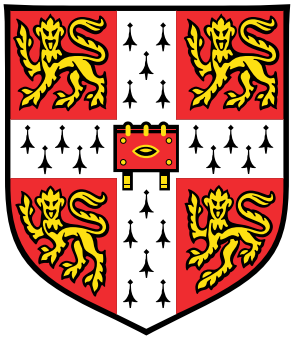
ScatterNet Hybrid Frameworks for Deep Learning



Amarjot Singh

Department of Engineering

University of Cambridge

This is submitted for the

Doctor of Philosophy

Jesus College September 2018

I would like to dedicate this thesis to my loving parents, my brother and sister-in-law, my niece (Banni) and my girlfriend, Samina!

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. This dissertation is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and Acknowledgements. This dissertation contains fewer than 65,000 words including appendices, bibliography, footnotes, tables and equations and has fewer than 150 figures.

Amarjot Singh
September 2018

Acknowledgements

My time at the University of Cambridge has been a unique experience, and I had the opportunity to meet and work with many outstanding people who supported and inspired me throughout my studies, and to whom I would like to express my deepest gratitude. First and foremost I would like to thank my supervisor Prof. Nick Kingsbury, who has significantly altered my views on computer vision, deep learning, and research in general. I also sincerely thank Cambridge Trust for providing me the scholarship for my Ph.D., without which It would have been impossible for me to pursue this journey. I joined Prof. Kingsbury's group with the impression that there is no or very little deep learning used in the lab and most of my working would be on designing handcrafted feature detectors. However, it turns out that there is a lot of deep learning research in our lab, although of a slightly different kind. It happens during the countless hours we spend looking at data and trying to get an intuition about what is essential. It is our intuition that we should develop first – that is the unwritten rule in our lab. Nick has a unique teaching style: he doesn't tell us the answers; he steers us towards them so that we can build our intuition in the process. He doesn't directly ask us to do work. Instead, he uses his unbounded enthusiasm to motivate us, and it is quite infectious! Nick has led me to conclude that the critical component in designing deep networks is not to learn end-to-end in a supervised fashion but to create hybrid systems that make use of mainly unlabelled data to learn invariant and discriminative representation rapidly. This philosophy has led me to develop the ScatterNet Hybrid Deep Learning Networks (SHDL) which I have presented throughout this thesis. I am grateful for the countless hours he spent working with me on the SHDL ideas and teaching me so much in the process.

I am also thankful to my advisor Dr. Joan Laseby who always motivated and supported me throughout my PhD. She continuously provided many valuable inputs on my work which have helped me significantly improve my research. I am also very grateful for her help with several of my Postdoctoral fellowship applications.

Throughout my PhD, I was privileged to have worked with amazing engineers, scientists, and friends. To begin with, my colleague, Fergal cotter, with whom I shared

most of my experiences in the signal processing lab. I greatly enjoyed working with him, and I am thankful for his friendship and support on several projects. I further would like to thank my colleagues Oliver Bonner, Kuan Hsieh, Josias Van Der Westhuizen, Dr. Pavan Koteshwar Srinath and Marina Riabiz for making my time in the signal processing lab so enjoyable. I would also like to thank my peers Fabio Giardina, Priyanka Joshi, Utkarsh Sharma, and Abhimanyu Sharma, for being both personally and intellectually supportive to me.

This dissertation ties together a large body of work completed during my Ph.D., and below I detail the papers emanated from this work.

Chapter 4 presents two hand-crafted architectures presented at the:

- *International Conference on Mathematics in Signal Processing (IMA), 2016*, in Birmingham, UK, and the
- *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2017*, in New Orleans, USA.

Chapter 5 presents the work which uses the hand-crafted front-end proposed in chapter 4 to accelerate the learning of deep supervised networks. This work presented at the Compact and Efficient Feature Representation and Learning in Computer Vision (CEFRL) workshop at *IEEE International Conference on Computer Vision (ICCV), 2017*, in Venice, Italy.

Chapter 6 presents two SHDL architectures for object recognition and semantic image segmentation presented at the:

- *IEEE International Workshop on Machine Learning for Signal Processing (MLSP), 2017*, in Tokyo, Japan and the
- *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2018*, in Calgary, Canada.

Abstract

Image understanding is the task of interpreting images by effectively solving the individual tasks of object recognition and semantic image segmentation. An image understanding system must have the capacity to distinguish between similar looking image regions while being invariant in its response to regions that have been altered by the appearance-altering transformation. The fundamental challenge for any such system lies within this simultaneous requirement for both invariance and specificity.

