [ones(N_{positives}), zeros(N_{negatives})]$.
10:     Find the linear discriminant analysis (LDA) between $S$ and $T$: $L =$
        $\mathrm{LDA}(S, T)$.
11:     Find the perfromance ($ErrorRate$) of $S$ using $L$.
12:     if $ErrorRate < tol$ then
13:         $weights \leftarrow [weights, LDA'sweights]$.
14:         $bias \leftarrow [bias, LDA'sbias]$.
15:         $i \leftarrow i + 1$.
16:     end if
17: end while
```

4.3.2 Experiments

In this section, we conduct two experiments on two datasets to compare the performance of the Stacked-LDA model with different relabelling methods. The first experiment, which was described in Section 4.3.3, compared the three relabelling methods (I, II and III) on the MNIST database. The second method discriminated between the relabelling-I and II approaches using the CIFAR-10 dataset (Section 4.3.4).

4.3.3 Digits Classification on the MNIST Database

This experiment aimed to compare the Stacked-LDA model's performance using three relabelling techniques, I (4.2.1), II (4.2.1) and III (4.3.1), on the MNIST dataset. The number of the negative and positive examples ($N_{negatives}$ and $N_{positives}$) in the third relabelling approach was set to 32 and 2, respectively. We chose the number of new labels to be equivalent to those generated previously by the relabelling I and II methods (Tables 4.4 and 4.5). The tolerance (tol) in Algorithm 4 was set to zero. For a fair comparison between the methods, and because we used tenfold cross-validation in the previous experiment (Section 4.2.2.2), we ran the Stacked-LDA with relabelling-III ten times and averaged the accuracies.

Table 4.7 shows the accuracy of the Stacked-LDA using the three labelling techniques, I, II and III, on the test set of the MNIST database. When the number of classes was small, the performance of the Stacked-LDA model using relabelling-III was worse than the performance when using both I and II relabelling techniques. The big difference in the accuracy was $\sim$ 2% between relabelling-I and III with 98 classes. On the other hand, the performance of the Stacked-LDA model with relabelling-III was enhanced by using a more significant number of labels. The best performances reported on the MNIST database in Section 4.2.2.2 were 96.812% using relabelling-I and 9298 classes and 96.772% using relabelling-II and 6372 labels. Using relabelling-III with the same number of classes, we achieved 0.5% better accuracy. Moreover, unlike relabelling-I and II, the model's performance was enhanced with more hidden units, such as 6603 and 14,184, as shown in Table 4.7.

TABLE 4.7: Accuracy (%) on MNIST database using three relabelling methods.

Number of Classes	**Accuracy**		
	relabelling-I	**relabelling-II**	**relabelling-III**
98	**89.246**	—	87.758
1396	95.365	—	**95.793**
4779	96.682	—	**97.023**
9298	96.812	—	**97.399**
14184	96.584	—	**97.61**
160	—	**90.4**	90.28
524	—	93.852	**93.999**
3303	—	96.17	**96.709**
6372	—	96.739	**97.167**
6603	—	96.772	**97.212**

4.3.4 Pattern Recognition on CIFAR-10 Database

This experiment compared the Stacked-LDA's model performance using different labelling techniques (Section 4.2.1 and 4.3.1) on a more challenging database (CIFAR-10). Relabelling-I and II generated the same error rate, albeit with more iterations in the second method. Therefore, in this experiment, we focused on the relabelling-I method. To preprocess the images, we used the preprocessing steps mentioned in Sections 4.2.2.2 and 4.3.3. The accuracy achieved by applying the regular LDA classifier to the training and testing data of CIFAR-10 is shown in Table 4.8.

Table 4.9 displays the accuracy of the Stacked-LDA model on the CIFAR-10 database using relabelling-I and III methods. Using validation sets, the performance of the Stacked-LDA using relabelling-I stopped enhancing after the ninth iteration with 18,402 classes. The two relabelling methods I and III generated almost the same accuracy when the number of classes was 100. With more labels, the relabelling-III method produced better performance than the relabelling-I method. The difference in performance between the two approaches was around 3%.

TABLE 4.8: Accuracy (%) after applying LDA on CIFAR-10

Validation set	**Test set**
40.884	40.425

TABLE 4.9: Accuracy (%) on the CIFAR-10 database using relabelling methods I and III

Number of classes	**Accuracy %**	
	relabelling-I	**relabelling-III**
100	**39.971**	39.834
3476	47.601	**52.77**
10262	51.4445	**55.296**
14115	52.496	**55.848**
16128	53.042	**56.01**
17213	53.235	**56.218**
17840	53.409	**56.224**
18192	53.46	**56.283**
18402	53.485	**56.398**
18519	53.549	**56.41**

4.3.5 Section Summary

In this section, we presented the relabelling-III technique (Section 4.3.1). The advantage of this method over the other relabelling approaches (described in Section 4.2.1) is its ability to generate the desired number of classes in a non-iterative form. The experiments (Section 4.3.2) showed the effectiveness of the new relabelling technique in producing new classes immediately and improving the model's accuracy.

Inspired by the promising results we obtained in Chapter 3, which showed that convolutional layers are highly effective in achieving good classification accuracy, the following section aims to explore the effectiveness of using the Stacked-LDA method to generate filters for these layers.

4.4 Convolutional Stacked-LDA model

The Stacked-LDA model was evaluated on two databases (Sections 4.3.2 and 4.2.2.2), and it showed improved accuracy compared to the original LDA classifier. The architecture reached an accuracy of 97.212% on the MNIST database and 56.41% on the CIFAR-10 database, which was much higher than the accuracy obtained by the standard LDA classifier. The relabelling-III method outperformed the other techniques in terms of accuracy and training speed.

Despite the excellent performance of the Stacked-LDA model, it is still below the accuracy achieved by convolutional architecture such as Multi-Layer PCANet (described in Chapter 3). To overcome this issue, we study in this section how to use the Stacked-LDA model as a convolutional kernel to replace PCA filters in Multi-Layer PCANet (Chapter 3). The first part of this section (Subsection 4.4.1) describes the architecture of the convolutional Stacked-LDA network. The experimental part (4.4.2) compares the performance of the proposed network to Multi-Layer PCANet across four benchmarks. It also studies the impact of the network hyperparameters on the model's accuracy.

4.4.1 Network Structure

Convolutional Stacked-LDA is a shallow depth network, and its structure is shown in Figure 4.6. Like the Multi-Layer PCANet architecture described in Chapter 3, the framework comprises three essential parts and one optional component. The key components include the Stacked-LDA convolutional layer, second-order features and final LDA classifier. Spatial pyramid pooling is an optional step that can be applied between the second-order features and the LDA classifier to reduce the number of dimensions. A non-linear activation function is applied after each convolutional layer, albeit not between the layers. Section 4.4.1.1 describes the Stacked-LDA convolutional layer. The other network components were described in Chapter 3 (Sections 3.2.2 and 3.2.3).

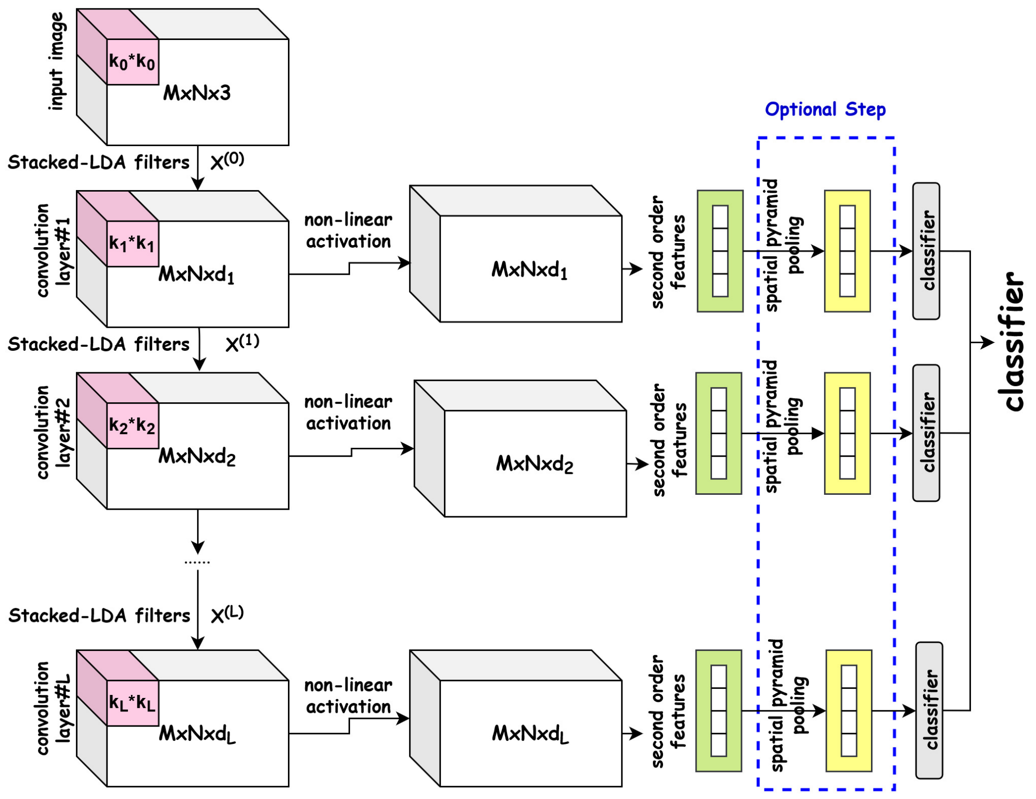


FIGURE 4.6: Convolutional Stacked-LDA model.

4.4.1.1 Stacked-LDA convolutional layer

For N feature maps at a layer $L-1$ ($X^{(L-1)}$: $\{X_i^{(L-1)} \in \mathbb{R}^{m\times n\times c}\}$) with actual classes $Target \in \mathbb{R}^{N\times 1}$, where m and n are the spatial dimensions of the image and c is the number of channels, we compute the Stacked-LDA filters as follows:

1. Given a single image $X_i^{(L-1)} \in \mathbb{R}^{m\times n\times c}$ and a filter size $k_L \times k_L$, we extract and vectorise all overlapping patches of size $k_L \times k_L \times c$ each. The resulting matrix is $Y_i \in \mathbb{R}^{(k_L^2 c)\times \tilde{m}\tilde{n}}$, where $\tilde{m} = (m-k_L)+1$, $\tilde{n} = (n-k_L)+1$ and m and n are the spatial dimensions of the image, respectively. Note that $k_L \times k_L$, which denotes the size of patches, is a user predefined parameter;

2. We repeat the previous step for all images in the dataset to obtain $Y \in \mathbb{R}^{(k_L^2 c)\times \tilde{m}\tilde{n}N}$;

3. We create a vector $T \in \mathbb{R}^{1\times \tilde{m}\tilde{n}N}$ that contains the class labels of the patches. A single patch is assigned a label equivalent to the class of its full image;

4. Using random samples and specific tolerance, we apply Algorithm 4 on Y and T to obtain the Stacked-LDA filters' weights W_s^L and bias B_s^L;

5. We can express the Stacked-LDA filters as follows:

$$\begin{aligned} W_s^L &= \underset{k_L\times k_L\times c}{\text{mat}}\ q_s,\ s = 1,2,\ldots,d_L, \\ B_s^L &= \underset{1\times 1\times c}{\text{mat}}\ q_s,\ s = 1,2,\ldots,d_L, \end{aligned} \tag{4.2}$$

   where d_L is the number of filters chosen by the user, which is equivalent to the number of classes in Algorithm 4;

6. We convolve the original images with the filters as follows:

$$X_i^L = X_i^{L-1} * W_s^L + B_s^L \in \mathbb{R}^{m\times n\times d_L}, \tag{4.3}$$

   where $s = 1,2,\ldots,d_L$ and X_i^{L-1} is zero-padded to get the same image size;

7. After computing the convolutional layer output, we apply a non-linear activation function that converts the data into a non-linear space.

4.4.2 Experiments

4.4.2.1 Network Hyper Parameters

This section aims to study the effect of two network parameters on the Stacked-LDA model's accuracy and compare its performance with Multi-Layer PCANet (described in Chapter 3) on the CIFAR10 database. The two parameters include the number of filters and the number of layers; therefore, we have three main experiments, as follows:

- Experiment 1: testing the impact of the number of filters on the Stacked-LDA network.

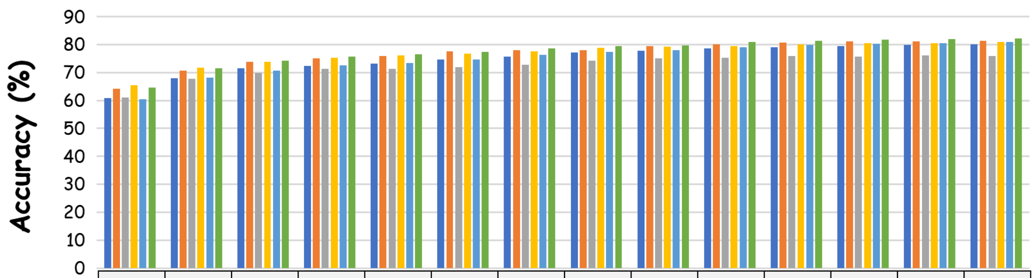
- Experiment 2: studying the effect of the number of layers on the Stacked-LDA network's accuracy.

- Ex[eriment 3: showing the accuracy when combining two types of filters (PCA and stacked-LDA) on the CIFAR-10 database.

4.4.2.1.1 Experiment 1. To study the impact of the number of filters on the Stacked-LDA model's accuracy, we used a simple network consisting of a single convolutional layer followed by ReLU activation, second-order pooling, spatial pyramid pooling and a support vector machine (SVM) classifier. The filter size was set to 5×5, and the number of filters varied between 10 and 75. The second-order block size was set to 8×8 with stride $= 1$. A three-level spatial pyramid pooling was attached to the second-order features with 1×1, 2×2 and 4×4 subregions. We used the LIBLINEAR library described in [89] to classify the features. To compute the Stacked-LDA filters, we set the number of positive samples ($N_{positives}$) to 2, while the number of negative samples ($N_{negatives}$) was 32. The tolerance (tol) in Algorithm 4 was set to zero.

Figure 4.7 compares the accuracy of Stacked-LDA to that of Multi-Layer PCANet on the CIFAR-10 database. Stacked-LDA-1 and PCA-1 in Figure 4.7 represent the accuracy of a single-layer network using two types of filters (Stacked-LDA and PCA) on the CIFAR-10 database. When the number of filters was small, such as 10 filters in Figure 4.7, the recognition rate of the single-layer network using the two types of filters (Stacked-LDA and PCA) was the same. On the other hand, a noticeable improvement in performance using the Stacked-LDA filters could be seen when the number of filters increased. The best performance achieved using the PCA filters was 76.14% with 70 filters. In contrast, we obtained an accuracy of 80.2% with 75 Stacked-LDA filters, which was $\sim 5\%$ higher than the accuracy achieved by the PCA filters. Therefore, the number of filters is an essential factor for affecting the recognition rate in the Stacked-LDA network. When the number of filters increases, more discriminant information is obtained, and the model's accuracy is enhanced. Additionally, the difference in the performance between the PCA and the Stacked-LDA filters becomes higher with more filters.

4.4.2.1.2 Experiment 2. We used a two-level network with the same number of filters in each layer to study the effect of the number of layers on the Stacked-LDA model's performance. The second layer had the same parameters settings as the first layer, as described in Experiment 1 (4.4.2.1.1). The posteriors of the



	10	**15**	**20**	**25**	**30**	**35**	**40**	**45**	**50**	**55**	**60**	**65**	**70**	**75**
Stacked-LDA-1	60.93	67.9	71.5	72.41	73.28	74.65	75.79	77.14	77.81	78.57	79.07	79.46	79.94	80.2
Stacked-LDA-2	64.18	70.65	73.81	75.03	75.9	**77.5**	77.93	78.02	79.56	80.1	80.67	81.09	81.12	81.44
PCA-1	61.09	67.68	69.88	71.38	71.25	71.94	72.83	74.16	75.1	75.36	75.84	75.7	76.14	75.96
PCA-2	**65.39**	**71.68**	73.79	75.3	76.18	76.72	77.68	78.77	79.25	79.57	80.15	80.42	80.59	80.87
LDA-PCA-1	60.41	68.18	70.74	72.61	73.41	74.72	76.36	77.47	78.04	79.08	79.84	80.29	80.6	80.87
LDA-PCA-2	64.56	71.58	**74.16**	**75.78**	**76.45**	77.4	**78.74**	**79.55**	**79.63**	**80.89**	**81.35**	**81.75**	**82.01**	**82.26**

FIGURE 4.7: Accuracy of the CIFAR-10 database using different types of filters and different numbers of layers.

two layers were averaged and sent to SVM for the final prediction.

The accuracy of the two-layer network using two types of filters (PCA and Stacked-LDA) with several filters varied between 10 and 75 is described in Figure 4.7 (see Stacked-LDA-2 and PCA-2). From the results, we observed the following:

- Increasing the depth of the two networks (Stacked-LDA and Multi-Layer PCANet) led to better accuracy of results.
- Multi-Layer PCANet's accuracy increased sharply with more layers. Therefore, the gap in performance between the two networks (Stacked-LDA and Multi-Layer PCANet) shrunk with more layers. The best accuracy obtained by the Multi-Layer PCANet was 80.87%, with 70 filters in each layer. On the other hand, the best performance recorded for the two-layers Stacked-LDA network was 81.44%, which was around 1% better than the one achieved by its PCANet counterpart.
- The performance of the two-layer architectures (Stacked-LDA and Multi-Layer PCANet) was the same when the number of filters was between 10 and 50. With more filters, the recognition rate of the Stacked-LDA tended to increase more than its PCANet counterpart, indicating the importance of the number of filters on the accuracy of the model.

4.4.2.1.3 Experiment 3. The Stacked-LDA network presented high accuracy with a single-layer architecture compared to its PCANet counterpart. On the other hand, Multi-Layer PCANet showed promising results with more layers added. For this reason, we built a filter bank that was a combination of PCA and Stacked-LDA filters and evaluated its performance using single-layer and two-level networks on the CIFAR-10 database. The parameter settings in this experiment were similar to those described in Experiments 1 and 2 (4.4.2.1.1 and 4.4.2.1.2). In this experiment, we used half of the filters as Stacked-LDA filters and the other half as PCA filters. For example, with 15 filters, the filter bank consisted of 7 Stacked-LDA filters and 8 PCA filters.

Figure 4.7 shows the performance of the single-layer and two-layer LDA-PCA network on the CIFAR-10 database. We noticed that the model's accuracy increased gradually with the addition of more filters. The architecture reached an accuracy of 82.26% with two layers and 75 filters, which was about 2% better than the one achieved by PCA filters alone. Therefore, the filter type is an essential factor for enhancing the recognition performance of the convolutional networks.

4.4.2.2 Image Classification on four Databases

This section aims to show the performance of the convolutional Stacked-LDA network on four datasets, MNIST, CIFAR-10, CIFAR-100 and TinyImageNet, described in Chapter 3 (Section 3.3.1). We start by defining the general settings of the network parameters and the image processing procedures (Sections 4.4.2.2.1 and 4.4.2.2.2). We then analyse the performance of the Stacked-LDA model using the four databases and compare it with Multi-Layer PCANet (Chapter 3).

4.4.2.2.1 Image Processing. The databases in this section were used with no data augmentation. To preprocess the images, we only applied min-max normalisation to every image in the database individually. Min-max normalisation, which transforms data into a range between 0 and 1, is defined using equation 3.11.

4.4.2.2.2 Parameter Settings. We investigated multiple architectures to evaluate the convolutional Stacked-LDA network on four databases, namely MNIST, CIFAR-10, CIFAR-100 and TinyImageNet. The best configurations we found for the four datasets are described in Table 4.10. We could not immediately use the architectures described in Table 3.4 in Chapter 3, as we noticed from experiment 1 (4.4.2.1.1) that we could achieve high accuracy with more filters being added. Therefore, we chose to implement these configurations with a larger number of filters. We also used a 3×3 filter size for all databases except for the MNIST, whose filter size was set to 13×13. Moreover, we ran an SVM classifier in each layer, and the posteriors were averaged and sent to a final SVM classifier to obtain the accuracy of the models. The rectified linear activation function (ReLU) was used after each convolutional layer. Moreover, min–max normalisation was applied to the input of each convolutional layer.

TABLE 4.10: Configurations for CIFAR-10/100, MNIST and Tiny-ImageNet.

MNIST			
Layer number	Input Size	Filter size	Output size
1	$28 \times 28 \times 1$	$13 \times 13 \times 1 \times 60$	$28 \times 28 \times 60$
CIFAR-10			
Layer number	Input size	Filter size	Output size
1	$32 \times 32 \times 3$	$3 \times 3 \times 3 \times 200$	$32 \times 32 \times 200$
2	$32 \times 32 \times 200$	$3 \times 3 \times 200 \times 200$	$32 \times 32 \times 200$
CIFAR-100			
Layer number	Input size	Filter size	Output size
1	$32 \times 32 \times 3$	$3 \times 3 \times 3 \times 150$	$32 \times 32 \times 150$
2	$32 \times 32 \times 150$	$3 \times 3 \times 150 \times 100$	$32 \times 32 \times 100$
TinyImageNet			
Layer number	Input size	Filter size	Output size
1	$64 \times 64 \times 3$	$3 \times 3 \times 3 \times 40$	$64 \times 64 \times 40$
2	$64 \times 64 \times 40$	2×2 Max Pooling, stride$= 2$	$32 \times 32 \times 40$
3	$32 \times 32 \times 40$	$3 \times 3 \times 40 \times 90$	$32 \times 32 \times 90$
4	$32 \times 32 \times 90$	$3 \times 3 \times 90 \times 90$	$32 \times 32 \times 90$
5	$32 \times 32 \times 90$	$3 \times 3 \times 90 \times 90$	$32 \times 32 \times 90$

To compute the stacked-LDA filters (Algorithm 4), we set the number of negative samples ($N_{negatives}$) to 32, the number of positive examples ($N_{positives}$) to 2 and the tolerance (tol) to zero in all experiments.

4.4.2.2.3 Performance Analysis. In the following subsections, we discuss the architectures described in Table 4.10 for each database (CIFAR10, CIFAR-100, MNIST and TinyImageNet) and analyse their performance compared with other methods, such as Multi-Layer PCANet (Chapter 3).

4.4.2.2.3.1 Digits Recognition on the MNIST Database. The model used in this experiment consisted only of a single layer with 60 filters, as described in Table 4.10. Adding more filters or layers did not improve the accuracy of this database. To extract the second-order features, we used a 7×7 second-order pooling block size with stride $= 7$.

Table 4.11 compares the accuracy of the Stacked-LDA network with PCANet [15], LDANet [15], Multi-Layer PCANet (Chapter 3) and its single-layer PCANet counterpart. The single-layer PCANet had the same structure and parameter settings as the Stacked-LDA network. However, we replaced the Stacked-LDA filters with the PCA filters. From the results in Table 4.11, the Stacked-LDA network achieved 0.1% greater accuracy than its single-layer PCANet counterpart. In fact, the architecture provided the same accuracy as the other models, including PCANet, LDANet and Multi-Layer PCANet. However, the Stacked-LDA reached this accuracy with fewer features than PCANet and LDANet and with fewer layers than Multi-Layer PCANet.

TABLE 4.11: Accuracy of the Stacked-LDA network compared with different methods on the MNIST database with no data augmentation.

Method	**Accuracy (%)**
PCANet-1 [15]	99.06
PCANet-2 [15]	99.34
LDANet-1 [15]	99.02
LDANet-2 [15]	99.38
PCANet-1 ($k = 13$) [15]	99.38
Multi-Layer PCANet	**99.40**
Stacked-LDA	99.39
Single-layer PCANet	99.29

4.4.2.2.3.2 Testing on the CIFAR-10 Dataset. To evaluate the Stacked-LDA network on the CIFAR-10 database, we used a two-layer network with the number of filters $= 200$ in each layer, as shown in Table 4.10. In each layer, the second-order pooling block size was set to 8×8 with a stride $= 1$. A three-level spatial pyramid pooling was attached to each layer's second-order pooling with 4×4, 2×2 and 1×1 subregions.

Table 4.12 shows the performance of the Stacked-LDA network compared with methods such as ResNet, PCANet [15], Multi-Layer PCANet (Chapter 3) and NoFilterNet on the CIFAR-10 database. To show the effectiveness of the convolutional layers on the model's performance, we proposed the NoFilterNet architecture. The proposed architecture has the same structure as the

TABLE 4.12: Accuracy of the Stacked-LDA network compared to some methods on the CIFAR-10 and CIFAR-100 databases with no data augmentation.

Model	**#Params**		**Accuracy %**	
	CIFAR-10	**CIFAR-100**	**CIFAR-10**	**CIFAR-100**
Residual networks reported by [90]				
ResNet-18	11.05M	10.96M	86.29	59.15
ResNet-34	21.07M	21.16M	**87.97**	56.05
PCANet [15] and Multi-Layer PCANet (Chapter 3)				
PCANet-2	2.16M	21.52M	77.14	51.62
Multi-Layer PCANet	2.18M	12.06M	81.72	57.86
No convolutional layers used				
NoFilterNet	0.09M	0.85M	65.68	39.35
Stacked-LDA descibed in this chapter				
Stacked-LDA-1	1.08M	10.81M	81.73	56.88
Stacked-LDA	8.89M	35.05M	84.55	**59.41**

Stacked-LDA model but without the convolutional layers and is a single-layer model. Therefore, we divided the original images (after zero-padding them) into patches of size 3×3 each, obtaining 27 dimensions to replace the filters in the Stacked-LDA model. Then we applied second-order pooling, spatial pyramid pooling and the SVM classifier similar to the Stacked-LDA network.