Many image understanding systems have been proposed that capture geometric properties such as shapes, textures, motion and 3D perspective projections using filtering, non-linear modulus, and pooling operations. Deep learning networks ignore these geometric considerations and compute descriptors having suitable invariance and stability to geometric transformations using (end-to-end) learned multi-layered network filters. These deep learning networks in recent years have come to dominate the previously separate fields of research in machine learning, computer vision, natural language understanding and speech recognition.

Despite the success of these deep networks, there remains a fundamental lack of understanding in the design and optimization of these networks which makes it difficult to develop them. Also, training of these networks requires large labeled datasets which in numerous applications may not be available.

In this dissertation, we propose the ScatterNet Hybrid Framework for Deep Learning that is inspired by the circuitry of the visual cortex. The framework uses a hand-crafted front-end, an unsupervised learning based middle-section, and a supervised back-end to rapidly learn hierarchical features from unlabelled data. Each layer in the proposed framework is automatically optimized to produce the desired computationally efficient architecture. The term 'Hybrid' is coined because the framework uses both unsupervised as well as supervised learning.

We propose two hand-crafted front-ends that can extract locally invariant features from the input signals. Next, two ScatterNet Hybrid Deep Learning (SHDL) networks (a generative and a deterministic) were introduced by combining the proposed front-ends with two unsupervised learning modules which learn hierarchical features. These

hierarchical features were finally used by a supervised learning module to solve the task of either object recognition or semantic image segmentation. The proposed front-ends have also been shown to improve the performance and learning of current Deep Supervised Learning Networks (VGG, NIN, ResNet) with reduced computing overhead.

Table of contents

List of figures **xv**

List of tables **xix**

1 Introduction **1**
1.1 Introduction . . . 1
1.2 Image Understanding Systems . . . 3
1.3 Contributions . . . 7
1.4 Thesis overview . . . 8

2 Literature Review **11**
2.1 Handcrafted Architectures . . . 11
2.1.1 Scale-Invariant Feature Transform (SIFT) . . . 12
2.1.2 Scatter Net . . . 14
2.2 End-to-end Learned Networks . . . 20
2.2.1 Convolutional Neural Network . . . 21
2.2.2 AlexNet . . . 26
2.2.3 Network in Network (NIN) . . . 27
2.2.4 Visual Geometry Group (VGG) . . . 28
2.2.5 Residual Networks . . . 29
2.2.6 Restricted Boltzmann Machine . . . 30
2.3 Hybrid Architectures . . . 33
2.3.1 Bag of Features . . . 33
2.3.2 Deep Sparse Coding . . . 35
2.3.3 HMAX Model . . . 36

3 ScatterNet Hybrid Framework for Deep Learning **41**
3.1 ScatterNet Hybrid Deep Learning Framework . . . 44
3.1.1 ScatterNet Front-end . . . 45

3.1.2 Unsupervised Learning Mid-section Module 45
3.1.3 Supervised Learning Back-end 46

4 Hand-crafted Front-end 47
4.1 Multi-Resolution Region Pooling ScatterNet 49
4.2 Overview of Results 52
4.2.1 US Postal Service Dataset 53
4.2.2 The UCI Isolet Dataset 54
4.2.3 The UCI Yeast Dataset 55
4.2.4 The UCI Glass Dataset 55
4.3 Computational Complexity 56
4.4 Discussions 56
4.5 Multi-resolution Parametric Log ScatterNet 57
4.6 Overview of Results 61
4.7 Computational Complexity 63
4.8 Discussions 64
4.9 Comparison between the Proposed ScatterNets 64

5 Efficient Learning using ScatterNets 67
5.1 DTCWT ScatterNet Convolutional Neural Network (DTSCNN) 68
5.2 Experimental Results 70
5.2.1 Datasets 70
5.2.2 Evaluation and Comparison on Classification Error 71
5.2.3 Analysis on Computational Efficiency and Learning 73
5.2.4 Comparison with Pre-trained CNN First Layers 75
5.2.5 Comparison with the state-of-the-art 75
5.3 Discussions 76