As shown in Table 4.12, the NoFilterNet achieved the worst result, with an accuracy of 65.68%, indicating the importance of the filters for gaining good accuracy. The other networks, such as ResNet, PCANet and Multi-Layer PCANet, presented good accuracy. The best accuracy achieved by the Stacked-LDA model was 84.55%, and it improved by 3% on Multi-Layer PCANet and 7% on PCANet [15]. However, the Stacked-LDA model presented a higher number of parameters, namely $4\times$ the number of parameters in PCANet and Multi-Layer PCANet. For this reason, we tested the Stacked-LDA on a single-layer architecture with only 100 filters; this is named Stacked-LDA-1 in Table 4.12. Stacked-LDA-1 provided the same accuracy as Multi-Layer PCANet, albeit with fewer trained parameters. The best accuracy result in Table 4.12 was 87.97%, which was achieved by ResNet-34 and was 3% higher than the Stacked-LDA model but with $2\times$ the number of parameters. Therefore, the accuracy results demonstrate the effectiveness of the Stacked-LDA filters in the classification problem because they include discriminative information, which can explain the good accuracy results.

4.4.2.2.3.3 Testing on the CIFAR-100 Dataset. The architecture used to evaluate the Stacked-LDA network on CIFAR-100 consisted of two layers, as shown in Table 4.10. The first layer had 150 filters, while the latter had 100 filters. As with the CIFAR-10 database, we used an 8×8 second-order pooling block size with a stride of 1. A spatial pyramid pooling with three levels was used to pool each layer's second-order features with the number of bins $= 16$, 8 and 1, respectively.

Table 4.12 compares the accuracy of the Stacked-LDA with other convolutional architectures on the CIFAR-100 database. The NoFilterNet was trained using the settings described in 4.4.2.2.3.2, albeit with CIFAR-100 images. The results showed that the Stacked-LDA network provided the best accuracy, at around 2% better than Multi-Layer PCANet, 9% better than PCANet, 3% higher than ResNet-34 and about the same performance as ResNet-18. However, the number of the trained parameters of the Stacked-LDA model was much higher than those of the other architectures. Therefore, we trained a single-layer Stacked-LDA model with 100 filters to reduce the number of parameters (Stacked-LDA-1 in Table 4.12). The single-layer Stacked-LDA model showed good accuracy of 56.88%, at around 1% lower than Multi-Layer PCANet, albeit with significantly reduced trained parameters. Again, the NoFilterNet obtained the worst accuracy, showing the implication of the convolutional filters for enhancing the performance of the networks.

4.4.2.2.3.4 Testing on the TinyImageNet Dataset. As shown in Table 4.10, the model used to evaluate the Stacked-LDA on TinyImageNet consisted of five layers. The first layer was a convolutional layer with 40 filters. The second layer was a max-pooling layer with a block size $= 2 \times 2$ and stride $= 2$. The use of the max-pooling layer reduced the spatial dimensions of the images and thus reduced the processing time. The last three layers were convolutional layers with 90 filters each. We used a 16×16 second-order pooling block size with a stride of 1 for every convolutional layer to extract the second-order features. A three-level spatial pyramid pooling with 4×4, 2×2 and 1×1 subregions was used to pool the second-order features of the first layer. For the remaining layers, we used a two-level spatial pyramid pooling with 2×2 and 1×1 subregions.

Table 4.13 compares the accuracy of the Stacked-LDA model with other convolutional networks, including ResNet, PCANet and Multi-Layer PCANet, on the TinyImageNet database. In comparison with PCANet, the Stacked-LDA model obtained better accuracy with a notable reduction in the number of trainable parameters. The Stacked-LDA also improved on the Multi-Layer PCANet by around 2%, albeit with more parameters. The two residual networks obtained similar accuracy results, at around 1.5% better than the Stacked-LDA model. However, the number of parameters in ResNet-34 was significantly larger than that obtained by the Stacked-LDA. In general, the Stacked-LDA model provided good results on par with other convolutional structures with no data augmentation.

TABLE 4.13: Comparison of the accuracy of some methods on the TinyImageNet database with no data augmentation.

Method	Accuracy%	#Params
ResNet-18 [90]	**43.02**	11.15M
ResNet-34 [90]	42.65	21.26M
PCANet-2	30.00	32.26M
Multi-Layer PCANet (ours)	40.87	9.47M
Stacked-LDA	41.76	16.34M

4.5 Conclusions

In this chapter, we first proposed a novel Stacked-LDA algorithm as an alternative for the single-layer neural network. The new architecture, constructed by stacking two LDA classifiers, relies on dividing the data into clusters and considering the classification errors iteratively. The experimental results indicate the effectiveness of our algorithm, as it enhanced by more than 9% the standard LDA classifier and was on par with other full-image architectures. We then studied the effect of the relabelling techniques on the Stacked-LDA model's performance and introduced the relabelling-III method. Relabelling-III provided better accuracy results and did not require many iterations to adjust the weights of the hidden layer. We followed this by presenting a convolutional architecture with filters being trained using the Stacked-LDA algorithm. Like Multi-Layer PCANet, the convolutional Stacked-LDA model includes three key components: (i) a convolutional layer with filters being learnt through the use of the Stacked-LDA method; (ii) a ReLU non-linear processing layer; and (iii) a second-order pooling layer followed optionally by a spatial pyramid pooling layer. We tested the proposed architecture on four benchmarks: MNIST, CIFAR-10, CIFAR-100 and TinyImageNet. The results showed that the convolutional Stacked-LDA model outperformed Multi-Layer PCANet in image classification by 3% on CIFAR-10 and by 2% on CIFAR-100 and Tiny-ImageNet. Moreover, the proposed network obtained accuracy that was 3% higher than ResNet-34 on the CIFAR-100 database. Despite the good accuracy of the Stacked-LDA network, the number of parameters increased gradually. However, we showed that a single-layer Stacked-LDA obtained the same accuracy results as Multi-Layer PCANet and with fewer parameters.

Based on the scenarios and the preliminary experiments in this chapter, we can draw the following conclusions:

- The Stacked-LDA algorithm can be used as a classifier to classify data, and the two LDAs can be replaced with other types of classifiers, including SVM;
- To increase data representation capacity, we need to use many Stacked-LDA filters, as our experiments showed that using a small number of filters is not sufficient to provide high accuracy;
- Increasing the number of layers in the convolutional Stacked-LDA network leads to better accuracy compared to single-layer architecture. However, this enhancement is not significant when compared with using PCA filters, as PCA filters seem to take more advantage of network depth than the Stacked-LDA network;
- In general, the convolutional Stacked-LDA network shows good accuracy results, which are comparable with architectures such as ResNet and Multi-Layer PCANet, indicating the efficiency of the network.

In the following chapters, and based on our findings, we highlight the following directions:

1. Strategies for incorporating more convolutional layers while maintaining or improving accuracy, training speed, and computational cost;
2. Development of novel convolutional architectures inspired by traditional deep learning designs;
3. Optimised implementation of a clustering technique to enhance the accuracy of the Stacked-LDA network;
4. Investigating the possibility of replacing backpropagation with nonlinear approaches for updating convolutional layer weights.

Chapter 5

Class-Embedding Networks

5.1 Introduction

Although PCANet [15] and Multi-Layer PCANet [1] achieve high accuracy on a variety of classification benchmarks, PCA is not designed to maintain the relationship between classes, which is often essential for pattern recognition tasks; hence it is possible that these networks may not extract discriminative features. To address this problem, the supervised stacked linear discriminant analysis algorithm (Stacked-LDA) (Chapter 4) is used to produce the filter banks. This algorithm incorporates class information by grouping some patches while ensuring that the LDA classifier can discriminate between the new groups. The Stacked-LDA network presents the following advantages over PCANet, Multi-Layer PCANet and LDANet [15]:

- In a Stacked-LDA network, there is no restriction on the number of filters that we can choose for any given filter size. On the other hand, the number of filters in the other architectures, namely PCANet and Multi-Layer PCANet, cannot exceed the maximum number of eigenvectors of the image-based patches' covariance matrix. The number of filters in the LDANet is also limited to the maximum number of eigenvectors of the scattering matrix (the ratio of within-class to between-class scatter matrix);

- In contrast to PCANet, Multi-Layer PCANet and LDANet, the Stacked-LDA weights may represent more discriminative information about the same object by considering the classification errors;

- When compared against multiple classification benchmarks (CIFAR-10, CIFAR-100, TinyImageNet and MNIST), the Stacked-LDA network provided the best accuracy results and significantly improved on PCANet and Multi-Layer PCANet.

According to [30], the accuracy of different shallow-depth structures varies depending on the database employed. In order to determine the best generalisable architecture for the classification tasks, this chapter investigates four supervised architectures for feature extraction and learning tasks. Multi-layer PCANet (Chapter 3) was chosen as the fundamental framework for the proposed architectures. All supervised networks follow the same concepts used by Multi-Layer PCANet and use a similar architecture; however, the networks' filter banks are produced by different supervised approaches; hence, every network generates unique features. In other words, the primary objective of this chapter is to investigate four supervised strategies, which preserve class information, for producing different filter banks to improve the feature representations. The first network, called the clustering network, was inspired by the Stacked-LDA network, but it groups the images' patches based on the spectral clustering algorithm. This network studies if alternative grouping techniques improve the Stacked-LDA network results. Despite producing discriminative features, the Stacked-LDA network's filters are linear. To add nonlinearity, we propose a supervised Laplacian eigenmaps network (SLE) to generate filters that map the input into a nonlinear space based on the classification labels. The SLE network generates new filters based on the separation criteria in which the distance between samples from different classes is maximised and the distance within the class is minimised. The Hilbert–Schmidt independence criterion network (HSIC Net), which uses the information criteria to increase the dependency on the labels while compressing the inputs, is also presented. The weights of the HSIC network are calculated using a linear kernel-based approach. The final network, known as the supervised extreme learning machine (S-ELM), attempts to compress the image patches' features while preserving discriminative information. The filter bank is obtained by training the extreme learning machine autoencoder with both image-based patches and their corresponding classes.

Therefore, this chapter provides the following contributions:

- We develop four new filter banks based on spectral clustering, supervised Laplacian eigenmaps, the Hilbert–Schmidt independence criterion and supervised extreme learning machine auto-encoder. In contrast to PCANet, these filters are generated using discriminant information obtained from supervised data;
- We propose four semi-supervised networks that resemble the supervised

networks, namely SLE, HSIC, S-ELM, and clustering networks, with the difference being that the filters of each supervised technique are coupled with PCA filters at a 50% ratio in the semi-supervised networks versions;

- We present a new supervised network developed by integrating the convolutional outputs of the four proposed networks, namely clustering, SLE, S-ELM and HSIC networks, with 25% of each type of filter.

In summary, this chapter is structured as follows. Section 5.2 gives a detailed description of the proposed networks (clustering, SLE, S-ELM and HISC). The section starts with explanations of preliminary concepts such as spectral clustering (Section 5.2.1.1), extreme learning machine (Section 5.2.1.2), supervised laplacian eigenmaps (Section 5.2.1.3) and Hilbert Schmidt independence criterion (Section 5.2.1.4). Then we describe the networks' architecture and the procedures for generating the filter banks in each network, namely clustering architecture (Section 5.2.3.1), SLE (Section 5.2.3.3), S-ELM (Section 5.2.3.2) and HSIC (Section 5.2.3.4). Section 5.3 summarises the accuracy results of the four proposed networks using four benchmarks: CIFAR-10, CIFAR-100, MNIST and TinyImageNet. The experiments include: (i) evaluating the four proposed networks on the four databases with a single-layer network architecture (Section 5.3.3); (ii) testing the new networks on the four classification benchmarks with a two-layer architecture (Section 5.3.4); and (iii) evaluating the proposed networks using a combination of filters regardless of their learning paradigm with single-layer and two-layer network architectures (Section 5.3.5). In each experiment, the proposed networks are compared with Multi-Layer PCANet (Chapter 3) and Stacked-LDA (Chapter 4). Finally, Section 5.4 presents the conclusions and a discussion.

5.2 Deep Supervised Networks

In this section, we propose four supervised shallow-depth frameworks (clustering network, S-ELM Net, SLE Net and HSIC Net). The networks have a unified architecture, similar to the one described in Chapter 4, composed of several convolutional layers. Each layer is followed by non-linear activation, second-order pooling, an optional spatial pyramid pooling and a classifier. The final decision is made by combining the posteriors of all convolutional layers. Figure 5.1 describes the conceptual framework for all of these supervised networks. The difference between the four architectures is in the filter learning process, as each network produces different local receptive fields and, accordingly, different features. Besides employing class information to increase accuracy, each network has a specific objective that can be described in the following paragraphs:

The clustering network is conceptually related to the Stacked-LDA network described in the previous chapter. As described in Chapter 4, the Stacked-LDA network consists of two linear discriminant analysis layers: the first is trained using new labels derived from the original ones, while the second is trained with the actual classes. Additionally, in Chapter 4, we demonstrated the Stacked-LDA network's effectiveness and the impact of the relabeling strategies on its accuracy. The clustering network discussed in this chapter is a special case of the Stacked-LDA network; however, the spectral clustering method is used as the relabeling technique. The spectral clustering algorithm [91] involves mapping the data into a new embedding based on the similarities between the samples and then dividing the embedding data points into clusters. Other clustering approaches might be utilised; however, the clustering network aims to determine whether using clustering techniques as the relabeling methods in the Stacked-LDA would improve its performance. In other words, the clustering network tries to identify an alternative technique to group image-based patches using spectral clustering and investigate whether this clustering method would improve Stacked-LDA accuracy.

All previously studied networks, including PCANet [15], Multi-Layer PCANet [1], and Stacked-LDA (Chapter 4), use linear filters created by a linear transformation of the data, as in the PCANet or Multi-Layer PCANet networks, or by a linear classifier, such as the LDA, in the Stacked-LDA architecture. Consequently, the primary objective of the SLE and HSIC networks is to explore non-linear approaches to design the filter banks and investigate if such filters are effective for providing good representations and enhancing the network's performance. Precisely, the SLE network relies on the use of supervised Laplacian eigenmaps [92] to generate non-linear embedding of image-based patches. This embedding is generated using the separation criterion, which maximises the distance between samples from different classes while minimising it within the class. We then use an extreme learning machine [17] to learn the weights that map the image-based patches into the non-linear embedding space; these weights serve as SLE filter banks. HSIC network, on the other hand, employs information criterion rather than separation criterion to maximise dependency on the labels while compressing input data.

Extreme learning machine (ELM) is an alternative training approach for neural networks that offers less computational complexity than gradient descent and backpropagation [17]. The ELM was specifically designed to train a single hidden layer feed-forward neural network, and it consists of three basic steps: random projection, non-linear transformation, and regression model [93]. The low computational cost of the ELM has attracted the interest of several academics, particularly those dealing with high-dimensional and huge data sets [94]. The ELM is then generalised to deep neural networks such as autoencoders. The extreme learning machine autoencoder (ELM-AE) efficiently used the straightforward procedures of the original ELM to learn compressed representations [19]. The S-ELM network relies on using ELM-AE to extract features. However, the network aims to use a supervised version of the ELM-AE to learn compact representations while maintaining class information.

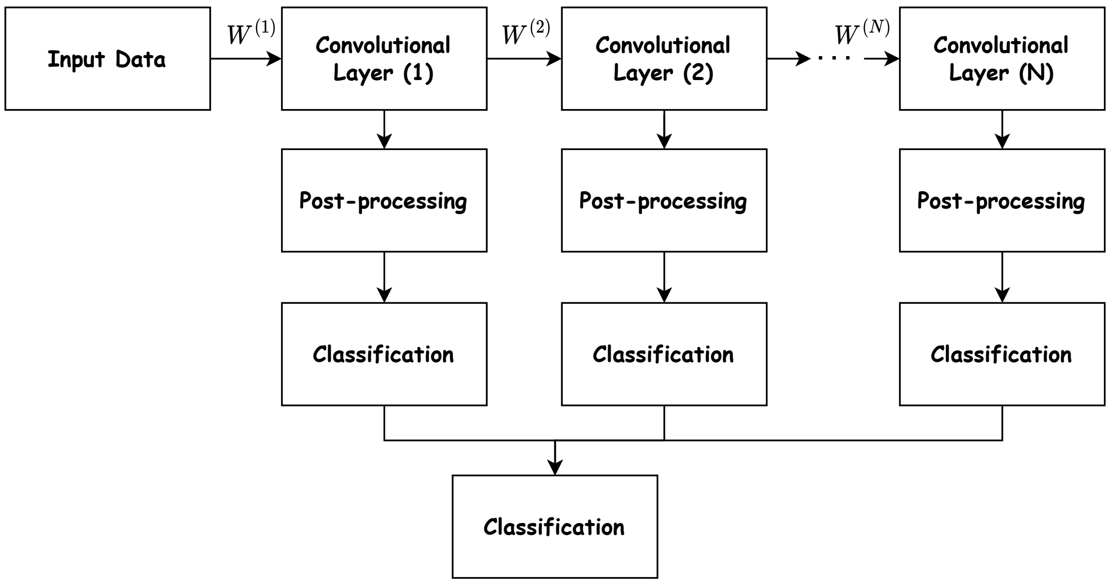


FIGURE 5.1: Conceptual illustration of the proposed shallow depth networks. The architecture consists of several convolutional layers that learn filter banks through supervised methods. The post-processing step involves applying a non-linear function to convert the data into a non-linear space, followed by second-order pooling. To reduce the high dimensionality of the features, the second-order pooling can be followed by a spatial pyramid pooling. Then we apply a classifier to the output of each layer and combine the classifier's posteriors to make the final decision.

The content of this section is organized as follows. We start by describing preliminary concepts such as spectral clustering (Section 5.2.1.1), extreme learning machine (Section 5.2.1.2), supervised laplacian eigenmaps (Section 5.2.1.3) and Hilbert Schmidt Independent Criteria (Section 5.2.1.4). Then, we describe the procedures for generating the filter banks in all architectures, including clustering network (Section 5.2.3.1), S-ELM Net (Section 5.2.3.2), SLE Net (Section 5.2.3.3) and HSIC Net (Section 5.2.3.4). The problem settings are also described in Section 5.2.2.

5.2.1 Preliminary Principles

5.2.1.1 Spectral Clustering

Spectral clustering, described by Algorithm 5, is a popular unsupervised technique that usually outperforms traditional clustering algorithms such as k-means [91]. The algorithm training involves three main steps: (i) constructing a graph; (ii) calculating the Laplacian matrix in the defined neighbourhood; and (iii) finding K eigenvectors of the Laplacian matrix to split the graph into K clusters. A set of data points $\{x_i\}_1^N, x_i \in \mathbb{R}^d$ can be represented as a graph $G = (\mathcal{V}, \mathcal{E})$, where the vertices $\mathcal{V}$ represent the data points and the edges $e_{ij} \in \mathcal{E}$ connect two vertices v_i and v_j with a specific weight, using one of the following constructions:

- ϵ-neighbourhood graph, where all data points with distances smaller than a value called ϵ are connected;
- k-nearest neighbour graph, which connects two vertices v_j and v_j with an undirected edge if v_i is among the k-nearest neighbours of v_j;
- a fully-connected graph, which connects all pairwise data points using a predefined similarity measure s_{ij}.

To define the edges connecting vertices in a graph, including ϵ-neighbourhood, k-nearest neighbour or a fully-connected graph, pairwise distances $||x_i - x_j||$ between all points i and j in the neighbourhood are computed. These distances are then transformed into similarity measures using a kernel transformation as follows:

$$s(x_i, x_j) = \exp(-\frac{||x_i - x_j||^2}{2\sigma^2}), \tag{5.1}$$

where σ defines the width of the neighbourhood.

Following the computation of the graph's similarity matrix, we compute the unnormalised Laplacian matrix as explained in step 3 of Algorithm 5. It is also possible to use the normalised versions of the Laplacian matrix as described in step 4 of Algorithm 5. Next, we create the new embedding $Y \in \mathbb{R}^{N\times K}$, where its columns are the K eigenvectors corresponding to the K smallest eigenvalues of the Laplacian matrix. Using the normalised Laplacian matrix described in [95], we normalise each row of the eigenvectors to have a unit length (step 7 in Algorithm 5). Finally, treating each row of $Y \in \mathbb{R}^{N\times K}$ as a point, we divide the N points into K clusters using the k-means method.

Algorithm 5 Spectral Clustering

```
Input: Data points: $X \in \mathbb{R}^{N\times d}$, where $\{x_i^N\ ,\ x_i \in \mathbb{R}^d\}$; number of clusters: $K$
Output: $K$ clusters $(C_1, C_2, \ldots, C_K)$
 1: Define the local neighbourhood for each point $x_i$ in the dataset using $\epsilon$-neighbourhood, k-nearst neighbourhood or a fully-connected graph.
 2: Construct $(N \times N)$ affinity matrix $S$, where $s_{ij}$ denotes the similarity between the points $x_i$ and $x_j$ in the neighbourhood.
 3: Calculate the unnormalised Laplacian matrix as $L = D - S$, where the degree matrix $D$ is defined as the row sum of the affinity matrix: $D_i = \sum_j S_{i,j}$.
 4: Optionally, normalise the Laplacian matrix $(L)$ by one of the following equations:
    $L = D^{-1}L$ [96],
    $L = D^{-\frac{1}{2}}LD^{-\frac{1}{2}}$ [95].
 5: Compute the largest $K$ eigenvalues $(\lambda_k)$ and eigenvectors $(v_k)$ of $L$.
 6: The embedding of each point $(x_i)$ is the vector $(y_i)$, with $(y_{ik})$ representing the $i^{th}$ element of the $k^{th}$ principal eigenvector $v_k$ of $L$.
 7: When using the normalised equation [95] to normalise the Laplacian matrix, $Y = \{y_i\}_1^N, y_i \in \mathbb{R}^K$ should be normalised as the following:
    $y_{ij} = y_{ij} / \left(\sum_k y_{ik}^2\right)^{\frac{1}{2}}$.
 8: Cluster the points $(y_i)$ $i = 1, \ldots, N$ with the k-means algorithm into $K$ clusters $(C_1, \ldots, C_K)$.
```

5.2.1.2 Extreme Learning Machine

5.2.1.2.1 Description and Algorithm An ELM is a fast training single-layer feedforward neural network (SLFN) that avoids gradient calculations to update the network weights. Figure 5.2 describes the basic architecture of the ELM mechanism. Unlike gradient-based networks, the hidden node parameters are assigned randomly and never updated during the network's training.

The output layer weights are usually learnt by the least-square method in a single step [19].

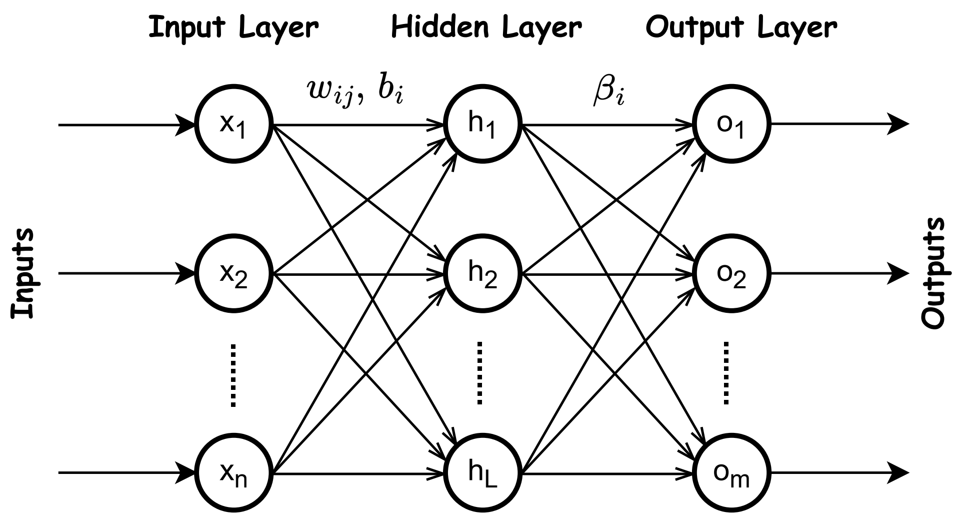


FIGURE 5.2: Structure of single-layer feedforward neural network.

Algorithm 6 describes the procedures of training a single-layer feedforward network through the ELM method. Given a training set $S = \{(x_i, t_i) : x_i \in \mathbb{R}^{1\times n}, t_i \in \mathbb{R}^{1\times m}, \text{ and } i = [1, 2, \dots, N]\}$, where x_i represents the inputs and t_i denotes the desired target, the algorithm starts by calculating the output of the hidden layer with L neurons, as follows:

$$h_j = g(\sum_{i=1}^{n} x_i w_{ij} + b_j), \ \ j = [1, 2, \dots, L], \tag{5.2}$$

where $g(x)$ represents a non-linear activation function, w_{ij} and b_i are the hidden layer's weights assigned randomly and fixed during the training and h_j is the network response at the single hidden layer node. The ELM aims to minimise the mean square error (MSE) between the actual targets and the ELM outputs, as follows:

$$\text{Minimise}||H\beta - T||^2, \tag{5.3}$$

where $T = [t_1, t_2, \dots, t_m]$ represents the desired targets, $H = [h_1, h_2, \dots, h_L]$ is the hidden layer's outputs and the output weights β can be computed using the Moore–Penrose inverse of matrix H, as follows:

$$\beta = H^{\dagger}T. \tag{5.4}$$

The authors of [17] investigated an alternative method that was better for calculating β by introducing a regularisation parameter C, as follows:

$$\beta = \begin{cases} H^T \left(\frac{I}{C} + HH^T\right)^{-1} T & \text{if } N < L \\ \left(\frac{I}{C} + H^T H\right)^{-1} H^T T & \text{otherwise,} \end{cases} \tag{5.5}$$

where L denotes the number of the hidden layer's nodes, N is the number of training examples and I is the identity matrix.

Algorithm 6 Training SLFN through the ELM learning method

```
Input: Set of inputs: $X \in \mathbb{R}^{N \times n}$, where $\{x_i^N , x_i \in \mathbb{R}^n\}$; Targets: $T \in \mathbb{R}^{N \times m}$,
    where $\{t_i^N , t_i \in \mathbb{R}^m\}$; number of hidden nodes: $L$
Output: $\beta_i$, where $i = [1, 2, \dots, m]$

    Initialisation:
    1. Generate random weights ($W \in \mathbb{R}^{n \times L}$, $B \in \mathbb{R}^{1 \times L}$) between the input and
    hidden layers ($w_{ij}$, $b_i$ in Figure 5.2).
    2. Calculate the outputs of the hidden layer: $H = XW + B$.
    3. Apply a non-linear activation function to the hidden layer's outputs: $H =$
    $g(H)$.
    Analytical solution:
    Use Moore–Penrose inverse of H to find the weights between the hidden and
    the output layers ($\beta$): $\beta = H^{\dagger} T$.
```

5.2.1.2.2 ELM Autoencoder ELM is extended to ELM autoencoder (ELM-AE), which works with unsupervised feature representations. Figure 5.3-a describes a single-layer ELM-AE architecture that modifies the original ELM structure by using the input data as the output data. The random parameters of the hidden layer (w_{ij}, b_i) are chosen to be orthogonal, according to [18]. As with conventional auto encoders, the ELM-AE is composed of encoder and decoder parts. The encoder maps the data points $X \in \mathbb{R}^{N \times n}$ into a non-linear space using random weights and bias (equation 5.2). The decoder maps the output of the hidden layer ($H \in \mathbb{R}^{N \times L}$) back to $X \in \mathbb{R}^{N \times n}$ through the output weights $\beta \in \mathbb{R}^{L \times n}$, which can be estimated using either equation 5.4 or 5.5. The ELM-AE can learn different representations of the data input, which include: (i) compressed features if $L < n$; (ii) equal dimension feature representation if $L = n$; and (iii) sparse representation if $L > n$. For any new data point $x_i \in \mathbb{R}^{1 \times n}$, the new representation can be found as follows, which is also described in Figure 5.3-b.

$$H_{new} = X\beta^T \tag{5.6}$$

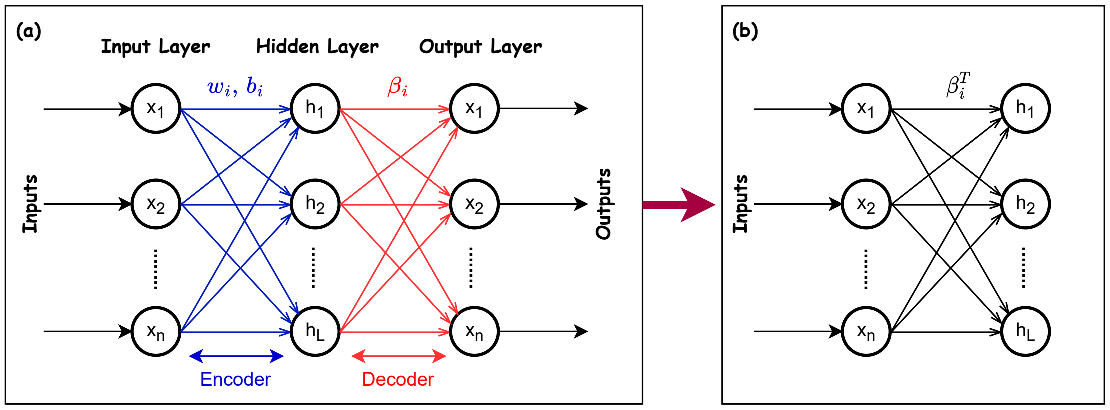


FIGURE 5.3: Structure of single-layer ELM-AE.

5.2.1.3 Supervised Laplacian Eigenmaps

SLE is a type of manifold learning [92]. Given N data points $\{x_i\}_{i=1}^N$ in a high dimensional space $\mathbb{R}^p$, the SLE aims to map the data points $X \in \mathbb{R}^{N\times p}$ into a non-linear lower-dimensional space $Z \in \mathbb{R}^{N\times d}$, where $d \ll p$. The non-linear embedding obtained by SLE should maximise the separation between samples from different classes while minimising the distances between examples from the same class. The following equation defines the objective function of the SLE:

$$\begin{aligned} \min_Z \quad & \mathrm{tr}(Z^T L_w Z) \\ \max_Z \quad & \mathrm{tr}(Z^T L_b Z) \\ \text{s.t.} \quad & Z^T D_w Z = I, \end{aligned} \tag{5.7}$$

where:

- $L_b \in \mathbb{R}^{N\times N}$ denotes the between-class Laplacian matrix, which is defined as follows:
$$L_b = D_b - W_b; \tag{5.8}$$
- $L_w \in \mathbb{R}^{N\times N}$ represents the within-class Laplacian matrix, which is defined as
$$L_w = D_w - W_w; \tag{5.9}$$
- The weights matrices W_w and W_b can be defined for a set of classes T, as the follows:
$$W_w(i,j) = \begin{cases} \exp(-\frac{||x_i - x_j||^2}{\sigma^2}) & \text{if } T_i = T_j \\ 0 & \text{Otherwise,} \end{cases} \tag{5.10}$$

$$W_b(i,j) = \begin{cases} 1 & \text{if } T_i \neq T_j \\ 0 & \text{Otherwise;} \end{cases} \tag{5.11}$$

- D_w and D_b are the within-class and between-class degree matrices that are defined as

$$D_w = \sum_{k=1}^{N} W_w(i,k) \quad i = [1, 2, \ldots, N], \tag{5.12}$$

$$D_b = \sum_{k=1}^{N} W_b(i,k) \quad i = [1, 2, \ldots, N]; \tag{5.13}$$

- I is the identity matrix of size $N \times N$.

- The constraint ensures that Z has unit norm rows and is orthogonal with respect to the within-class degree matrix defined by D_w.

The optimal solution for the optimisation problem (equation 5.7) using the eigendecomposition, according to [92], is described as

$$Bz = \lambda D_w z, \tag{5.14}$$

where $B = \gamma L_b + (1-\gamma) W_w$, γ is a user-predefined parameter that is used to balance equation 5.14, W_w is within class similarity matrix defined by equation 5.10, L_b is the between-class Laplacian matrix defined by equation 5.8 and λ represent the eigenvalues in descending order.

5.2.1.4 Hilbert–Schmidt Independence Criterion

5.2.1.4.1 HSIC Description HSIC is a kernel-based measure that relies on estimating the Hilbert–Schmidt norm of the cross-covariance operator between the distributions in reproducing kernel Hilbert spaces [97]. The following formula describes the formulation of the HSIC in two separable Hilbert spaces $\mathcal{H}$ and $\mathcal{G}$:

$$\begin{aligned} \text{HSIC}(\mathbb{P}_{XY}, \mathcal{H}, \mathcal{G})) &= ||C_{XY}||^2 \\ &= \mathbb{E}_{XYX'Y'}[K_X(X,X')K_{Y'}(Y,Y')] \\ &+ \mathbb{E}_{XX'}[K_X(X,X')]\mathbb{E}_{Y'}[K_Y(Y,Y')] \\ &- 2\mathbb{E}_{XY}[\mathbb{E}_{X'}[K_X(X,X')]E_{Y'}[K_Y(Y,Y')]], \end{aligned} \tag{5.15}$$

where $\mathbb{P}_{XY}$ denotes probability distributions over $(\mathcal{X} \times \mathcal{Y})$, K_X and K_Y are kernel functions and $\mathbb{E}_{XY}$ is Expectation over X and Y.

Given a set of m independent data points $\mathcal{S} = \{(x_1, y_1), \ldots, (x_m, y_m\}$ drawn from $\mathbb{P}_{XY}$, the empirical estimation of HSIC can be described by the following equation:

$$\mathrm{HSIC}(\mathcal{S}, \mathcal{H}, \mathcal{G}) = (m-1)^{-2}\,\mathrm{tr}(K_X H K_Y H), \tag{5.16}$$

where $K_X, K_Y \in \mathbb{R}^{m \times m}$ represent the kernel functions of X and Y, respectively, and H is the centring matrix that is defined as follows:

$$H = I_m - \frac{1}{m} 1_m 1_m^T, \tag{5.17}$$

where $I \in \mathbb{R}^{m \times m}$ is the identity matrix.

5.2.1.4.2 Information Bottleneck Given the two random variables X and Y, where X represents the input and Y is the class labels, information bottleneck (IB) aims to find the minimal representation that is independent of the dimensions by capturing as much mutual information between X and Y as possible [98].

The BI objective function with respect to the conditional distribution $p(t|x)$ can be expressed as follows:

$$\min_{p(t|x)} I(X;T) - \beta I(X;Y), \tag{5.18}$$

where the Lagrange multiplier β controls the trade-off between the compression and preserving more information about X (with large β, we attain more information about X, while small β implies more compressed data), and T is the hidden representation. The mutual information between two random variables X and Y with joint distribution $p(x, y)$ is defined as follows:

$$I(X;Y) = H(X) - H(X|Y), \tag{5.19}$$

where H is the entropy that measures the information or randomness in X. In general, the IB objective function (5.18) tries to preserve the information of the hidden representations about the label and compress information about the input data.

5.2.1.4.3 HSIC Bottleneck The authors of [99] proposed a training mechanism named HSIC training to train deep neural networks without backpropagation. HSIC training relies on replacing the mutual information in the IB optimisation problem (equation 5.18) with a normalised version of HSIC, as follows:

$$\min_{\theta_i} \text{nHSIC}(T_i(Z_{i-1}, \theta_i), X) - \text{nHSIC}(T_i(Z_{i-1}, \theta_i), Y), \tag{5.20}$$

where:

- X and Y are the input data and the class labels, respectively;
- T_i is the transformation applied to the current layer;
- Z_{i-1} is the output of the previous hidden layer;
- nHSIC is the normalised Hilbert–Schmith criterion (information criteria), which can be defined from independent and identically distributed (i.i.d.) samples $\mathcal{D} = \{(x_i, y_i) : [i = 1, \ldots, m], \text{where: } x_i \in \mathbb{R}^{d_x} \text{ and } y_i \in \mathbb{R}^{d_y}\}$, as follows:

  $$\text{nHSIC}(\mathcal{D}, \mathcal{H}, \mathcal{G}) = \text{tr}(\tilde{K}_x \tilde{K}_y), \tag{5.21}$$

  where $\tilde{K}_x = (\bar{K}_x(\bar{K}_x + \lambda m I_m))$, $\tilde{K}_y = (\bar{K}_y(\bar{K}_y + \lambda m I_m))$ are the centered kernels for X and Y and $\mathcal{G}$ and $\mathcal{H}$ are the Hilbert spaces.

5.2.1.4.4 HSIC Training with Linear Kernels We aim to solve equation 5.20 using a straightforward solution that allows us to achieve fast feature learning without iterations or intensive network parameter updates. One possible way to achieve this goal is to rewrite the objective function of the HSIC bottleneck as follows:

$$\begin{aligned} &\min_{\theta} K_Z^{0.5}(HK_XH)K_Z^{0.5} \\ &\max_{\theta} K_Z^{0.5}(HK_YH)K_Z^{0.5}, \end{aligned} \tag{5.22}$$

where:

- HK_XH and HK_YH are fixed symmetric matrices;
- K_X and K_Y represent the kernel functions of the input data and class labels, respectively;
- H is the centring matrix defined for m data points by equation 5.17;
- $K_Z = K(T_i(Z_{i-1}, \theta))$, where Z_{i-1} denotes the output of the previous hidden layer and T_i is the transformation applied to the current layer.

Finding a single-step solution to equation 5.22 using non-linear kernels is difficult. Therefore, we solved the optimisation problem in equation 5.22 using linear kernels. A detailed description of our solution is described in Appendix A. Our solution shows that the transformation weights matrix (θ) is provided by solving the following eigenvector problem:

$$Z^T(HK_XH)Z\theta = \lambda Z^T(HK_YH)Z\theta, \tag{5.23}$$

where Z is the previous layer's outputs and θ is represented by the eigenvectors corresponding to the smallest eigenvalues (λ). The eigenvectors are sorted by the smallest eigenvalues as the objective function (equation 5.22) is re-written as a minimisation problem (Appendix A).

5.2.2 Problem Settings

Let us consider a learning problem with N images output from a previous layer $L-1$ $\{X^{(L-1)} : X_i \in \mathbb{R}^{m\times n\times d_{L-1}}\}$ and C classes $Target \in \mathbb{R}^{N\times 1}$, where m and n represent the image's spatial dimensions and d_{L-1} is the number of filters in layer $L-1$. The primary goal of the supervised architectures, such as clustering, SLE, S-ELM and HSIC networks, is to generate supervised filters that maximise the classification results subject to the training data available. To achieve this goal, we first divide each image $X_i \in \mathbb{R}^{m\times n\times d_{L-1}}$ into patches $P_i \in \mathbb{R}^{(k_L^2 d_{L-1})\times \tilde{m}\tilde{n}}$, where $k_L \times k_L$ represents the kernel size in layer L, $\tilde{m} = (m-k_L)+1$ and $\tilde{n} = (n-k_L)+1$. Likewise, we extract all overlapping patches $P \in \mathbb{R}^{(k_L^2 d_{L-1})\times(\tilde{m}\tilde{n}N)}$ of all images in the database. Next, we create a vector $T \in \mathbb{R}^{\tilde{m}\tilde{n}N\times 1}$ containing the class labels of the patches $P \in \mathbb{R}^{(k_L^2 d_{L-1})\times(\tilde{m}\tilde{n}N)}$. Each patch is assigned a class identical to its full-image one. Finally, using a random subspace of P and T, we compute the supervised filters using the methods explained in sections 5.2.3.1, 5.2.3.2, 5.2.3.3 and 5.2.3.4.

5.2.3 Producing Networks Filter Banks

5.2.3.1 Clustering Network

Given a random set with M patches $P_s \in \mathbb{R}^{(k_L^2 d_{L-1})\times M} \subset P \in \mathbb{R}^{(k_L^2 d_{L-1})\times(\tilde{m}\tilde{n}N)}$ and C classes $T_s \in \mathbb{R}^{M\times 1} \subset T \in \mathbb{R}^{\tilde{m}\tilde{n}N\times 1}$, we compute the filters of the clustering network as follows:

1. For each class $c \in C$, we find all instances P_c that belong to c, as $\{P_c \in P_s : \text{if } T_s = c\}$;

2. We divide each class c into K clusters using Algorithm 5 with the subset P_c;

3. We create a vector containing the new classes for all training images, where the number of the new labels $= K \times C$;

4. We apply the multi-class LDA classifier to distinguish between P_s and the new classes;

5. The filters W_s^L obtained by the multi-class LDA can be expressed as follows:

$$W_s^L = \underset{k_L \times k_L \times d_{L-1}}{\text{mat}} \; q_s, \; s = [1, 2, \ldots, d_L], \tag{5.24}$$

where d_L represents the number of filters chosen for the current layer L;

6 We convolve the images in layer $(L-1)$ with the computed filters as the follows:

$$X_i^L = X_i^{L-1} * W_s^L, \tag{5.25}$$

where $*$ denotes the convolution operation.

5.2.3.2 Supervised Extreme Learning Machine

The S-ELM-based filters can be computed as follows:

1. We create a matrix $T_s \in \mathbb{R}^{C \times (\tilde{m}\tilde{n}N)}$, which represents the one-hot encoding of the original classes T;

2. We subtract the mean from T_s as follows:

$$T_s(i,j) = T_s(i,j) - \mu_i, \quad i = [1, 2, \ldots, \tilde{m}\tilde{n}N], \tag{5.26}$$

where μ_i denotes the mean of each column in matrix T_s and is described by the following equation:

$$\mu_i = \frac{1}{C} \sum_{j=1}^{C} T_s(j,i), \quad i = [1, 2, \ldots, \tilde{m}\tilde{n}N]; \tag{5.27}$$

3. We construct a matrix $P_s \in \mathbb{R}^{((k_L^2 d_{L-1})+C) \times (\tilde{m}\tilde{n}N)}$ by concatenating the images' patches $P \in \mathbb{R}^{(k_L^2 d_{L-1}) \times (\tilde{m}\tilde{n}N)}$ and the zero-mean targets $T_s \in \mathbb{R}^{C \times (\tilde{m}\tilde{n}N)}$;

4. We find β in equation 5.4 using the data points P_s and the ELM-AE concept described in Section 5.2.1.2.2;

5. The S-ELM-based filters can be expressed as follows:

$$W_s^L = \underset{k_L \times k_L \times d_{L-1}+C}{\text{mat}} q_s, \; s = [1, 2, \ldots, d_L], \tag{5.28}$$

   where $W_s^L = \beta$ and d_L is a user-predefined parameter that represents the number of filters in layer L.

To convolve an image $X_i^{L-1} \in \mathbb{R}^{m \times n \times d_{L-1}}$ with the S-ELM-based filters W_s^L, we first divide the image into patches $P \in \mathbb{R}^{mn \times {k_L}^2 d_{L-1}}$ using a kernel size $= k_L \times k_L$ after zero-padding it. Then, to replace the zero-mean targets in the training phase, we add C dimensions to P and fill them with zeros. The convolution operation can be expressed by the following matrix multiplication:

$$X_i^L \in \mathbb{R}^{mn \times d_L} = P \in \mathbb{R}^{mn \times (k_L^2 d_{L-1} + C)} \times W_s^L \in \mathbb{R}^{(k_L^2 d_L + C) \times d_L}, \tag{5.29}$$

where d_L is the number of the filters in layer L. We can obtain the convolutional outputs by reshaping the resulting matrix into $X_i^L \in \mathbb{R}^{m \times n \times d_L}$.

5.2.3.3 Supervised Laplacian Eigenmaps

Given a subset of patches $P_s \in \mathbb{R}^{(k_L^2 d_{L-1}) \times M}$ and C classes $T_s \in \mathbb{R}^{M \times 1} \subset T \in \mathbb{R}^{\tilde{m}\tilde{n}N \times 1}$, the procedures for computing SLE-based filters involve two primary steps:

(i) We convert P_s to a non-linear embedding space $Z \in \mathbb{R}^{M \times d_L}$ using the supervised Laplacian eigenmaps approach described in Section 5.2.1.3, where d_L is the number of filters in layer L;

(ii) We determine the weights $W_1^{(L)}$ and $W_2^{(L)}$ that map P_s to Z using the ELM mechanism defined in Section 5.2.1.2.

To generate the non-linear embedding $Z \in \mathbb{R}^{M \times d_L}$ of the input data P_s and T_s, the within-class and between-class weight matrices are first computed using Equations 5.10 and 5.11. Using Equations 5.13 and 5.12, the degree matrices within and between classes are calculated. Ultimately, we solve the generalised eigenvector problem represented by equation 5.14 for Z, where the L_b can be obtained using equation 5.8.

The convolutional filters $W_1^{(L)}$ and $W_2^{(L)}$ that map the input data P_s into Z can be computed using the ELM method, as follows:

1. We project P_s into a non-linear space using random weights $W_1^{(L)} \in \mathbb{R}^{k_L^2 d_{L-1} \times d_h}$ and bias $B_1^{(L)}$, as follows:
$$H = g(P_s^T W_1^{(L)} + B_1^{(L)}), \tag{5.30}$$
where d_h is a user-defined parameter that represents the number of dimensions in the ELM's hidden layer and g is a non-linear activation function;

2. We convert $H \in \mathbb{R}^{M \times d_h}$ into $Z \in \mathbb{R}^{M \times d_L}$ using the weights β, which can be estimated using equation 5.4 or 5.5. Then we set $W_2^{(L)} = \beta$;

3. The weights of the convolutional layers $W_1^{(L)} \in \mathbb{R}^{K_L^2 d_{L-1} \times d_h}$ and $W_2^{(L)} \in \mathbb{R}^{d_h \times d_L}$ can be represented as follows:
$$\begin{aligned} W_1^{(L)} &= \underset{k_L \times k_L \times d_{L-1}}{\text{mat}} q_s, \quad s = 1, 2, \ldots, d_h, \\ B_1^{(L)} &= \underset{1 \times 1 \times d_{L-1}}{\text{mat}} q_s, \quad s = 1, 2, \ldots, d_h, \\ W_2^{(L)} &= \underset{1 \times 1 \times d_h}{\text{mat}} q_s, \quad s = 1, 2, \ldots, d_L, \end{aligned} \tag{5.31}$$
where d_h and d_L are predefined user parameters;

4. To convolve $X_i^{(L-1)}$ with SLE filters, we use the formula
$$X_i^{(L)} = g(X_i^{(L-1)} * W_1^{(L)} + B_1^{(L)}) * W_2^{(L)}, \tag{5.32}$$
where $*$ is the convolution operation and g represents a non-linear activation function.

5.2.3.4 Hilbert–Schmidt Independent Criteria Network

If we have N images $X^{(0)} = \{X_i^{(0)} \in \mathbb{R}^{m \times n \times d_0},\ d_0 \in \{1, 3\}\}$, a set of classes $Target \in \mathbb{R}^{N \times 1}$ corresponding to them and a set of feature maps in layer $L-1$ $X^{(L-1)} = \{X_i^{(L-1)} \in \mathbb{R}^{m \times n \times d_{L-1}}\}$, then the HSIC-based filters can be computed as follows:

1. After zero-padding the original images $X^{(0)}$, we divide them into patches $P^{(0)} \in \mathbb{R}^{k^2 d_0 \times mnN}$, where $k \times k$ is the size of the kernel and n and m are the image's spatial dimensions;

2. We divide the zero-padded feature maps $X^{(L-1)}$ into patches $P^{(L-1)} \in \mathbb{R}^{k_L^2 d_{L-1} \times mnN}$ using a kernel size of $k_L \times k_L$;

3. We create a vector $T \in \mathbb{R}^{1 \times mnN}$ containing the class labels for patches, where each patch is allocated a class identical to its full-image class;

4. Using the one-hot encoding approach, we transform T to $T_s \in \mathbb{R}^{C \times mnN}$, where C denotes the number of classes;

5. Using random subsets with M patches $S^{(0)} \in \mathbb{R}^{k^2 d_0 \times M} \subset P^{(0)}$, $S^{(L-1)} \in \mathbb{R}^{k_L^2 d_{L-1} \times M} \subset P^{(L-1)}$ and $Y \in \mathbb{R}^{C \times M} \subset T_s$, we compute the Gaussian kernel defined by equation 5.1 on $S^{(0)}$ and Y to obtain K_X and K_Y;

6. We solve the optimisation problem (equation 5.22) for θ using K_X, K_Y and $S^{(L-1)}$ as Z in equation 5.22;

7. The HSIC convolutional weights $W_s^{(L)}$ can be expressed as follows:

$$W_s^{(L)} = \underset{k_L \times k_L \times d_{L-1}}{\operatorname{mat}} q_s, \quad s = 1, 2, \ldots, d_L, \tag{5.33}$$

   where d_L represents the number of filters chosen for layer L and $W_s^{(L)} = \theta$;

8. We convolve any image $X_i^{(L-1)}$ with the HSIC-based filters, as follows:

$$X_i^{(L)} = X_i^{(L-1)} * W_s^{(L)}, \tag{5.34}$$

   where $*$ denotes the convolution operation.

5.3 Experiments and Results

In this section, we analyse the performance of the four proposed networks using several classification benchmarks, namely CIFAR-10, CIFAR-100, TinyImageNet and MNIST. Our experiments are organised into three main sections. In Section 5.3.3, we analyse the efficacy of the proposed networks employing a single-layer architecture with a sufficient number of filters. The second experiment, described in Section 5.3.4, investigates the impact of network depth on the accuracy of the proposed networks. In the last experiment (Section 5.3.5), we study the effect of combining different filters in the convolutional layers on the accuracy of the models. In addition, in Section 5.3.2, we describe the complexity of the networks used in our experiments, and in Section 5.3.1, we explain how we set the networks' parameters.

5.3.1 Selecting Hyperparameters for Proposed Architectures

The selection of optimal parameters is essential for achieving good accuracy in any machine learning algorithm. In all experiments, we evaluated the performance of our models using different parameter settings on a validation set and chose the parameters that led to the best accuracy results. For instance, with the spectral clustering network, we evaluated the model's performance using different graphs and similarity matrices with different parameters. We then selected the parameters that led to the highest accuracy results on the validation set. Similarly, we tested various parameter values using a validation set for the other networks and chose the hyperparameters that resulted in the best performance.