6 ScatterNet Hybrid Deep Learning (SHDL) Networks 79
6.1 Deterministic ScatterNet Hybrid Deep Learning (D-SHDL) network . . 82
6.1.1 ScatterNet Hand-crafted Descriptors 82
6.1.2 Unsupervised Learning Module: PCA-Net Layers 83
6.1.3 Supervised Learning Module: OLS and G-SVM 85
6.2 Overview of the D-SHDL Results 87
6.2.1 ScatterNet feature extraction 87
6.2.2 PCA Layers: features and layer optimization 87

6.2.3 Classification performance . . . 89
6.2.4 Comparison with the state-of-the-art . . . 90
6.2.5 Advantage over supervised learning . . . 91
6.2.6 Computational Complexity . . . 91
6.3 Discussions . . . 93
6.4 Generative ScatterNet Hybrid Deep Learning (G-SHDL) network . . . 94
6.4.1 ScatterNet Hand-crafted Descriptors . . . 94
6.4.2 Unsupervised Learning: RBM with Priors . . . 95
6.4.3 Supervised CRF Segmentation . . . 97
6.5 Overview of the G-SHDL Results . . . 98
6.5.1 Handcrafted Front-end: ScatterNet . . . 98
6.5.2 Unsupervised Mid-section: RBM with PCA priors . . . 99
6.5.3 Classification performance . . . 99
6.5.4 Comparison with the state-of-the-art . . . 100
6.5.5 Advantage over Deep Supervised Networks . . . 101
6.5.6 Computational Complexity . . . 102
6.5.7 Discussions . . . 104

7 Conclusions 105
7.1 Summary of Key Results . . . 105
7.2 Future Work . . . 109
7.2.1 SHDL Back-end . . . 109
7.2.2 Artificial General Intelligence: Sequential Learning . . . 110
7.3 Parting Note . . . 111

Appendix A Training of Convolutional Neural Networks 113
A.1 Back-propagation Algorithm . . . 113

Appendix B Feature Selection 115
B.1 Orthogonal Least Squares(OLS) . . . 115

Appendix C PCA Network (PCANet) 117

Appendix D Software 119
D.1 ScatterNet . . . 119
D.2 Deep Convolutional Networks . . . 119
D.3 PCA Network . . . 120
D.4 Convolutional RBM . . . 120

Appendix E Datasets **121**

Appendix F Learned Filters **125**

Appendix G Publication List **127**

References **129**

List of figures

1.1 Illustration of the ideal case in which the distance measured by a trained classifier between two feature vectors extracted from both same images is zero . . . 2
1.2 Illustration presents the classifier which learns local and global regularities from the training set. . . . 3
1.3 Illustration of real-world scenario in which the distance measured by a trained classifier between feature vectors extracted from the image (x_1) and it's corrupted version (x_f), is significant. . . . 5

2.1 SIFT Difference-of-Gaussians pyramid . . . 13
2.2 SIFT descriptor . . . 14
2.3 Real and Imaginary parts of the complex Morlet wavelets at multiple scales and orientations . . . 15
2.4 Scattering operations performed on a signal to extract filtered response with translation invariance and to recover the lost frequency components. 16
2.5 Invariant scatter convolutional network. . . . 17
2.6 Sparse and Deep autoencoder trained to car dataset. The filter weights for each layer are visualized to present low, mid and high-level features learned from the car class. . . . 19
2.7 Comparison of earlier and convolutional architecture . . . 20
2.8 Convolutional Neural Network Architecture . . . 22
2.9 Filters applied on an Image to extract low level image features (edges) 23
2.10 Pooling performed within and across feature maps to achieve translation and rotation invariance. . . . 24
2.11 Convolutional Neural Network trained on car class of the ImageNet dataset. The trained network is further applied on an image. . . . 25
2.12 The architecture of the AlexNet . . . 26

2.13 The illustration presents the Low-Dimensional Embedding (LDE) of the Network in Network (NIN) architecture 27
2.14 The illustration presents the architecture of the Visual Geometry Group (VGG) 28
2.15 The illustration presents the benifit of residual bloacks proposed in the residual network over other deeper architectures. 29
2.16 Restricted Boltzmann Machine trained using contractive divergence on MNIST dataset. 31
2.17 Bag of features model 34
2.18 The illustration shows the three-layer deep sparse coding framework. . 35
2.19 Complex cells pooling responses from Simple cells at different locations and scales (same orientation) to achieve translation and scale invariance. 37
2.20 HMAX Model 38