5.3.2 Networks Complexity

In this section, we discuss the computational complexity of the four proposed networks, namely SLE, HSIC, S-ELM and clustering networks. We also compare their computational costs to the Multi-Layer PCANet (Chapter 3) and the Stacked-LDA (Chapter 4). In all experiments (Sections 5.3.3, 5.3.4 and 5.3.5), we used the same network architecture, either with a single-layer or two-layer design. Therefore, the number of parameters in all networks in our experiments was the same. However, the complexity of the networks varied based on the algorithm used to compute the convolution filters.

For image-based patches with n data points and d dimensions, the convolutional filters for both the clustering network and SLE-Net rely on computing the Laplacian matrix and performing eigendecomposition on the resulting matrix. The complexity of computing the graph Laplacian is typically $O(n^2)$ while performing eigendecomposition requires $O(n^3)$ in the worst-case scenario. Therefore, the complexity of these steps is approximately $O(n^3 + n^2)$. Running k-means clustering, in the case of spectral clustering, for t iterations and k clusters results in an $O(tknd)$ complexity. Therefore, the overall complexity of the spectral clustering algorithm is $O(n^3 + n^2 + tknd)$. To generate the filters in the clustering network, we also applied an LDA between the image-based patches and the labels generated by the spectral clustering, which adds around $O(nd^2)$ more complexity. In the case of the SLE network, we generalised to the unseen data using the extreme learning machine, which involves two steps: random projection, which transforms the input into hidden representation and linear regression model, which transforms the hidden representation into the non-linear embedding space computed by the SLE. As so, the complexity added to the computation of the SLE is about $O(nd + nm^2 + m^3)$, where m is the size of the embedded features computed by SLE. The computation of the S-ELM filters involves using the supervised ELM-auto-encoder, which is similar to the original ELM, but its output is equal to its inputs. For this reason, the complexity of these filters is around $O(nd + nd^2 + d^3)$. Finally, the computational complexity of HSIC filters is dominated by the computation of the K_x and K_y kernels (equation 5.23), which require around $O(n^2)$ computational complexity. Table 5.1 provides a summary of the estimated complexity of the filters generated by the four proposed architectures (SLE, HSIC, S-ELM, and clustering) in comparison to those associated with generating PCA and LDA filters (Chapters 3 and 4).

TABLE 5.1: The estimated computational complexity of the filters generated by the four proposed networks: SLE, HSIC, S-ELM, and clustering, in comparison to those associated with generating PCA and LDA filters (Chapters 3 and 4).

Network's name	**Complexity**
Multi-Layer PCANet (Chapter 3)	$O(n)$
Stacked-LDA (Chapter 4)	$O(n)$
SLE Net	$O(n^3)$
Clustering network	$O(n^3)$
S-ELM network	$O(n)$
HSIC Net	$O(n^2)$

In summary, the computational complexity discussed in this section is exclusively computed for the filters of the four proposed networks (SLE, HSIC, S-ELM, and clustering) compared to those generated in Chapters 3 and 4. The other components of the networks, including second-order pooling, spatial pyramid pooling and the classifier, are the same across all networks. For this reason, we do not compute their complexity.

5.3.3 Single-Layer Networks

5.3.3.1 Parameter Settings

This experiment aimed to determine if the addition of supervised information would improve the discriminational ability of the proposed networks, namely the clustering network, SLE Net, S-ELM Net and HSIC Net when compared with PCANet. To perform this experiment, we used a single-layer network architecture with 75, 75, 60 and 27 filters for the CIFAR-10, CIFAR-100, MNIST and TinyImageNet databases, respectively. Following each network's convolutional layer, a ReLU activation function, second-order pooling and support vector machine (SVM) classifier were applied. The filter size was set to 5×5 using the CIFAR-10 and CIFAR-100 databases, 13×13 for the MNIST database and 3×3 for the TinyImageNet database. For the CIFAR-10 and CIFAR-100 databases, we employed an 8×8 second-order pooling block size with a stride of 1 and connected the second-order features to a three-level spatial pyramid pooling of 4×4, 2×2 and 1×1 subwindows. We extracted the second-order features for the MNIST database using a second-order pooling block size of 7×7 and a stride of 7. In the TinyImageNet database, we used a 16×16 second-order pooling block size with a stride of 1 followed by a spatial pyramid pooling of 4×4, 2×2 and 1×1 subregions.

To implement the clustering network, we defined the local neighbourhood for each point in the dataset using a k-nearest neighbour graph with $k = 5$. The similarity matrix was computed using the Gaussian kernel (equation 5.1) with the kernel scale $\sigma = 1$. To compute the Laplacian matrix, we used the normalised random walk Laplacian matrix described in [96] (step 4 of Algorithm 5). In addition, we used the standard deviation of the patches rather than the patches themselves as input for Algorithm 5. The number of clusters per class in all databases was set to 15.

In order to implement the S-ELM Net, we assigned a uniformly distributed random number to the W and B variables in equation 5.2 and defined g as a ReLU activation function. We also estimated β using equation 5.5 with a regularisation factor $C = 0.001$.

After finding the patch-based images' embedding as detailed in Section 5.2.3.3 for the SLE Net, we assigned uniformly random numbers to $W_1^{(L)}$ and B (equation 5.30). The number of filters d_h in equation 5.30 was selected to be twice the number of embedding features. For instance, in this section and for the CIFAR-10 database, we employed a single-layer architecture with 75 filters; hence, we assigned 150 to the middle layer's number of filters d_h. In addition, β was calculated using equation 5.5, and a ReLU activation function was used between the layers.

For the implementation of the HSIC Net, we employed the Gaussian kernel defined by equation 5.1 with kernel scale $\sigma = 1$ for both training patches and hot-encoding targets.

5.3.3.2 Performance Analysis

Table 5.2 compares the accuracy of the proposed supervised architectures to that of Multi-Layer PCANet (Chapter 3) and Stacked-LDA (Chapter 4) using single-layer structures on four databases (CIFAR-10, CIFAR-100, MNIST and TinyImageNet). For an adequate comparison, we used the same parameters as the supervised models described in Section 5.3.3.1 for the PCANet and Stacked-LDA architectures. In addition, we generated the Stacked-LDA filters using the settings provided in Section 4.4.2.1.1 of Chapter 4.

TABLE 5.2: Accuracy (%) using different networks on the CIFAR-10, CIFAR-100, MNIST and TinyImageNet databases. PCANet in the table refers to Multi-Layer PCANet (Chapter 3).

Single-Layer Network					
Filter Type	**CIFAR-10**	**CIFAR-100**	**MNIST**	**TinyImageNet**	**Learning Paradigm**
Stacked-LDA-1	**80.20**	**54.9**	99.39	30.20	Supervised
PCANet-1	75.96	53.68	99.29	24.14	Unsupervised
Clustering Net-1	77.31	52.38	99.30	25.28	Supervised
S-ELM Net-1	77.08	52.82	**99.41**	**30.38**	Supervised
SLE Net-1	77.00	51.64	99.20	26.80	Supervised
HSIC Net-1	76.16	52.12	99.23	26.32	Supervised
Two-Layer Network					
Filter Type	**CIFAR-10**	**CIFAR-100**	**MNIST**	**TinyImageNet**	**Learning Paradigm**
Stacked-LDA-2	**81.44**	55.94	**99.40**	32.3	Supervised
PCANet-2	80.87	**58.74**	99.30	31.06	Unsupervised
Clustering Net-2	77.91	52.1	99.28	25.9	Supervised
S-ELM Net-2	78.82	54.37	99.35	**32.38**	Supervised
SLE Net-2	78.80	52.26	99.18	25.86	Supervised
HSIC Net-2	77.91	52.05	99.13	27.33	Supervised
Combination of Filters in Single-Layer Architectures					
Filter Type	**CIFAR-10**	**CIFAR-100**	**MNIST**	**TinyImageNet**	**Learning Paradigm**
Stacked-LDA+PCA-1	**80.87**	57.75	99.33	**30.88**	Semi-supervised
Clustering+PCA Net-1	79.76	57.6	**99.35**	26.27	Semi-supervised
S-ELM+PCA Net-1	79.76	**58.29**	**99.35**	29.82	Semi-supervised
SLE+PCA Net-1	80.20	57.89	99.31	28.56	Semi-supervised
HSIC+PCA Net-1	78.08	57.32	99.30	27.91	Semi-supervised
Supervised Net-1	79.3	56.23	99.32	28.24	Supervised
Combination of Filters in Two-Layer Architectures					
Filter Type	**CIFAR-10**	**CIFAR-100**	**MNIST**	**TinyImageNet**	**Learning Paradigm**
Stacked-LDA+PCA-2	**82.26**	58.65	99.33	33.34	Semi-supervised
Clustering+PCA Net-2	81.39	58.47	99.36	29.67	Semi-supervised
S-ELM+PCA Net-2	81.51	59.20	99.33	**33.79**	Semi-supervised
SLE+PCA Net-2	81.65	**59.39**	**99.43**	32.79	Semi-supervised
HSIC+PCA Net-2	80.23	59.08	99.33	31.78	Semi-supervised
Supervised Net-2	81	57.28	99.30	31.19	Supervised

For the CIFAR-10 database, as shown in Table 5.2, the supervised networks presented interesting results regarding accuracy when compared with the unsupervised PCANet. All of the proposed networks outperformed the accuracy metric of the PCANet by about 2%, showing that the feature representation in these networks was sufficient for reaching good performance results. However, one can note that the Stacked-LDA network outperformed all other models in terms of accuracy, achieving approximately 3% higher accuracy than all the supervised approaches and around 5% higher than PCANet, demonstrating its good generalisation ability in this benchmark dataset.

According to Table 5.2, PCANet and Stacked-LDA outperformed all the new supervised methods (clustering, S-ELM, SLE and HSIC networks) on the CIFAR-100 database. The best accuracy result obtained by the Stacked-LDA network did not demonstrate a significant improvement over PCANet, being only about 1% better. Likewise, there was no significant difference between the accuracy values achieved by PCANet and the new supervised approaches, since PCANet outperformed them by around 1%. Therefore, all of the models provided satisfactory results in terms of accuracy in the CIFAR-100 benchmark. Nevertheless, PCANet and Stacked-LDA Net were the optimal models for this database due to their lower computational cost when compared with other methods.

For the MNIST database, all networks presented good accuracy results of over 99%, as shown in Table 5.2. The best accuracy result was 99.41%, which was achieved by using the S-ELM network, showing the excellent representativeness of the extracted features and generalisation ability of ELM models. However, the Stacked-LDA provided a result comparable with the one obtained by S-ELM in terms not only of accuracy but also of computational cost. Although SLE Net provided the lowest accuracy (99.20%), this was still comparable with the results produced by PCANet and other models.

As shown in Table 5.2, in the TinyImageNet database, all of the supervised architectures presented better accuracy results than PCANet. The S-ELM architecture reached an accuracy of 30.38% in the database, which was equivalent to the accuracy obtained by the Stacked-LDA network and around 6% higher than that of PCANet. The other supervised networks provided similar performance, which improved on PCANet by about 2%, except for the clustering network, which improved by 1% over PCANet. Therefore, the use of supervised networks increased the discriminability of features and efficiency, even with a limited number of training instances per class, as was the case in this database. However, the Stacked-LDA and S-ELM networks provided the best accuracy results, which made them the optimal networks to use with this benchmark dataset.

In general, and based on the results obtained by different single-layer architectures using CIFAR-10, CIFAR-100, MNIST and TinyImageNet databases, we highlight the following key findings:

- The filter type has a significant influence on the model's performance. For example, the Stacked-LDA network obtained 5% higher accuracy than PCANet on the CIFAR-10 database. Likewise, the S-ELM network represented an approximately 6% enhancement over PCANet on the TinyImageNet database. Therefore, if there is time to perform more research, it should be utilised to identify filters that generalise well for the database in question.
- Supervised architectures are not necessarily superior to unsupervised approaches, especially for databases with significantly similar classes, such as CIFAR-100. Because the filters are learned in a single step with no mechanism to guarantee that the patches are separated appropriately, the supervised techniques are prone not only to a problem with overfitting but also to classification errors that could propagate through the network.
- When compared with all the other single-layer architectures across all the databases, the Stacked-LDA provided the highest accuracy results. The S-ELM network showed equivalent performance on TinyImagenet and MNIST, making it an excellent alternative to the Stacked-LDA model in these two databases. Learning the stacked-LDA filters involves grouping some patches and ensuring that the chosen groups are separable. Consequently, the Stacked-LDA model produced expressive results across all the datasets for object recognition and written hand-digit classification, indicating that this model had significant generalisation capacity over all the other models.

5.3.4 Two-layer Networks

5.3.4.1 Parameter Settings

The aim of this experiment was to show the impact of the network depth on the performance of the proposed networks (clustering, HSIC, S-ELM and SLE). To achieve this goal, we added a layer to the single-layer structures described in section 5.3.3 using the same parameter settings with the exception of the MNIST database, for which we utilised a 1×1 filter size instead of a 13×13 filter size. We used a small filter size for the second layer of the MNIST database as using a larger filter size did not improve the performance and instead imposed a higher overhead. In addition, we compared the results of the proposed networks with those of two-layer PCANet and Stacked-LDA using the parameters stated in Section 4.4.2.1.1 of Chapter 4.

5.3.4.2 Performance Analysis

Table 5.2 reports the accuracy results of the four proposed networks on different classification benchmarks using two-layer architectures. The proposed architectures were also compared with the two-layer PCANet and Stacked-LDA models. As shown in Table 5.2, different architectures yielded different accuracy results for each database, emphasising our hypothesis about the impact of the filter type on the model's performance.

According to Table 5.2, increasing the number of layers improved the accuracy of all models using the CIFAR-10 database. This improvement did not appear to be significant for any of the supervised approaches, since it was only around 1% better than their single-layer structures. In comparison, the PCANet's performance increased from 75.97% to 80.87%, outperforming all supervised network except the Stacked-LDA. While the two-layer Stacked-LDA network outperformed the other networks in terms of accuracy, the single-layer Stacked-LDA network produced results that were on par with those of the two-layer PCANet network. As a result, the Stacked-LDA network is an excellent option to use in this database. The other supervised networks (HSIC, S-ELM, SLE and clustering) produced equivalent results that were 2% lower than PCANet in the two-layer architectures, indicating they are more appropriate for wider networks (fewer layers with more filters in each layer) than deeper ones.

For the CIFAR-100 database, adding a layer to PCANet resulted in a considerable improvement of roughly 5% over its single-layer architecture, as shown in Table 5.2. Increasing the network depth also improved the recognition rates of two supervised models, namely the S-ELM and Stacked-LDA networks, in the same dataset. However, the improvement obtained by these two networks was not statistically significant, at approximately 1% for the Stacked-LDA model and 2% for the S-ELM network. The accuracy of the HSIC, SLE and clustering networks did not improve with the network depth. Overall, two-layer PCANet provided the best accuracy result, at 58.74%, which was around 3% higher than Stacked-LDA, 4% better than S-ELM and 7% more than the other supervised techniques. According to the results, PCANet is the best network for this database.

Table 5.2 demonstrates that, for the MNIST database, all two-layer structures, regardless of the learning paradigm, provided nearly the same recognition rates as their single-layer counterparts, with the exception of the SLE network, whose two-layer architecture performed 0.10% worse than its single-layer architecture. This implies that when additional layers are added, the generalisation capability of these shallow-depth networks using this database decreases, which may lead to an overfitting problem. As a result, in this database, using a single-layer architecture is enough to reach good accuracy results.

Based on the results reported in Table 5.2 when using the TinyImageNet database, we noticed that the two-layer Stacked-LDA and S-ELM networks achieved the best recognition rates, with a 2% improvement over their single-layer counterparts. Two-layer PCANet, on the other hand, provided very competitive results, with an accuracy of 31.06%, which was 1% lower than the best results obtained and 6% higher than its single-layer architecture. The results also showed that the two-layer SLE and clustering networks provided the worst results, since they did not improve on their single-layer architectures and were 7% lower than the best accuracy results. The two-layer HSIC network showed a 1% improvement over its single-layer counterpart but was 5% lower than the highest accuracy. Consequently, three of the supervised techniques, the HSIC, SLE and clustering networks, are inappropriate for use as deep architectures, while the Stacked-LDA and S-ELM networks are the two optimal models for this database.

To summarise the results of the two-layer network shown in Table 5.2, we highlight the following two main findings:

- In the context of supervised networks, increasing the depth of the Stacked-LDA and S-ELM networks always resulted in improved accuracy for all databases except for the MNIST database, for which a single-layer structure was sufficient for obtaining the desired results, even though this improvement was not significant. The SLE, HSIC and clustering networks, on the other hand, did not perform well when additional layers were added. As a result, adopting supervised architectures with more filters and a single layer was sufficient to achieve good performance.
- Two-layer PCANet provided a significant improvement over its single-layer counterpart. This improvement was about 5% or even higher using the CIFAR-10, CIFAR-100 and TinyImageNet databases. Consequently, PCANet became competitive with the supervised shallow-depth networks obtaining the best accuracy results. For example, the two-layer PCANet provided 1% less accuracy than the Stacked-LDA network using two databases, CIFAR-10 and TinyImageNet, while achieving the best accuracy compared with the other architectures on the CIFAR-100 database.

5.3.5 Combination of Filters

5.3.5.1 Motivation and Parameter Settings

From the previous experiments (Sections 5.3.3 and 5.3.4), we noticed that supervised architectures generally had good convergence and better performance than unsupervised PCANet with single-layer architectures. However, PCANet provided competitive results when the network depth increased. As a consequence, we combined in this experiment the convolutional outputs of the supervised techniques with the feature maps produced by the unsupervised PCANet, which we refer to in Table 5.2 as semi-supervised methods, in order to exploit the unique characteristics of both supervised and unsupervised approaches. Figure 5.4 shows an example of a two-layer semi-supervised network. The post-processing step in the figure involves nonlinearity, second-order pooling and an optional spatial pyramid pooling.

For the implementation of semi-supervised networks, the same parameters mentioned in Sections 5.3.3 and 5.3.4 were employed. However, we used 50% of the total number of filters to produce supervised feature maps and

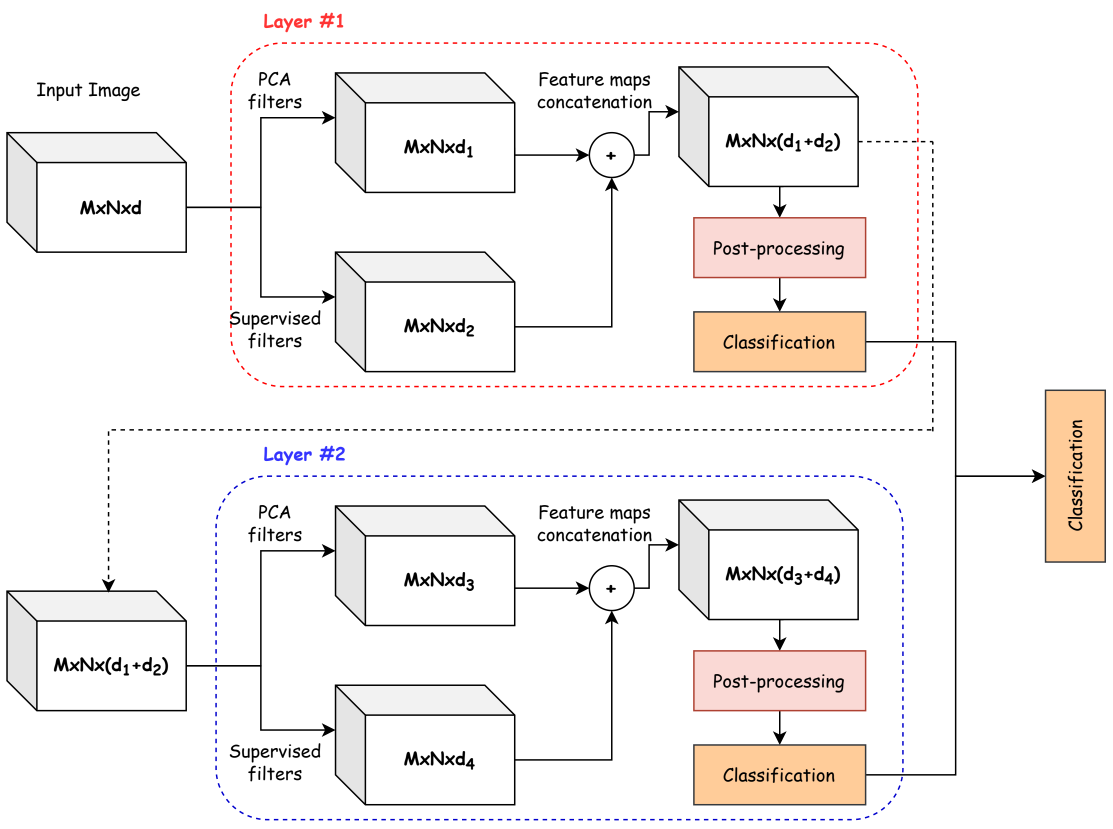


FIGURE 5.4: Two-layer semi-supervised network.

the remaining 50% to construct unsupervised feature maps. In the CIFAR-10 database, for instance, there were 75 filters; hence, 38 supervised and 37 unsupervised feature maps were combined. We also evaluated a semi-supervised Stacked-LDA network in which the Stacked-LDA followed the configurations outlined in Section 4.4.2.1.1 of Chapter 4. In addition, we developed a supervised network named "Supervised Net" in Table 5.2 to show how the performance would be affected by integrating only supervised filters. This network used the same settings as the other networks, but only 25% of the total number of filters were used to generate the feature maps for all the supervised networks (HSCI, SLE, S-ELM and clustering). For instance, in the TinyImageNet database, we used 27 filters; therefore, we combined seven feature maps of each network, including SLE, clustering and S-ELM, with six HSIC filters' responses.

5.3.5.2 Performance Analysis

Table 5.2 compares the accuracy of the semi-supervised networks against each other and against the supervised network "Supervised Net" using single-layer and two-layer architectures. The supervised network incorporated SLE, clustering, S-ELM and HSIC feature maps, using 25% of the total number of filters for each filter type.

According to Table 5.2, the investigated shallow-depth networks performed similarly in both single-layer and two-layer architectures, independent of the learning paradigm employed. There were cases where it was evident that one learning paradigm would provide a slight improvement over the other. For instance, when the classes were similar, such as in the CIFAR-100 database, the semi-supervised techniques were superior to the supervised ones. Additionally, it was noticeable that some networks performed significantly better than others, depending on the datasets used. More specifically, in the TinyImageNet database, the Stacked-LDA and S-ELM networks outperformed the other approaches by 2% to 4% on single-layer and two-layer structures, respectively. This observation suggests that the semi-supervised S-ELM and Stacked-LDA networks are the optimal architectures for the classification task.

As shown in Table 3.3, for the CIFAR-10 database, all networks provided competitive accuracy results, with the semi-supervised Stacked-LDA network obtaining the best accuracy in the single-layer and two-layer architectures. The accuracy of all semi-supervised networks was around 3% higher than that of their supervised counterparts, with the exception of the Stacked-LDA, which improved by approximately 1% over its supervised counterpart. The Supervised Net, created by combining four supervised networks' feature maps (HSIC, S-ELM, SLE and clustering), achieved substantially higher accuracy than any of them individually. This finding implies that we can still attain excellent accuracy by combining filters with the same learning paradigms. In all the networks, two-layer semi-supervised structures outperformed their single-layer architectures by approximately 2%. This improvement was not significant compared with that reached by unsupervised PCANet as its depth increased, but it was superior to that attained by any of the supervised structures as their depth increased.

Using the CIFAR-100 database as an example, Table 5.2 demonstrates that combining different filters significantly improved the performance compared with using each filter type alone. For example, the semi-supervised Stacked-LDA outperformed its supervised counterpart by around 3%. Other semi-supervised networks outperformed their supervised counterparts by more than 5% in terms of accuracy. This suggests that combining several convolutional outputs might produce more robust representations than employing a single filter type. The semi-supervised S-ELM network obtained the best accuracy with the single-layer architecture, which was equivalent to two-layer PCANet accuracy and 5% better than its supervised architecture. In two-layer semi-supervised networks, the SLE network showed superior performance, whereas the HSIC and S-ELM networks produced comparable results. On the other hand, Supervised Net achieved the worst performance in single-layer and two-layer architectures. However, the degradation was only 2% worse than the best accuracy, indicating that combining supervised and unsupervised techniques would be preferable when dealing with a database of similar classes. Moreover, increasing the network depth across all networks improved performance by 1% to 2%.

According to Table 5.2, when using the MNIST database, all semi-supervised networks insignificantly outperformed their supervised counterparts, with the exception of the S-ELM and Stacked-LDA networks, whose accuracy dropped by 0.06%. The semi-supervised S-ELM network provided the best accuracy results when compared with all the other architectures, with an accuracy of 99.35%, which was 0.06% lower than its supervised architecture. Moreover, increasing network depth resulted in the same accuracy for all the networks, except for the SLE, which reached the highest accuracy in this database, at 99.43%. However, this accuracy was equivalent to the single-layer supervised Stacked-LDA and S-ELM networks. This findings suggest that one layer of supervised Stacked-LDA or S-ELM networks for the MNIST database would be sufficient to provide results with good accuracy.

For the TinyImageNet database, Table 5.2 shows that the semi-supervised Stacked-LDA and S-ELM provided results with the same accuracy as their supervised architectures in single-layer architecture. The other networks improved the accuracy by around 1% to 2% over their supervised counterparts. The results also stated that increasing the network depth led to 3% to 5% better performance. The S-ELM network achieved the best accuracy in semi-supervised two-layer architectures. This performance of 33.79% was comparable with that of the Stacked-LDA network, 2% better than the HSIC and Supervised networks and 4% higher than the clustering network. Based on the results, we can conclude that semi-supervised Stacked-LDA and S-ELM networks have good generalisation performance in this database.

Based on the experimental results presented in this section, we can draw the following conclusions:

- The best performance across all databases was achieved by semi-supervised networks, indicating that using a combination of different filters in the convolutional layers could result in a good generalisation capacity.
- Adding a layer to the architectures with integrated filters always boosted performance by 1% to 3% on average.
- Combining filters in convolutional layers improved accuracy in most situations, independent of the learning paradigm used by the filters.

5.4 Conclusions

In this chapter, we introduced four supervised networks, namely the clustering, supervised extreme learning machine (S-ELM), supervised Laplacian eigenmaps (SLE) and Hilbert–Schmidt independence criterion (HSIC) networks. The proposed networks followed a similar architecture that consisted of convolutional layers, non-linear activation, second-order pooling and a classifier. The spatial pyramid pooling could be added between the second-order pooling and the classifier. With multilayer architectures, we combined the classification decisions of all layers to make the final decision. The four proposed architectures differed in their filter types, with each network using a different filter type and extracting different features. In the following, we summarise the objective of each network:

- The clustering network aims to optimise the Stacked-LDA implementation by grouping the image-based patches using the spectral clustering algorithm;
- The SLE network attempts to replace linear filters with filters that map data into a non-linear space using the separation criteria, which try to maximise the distance between samples from different classes while minimising the distance inside the class;
- The HSIC network's objective is to create a filter bank using the information criteria rather than the separation criteria, where the information criteria aims to maximise the dependency on the labels while compressing the input data;

- The S-ELM uses the supervised extreme learning machine autoencoder to compress the features while preserving information about the classes.

The proposed networks were evaluated on four classification databases, CIFAR-10, CIFAR-100, MNIST and TinyImageNet, using single-layer and two-layer architectures (Sections 5.3.3 and 5.3.4). The new networks were also compared to the Multi-Layer PCANet (Chapter 3) and Stacked-LDA (Chapter 4). Using sufficient training data and based on our experiments in Sections 5.3.3 and 5.3.4, we could generalise the following findings:

- The supervised architectures outperformed the Multi-Layer PCANet in single-layer architectures. However, as the number of layers increased, the performance of supervised networks improved somewhat less than that of the unsupervised Multi-Layer PCANet when an additional layer was added. This observation makes the performance of the unsupervised network (Multi-Layer PCANet) comparable with that of other supervised approaches as network depth increases;
- While the S-ELM network achieved competitive accuracy results in some databases, the Stacked-LDA network presented better data representation and generalisation capabilities than the other networks in most cases. A single-layer Stacked-LDA architecture with a large number of filters also outperformed two-layer PCANet architecture.

In order to make use of the unique characteristics of the supervised and unsupervised networks, we investigated four semi-supervised architectures that were similar to the supervised networks (clustering, HSIC, SLE and S-ELM). However, we combined 50% supervised and 50% PCA filters to produce the convolutional layers' outputs. In addition, to investigate the efficacy of the combination mechanism employing different learning paradigms, we constructed a supervised network to combine the convolutional outputs of the four supervised networks, with 25% of each filter type. We evaluated the performance of the proposed networks on four classification benchmarks, CIFAR-10, CIFAR-100, MNIST and TinyImageNet, utilising both single-layer and two-layer structures. Our experiments (Section 5.3.5) can be summarised as follows:

- Regardless of the learning paradigm used, combining the convolutional outputs of many filters resulted in better performance than using each filter type alone;

- Two-layer architectures improved on their single-layer counterparts; however, this improvement of 1–3% was smaller than the one of 5–6% improvement that was achieved by adding a layer to PCANet;
- Compared with all the other networks, the semi-supervised S-ELM and Stacked-LDA networks achieved high accuracy and generalised well across all datasets.

Despite the many advantages of shallow-depth architectures, such as reasonable training time and good accuracy, several open challenges may serve as motivation for future research, which is the focus of the following chapter.

The number of layers affects a neural network's generalisation capacity. CNNs, for example, achieve excellent accuracy through the use of many convolution layers. However, when the number of layers increases in both supervised and unsupervised architectures, a problem with the loss of generalisation capacity emerges, which may lead to an overfitting problem. We believe that the subsampling mechanism in these shallow-depth networks causes this limitation. Using PCANet as an example, we use a subset of the eigenvectors generated by PCA to compute the convolutional outputs of the first layer. In the second layer, we generate filters using the first subspace's processed images, which might represent 90% of the original dataset. As we progress by adding more layers, we lose more information about the original dataset, resulting in weak network representation and poor accuracy results. The residual blocks for each layer are introduced in the following chapter to alleviate this limitation.

In many cases, the Stacked-LDA network showed good performance compared with other shallow-depth networks. We believe its high performance is due to the fact that it involves procedures that consider classification errors. Because shallow-depth networks are trained in a single step, classification errors are likely to propagate across the network. Therefore, one of the challenges addressed in the next chapter is to increase the network depth while including a mechanism that corrects classification errors when traversing the network.

It is noticeable that unsupervised networks have an advantage over the supervised ones when the number of training samples per class is small, as shown in the case of the CIFAR-100 database (Section 5.3.4). The following chapter explores augmenting data to increase the training sample size.

Chapter 6

Deep Residual Compensation Convolutional Network

6.1 Introduction

The findings obtained in Chapter 4 provided evidence that the filter type has a significant impact on the accuracy of the model. Motivated by the excellent performance of the supervised Stacked-LDA network (Chapter 4), we presented four more supervised networks (Chapter 5): clustering, supervised extreme learning machine (S-ELM), supervised Laplacian eigenmaps (SLE) and Hilbert–Schmidt independence criterion (HSIC) networks. All networks use the same architecture consisting of convolutional layers; each convolutional layer is followed by non-linear activation, second-order pooling and a classifier. However, the four networks produce different supervised features depending on the filter type employed. The clustering network uses the spectral clustering approach to create new classes for randomly chosen image-based patches. After that, the filters are obtained using LDA applied with the newly generated classes, rather than the original ones. The SLE network utilises non-linear filters to maximise the distance between data from different classes while decreasing the distance between samples within a class. The HSIC network creates a filter bank using the information criterion, which tries to maximise the dependence on labels while compressing original features. The S-ELM generates its filter bank using the supervised extreme learning machine autoencoder, which helps in the compression of features while preserving class information. The four networks were evaluated on the CIFAR-10, CIFAR-100, TinyImageNet and MNIST databases, demonstrating their effectiveness compared with Multi-Layer PCANet in specific datasets. In the majority of situations, however, the experiments performed in Chapter 5 showed that the Stacked-LDA network achieved a higher level of recognition accuracy than did the other shallow-depth networks. The experimental results from Chapter 5 also showed that although the supervised networks used class labels to boost classification performance, the unsupervised Multi-Layer PCANet generated equivalent accuracy results as network depth increased. By combining supervised and unsupervised filters at a ratio of 50:50, we presented four semi-supervised shallow-depth networks (clustering, HSIC, SLE and S-ELM) capable of exploding both learning paradigms in an effective manner, resulting in a highly adaptable architecture and better accuracy results. To test the combination mechanism using the same learning paradigm, we created a supervised network to combine the convolutional outputs of HSIC, S-ELM, clustering and SLE with 25% of each filter type. The findings indicated that integrating several filters improved performance independent of the learning paradigm. While semi-supervised ELM performed comparably, the semi-supervised Stacked-LDA network outperformed all other approaches.

In general, the shallow-depth networks presented in Chapters 3, 4 and 5 provided good accuracy results. However, increasing the network depth leads to a considerable degradation in performance. Despite the importance of network depth for image classification, the maximum number of layers reached in our experiments was small (between six and nine). We believe that the subspace nature of these shallow-depth networks causes this limitation. As we progress by going deeper across the network, we lose more information about the original data, and unlike neural networks, there is no mechanism to adjust the weights concerning the classification errors; thus, errors propagate with more layers being added.

This chapter introduces a residual compensation convolutional learning architecture, which is inspired by the residual network [5] and is designed to obtain accuracy from considerably increased depth while effectively correcting network errors with more layers being added. The organisation of the chapter is as follows. The overall chapter is divided into two primary sections. The first section (Section 6.2) provides a detailed explanation of the network's design with its training algorithm. The second (section 6.3) consists of the experiments that were carried out to demonstrate the effectiveness of the proposed framework and is divided into three subsections. The first examines the influence of several network parameters, such as the number of filters (6.3.1.1) and the learning rate (6.3.1.2), on the accuracy of the deep residual compensation convolutional network. In Section 6.3.2, we evaluate the performance of the proposed networks on MNIST, CIFAR-10, CIFAR-100 and TinyImageNet against gradient-based and non-gradient architectures without data augmentation. The final subsection (Section 6.3.3) analyses the possibility of improving the proposed model's accuracy through the use of data augmentation. Finally, in Section 6.4, the findings are summarised along with suggestions for further study.

6.2 Network Architecture

Figure 6.1 shows the architecture of the proposed network. The first layer of the model is a single-layer architecture, similar to those discussed in Chapters 3, 4 and 5, consisting of a convolutional layer with any filter type, non-linear activation, second-order pooling and an LDA classifier trained using the original classes. The deeper layers are residual compensation layers, which have the same structure as the first layer. However, the LDA classifier in the residual compensation layers is trained using new classes learned from the residual information of the previous layers. The input to each residual compensation layer is the previous layer's output combined with the original features. In the second layer, for instance, the convolutional output of the first layer $\{O_i^{(1)} \in \mathbb{R}^{m\times n\times d_1} : i = [1, 2, \ldots, N]\}$ is concatenated with the original features $\{X_i^{(1)} \in \mathbb{R}^{m\times n\times d} : i = [1, 2, \ldots, N]\}$ to create input $\{X_i^{(2)} \in \mathbb{R}^{m\times n\times (d+d_1)} : i = [1, 2, \ldots, N]\}$ for the second layer. As network depth increases, the deeper layers compensate for earlier layers' errors; consequently, combining the predicted values of the residual compensation layers enables the model to reach high accuracy. Moreover, it is worth mentioning that the layers in our proposed network are added sequentially, one by one, and trained in a single-pass non-iterative manner without gradient descent or backpropagation.

Given a dataset with N points $\{X_i^{(1)} \in \mathbb{R}^{m\times n\times d} : i = [1, 2, \ldots, N]\}$ and C classes $T^{(1)} \in \mathbb{R}^{N\times 1}$, the first layer of the proposed model, as shown in Figure 6.1, starts by computing the convolutional outputs using d_1 filters of any type. The convolutional outputs $\{O_i^{(1)} \in \mathbb{R}^{m\times n\times d_1} : i = [1, 2, \ldots, N]\}$ are then post-processed by applying a ReLU non-linear activation followed by second-order pooling. We then train an LDA classifier using the second-order features and the original classes $T^{(1)}$ to produce the posterior probabilities $\tilde{Y}^{(1)} \in \mathbb{R}^{N\times C}$ of the first layer. $T^{(2)} \in \mathbb{R}^{N\times 1}$ in the figure represents the new classes that we use to train the LDA classifier of the next layer. To compute $T^{(2)}$, we first calculate the residual errors between the predicted outputs $\tilde{Y}^{(1)}$ and the actual classes $Y \in \mathbb{R}^{N \times C}$, as follows:

$$R^{(1)} = \lambda Y - \tilde{Y}^{(1)}, \tag{6.1}$$

where Y denotes the real classes in one-hot encoding, and $0 \leq \lambda \leq 1$ determines the maximum probability that a class may achieve. That is to say, the network's probabilities will be constrained to values between zero and λ. $T^{(2)}$ can then be defined as the classes with the maximum absolute residual errors, which can be expressed as follows:

$$T_i^{(2)} = \text{class}(|R_{ij}^{(1)}|, \forall j) \quad i = [1, \ldots, N], \tag{6.2}$$

where $\text{class}(x)$ denotes the name of the class whose residual error magnitude is the largest. For instance, having three classes, A, B and C, with residual error values of 0.4, -0.6, and 0, respectively, class B is the nominated class because it has the maximum absolute residual error.

As shown in Figure 6.1, the second layer receives the first layer's output combined with the original features as input and calculates d_2 filters of any type for the second convolutional layer. After convoluting the images $X^{(2)} \in \mathbb{R}^{m \times n \times (d_1 + d) \times N}$ with the d_2 filters, we apply a ReLU non-linear activation function and then extract the second-order features. After identifying the features of the second layer, an LDA classifier is trained using these features and the newly generated classes $T^{(2)}$ to provide a correction term that is then added to the first layer's classification results $\tilde{Y}^{(1)}$ to obtain more accurate predictions $\tilde{Y}^{(2)}$. This correction term may take either a positive or negative value to maintain the network probabilities at the second layer $\tilde{Y}^{(2)}$ to be within the range of 0 to λ (equation 6.1). In other words, the first layer's posteriors $\tilde{Y}^{(1)}$ can be corrected by adding or subtracting the posteriors of the second layer to obtain more accurate classification results ($\tilde{Y}^{(2)}$). To do so, we introduce an indicator variable of the same size as the new labels ($T^{(2)}$), but with only two possible values, namely -1 or 1. This variable is defined as the signum function of the maximum absolute residual errors of the previous layer, as follows:

$$s_i^{(2)} = \text{sign}(R_{i^*}^{(1)}), \quad i^* = \arg\max_i |R_i^{(1)}|. \tag{6.3}$$

The indicator variable is responsible for indicating when to add or subtract from the previous layer's probabilities to maintain the correct range for the combined probabilities. With a positive indicator value, both the first- and second-layer probabilities are added. In contrast, negative indicator values indicate that the predicted probabilities of the second layer are subtracted from the predicted probabilities of the first layer. Suppose we have N images ($\{X_i^{(1)} \in \mathbb{R}^{m\times n\times d} : i = [1, 2, \dots, N]\}$) that need to be classified into one of three groups (A, B or C). For simplicity, let's assume that given an image $X_i^{(1)}$ of actual class A, the first layer predicts probabilities of 0.4, 0.6 and 0 for the classes A, B and C, respectively. To add a second layer, the residual errors of the classes A, B and C are computed as 0.4, -0.6 and 0 using equation 6.1 and $\lambda = 0.8$. Consequently, the new label for this image is B, since it has the highest magnitude, and the indicator is -1, as the maximum absolute residual error has a negative sign. Suppose that the second layer's features were trained using class B and accurately predicted probabilities of 0, 1 and 0 for the three classes, A, B and C, respectively. Because the indicator variable has a negative sign, subtracting the predicted probabilities of the second layer from those for the first layer provides final predictions of 0.4, -0.4 and 0 for the three classes, A, B and C, respectively. Therefore, this approach enables the network to correct the first layer's outputs and accurately predict that the image belongs to the A class, which is its actual label. In order to implement the previously mentioned idea and extend it to the test set whose classes are not seen, we divide the training dataset into negative and positive samples according to the values of their indicator variables. The positive instances are those with positive indicator values, whereas the negative examples have negative ones. Then we train two LDA classifiers for each layer, one for positive samples with their new classes $\{T_p^{(2)} \subset T^{(2)} : s^{(2)} = 1\}$ and the other for negative instances with their new labels $\{T_n^{(2)} \subset T^{(2)} : s^{(2)} = -1\}$. It should be emphasised that both classifiers have complete access to all training images. That is to say; the negative classifier is trained by treating each class in the positive samples as a negative class (its class is zero in one-versus-all decomposition). To the same extent, each class in the negative samples is considered a negative class during the training of the positive classifier. Ultimately, the output of the network at the second layer $\tilde{Y}^{(2)}$ is represented as follows:

$$\tilde{Y}^{(2)} = \tilde{Y}^{(1)} + \alpha[\frac{n_p}{N}\tilde{Y_p}^{(2)} - \frac{n_n}{N}\tilde{Y_n}^{(2)}], \tag{6.4}$$

where $\tilde{Y_n}^{(2)}$ and $\tilde{Y_p}^{(2)}$ represent the N predictions made by the classifiers trained on negative and positive samples, n_n and n_p denote the number of negative and positive samples, respectively, N is the total number of training samples and α

is the learning rate. The learning rate, similar to that used in neural networks, is introduced to reduce oscillations when adding more layers.

To add a third layer, we first find the new labels $T^{(3)}$ by computing the residual errors between the network's outputs at the second layer $\tilde{Y}^{(2)}$ and the actual classes in one-hot encoding $Y \in \mathbb{R}^{N\times C}$, as shown below:

$$\begin{aligned} T_i^{(3)} &= \text{class}(|R_{ij}^{(2)}|, \forall j) \\ &= \text{class}(|Y_{i,j} - \tilde{Y}_{i,j}^{(2)}|, \forall j), \quad i = [1, \dots, N] \end{aligned} \tag{6.5}$$

where class(x) represents the name of the class with the largest residual error magnitude, and N is the total number of instances in the database. The indicator variable for the third layer can be defined using the residual information of the previous layers, as follows:

$$s_i^{(3)} = \text{sign}(R_{i^*}^{(2)}), \quad i^* = \arg\max_i |R_i^{(2)}|. \tag{6.6}$$

Once the indicator variable and new labels for the third layer have been identified, the third layer employs the same procedures as the second layer. To calculate the output of the third convolutional layer, the output of the second layer is first concatenated with the original features to generate d_3 filters. Following the calculation of the convolutional layer, a ReLU activation is applied, and the second-order features are extracted. Moreover, the output of the third layer, $\tilde{Y}^{(3)}$, can be calculated using the following equation:

$$\tilde{Y}^{(3)} = \tilde{Y}^{(2)} + \alpha[\frac{n_p}{N}\tilde{Y_p}^{(3)} - \frac{n_n}{N}\tilde{Y_n}^{(3)}], \tag{6.7}$$

where

- $\tilde{Y_p}^{(3)}$ represents the probabilities predicted by a classifier trained using positive samples whose classes $T_p^{(3)} \subset T^{(3)}$ have positive indicator values $s^{(3)} = 1$;
- $\tilde{Y_n}^{(3)}$ shows the probabilities predicted by a classifier trained on negative examples with classes $T_n^{(3)} \subset T^{(3)}$ containing negative indicator values $s^{(3)} = -1$;
- n_p and n_n represent the number of the positive and negative samples, respectively;
- N is the total number of instances in the database;

- α, whose value is between 0 and 1, is a learning rate that prevents the oscillation in performance when additional layers are added.

Algorithm 7 summarises the training procedures of the deep residual compensation convolutional network with L layers. For deeper residual compensation layers, it is possible to compute the network outputs at layer L ($\tilde{Y}^{(L)}$) using the new classes ($T^{(L)}$) learnt from the residual errors of the previous layers, which can be defined as follows:

$$\begin{aligned} T^{(L)} &= \text{class}(|R^{(L-1)}|) \\ &= \text{class}(|Y - \tilde{Y}^{(L-1)}|) \\ &= \text{class}(|Y - [\tilde{Y}^{L-2} + \frac{\alpha}{N}(n_p^{(L-2)}\tilde{Y}_p^{(L-2)} - n_n^{(L-2)}\tilde{Y}_n^{(L-2)})]) \\ &= \dots \\ &= \text{class}(|Y - [\tilde{Y}^{(1)} + \frac{\alpha}{N}\sum_{i=2}^{L-1}(n_p^i\tilde{Y}_p^i - n_n^i\tilde{Y}_n^i)]), \end{aligned} \tag{6.8}$$

where Y is the original classes in one-hot encoding, n_p^i and n_n^i are the number of positive and negative samples in layer i, respectively, and N denotes the total number of samples in the database.

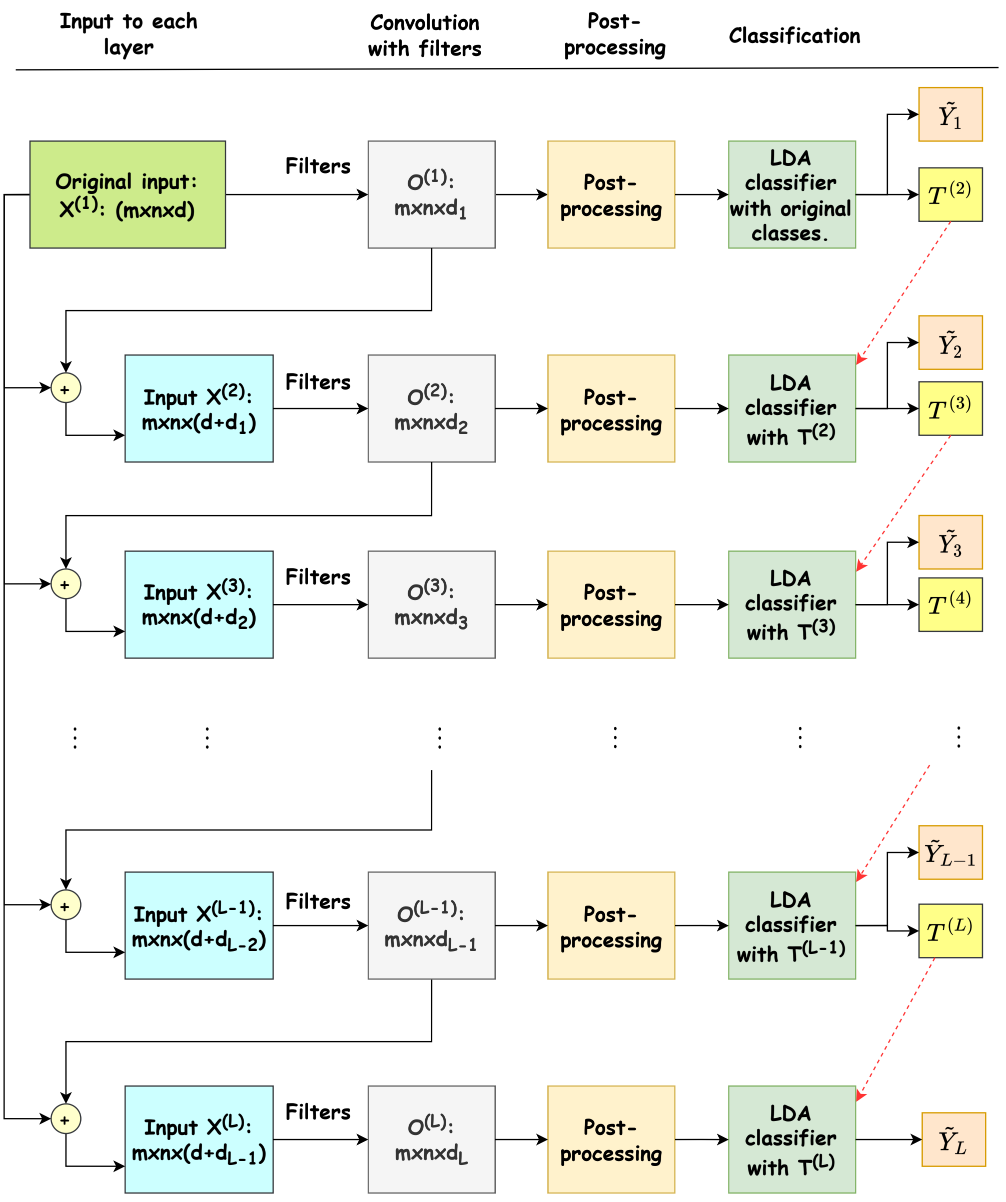


FIGURE 6.1: Deep residual compensation convolutional network.

Algorithm 7 Deep Residual Compensation Convolutional Network Training

```
Input: Training images: $\{X_i^{(1)} \in \mathbb{R}^{m\times n\times d} : i = [1, 2, \ldots, N]\}$, $C$ classes $T^{(1)} \in \mathbb{R}^{N\times 1}$, learning rate: $\alpha$, number of layers: $L$ and $\lambda$ to determine the highest probability a class can reach.
Output: Model's accuracy: $accuracy$
 1: Generate $Y \in \mathbb{R}^{N\times C}$, which is the one-hot encoding of $T^{(1)}$.
 2: $i \leftarrow 1$.
 3: $\tilde{Y}^{(0)} \leftarrow 0$
 4: $s^{(i)} = 1^{N\times 1}$   ▷ fill all entries of the indicator variable of the first layer with the value 1.
 5: while $i < L$ do
 6:     if $i > 1$ then                ▷ for the residual compensation layers
 7:         Normalise $O^{(i-1)}$ such that all of its elements have values between 0 and 1, as follows:
                $$O^{(i-1)} = \frac{O^{(i-1)} - \min(O^{(i-1)})}{\max(O^{(i-1)}) - \min(O^{(i-1)})}. \quad (6.9)$$
 8:         Concatenate the output of the previous layer, $O^{(i-1)} \in \mathbb{R}^{m\times n\times d_{i-1}\times N}$, with the original features, $X^{(1)} \in \mathbb{R}^{m\times n\times d\times N}$, to get the input of the current layer, $X^{(i)} \in \mathbb{R}^{m\times n\times (d_{i-1}+d)\times N}$.