3.1 Illustration of an image with two class objects mapped onto their respective manifolds. 42
3.2 ScatterNet Hybrid Deep Learning (SHDL) framework inspired by the circuitry of the visual pathway is presented. The front-end of the SHDL network is a hand-crafted ScatterNet similar to the V1 of the pathway, which decomposes the input signal into features at different scales and orientations using complex wavelets [119, 155]. The mid-section then uses unsupervised learning on these hand-crafted features to rapidly encode invariant hierarchal feature similar to what is believed to be the functions of V2 and V4 regions [149–152]. Finally, the back-end of the framework uses supervised learning to assign class specific labels to the features obtained from the last layer of the unsupervised module using few labeled examples [120, 148]. 44

4.1 Multi-Resolution Region Pooling ScatterNet 51
4.2 Multi-resolution Parametric Log Scattering Network 58
4.3 Examples of features representations before and applying the parametric Log non-linearity 59

5.1 DTCWT ScatterNet Convolutional Neural Network (DTSCNN) 68
5.2 Graphs show the faster convergence and rate of learning of the DTSCNN derived architectures (AS-1 to AS-4) compared to the CNN (A-1 to A-4) architectures for a range of small and large training data sizes 72

5.3 Computational time of the DTSCNN derived architectures (AS-1 to AS-4) compared to the CNN (A-1 to A-4) architectures for a range of small and large training data sizes 73
5.4 Graphs show the faster convergence and rate of learning of the DTSCNN standard deep architectures (AS-5 to AS-7) compared to the CNN (A-5 to A-7) architectures for a small (5000) and large (50000) training dataset 74

6.1 Deterministic ScatterNet Hybrid Deep Learning (D-SHDL) network . . 81
6.2 Illustration shows the DTCWT real filters at two scales used at Layer L1 and L2 along with the filters learned by the PCA-Net at L3 and L4 stage. 82
6.3 Illustration presents the optimization for the number of filters (K_{L3}) learned at L3 layer as well as the log non-linearity parameter k_{L3} applied on the L3 layer feature representations using 5-CV classification 84
6.4 Illustration presents the optimization for the number of filters (K_{L4}) learned at L4 layer as well as the log non-linearity parameter k_{L4} applied on the L4 layer feature representations using 5-CV classification 85
6.5 Illustration presents the effect of the parametric log transformation applied to the features extracted at layers, L3 and L4 of the D-SHDL network. 86
6.6 The illustration presents the computational time required to learn the filters at Layers, L3 and L4 of the D-SHDL network 92
6.7 Generative ScatterNet Hybrid Deep Learning (G-SHDL) network . . . 95
6.8 The illustration presents the improvement in rate of learning for the RBM with the use of PCA priors 96
6.9 Figure shows two images from MSRC dataset with their ground truth and segmentation obtained at L2 to L6 of G-SHDL 97
6.10 The illustration shows the L6 RBM features thresholded to the top 10, 20 and 30 activations and back-projected to the input image selected from the Caltech-101 dataset. 98
6.11 The illustration presents the computational time required to learn the PCA priors and RBM filters at different Layers of the G-SHDL network 103

7.1 The illustration presents the memory augmented ScatterNet Hybrid Deep Learning Network for sequential learning. 111

E.1 The illustration presents a visualization of a randomly selected training sample from each of the datasets. 122

E.2 Examples of images with class labels from the CIFAR and Caltech datasets. 123
E.3 Example Images, Semantic Labels, and Regions from the a) Stanford Background b) MSRC dataset. 123

F.1 The illustration presents the PCA filters learned at different layers of the deterministic SHDL network presented in section 6.1. 125
F.2 The illustration presents the RBM filters at different Layers of the G-SHDL network presented in section 6.4. 126

List of tables

4.1 Classification error (%) on different datasets for each part of the proposed network 53

4.2 Classification error (%) comparison on USPS Dataset 54

4.3 Classification error (%) comparison on Isolet Dataset 54

4.4 Classification error (%) comparison on Yeast Dataset 55

4.5 Classification error (%) comparison on Glass Dataset 56

4.6 Accuracy (%) on CIFAR-10 for both R1 and R2 for each scale (J) and coefficients at $m = 1$, with and without applying log transformation . . 62

4.7 Accuracy (%) and comparison on both datasets. Pro.: Proposed, Sup: Supervised and Unsup: Unsupervised, learning. 62

4.8 Arc.: Architectures, Pro.: Proposed, R1, R2: Resolution - 1,2 pipeline, FVL: Feature vector length, SD: Selected dimensions using OLS, FR: Feature richness (%) 63