 9:         Compute the residual errors between the previous layer's output and the original classes $Y$, as follows:
                $$R^{(i-1)} = \lambda Y - \tilde{Y}^{(i-1)}. \quad (6.10)$$
10:         The current layer's new labels are defined as the maximum absolute residual error, as follows:
                $$T_j^{(i)} = \text{class}(|R_{jk}^{(i-1)}|, \forall k), \quad j = [1, 2, \ldots, N]. \quad (6.11)$$
11:         Define the indicator variable for the current layer as the signum function of the maximum absolute residual errors, as follows:
                $$s_j^{(i)} = \text{sign}(R_{j^*}^{(i-1)}), \quad j^* = \arg\max_j |R_j^{(i-1)}|. \quad (6.12)$$
12:     end if
13:     Compute $d_i$ filters $(W^{(i)})$ using the inputs $X^{(i)}$.
14:     Find the convolutional outputs of the current layer $O^{(i)}$ using $X^{(i)}$ and the $d_i$ filters, as follows:
                $$O^{(i)} = X^{(i)} * W^{(i)}, \quad (6.13)$$
        where $X^{(i)}$ is zero-padded to generate images with the same spatial dimensions as the input images.
```

```
15:     Apply a ReLU non-linear activation to the convolutional outputs $O^{(i)}$,
        as follows:
                    $$Out^{(i)} = \text{ReLU}(O^{(i)}). \quad (6.14)$$
16:     Extract the second-order features ($F^{(i)}$) of $Out^{(i)}$.
17:     Find $F_n^{(i)} \subset F^{(i)}$ and $T_n^{(i)} \subset T^{(i)}$, for which their indicator values are
        negative.
18:     Find $F_p^{(i)} \subset F^{(i)}$ and $T_p^{(i)} \subset T^{(i)}$, which have positive indicator values.
19:     if $T_p^{(i)} \neq \emptyset$ then
20:         Find an LDA classifier between $F_p^{(i)}$ and $T_p^{(i)}$: $L_1 = \text{LDA}(F_p^{(i)}, T_p^{(i)})$.
21:         Predict the posterior probabilities $\tilde{Y}_p^{(i)}$ using $L_1$, as follows:
                    $$\tilde{Y}_p^{(i)} = \text{prediction}(L_1, F^{(i)}). \quad (6.15)$$
22:     else
23:         $\tilde{Y}_p^{(i)} \leftarrow 0$.
24:     end if
25:     if $T_n^{(i)} \neq \emptyset$ then
26:         Find an LDA classifier between $F_n^{(i)}$ and $T_n^{(i)}$: $L_2 = \text{LDA}(F_n^{(i)}, T_n^{(i)})$.
27:         Predict the posterior probabilities $\tilde{Y}_n^{(i)}$ using $L_2$, as follows:
                    $$\tilde{Y}_n^{(i)} = \text{prediction}(L_2, F^{(i)}). \quad (6.16)$$
28:     else
29:         $\tilde{Y}_n^{(i)} \leftarrow 0$.
30:     end if
31:     Compute the current's layer output $\tilde{Y}^{(i)}$, as follows:
                    $$\tilde{Y}^{(i)} = \tilde{Y}^{(i-1)} + \alpha[\frac{n_p}{N}\tilde{Y_p}^{(i)} - \frac{n_n}{N}\tilde{Y_n}^{(i)}], \quad (6.17)$$
        where, $n_p$ and $n_n$ represent the number of the positive and negative
        samples, respectively.
32:     $i \leftarrow i + 1$
33: end while
34: Compute the accuracy of the model at layer $L$ using $\tilde{Y}^{(L)}$.
```

6.3 Experiments and Results

This section has three primary subsections. The first subsection (Section 6.3.1) aims to investigate how increasing the number of filters (Section 6.3.1.1) or the learning rate (Section 6.3.1.2) affects the performance of the deep residual compensation convolutional network. In the second subsection (Section 6.3.2), we evaluate the proposed network over four databases, namely the MNIST, CIFAR-10, CIFAR-100 and TinyImageNet, and compare the network's accuracy to that of some gradient-based and non-gradient-based architectures with no data augmentation. The final subsection (Section 6.3.3) looks at enhancing the model's accuracy using data augmentation.

6.3.1 Network Parameters

This section aims to analyse how increasing the number of filters or the learning rate would affect the performance of the deep residual compensation convolutional network. The database used in this section is the CIFAR-10, and the experiments are organised into two subsections as follows:

- **Experiment 1**: Testing the effect of increasing the number of filters on the deep residual compensation convolutional network's accuracy;
- **Experiment 2**: Studying the effect of changing the learning rate on the model's performance.

6.3.1.1 Experiment 1

6.3.1.1.1 Parameter Settings This experiment aimed to determine how increasing the number of filters affects the accuracy of the deep residual compensation convolutional network. In this experiment, and throughout the rest of the chapter, we used the same number of filters for all layers and stopped adding layers when the training error rate reached 0%. In order to achieve our objective of investigating how the number of filters affects the network's performance, we created two deep residual compensation convolutional networks with the same network parameters but different numbers of filters. The first network used 30 filters across all of its layers, whereas the second employed 50 in the same way. The two networks used the semi-supervised Stacked-LDA filters described in Chapter 5, which consisted of 50% PCA and 50% Stacked-LDA filters. For instance, in the network with 30 filters across all of its layers, the total number of filters in each layer was calculated by combining 15 PCA filters and 15 stacked LDA filters. The Stacked-LDA filters were computed using the settings discussed in Section 4.4.2.1.1 of Chapter 4. After obtaining the output of each convolutional layer, we used a ReLU non-linear activation function; however, this function was not utilised between the layers. Across all layers of the two networks, the second-order pooling block size was set to 8×8 with a stride of 1 pixel. The output of the second-order pooling was linked to three-level spatial pyramid pooling with 16, 4 and 1 bin, respectively, at each level. In addition, the learning rate (α) was set to 0.4, while λ (equation 6.10) was fixed to 0.8. LDA using the one versus all decomposition approach, discussed in Section 4.2.2.1, was used as the classifier in each layer of the two networks. To calculate the posterior probabilities, we used the sigmoid function with the scaling parameter set to 16, as shown in equation 4.1 of Chapter 4.

6.3.1.1.2 Performance Analysis In this section, we report the performance of the two investigated networks using 30 and 50 filters on the CIFAR-10 database. For simplicity, we refer to the residual compensation layer network with 30 filters in all of its layers as ResCNet-30, while the network with 50 filters in all of its layers is referred to as ResCNet-50.

Figures 6.2 and 6.3 show the error rates of the training and testing sets of the CIFAR-10 database using the ResCNet-30 and ResCNet-50 networks, respectively. According to the figures, the performance was the same for both ResCNet-30 and ResCNet-50, despite the fact that ResCNet-30 required more layers to achieve the same level of performance. ResCNet-50 obtained a training error rate of 0% at layer 127, while ResCNet-30 achieved that of 0% at layer 781. The test error rate for both 127- and 781-layer networks was around 13.72%. This finding suggested that increasing the number of filters in the ResCNet architecture from 30 to 50 did not significantly improve performance. Surprisingly, we achieved comparable performance results using either a smaller (30) or larger (50) number of filters. Nonetheless, using a smaller number of filters would require many layers to reach the same level of performance. Therefore, throughout the rest of this chapter, we evaluated the deep residual compensation convolutional network using the maximum possible number of filters.

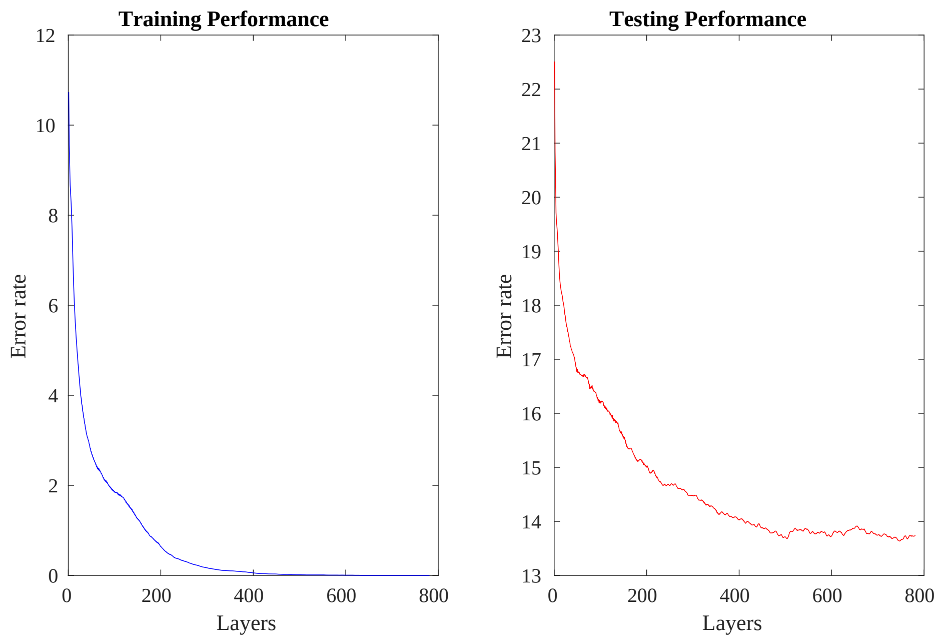


FIGURE 6.2: The performance (error rate %) of a deep residual compensation convolutional network with 30 filters throughout all of its layers on the CIFAR-10 database.

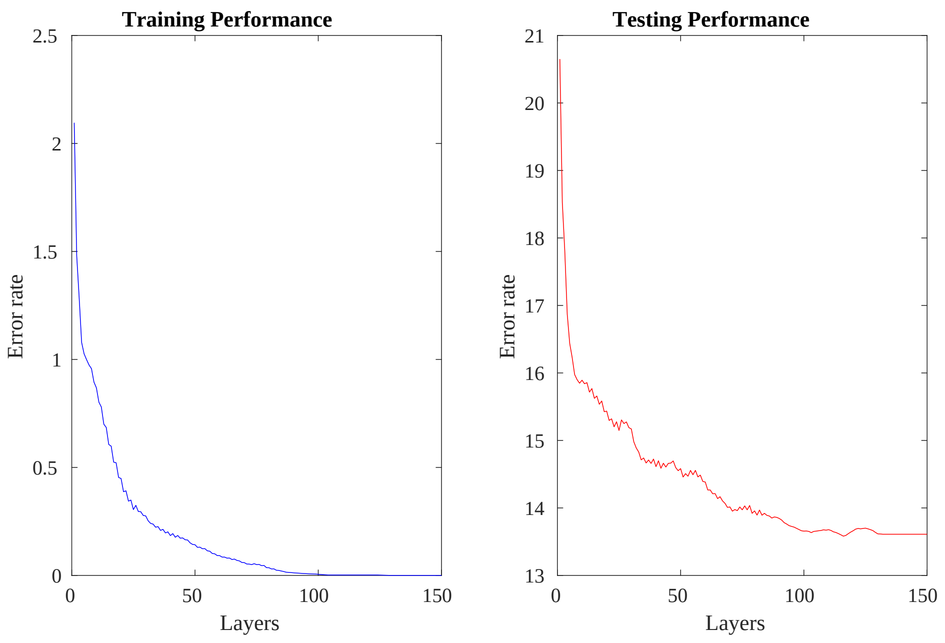


FIGURE 6.3: The performance (error rate%) of a deep residual compensation convolutional network with 50 filters throughout all of its layers on the CIFAR-10 database.

From Figures 6.2 and 6.3, we can observe that the error rate of both ResCNet-30 and ResCNet-50 dropped rapidly in the first few layers and remained relatively stable when the training error rate was close to zero. ResCNet-50's performance, for instance, remained steady after 100 layers, but ResCNet-30's error rate curve stabilised after 500 layers. In other words, training ResCNet-50 with 100 layers and ResCNet-30 with 500 layers was sufficient to obtain good performance. The test error rate obtained by 500-layer ResCNet-30 and 100-layer ResCNet-50 was 13.67%.

In general, the deep residual compensation network achieved high accuracy (86.33%) on the CIFAR-10 database, with no data augmentation. This performance demonstrated the effectiveness of our network over all of the shallow-depth networks described in Chapters 3, 4 and 5. Our findings in this section showed that while increasing the number of filters did not necessarily improve ResCNet's accuracy, using a higher number of filters is generally better than relying on a small number of filters. If no improvement has been observed in accuracy, the model converges faster with a large number of filters. It is also worth mentioning that we used only two filters values to study the impact of increasing the number of filters in the ResCNet architecture, as reducing the number of filters (< 30) would require adding many layers while using a larger number of filters (e.g., 100) would generate a large number of features.

6.3.1.2 Experiment 2

6.3.1.2.1 Parameter Settings This experiment aims to show the impact of the learning rate, denoted by α in Equation 6.17, on the proposed model's accuracy. The learning rate in our model is intended to avoid oscillations and produce faster convergence, similar to that of neural networks. In contrast to neural networks, the learning rate in our model also serves as a weight to integrate the probabilities of multiple layers, similar to weighted sum techniques.

To understand the influence of the learning rate on the model's performance, we created different networks with identical configurations but different learning rates (α) and evaluated them on the CIFAR-10 database. All networks used 50 filters throughout all of their layers, with α being set to different numbers ranging from 0.2 to 1. The number of layers in all networks was fixed at 30. For convenience, we referred to a network with $\alpha = L$ and 50 filters in each of its layer as ResCNet-50–L; hence the network with $\alpha = 0.4$ is referred to as ResCNet-50–0.4. In all networks, we implemented the 3×3 semi-supervised Stacked-LDA filters described in Chapter 4 by combining 50% PCA filters and 50% stacked-LDA filters. In addition, the Stacked-LDA filters were generated using the parameters defined in Section 4.4.2.1.1 of Chapter 4. After each convolutional layer, the ReLU activation function was applied, and the second-order features were retrieved using 16×16 blocks with stride $= 1$. The second-order features were then pooled using three-level spatial pyramid pooling with the number of bins equal to 16, 4 and 1. Moreover, the linear discriminant analysis with one-versus-all binary decomposition was the classifier used for each layer. The probabilities were obtained using the sigmoid function with a scale parameter of 16, as in equation 4.1 (Chapter 4).

6.3.1.2.2 Performance Analysis Figure 6.4 shows the training and testing error rates of ResCNet-50 with different learning rates ranging from 0.2 to 1 on the CIFAR-10 dataset. As shown in the figure, both the training and testing error rates consistently decreased over time. However, the convergence was slower when using lower learning rates. For instance, at layer 30, the training error rate of ResCNet-50–1 was 3.6% (accuracy=96.40%), while the testing error rate was 15.7% (accuracy = 84.30%). In comparison, at the same layer, the training error rate of ResCNet-50–0.2 was much higher at 7.05% (accuracy = 92.95%), while the testing error rate was 16.87% (accuracy = 83.18%). Overall, running ResCNet-50 for 30 layers with a learning rate of 1 provided the best accuracy and faster convergence rate. However, selecting an appropriate learning rate to balance the trade-off between speed and accuracy is essential.

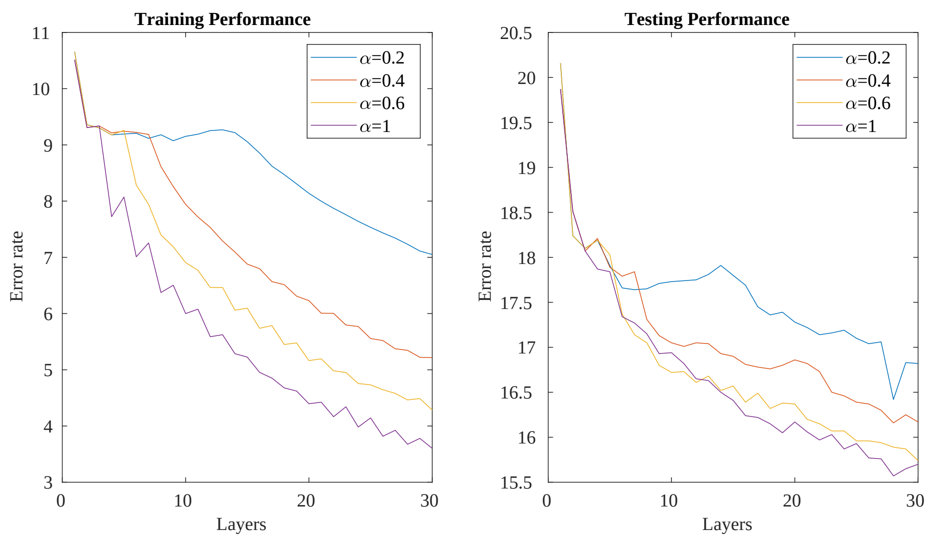


FIGURE 6.4: The performance (error rate %) of a deep residual compensation convolutional network with 50 filters throughout all of its layers on the CIFAR-10 database. The experiment was evaluated by setting α (in equation 6.17) to different values ranging from 0.2 to 1, and the number of layers was set to 30.

To further illustrate the impact of the learning rate on the accuracy of the deep residual compensation convolutional network, we trained ResCNet-50–1 and ResCNet-50–0.4 until their training error rates reached 0%. Figures 6.5 and 6.6 show the error rates of training and testing these two networks on the CIFAR-10 database. In ResCNet-50-1, the training error rate went to 0% at layer 348, whereas the testing error rate at that layer was 13.09% (testing accuracy was 86.91%). In contrast, ResCNet-50–0.4 achieved a training error rate of 0% at layer 970, with a testing error rate of 12.59% (accuracy of 87.41%). The best testing error rate obtained by ResCNet-50–1 was 12.98% at layer 208 (accuracy: 87.02%). In contrast, ResCNet-50–0.4 reached its best testing error rate, 12.46% (accuracy: 87.54%), at layer 937. Figure 6.6 shows that the training error rate of ResCNet-50–0.4 did not significantly improve after 600 layers; hence, 600 layers were sufficient to train ResCNet-50–0.4, for which the testing error rate at that layer was 12.91% (accuracy: 87.09%). The above findings are summarised in Table 6.1.

TABLE 6.1: Accuracy (%) on the CIFAR-10 database test set using ResCNet-50–1 and ResCNet-50–0.4 with different numbers of layers.

Network	# Layers	Accuracy (%)
ResCNet-50–1	348	86.91
ResCNet-50–1	208	87.02
ResCNet-50–0.4	970	87.41
ResCNet-50–0.4	937	**87.54**
ResCNet-50–0.4	600	87.09

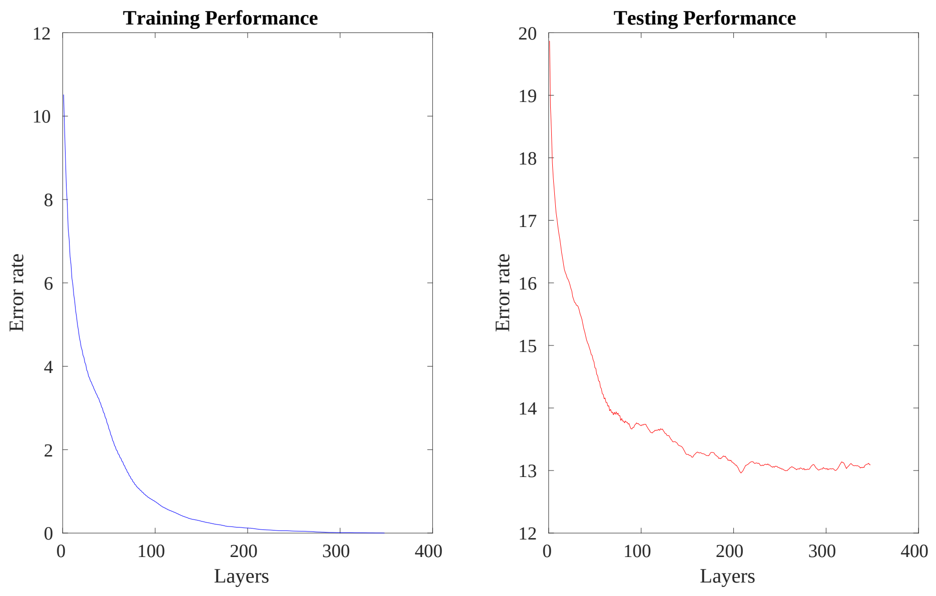


FIGURE 6.5: The performance (error rate %) of a deep residual compensation convolutional network with 50 filters throughout all of its layers on the CIFAR-10 database, where α (in equation 6.17) was set to 1.

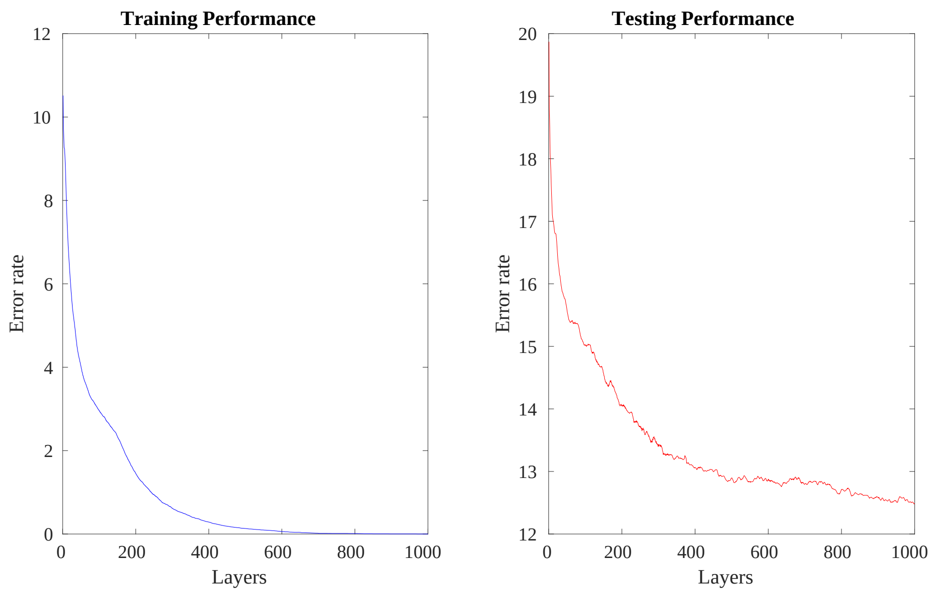


FIGURE 6.6: The performance (error rate %) of a deep residual compensation convolutional network with 50 filters throughout all of its layers on the CIFAR-10 database, where α (in equation 6.17) was set to 0.4.

Based on the findings of this section and Section 6.3.1.1, we can derive the following conclusions:

- Using the CIFAR-10 database, the second-order block size had an impact on the accuracy of the deep residual compensation convolutional network. For instance, the highest accuracy attained in this section (87.54%) was around 1% higher than the best accuracy achieved in Section 6.3.1.1 (86.33%), with the only difference between the models being the modification of the second-order block size. In the previous section, the models used 8×8 second-order blocks; however, in this section, 16×16 second-order blocks were utilised. Even though the improvement was just 1%, time could be spent modifying the model's parameters, such as the filter type, second-order block size and non-linear activation, to improve the model's accuracy further.

- The learning rate determines how fast the model adapts to the problem, and this section's findings indicate that it was a crucial aspect in training the deep residual compensation convolutional network. Using the CIFAR-10 database as an example, the results in this section demonstrated that the learning rate $\alpha = 0.4$ was small, since the model needed a number of layers for coverage (600 layers). If the training error rate oscillates, which is not the case in the CIFAR-10 database, the learning rate must be decreased to avoid oscillations.

6.3.2 Image Classification over Four Benchmarks with No Data Augmentation

In this section, we evaluate the deep residual compensation convolutional network on four benchmarks, namely the MNIST (Section 6.3.2.1), CIFAR-10 (Section 6.3.2.2), CIFAR-100 (Section 6.3.2.3) and TinyImageNet (Section 6.3.2.4). In the previous section (Section 6.3.1), our experiments showed that we could use either a big or small number of filters and still achieve the same level of accuracy, albeit with more layers in the latter case. However, the accuracy was shown to be affected by the second-order pooling block size, as stated in Section 6.3.1.2. As a result, the number of filters we use in all experiments described in this section has been determined to be as large as possible, given the available resources. To determine the values for the other network parameters, such as the second-order pooling block size and stride, we examine a variety of architectural configurations, each of which having different parameters, and then report the results of the configuration that works the best. Our experiments in this section indicate that, in terms of accuracy, our network outperforms all non-differentiable networks and is comparable with standard gradient-based models. The experiments also show that our network presents an advantage over gradient-based models regarding the number of floating operations (FLOPs) required.

6.3.2.1 Handwritten Digit Recognition on the MNIST Database

6.3.2.1.1 Parameters Settings The optimal settings that we identified for the MNIST database after examining several parameters are listed in Table 6.2. As shown in Table 6.2, the network architecture consisted of 231 layers, each containing 60 filters. Based on the results of our experiments in Chapters 3, 4 and 5, it was found that there is no remarkable improvement in the accuracy of the MNIST database when different filter types are being used. As a result, we used PCA to generate the filters for all layers in this experiment, as this method is generally considered a computationally efficient algorithm. The implementation of the PCA filters was discussed in Section 3.2.1 of Chapter 3. The initial layer of the architecture in this experiment used a 13×13 filter size, while the subsequent layers utilised a 3×3 filter size. In addition, we used a ReLU activation function after each convolutional layer but not between layers. The filter responses of each layer were post-processed using second-order pooling with a block size of 7×7 and a stride of 4 pixels. The second-order features of each layer were then pooled using three-level spatial pyramid pooling with 16, 4 and 1 bins per level. We ran the LDA classifier for each layer with the probabilities extracted using the softmax function, as follows:

$$\text{softmax}(y_i) = \frac{\exp(\beta y_i)}{\sum_{j=1}^{C} \exp(\beta y_j)}, \tag{6.18}$$

where β was assigned to 0.001, C denotes the number of classes and y represents the outputs of the LDA classifier.

TABLE 6.2: Network architectures using the MNIST, CIFAR-100 and TinyImageNet databases.

The MNIST database: $28 \times 28 \times 1$			
Filter size	**SOP**	**SPP (#bins)**	**Output size**
$13 \times 13 \times 1 \times 60$	7×7, Stride$= 4$	[16 4 1]	$28 \times 28 \times 60$
$[3 \times 3 \times 60 \times 60] \times 230$	7×7, Stride$= 4$	[16 4 1]	$28 \times 28 \times 60$
The CIFAR-100 database: $32 \times 32 \times 3$			
Filter size	**SOP**	**SPP (#bins)**	**Output size**
$3 \times 3 \times 3 \times 50$	16×16, Stride$= 4$	[16 4 1]	$32 \times 32 \times 50$
$[3 \times 3 \times 50 \times 50] \times 435$	16×16, Stride$= 4$	[16 4 1]	$32 \times 32 \times 50$
The TinyImageNet database: $64 \times 64 \times 3$			
Filter size	**SOP**	**SPP (#bins)**	**Output size**
$3 \times 3 \times 3 \times 40$	32×32, Stride$= 8$	[16 4 1]	$64 \times 64 \times 40$
$[3 \times 3 \times 40 \times 40] \times 511$	32×32, Stride$= 8$	[16 4 1]	$64 \times 64 \times 40$

6.3.2.1.2 Performance Analysis Figure 6.7 shows the training and testing error rates of 231-layer ResCNet-60 on the MNIST database, while Table 6.3 compares its accuracy with that of PCANet, LDANet, Stacked-LDA, Maxout network, Network in Network and Stochastic pooling. As shown in Figure 6.7, the training error rate decreased dramatically over the first few layers and fluctuated somewhat towards the end; nonetheless, the overall training error rate curve was smooth, since the training started with a very low error rate. Similarly, the testing error rate dropped rapidly at first and fluttered when it reached 0.54%. Adding more layers to the 231-layer network did not enhance the network's performance; thus, the training was terminated. With a 99.52% accuracy, ResCNet-60 outperformed all non-differentiable models such as PCANet, Multi-Layer PCANet, LDANet, and Stacked-LDA, as shown in Table 6.3. Our model improved on the best non-differentiable network results by around 0.12%. Furthermore, as shown in the table, our network produced results that were comparable to, but not better than, gradient-based networks such as the Maxout network, Network in Network, and stochastic pooling. Based on these results, it can be deduced that our network is reliable and on par with standard gradient-based models.

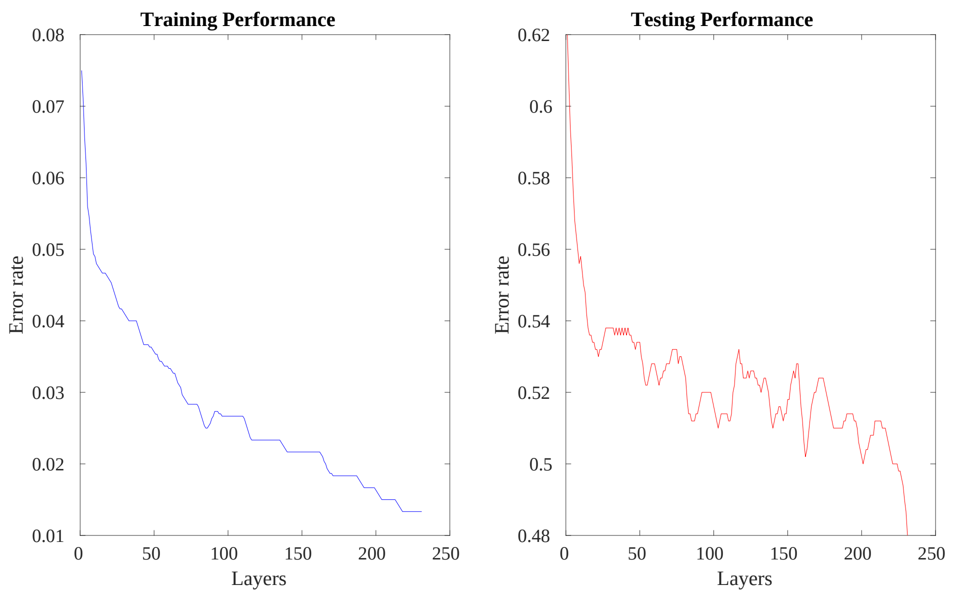


FIGURE 6.7: The training and testing error rates using the deep residual compensation network on the MNIST database.

TABLE 6.3: Accuracy of the deep residual compensation network compared with different methods on the MNIST database, with no data augmentation.

Non-differentiable networks	
Method	**Accuracy (%)**
PCANet-1 [15]	99.06
PCANet-2 [15]	99.34
LDANet-1 [15]	99.02
LDANet-2 [15]	99.38
PCANet-1 ($k = 13$) [15]	99.38
Multi-Layer PCANet (Chapter 3)	99.40
Stacked-LDA (Chapter 4)	99.39
ResCNet-60 (this chapter)	**99.52**
Gradient-based networks	
Method	**Accuracy (%)**
Stochastic pooling [8]	99.53
Maxout network [7]+Dropout	**99.55**
Network in network [31]+Dropout	99.53

6.3.2.2 Pattern Recognition on the CIFAR-10 Database

The ResCNet-50–0.4 model with 937 layers, presented in Section 6.3.1.2, achieved the best accuracy on the CIFAR-10 database, at 87.54%. Table 6.4 compares the accuracy achieved by this network with that obtained by other architectures, namely PCANet, Multi-Layer PCANet (Chapter 3), Stacked-LDA (Chapter 3), Maxout, Network in Network and Stochastic pooling. According to the results shown in the table, our non-differentiable deep residual compensation convolutional network (ResCNet) outperformed all of the other non-differentiable networks, namely PCANet, Multi-Layer PCANet and Stacked-LDA, in terms of accuracy. The accuracy of the proposed network (highlighted in blue in Table 6.4) was around 10% better than the performance achieved by the original PCANet, 6% higher than that of the Multi-Layer PCANet and 3% more than the one produced by the Stacked-LDA model. This achievement shows that our model was the best alternative to use compared with all other non-differentiable networks.

In order to demonstrate how well our model performs in comparison to neural networks, we compared its performance with that of other gradient-based models, namely Maxout, Network in Network and Stochastic pooling, as well as several residual networks. The results of these comparisons are presented in Table 6.4. Despite the fact that ResCNet's accuracy was around 1% lower than that of the Maxout network and 2% worse than that of Network in Network, our model achieved accuracy comparable with that of ResNet-34, 1% higher than ResNet-18 and ResNet-110, and 3% greater than stochastic pooling. In general, the results obtained by our model demonstrated its superiority over all existing non-differentiable networks and its competitiveness over standard gradient-based models.

TABLE 6.4: Accuracy of the deep residual compensation network compared with different methods on the CIFAR-10 and CIFAR-100 databases with no data augmentation.

Non-differentiable networks		
Method	**CIFAR-10**	**CIFAR-100**
PCANet-2 [15]	77.14	51.62
PCANet (combined) [15]	78.68	–
Multi-Layer PCANet (Chapter 3)	81.72	57.86
Stacked-LDA (Chapter 4)	84.44	59.41
ResCNet (this chapter)	**87.54**	**64.9**
Gradient-based networks		
Method	**CIFAR-10**	**CIFAR-100**
Stochastic pooling [8]	84.87	57.49
Maxout network [7] with dropout	88.32	61.43
Network in network [31] with dropout	**89.59**	64.32
Network in network [31] without dropout	85.49	–
110 ResNet, reported by [45, 75]	86.82	55.26
ResNet stochastic depth [45]	-	62.20
164-ResNet (pre-activation), reported by [75]	-	**64.42**
ResNet-18, reported by [90]	86.29	59.15
ResNet-34, reported by [90]	87.97	56.05

In terms of accuracy, the results that our model produced were on par with those produced by standard gradient-based models. However, to highlight the advantages of using our model over gradient-based models, we compared our model's total number of FLOPs[1] during the training phase on the CIFAR-10 database with those of different residual networks, namely ResNet-20, ResNet-32, ResNet-44 and ResNet-56. The results of these comparisons are described in Table 6.5. With the same settings as in [5] and [100], we re-ran the residual

[1]Measured in quadrillions (P)

networks using the CIFAR-10 images but with no data augmentation; the results reported in Table 6.5 are the average of five runs. Table 6.5 shows that compared to other residual networks, not only did our model have the best accuracy, but it also required the fewest FLOPs. ResCNet, with 937 layers, used somewhat fewer FLOPs than ResNet-20 in the worst-case scenario. The other versions of the ResCNet required a much fewer number of FLOPs compared with the residual networks. These findings suggest that our non-differentiable network can compete with standard gradient-based models in terms of accuracy and the number of FLOPs needed for training.

TABLE 6.5: Accuracy and number of FLOPs of the deep residual compensation network compared with different residual networks on the CIFAR-10 database with no data augmentation.

Residual networks		
Method	**Accuracy (%)**	**#FLOPs (P)**
Residual network (ResNet-20)	84.442	1.2951
Residual network (ResNet-32)	84.664	2.1444
Residual network (ResNet-44)	83.176	2.8953
Residual network (ResNet-56)	83.29	3.8433
Deep residual compensation convolutional network (this chapter)		
Method	**Accuracy (%)**	**#FLOPs (P)**
ResCNet-50–0.4 (937 layers), [Section 6.3.1.2]	**87.54**	1.1263
ResCNet-50–1 (208 layers), [Section 6.3.1.2]	87.07	0.2512
ResCNet-50 (100 layers), [Section 6.3.1.1]	86.33	**0.1256**

6.3.2.3 Image Classification on the CIFAR-100 Database

6.3.2.3.1 Parameter Settings Table 6.2 describes the architecture used to evaluate the deep residual compensation convolutional network on the CIFAR-100 database. As shown in the table, the architecture consisted of 436 layers with 50 filters used for each layer. We used a combination of 25 PCA filters and 25 Stacked-LDA filters per layer, with the filter size for each layer being 3×3. The Stacked-LDA filters were implemented with the configurations described in Section 4.4.2.1.1 of Chapter 4. After each convolutional layer, we applied a ReLU function followed by second-order pooling with a block size of 16×16 and a stride of 4. The second-order features were then pooled using three-level spatial pyramid pooling with 4×4, 2×2 and 1×1 subwindows. The LDA was employed as the classifier for each layer, with the sigmoid function utilised to extract the probabilities. The scale parameter for the sigmoid function was set to 16, as described by equation 4.1. λ in equation 6.1 was set to 0.8, and the initial value of the learning rate was 1, after which its value was reduced by 10% for every ten layers using the following formula:

$$\alpha = \alpha - \frac{10}{100}\alpha. \tag{6.19}$$

The learning rate continued to fall by 10% every ten layers until it hit 0.387, at which point it stopped decreasing further.

6.3.2.3.2 Performance Analysis Figure 6.8 shows the training and testing error rates of each layer of the proposed network on the CIFAR-100 database. As shown in the figure, the error rate of the training decreased considerably during the first 57 layers, but the improvement in error rate was small afterwards. The training accuracy at layer 57 was 99.72%, whereas the accuracy on the test set was 63.22%. The network's optimal performance was achieved at layer 436, where training accuracy was 99.92% and testing accuracy was 64.92%.

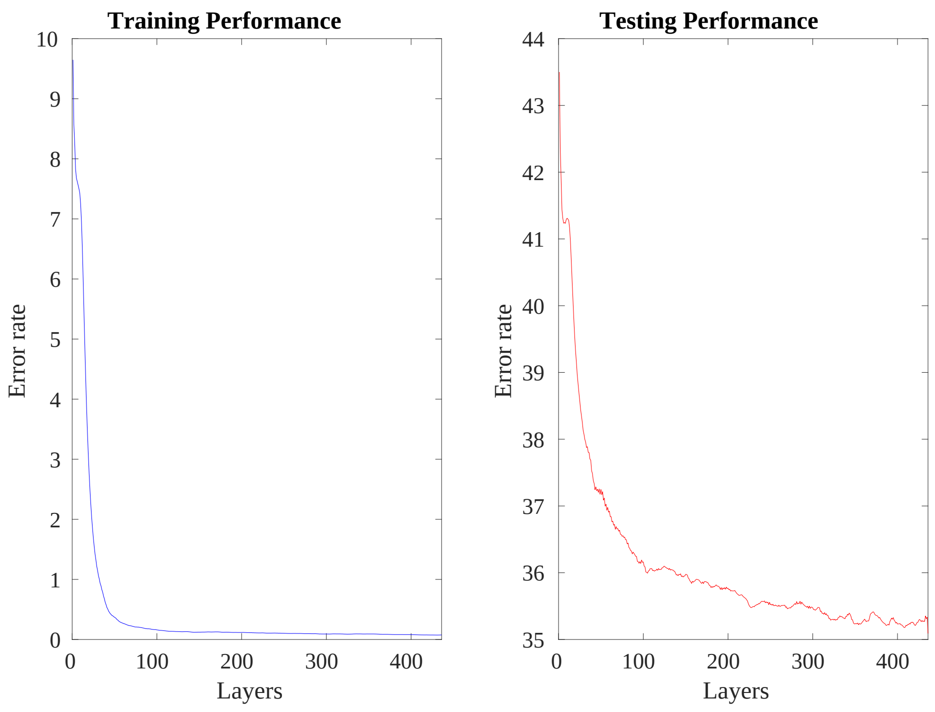


FIGURE 6.8: The training and testing error rates (%) for the deep residual compensation convolutional network trained on the CIFAR-100 database.

Table 6.4 compares the accuracy of the deep residual compensation convolutional network with that of several other networks, such as PCANet, Multi-Layer PCANet, Stacked-LDA, Stochastic pooling, Maxout network, Network in Network and other residual networks, on the CIFAR-100 database. The accuracy attained by our network, which is highlighted in blue in the table, was the highest among all the networks. The accuracy of 64.91% was 6% higher than that of Stacked-LDA and ResNet-18, around 8% higher than that of Multi-Layer PCANet and stochastic pooling, more than 9% higher than that of ResNet-110 and ResNet-34, 14% higher than the original PCANet and more than 2% higher than that of Maxout and ResNet with stochastic depth. Moreover, the results achieved by our network were approximately equivalent to those obtained by ResNet with 164 layers and Network in Network using the dropout technique. This gain in accuracy once again showed that our network outperformed every non-gradient-based model and was on par with standard gradient-based models that achieved the best results.

6.3.2.4 Classification on the TinyImageNet Database

6.3.2.4.1 Parameter Settings As shown in Table 6.2, the architecture used for the TinyImageNet database consisted of 512 layers, each of which had 40 filters. The input to each convolutional layer was normalised to have values of between 0 and 1, and each layer used 3×3 semi-supervised filters that were created by combining 20 PCA with 20 Stacked-LDA filters. The Stacked-LDA filters were produced using the parameters specified in Section 4.4.2.1.1 of Chapter 4. After each convolutional layer, a ReLU non-linear activation followed by a second-order pooling and spatial pyramid pooling were applied. The feature maps were pooled using a 32×32 second-order pooling with an 8-step stride, and then the pooled features were sent to a three-level spatial pyramid pooling with a total of 21 bins. The LDA was applied to each layer as a classifier, and the probabilities were calculated using the sigmoid function specified in equation 4.1. λ in equation 6.1 was set to 0.5, and the learning rate's initial value was set to 1. The value of the learning rate dropped by 10% every ten layers (equation 6.19) until it reached 0.478, at which point it stabilised.

6.3.2.4.2 Performance Analysis Figure 6.9 depicts the error rate of training and testing each layer of the deep residual compensation convolutional network on the TinyImageNet database. As indicated in the figure, the error rate dropped rapidly for the first 74 layers and more gradually afterwards. The testing accuracy at layer 74 was 43.01%, and the training accuracy was 91.06%. With 200 layers, the training accuracy increased to 98.37%, and the testing accuracy reached 43.23%. The best accuracy obtained by the network was achieved at layer 512, with 99.43% training accuracy and 44.37% testing accuracy. Table 6.6 compares the performance obtained by our network with that of some residual networks and non-gradient-based models. According to Table 6.6, our network achieved the highest accuracy compared with all other networks, without any data augmentation. This performance was around 3% greater than that of the Stacked-LDA model, 2% higher than that of the ResNet-34 model and 1% better than that of the ResNet-18 model. Again, our model with simple processing steps and no complicated functions or regulation approaches produced results equivalent to those of standard differentiable networks and superior to those of all non-gradient-based models.

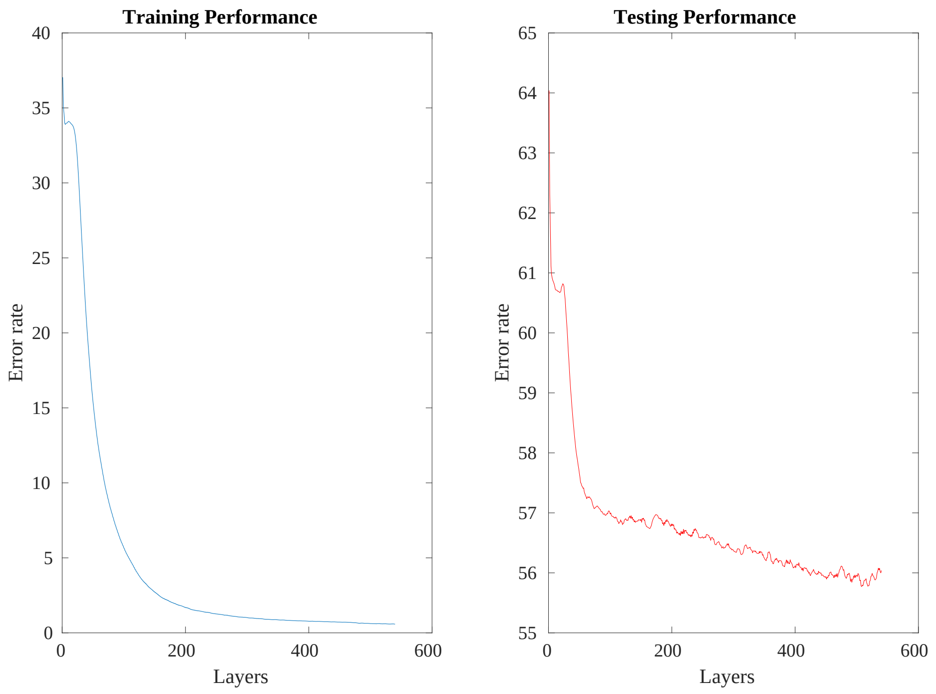


FIGURE 6.9: The training and testing error rates (%) for the deep residual compensation convolutional network trained on the TinyImageNet database without data augmentation.

TABLE 6.6: Accuracy (%) of the deep residual compensation convolutional network compared with other methods on the TinyImageNet database, without data augmentation.

Residual networks	
Method	**Accuracy (%)**
ResNet-18, reported by [90]	**43.02**
ResNet-34, reported by [90]	42.65
Non-differentiable networks	
Method	**Accuracy (%)**
PCANet-2	30.00
Multi-Layer PCANet (Chapter 3)	40.87
Stacked-LDA (Chapter 4)	41.76
ResCNet (this chapter)	**44.37**

6.3.3 Image Classification on Three Benchmarks with Data Augmentation

This section aims to enhance the classification performance by increasing the size of the training images using data augmentation. The horizontal flipping is the only type of data augmentation used in this section. We evaluate the deep residual compensation convolutional network using three databases, namely CIFAR-10, CIFAR-100 and TinyImageNet. The only image preprocessing used is the min–max normalisation before each convolutional layer, as described in Algorithm 7.

6.3.3.1 Parameter Settings

Table 6.7 provides an overview of the architectures that were implemented for CIFAR-10, CIFAR-100 and TinyImageNet with data augmentation. The three architectures shared the same design, consisting of several convolutional layers, each followed by a ReLU non-linear activation, second-order pooling, three-level spatial pyramid pooling and an LDA classifier. All architectures used 3×3-pixel filters created by combining 50% PCA and Stacked-LDA filters. For example, by combining 15 PCA filters with 15 Stacked-LDA filters, a total of 30 filters were utilised in each layer of the TinyImageNet database. The number of layers beyond which an accuracy gain was no longer observed was 490 for the CIFAR-10 database, 507 for the CIFAR-100 database and 480 for the TinyImageNet database. After convolving the images and applying ReLU non-linear activation on the feature maps, we utilised 16×16 second-order pooling with a stride of 4 pixels for the CIFAR-10 and CIFAR-100 databases. For the TinyImageNet database, we used 32×32 second-order pooling with a stride of

8 pixels. In all designs, the second-order features were subsequently pooled using three-level spatial pyramid pooling with 4×4, 2×2 and 1×1 subregions. The LDA classifier was used to classify the resulting features, with the sigmoid applied to obtain the probabilities (equation 4.1). During the training phase of the three architectures, the value of λ in equation 6.1 was set to 0.8 for the architectures that were designed for the CIFAR-10 and CIFAR-100 databases, and it was set to 0.5 for the TinyImageNet database. For the CIFAR-10 database, the learning rate (σ) was fixed at 0.35. However, for the other databases, the learning rate's initial value was set to 1 and dropped by 10% every ten layers, as shown in equation 6.19. The learning rate stopped decreasing when it reached 0.478 for the CIFAR-100 database and 0.81 for the TinyImageNet database.

TABLE 6.7: Network architectures for the MNIST, CIFAR-100 and TinyImageNet databases, with data augmentation.

The CIFAR-10 database: $32 \times 32 \times 3$			
Filter size	**SOP**	**SPP (#bins)**	**Output size**
$3 \times 3 \times 3 \times 50$	16×16, stride $= 4$	[16 4 1]	$32 \times 32 \times 50$
$[3 \times 3 \times 50 \times 50] \times 489$	16×16, stride $= 4$	[16 4 1]	$32 \times 32 \times 50$
The CIFAR-100 database: $32 \times 32 \times 3$			
Filter size	**SOP**	**SPP (#bins)**	**Output size**
$3 \times 3 \times 3 \times 50$	16×16, stride $= 4$	[16 4 1]	$32 \times 32 \times 50$
$[3 \times 3 \times 50 \times 50] \times 506$	16×16, stride $= 4$	[16 4 1]	$32 \times 32 \times 50$
The TinyImageNet database: $64 \times 64 \times 3$			
Filter size	**SOP**	**SPP (#bins)**	**Output size**
$3 \times 3 \times 3 \times 30$	32×32, stride $= 8$	[16 4 1]	$64 \times 64 \times 30$
$[3 \times 3 \times 30 \times 30] \times 479$	32×32, stride $= 8$	[16 4 1]	$64 \times 64 \times 30$

6.3.3.2 Performance Analysis

Table 6.8 reports the accuracy of the deep residual compensation convolutional network on the CIFAR-10, CIFAR-100 and TinyImageNet databases, both with and without the use of data augmentation. We have shown in Section 6.3.1 that the number of filters does not necessarily affect the accuracy of the proposed model, but second-order pooling does. Consequently, the reported accuracy for the CIFAR-100 and TinyImageNet databases without data augmentation was the same as those shown in Sections 6.3.2.3 and 6.3.2.4. For the CIFAR-10 database, however, and since we used a different second-order pooling block size in this section, we re-implemented the deep residual compensation convolutional network using the same settings illustrated in Section 6.3.3.1 but with 193 layers.

TABLE 6.8: Accuracy (%) of deep residual compensation convolutional network on the CIFAR-10, CIFAR-100 and TinyImageNet databases, with and without data augmentation.

Data augmentation	CIFAR-10	CIFAR-100	TinyImageNet
✗	86.82	64.9	44.33
✓	**88.35**	**67.8**	**45.37**

According to Table 6.8, using horizontal flipping to double the size of the databases enhanced the performance of the model across all three databases. This improvement was around 1% for the TinyImageNet database, 2% for the CIFAR-10 database and 3% for the CIFAR-100 database. For the CIFAR-10 database, the accuracy achieved with data augmentation was also 1% higher than the best result (87.54%) reported in Section 6.3.1.2. In general, the findings produced in this section indicated the importance of data augmentation for enhancing the model's generalisation ability and gain better accuracy. Incorporating other types of data augmentation is necessary to improve the model's accuracy further.

6.4 Conclusions and Future Work

In this chapter, we introduced the deep residual compensation convolutional network, a non-differentiable model that significantly increases network depth while simultaneously correcting classification errors when traversing it. The network's structure comprises several convolutional layers, each followed by a ReLU non-linear activation, second-order pooling, spatial pyramid pooling and a classifier. The LDA classifier of the first layer is trained using the original classes. In contrast, the LDA classifiers of the deeper layers, known as the compensation layers, are learned with new classes derived from the residual information of the previous layers. The new classes in each residual compensation layer are defined as the classes with the maximum absolute residual error between the predicted probabilities of preceding layers and original classes. The probabilities obtained by any residual compensation layer are then either added to or subtracted from the previous layers' outputs to correct their outputs and maintain the resulting probabilities within the range 0 to 1. The indicator variable, which is defined as the sign of maximum absolute residual error between the previous layers' predicted probabilities and the original classes, is responsible for indicating when to add to or subtract from the probabilities of a specific layer. With positive indicator values, the probabilities of a particular layer are added to those of previous layers. Negative indicator values, on the other hand, indicate subtraction from the probabilities of the earlier levels. In fact, the indicator divides the training samples into negative samples with negative indicator values and positive samples with positive indicator values. Two LDA classifiers are then run using the positive and negative examples with their corresponding classes. The probabilities predicted by LDA trained on positive samples are added to the probabilities of the preceding layer, whereas the probabilities predicted by LDA trained on negative samples are subtracted. As more layers are added, the succeeding layers compensate for previous layers' errors; therefore, by combining the predicted values of the residual compensation layers, the model can reach a high level of accuracy.

The proposed model was evaluated on different benchmarks, namely the CIFAR-10, CIFAR-100, MNIST and TinyImageNet databases. We summarise our findings as follows:

- The findings presented in Section 6.3.1.2 demonstrated that we were capable of increasing the depth of our network to more than 950 layers, and to the best of our knowledge, the model that we have proposed is the first non-differentiable model that is capable of reaching such a large number of layers.
- In contrast to traditional neural networks, which have many user-defined parameters, we used the same number of filters for each layer in all of our experiments, and the model's training was terminated when there was no noticeable improvement in the training error rate. However, someone still has to specify specific parameters, such as the second-order block size and the spatial pyramid pooling bins.
- Comparing our model to several residual networks on the CIFAR-10 database, Section 6.3.2.2 proved that our model used fewer FLOPs.
- Our model with basic preprocessing steps showed excellent results in all of the four databases, namely CIFAR-10, CIFAR-100, TinyImageNet and MNIST (Section 6.3.2). It outperformed the existing non-differentiable networks and was on par with the performance of certain gradient-based models. This was accomplished without the use of complicated functions, dropout or regulation techniques.
- Using a single form of data augmentation (horizontal flipping; Section 6.3.3), the accuracy of our models increased across all datasets, including

CIFRA-10, CIFAR-100 and TinyImageNet. This improvement was considerable in some datasets, such as CIFAR-100 (3%).

The performance of our model is improves as the size of the databases increases by utilising data augmentation techniques. However, increasing the number of samples leads to higher computational costs. To overcome this issue and in future work, we will investigate the possibility of transforming the current network into a batch-based system, similar to the neural network. In addition, the network may be developed further by modifying the filter types or by using other metrics to define the residual errors, and thus the new classes.

Chapter 7

Conclusions

The primary purpose of this research was to investigate deep, non-differentiable networks that are trained in a single pass with no gradient descent or back-propagation. The investigated networks were designed mainly for classification tasks. In all chapters, we used four standard databases to test our models, namely CIFAR-10, CIFAR-100, MNIST and TinyImageNet. Section 7.1 of this chapter summarises the key contributions, and Section 7.2 lists some of the potential future works in the field.

7.1 Summary of Contributions

In Chapter 3, we presented the Multi-Layer PCANet, which was inspired by the design of PCANet. The main contribution of this network is to reduce the number of features while increasing the network depth from two layers in the original PCANet structure to nine layers. Using second-order pooling and CNN-like filters decreased the data's dimensionality, allowing for deeper networks. In addition, the z-score approach had been shown to be an excellent alternative to the binarisation step in PCANet and its variants for maintaining more information. The late fusion also reduced the number of features further. The network was evaluated across four benchmarks, CIFAR-10, CIFAR-100, MNIST and TinyImageNet and proven to be effective, outperforming the original PCANet design in terms of accuracy and number of features generated.

In Chapters 4 and 5, new filters were implemented to replace the PCA filters in the Multi-Layer PCANet. Our experiments showed that different types of filters had different impacts on shallow-depth networks' efficiency. The performance of the networks was affected by the filters used, since each generated different features. Our investigations also showed that combining 50% PCA filters with 50% Stacked-LDA filters, referred to as semi-supervised Stacked-LDA, produced the best performance. When searching for separable classes, the Stacked-LDA filters were generated by repeatedly applying a linear discriminant analysis classifier to a different subset of image-based patches.

Chapter 6 introduced the deep residual compensation convolutional network (ResCNet). The network structure consisted of multiple convolutional layers, each followed by post-processing steps and a classifier. The first layer, known as the baseline, was trained using the original classes. The subsequent layers were residual layers and were trained with new labels learnt from the residual information of all its previous layers. As we progressed by going deeper into ResCNet, the subsequent layers compensated for the classification errors of the preceding layers. Our experiments showed that, unlike neural networks in which the structure needs to be known in advance, we could use a different number of filters and still achieve the same level of performance. When no improvement was seen in the training error rate, we stopped adding more layers to ResCNet. The experiments also showed that ResCNet had been trained with more than 900 layers, making it the first non-differentiable network to reach such a high number of layers. Regarding performance, ResCNet outperformed all existing non-differentiable networks and was on par with some gradient-based models, such as network in network [31], stochastic pooling [8] and many residual networks.

7.2 Future Research Directions

In terms of future development, we emphasise the following suggestions for improving the performance of ResCNet:

- Investigating the possibility of adding non-linearity between convolutional layers or attempting to identify a suitable non-linear filter that would improve the network's performance;

- Transforming the current ResCNet into a neural network–like batch-based system. Instead of training the model using all training examples at once, we aim to train it in batches. However, unlike typical neural networks, the weights connecting the layers will not be iteratively learnable. Instead, we expand to a new layer again using information about the new labels of all batches. The main challenge with transforming the current ResCNet into a batch-based system is that for any batch to be processed, we need to find the convolutional output and the probabilities using the information of all previous layers, which is time-consuming. As so, one of the future suggestions is to improve the inference time of the ResCNet and reduce its dependency on all layers;

- Examining if using alternative error metrics, post-processing steps, or filter types would enhance the existing ResCNet performance;
- Investigating the effect of incorporating some regularisation techniques, such as dropout and drop-connect, on ResCNet accuracy.

References

[1] Mubarakah Alotaibi and Richard C Wilson. “Multi-layer PCA Network for Image Classification”. In: *Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR)*. Springer. 2021, pp. 292–301.

[2] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. “Deep learning.” nature 521.7553 (2015): 436”. In: *DOI: https://doi. org/10.1038/nature14539 https://doi. org/10.1038/nature14539* ().

[3] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. “Imagenet classification with deep convolutional neural networks”. In: *Communications of the ACM* 60.6 (2017), pp. 84–90.

[4] Karen Simonyan and Andrew Zisserman. “Very deep convolutional networks for large-scale image recognition”. In: *arXiv preprint arXiv:1409.1556* (2014).

[5] Kaiming He et al. “Deep residual learning for image recognition”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[6] Benjamin Graham. “Fractional max-pooling”. In: *arXiv preprint arXiv:1412.6071* (2014).

[7] Ian Goodfellow et al. “Maxout networks”. In: *International conference on machine learning*. PMLR. 2013, pp. 1319–1327.

[8] Matthew D Zeiler and Rob Fergus. “Stochastic pooling for regularization of deep convolutional neural networks”. In: *arXiv preprint arXiv:1301.3557* (2013).

[9] Jost Tobias Springenberg et al. “Striving for simplicity: The all convolutional net”. In: *arXiv preprint arXiv:1412.6806* (2014).

[10] Mohammad Rastegari et al. “Xnor-net: Imagenet classification using binary convolutional neural networks”. In: *European conference on computer vision*. Springer. 2016, pp. 525–542.

[11] Gustav Larsson, Michael Maire, and Gregory Shakhnarovich. "Fractalnet: Ultra-deep neural networks without residuals". In: *arXiv preprint arXiv:1605.07648* (2016).

[12] Gao Huang et al. "Densely connected convolutional networks". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 4700–4708.

[13] Zhi-Hua Zhou and Ji Feng. "Deep forest". In: *arXiv preprint arXiv:1702.08835* (2017).

[14] Guang-Bin Huang et al. "Local receptive fields based extreme learning machine". In: *IEEE Computational intelligence magazine* 10.2 (2015), pp. 18–29.

[15] Tsung-Han Chan et al. "PCANet: A simple deep learning baseline for image classification?" In: *IEEE transactions on image processing* 24.12 (2015), pp. 5017–5032.

[16] Joan Bruna and Stéphane Mallat. "Invariant scattering convolution networks". In: *IEEE transactions on pattern analysis and machine intelligence* 35.8 (2013), pp. 1872–1886.

[17] Guang-Bin Huang et al. "Extreme learning machine for regression and multiclass classification". In: *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 42.2 (2011), pp. 513–529.

[18] Erik Cambria et al. "Extreme learning machines [trends & controversies]". In: *IEEE intelligent systems* 28.6 (2013), pp. 30–59.

[19] Deepak Ranjan Nayak et al. "Deep extreme learning machine with leaky rectified linear unit for multiclass classification of pathological brain images". In: *Multimedia Tools and Applications* 79.21 (2020), pp. 15381–15396.

[20] Cong Jie Ng and Andrew Beng Jin Teoh. "DCTNet: A simple learning-free approach for face recognition". In: *2015 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA)*. IEEE. 2015, pp. 761–768.

[21] Meng Xi et al. "Local binary pattern network: A deep learning approach for face recognition". In: *2016 IEEE international conference on Image processing (ICIP)*. IEEE. 2016, pp. 3224–3228.

[22] Yongqing Zhang et al. "ICANet: a simple cascade linear convolution network for face recognition". In: *EURASIP Journal on Image and Video Processing* 2018.1 (2018), p. 51.

[23] Chunxiao Fan et al. "PCANet-II: When PCANet Meets the Second Order Pooling". In: *IEICE TRANSACTIONS on Information and Systems* 101.8 (2018), pp. 2159–2162.

[24] Cheng-Yaw Low, Andrew Beng-Jin Teoh, and Kar-Ann Toh. "Stacking PCANet+: An overly simplified convnets baseline for face recognition". In: *IEEE Signal Processing Letters* 24.11 (2017), pp. 1581–1585.

[25] Xinghao Yang et al. "Canonical correlation analysis networks for two-view image recognition". In: *Information Sciences* 385 (2017), pp. 338–352.

[26] Dongshun Cui et al. "Compact feature representation for image classification using elms". In: *Proceedings of the IEEE International Conference on Computer Vision Workshops*. 2017, pp. 1015–1022.

[27] Bernardo B Gatto and Eulanda M dos Santos. "Discriminative canonical correlation analysis network for image classification". In: *2017 IEEE International Conference on Image Processing (ICIP)*. IEEE. 2017, pp. 4487–4491.

[28] Bernardo Bentes Gatto, Eulanda Miranda dos Santos, and Kazuhiro Fukui. "Subspace-based convolutional network for handwritten character recognition". In: *2017 14th IAPR international conference on document analysis and recognition (ICDAR)*. Vol. 1. IEEE. 2017, pp. 1044–1049.

[29] Mohammad Reza Mohammadnia-Qaraei, Reza Monsefi, and Kamaledin Ghiasi-Shirazi. "Convolutional kernel networks based on a convex combination of cosine kernels". In: *Pattern Recognition Letters* 116 (2018), pp. 127–134.

[30] Bernardo B Gatto et al. "A semi-supervised convolutional neural network based on subspace representation for image classification". In: *EURASIP Journal on Image and Video Processing* 2020.1 (2020), pp. 1–21.

[31] Min Lin, Qiang Chen, and Shuicheng Yan. "Network in network". In: *arXiv preprint arXiv:1312.4400* (2013).

[32] Md Zahangir Alom et al. "The history began from alexnet: A comprehensive survey on deep learning approaches". In: *arXiv preprint arXiv:1803.01164* (2018).

[33] Pierre Foret et al. "Sharpness-aware minimization for efficiently improving generalization". In: *arXiv preprint arXiv:2010.01412* (2020).

[34] Sangwook Kim, Swathi Kavuri, and Minho Lee. “Deep network with support vector machines”. In: *International Conference on Neural Information Processing*. Springer. 2013, pp. 458–465.

[35] Dan Wu et al. “Kernel principal component analysis network for image classification”. In: *arXiv preprint arXiv:1512.06337* (2015).

[36] Yann LeCun et al. “Gradient-based learning applied to document recognition”. In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[37] Kevin P Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.

[38] Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. “A fast learning algorithm for deep belief nets”. In: *Neural computation* 18.7 (2006), pp. 1527–1554.

[39] Ruslan Salakhutdinov and Geoffrey Hinton. “Deep boltzmann machines”. In: *Artificial intelligence and statistics*. 2009, pp. 448–455.

[40] Nurshazlyn Mohd Aszemi and PDD Dominic. “Hyperparameter Optimization in Convolutional Neural Network using Genetic Algorithms”. In: ().

[41] Christian Szegedy et al. “Going deeper with convolutions”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 1–9.

[42] Kaiming He et al. “Delving deep into rectifiers: Surpassing human-level performance on imagenet classification”. In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 1026–1034.

[43] Sergey Ioffe and Christian Szegedy. “Batch normalization: Accelerating deep network training by reducing internal covariate shift”. In: *International conference on machine learning*. PMLR. 2015, pp. 448–456.

[44] Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. “Highway networks”. In: *arXiv preprint arXiv:1505.00387* (2015).

[45] Gao Huang et al. “Deep networks with stochastic depth”. In: *European conference on computer vision*. Springer. 2016, pp. 646–661.

[46] Sergey Zagoruyko and Nikos Komodakis. “Wide residual networks”. In: *arXiv preprint arXiv:1605.07146* (2016).

[47] Kaiming He et al. “Identity mappings in deep residual networks”. In: *European conference on computer vision*. Springer. 2016, pp. 630–645.

[48] Sasha Targ, Diogo Almeida, and Kevin Lyman. "Resnet in resnet: Generalizing residual architectures". In: *arXiv preprint arXiv:1603.08029* (2016).

[49] Ke Zhang et al. "Residual networks of residual networks: Multilevel residual networks". In: *IEEE Transactions on Circuits and Systems for Video Technology* 28.6 (2017), pp. 1303–1314.

[50] Chen-Yu Lee, Patrick W Gallagher, and Zhuowen Tu. "Generalizing pooling functions in convolutional neural networks: Mixed, gated, and tree". In: *Artificial intelligence and statistics*. PMLR. 2016, pp. 464–472.

[51] Ming Liang and Xiaolin Hu. "Recurrent convolutional neural network for object recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 3367–3375.

[52] Zhibin Liao and Gustavo Carneiro. "Competitive multi-scale convolution". In: *arXiv preprint arXiv:1511.05635* (2015).

[53] Adriana Romero et al. "Fitnets: Hints for thin deep nets". In: *arXiv preprint arXiv:1412.6550* (2014).

[54] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. "Fast and accurate deep network learning by exponential linear units (elus)". In: *arXiv preprint arXiv:1511.07289* (2015).

[55] Jasper Snoek et al. "Scalable bayesian optimization using deep neural networks". In: *International conference on machine learning*. PMLR. 2015, pp. 2171–2180.

[56] Dmytro Mishkin and Jiri Matas. "All you need is a good init". In: *arXiv preprint arXiv:1511.06422* (2015).

[57] Kaiming He et al. "Deep residual learning for image recognition. CVPR. 2016". In: *arXiv preprint arXiv:1512.03385* (2016).

[58] Kaicheng Yu and Mathieu Salzmann. "Statistically-motivated second-order pooling". In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 600–616.

[59] Qilong Wang et al. "Deep cnns meet global covariance pooling: Better representation and generalization". In: *arXiv preprint arXiv:1904.06836* (2019).

[60] Peihua Li et al. "Towards faster training of global covariance pooling networks by iterative matrix square root normalization". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 947–955.

[61] Yunbo Wang et al. “Spatiotemporal pyramid network for video action recognition”. In: *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*. 2017, pp. 1529–1538.

[62] Hao Wang et al. “Multi-scale location-aware kernel representation for object detection”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 1248–1257.

[63] Fuzhen Zhuang et al. “A comprehensive survey on transfer learning”. In: *Proceedings of the IEEE* 109.1 (2020), pp. 43–76.

[64] Geoffrey Hinton. “The forward-forward algorithm: Some preliminary investigations”. In: *arXiv preprint arXiv:2212.13345* (2022).

[65] Kaiming He et al. “Spatial pyramid pooling in deep convolutional networks for visual recognition”. In: *IEEE transactions on pattern analysis and machine intelligence* 37.9 (2015), pp. 1904–1916.

[66] Atmane Khellal, Hongbin Ma, and Qing Fei. “Convolutional neural network based on extreme learning machine for maritime ships recognition in infrared images”. In: *Sensors* 18.5 (2018), p. 1490.

[67] Kaicheng Yu and Mathieu Salzmann. “Statistically-motivated second-order pooling”. In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 600–616.

[68] Joao Carreira et al. “Semantic segmentation with second-order pooling”. In: *European Conference on Computer Vision*. Springer. 2012, pp. 430–443.

[69] Qilong Wang et al. “RAID-G: Robust estimation of approximate infinite dimensional Gaussian with application to material recognition”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, pp. 4433–4441.

[70] Yu Liu et al. “Fusion that matters: convolutional fusion networks for visual recognition”. In: *Multimedia Tools and Applications* 77.22 (2018), pp. 29407–29434.

[71] Mike Ebersbach, Robert Herms, and Maximilian Eibl. “Fusion Methods for ICD10 Code Classification of Death Certificates in Multilingual Corpora.” In: *CLEF (Working Notes)*. 2017, p. 36.

[72] Alex Krizhevsky, Geoffrey Hinton, et al. “Learning multiple layers of features from tiny images”. In: (2009).

[73] Li Deng. “The mnist database of handwritten digit images for machine learning research [best of the web]”. In: *IEEE Signal Processing Magazine* 29.6 (2012), pp. 141–142.

[74] *Tiny ImageNet*. https://tiny-imagenet.herokuapp.com/.

[75] Gao Huang et al. “Densely connected convolutional networks”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 4700–4708.

[76] Zoheb Abai and Nishad Rajmalwar. “DenseNet models for tiny imagenet classification”. In: *arXiv preprint arXiv:1904.10429* (2019).

[77] Christopher M Bishop. *Pattern recognition and machine learning*. Vol. 4. 4. Springer.

[78] Alina Beygelzimer, John Langford, and Bianca Zadrozny. “Weighted one-against-all”. In: *AAAI*. 2005, pp. 720–725.

[79] Carlos N Silla and Alex A Freitas. “A survey of hierarchical classification across different application domains”. In: *Data Mining and Knowledge Discovery* 22.1-2 (2011), pp. 31–72.

[80] Susan Dumais and Hao Chen. “Hierarchical classification of web content”. In: *Proceedings of the 23rd annual international ACM SIGIR conference on Research and development in information retrieval*. 2000, pp. 256–263.

[81] Cecille Freeman. “Feature selection and hierarchical classifier design with applications to human motion recognition”. In: (2014).

[82] Daniel Silva-Palacios, Cesar Ferri, and Maria Jose Ramirez-Quintana. “Improving performance of multiclass classification by inducing class hierarchies”. In: *Procedia Computer Science* 108 (2017), pp. 1692–1701.

[83] Gjorgji Madzarov and Dejan Gjorgjevikj. “Multi-class classification using support vector machines in decision tree architecture”. In: *IEEE EUROCON 2009*. IEEE. 2009, pp. 288–295.

[84] Sungmoon Cheong, Sang Hoon Oh, and Soo-Young Lee. “Support vector machines with binary tree architecture for multi-class classification”. In: *Neural Information Processing-Letters and Reviews* 2.3 (2004), pp. 47–51.

[85] Daniel Silva-Palacios, Cesar Ferri, and Maria Jose Ramirez-Quintana. “Improving performance of multiclass classification by inducing class hierarchies”. In: *Procedia Computer Science* 108 (2017), pp. 1692–1701.

[86] Thomas G Dietterich and Ghulum Bakiri. “Solving multiclass learning problems via error-correcting output codes”. In: *Journal of artificial intelligence research* 2 (1994), pp. 263–286.

[87] Wiesław Chmielnicki and Katarzyna Stąpor. “Using the one–versus–rest strategy with samples balancing to improve pairwise coupling classification”. In: *International Journal of Applied Mathematics and Computer Science* 26.1 (2016), pp. 191–201.

[88] Bitna Kim and Young Ho Park. “Beginner's guide to neural networks for the MNIST dataset using MATLAB”. In: *The Korean Journal of Mathematics* 26.2 (2018), pp. 337–348.

[89] Rong-En Fan et al. “LIBLINEAR: A Library for Large Linear Classification”. In: *Journal of Machine Learning Research* 9 (2008), pp. 1871–1874.

[90] Pranav Jeevan et al. “Convolutional Xformers for Vision”. In: *arXiv preprint arXiv:2201.10271* (2022).

[91] Ulrike Von Luxburg. “A tutorial on spectral clustering”. In: *Statistics and computing* 17.4 (2007), pp. 395–416.

[92] Bogdan Raducanu and Fadi Dornaika. “A supervised non-linear dimensionality reduction approach for manifold learning”. In: *Pattern Recognition* 45.6 (2012), pp. 2432–2444.

[93] Wen-bing Huang, Fu-chun Sun, et al. “Building feature space of extreme learning machine with sparse denoising stacked-autoencoder”. In: *Neurocomputing* 174 (2016), pp. 60–71.

[94] Iago Richard Rodrigues et al. “Convolutional Extreme Learning Machines: A Systematic Review”. In: *Informatics*. Vol. 8. 2. MDPI. 2021, p. 33.

[95] Andrew Ng, Michael Jordan, and Yair Weiss. “On spectral clustering: Analysis and an algorithm”. In: *Advances in neural information processing systems* 14 (2001), pp. 849–856.

[96] Jianbo Shi and Jitendra Malik. “Normalized cuts and image segmentation”. In: *IEEE Transactions on pattern analysis and machine intelligence* 22.8 (2000), pp. 888–905.

[97] Arthur Gretton et al. “Measuring statistical dependence with Hilbert-Schmidt norms”. In: *International conference on algorithmic learning theory*. Springer. 2005, pp. 63–77.

[98] Mohammad Ali Alomrani. “A Critical Review of Information Bottleneck Theory and its Applications to Deep Learning”. In: *arXiv preprint arXiv:2105.04405* (2021).

[99] Wan-Duo Kurt Ma, JP Lewis, and W Bastiaan Kleijn. “The HSIC bottleneck: Deep learning without back-propagation”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 34. 04. 2020, pp. 5085–5092.

[100] Yerlan Idelbayev. *Proper ResNet Implementation for CIFAR10/CIFAR100 in PyTorch*. https://github.com/akamaster/pytorch_resnet_cifar10. Accessed: 20xx-xx-xx.

Appendix A

Solution to HSIC Optimisation Problem

As described in Chapter 5, our goal is to find a straightforward solution to the optimisation problem described by equation 5.22 using linear kernels. Since HK_XH and HK_YH are fixed symmetric metrics, the optimisation problem (equation 5.22) can be rewritten as follows:

$$\min_{K_z} \mathrm{tr}(K_z^{0.5} A K_z^{0.5}) - \mathrm{tr}(K_z^{0.5} B K_z^{0.5}), \tag{A.1}$$

where A represents HK_XH and HK_YH is denoted by B. The optimisation problem in equation A.1 can then be reformulated as follows:

$$\min_{K_z} \frac{\mathrm{tr}(K_z^{0.5} A K_z^{0.5})}{\mathrm{tr}(K_z^{0.5} B K_z^{0.5})}, \tag{A.2}$$

which is equivalent to:

$$\begin{aligned} &\min_{K_z} \mathrm{tr}(K_z^{0.5} A K_z^{0.5}) \\ &\textbf{s.t. } \mathrm{tr}(K_z^{0.5} B K_z^{0.5} = I), \end{aligned} \tag{A.3}$$

where $K_z = K(T(Z_{i-1}, \theta))$. Assuming we have linear transformation and we use linear kernels, as follows:

$$K_{zn} = Z_n Z_n^T, Z_n = Z\theta \tag{A.4}$$

By substituting A.4 into A.3, we obtain the following formula:

$$\begin{aligned} &\min_{\theta} \mathrm{tr}(((Z\theta)(Z\theta)^T)^{0.5} A((Z\theta)(Z\theta)^T)^{0.5}) \\ &\textbf{s.t. } \mathrm{tr}(((Z\theta)(Z\theta)^T)^{0.5} B((Z\theta)(Z\theta)^T)^{0.5} = I) \end{aligned} \tag{A.5}$$

Our loss function then can be rewritten as follows:

$$Loss = \min_{\theta} (\mathrm{tr}((Z\theta)(Z\theta)^T)A) - \lambda\, \mathrm{tr}((((Z\theta)(Z\theta)^T)B - I)) \tag{A.6}$$

We find the derivative of the loss function with respect to θ as follows:

$$\frac{\partial Loss}{\partial \theta} = \mathrm{tr}(2ZZ^T\theta A) - \mathrm{tr}(\lambda 2ZZ^T\theta B) = 0 \tag{A.7}$$

From the trace property:

$$Z^T AZ\theta - \lambda Z^T BZ\theta = 0 \ \Rightarrow Z^T AZ\theta = \lambda Z^T BZ\theta \tag{A.8}$$

Therefore, to find θ, we solve the following generalised eigenvalue/vector decomposition problem:

$$eig(Z^T AZ, Z^T BZ), \tag{A.9}$$

where Z is the previous layer's output.