4.9 Comparison of Proposed (Pro.) network on accuracy (%) with two supervised learning methods 64

4.10 Accuracy (%) and comparison on both datasets for the the Multi-Resolution Region Pooling ScatterNet and Multi-resolution Parametric Log ScatterNet. 65

5.1 Experiments are performed with CNN architectures (derived from LeNet designed for CIFAR-10 dataset that contain convolutional (CV) layers (L1 to L5) 69

5.2 Parameter values used by the architectures 69

5.3 Classification error (%) on the CIFAR-10 dataset for the original CNN architectures and their corresponding DTSCNN architectures. 71

5.4 Table shows the comparison on classification error (%) between the DTCWT ScatterNet (DTS) front-end and the first convolutional layer pre-trained on ImageNet for NIN [35] and VGG [36] architectures for Caltech-101 and CIFAR-10 datasets.T-NIN: Transfer-NIN, T-VGG: Transfer-VGG 75
5.5 Table shows the comparison on classification error (%) between the DTCWT ScatterNet ResNet (DTS-WResNet) Architecture with the state of the art architectures on the CIFAR-10 dataset 76
5.6 Table shows the comparison on classification error (%) between the DTCWT ScatterNet VGG (DTS-VGG) Architecture with the state of the art architectures on the Caltech-101 dataset 76

6.1 Accuracy (%) on CIFAR-10 for features extracted at different layers and resolutions. $S_{rs}[Layer]$, HC = S[L0, L1, L2] 88
6.2 5-CV Accuracy (%) on CIFAR-10 at L3 and L4. y_{L3}, y_{L4} output, $\hat{y}_{L3}$, $\hat{y}_{L4}$ optimal output and $\hat{y}_{L3,rs}$, $y_{L4,rs}$ relatively symmetric output, at L3 and L4. 88
6.3 Object classification accuracy (%) on CIFAR-10 and Caltech-101 for each module computed with OLS and G-SVM. The increase in accuracy with the addition of each layer is also shown. HC: Hand-crafted, PCA features $((Layer)_{filter-size})$: eg $(L3)_{s_{L3}=5}$ 89
6.4 Object classification accuracy (%) and comparison with other approaches on both datasets. Unsup: Unsupervised, Semi: Semi-supervised and Sup: Supervised. 90
6.5 Comparison of SHDL network on accuracy (%) with two supervised learning methods (VGG [36] and NIN [35] against different training dataset sizes on CIFAR-10. 91
6.6 5 fold cross validation performed on the training dataset of Stanford background (SB) ansd MSRC dataset to select the optimal numbers of filters for L3 to L6 RBM layers. L(size) = No. of Filters (a, a represents $a \times a$) 99
6.7 PA (%) on both dataset for each module computed with CRF. The increase in accuracy with the addition of each layer is also shown. HC: Hand-crafted. RBM Layers: L3, L4, L5 and L6. 100
6.8 PA (%) and comparison on both datasets. Unsup: Unsupervised, Semi: Semi-supervised and Sup: Supervised. 100

6.9 Comparison of G-SHDL on PA (%) with DeepLab-CRF [192] against different training dataset sizes on SB dataset. 102

Chapter 1

Introduction

1.1 Introduction

Imaging systems have become ubiquitous in our daily lives. These systems have been widely used for numerous applications including Self-driving cars [189], Intelligent Surveillance [190], Content-Based Image Retrieval (CBIR) [188], Human-Computer Interaction [191], etc. In order for them to function effectively, intelligent techniques are required to understand images.

Researchers have attempted to tackle the image understanding task by solving either object recognition or semantic image segmentation. Object recognition aims at classifying and localizing different objects present in the image. In the case of a single object within the image, only the object label is assigned to the image without localization. This is termed as object classification. Semantic image segmentation aims to assign a semantic label to every single pixel contained in the image, but not necessarily count the number of objects. In this thesis, we aim to solve both object classification and semantic image segmentation tasks.

To best solve the image understanding task, one may construct a metric such as Φ, that is applied to an image region x and the corresponding ground truth region category x', as shown:

$$D(x, x') = ||\Phi(x) - \Phi(x')|| \tag{1.1}$$

where $D(x, x')$ is the distance between the representations. Smaller distance between the region representation x and the ground truth region category x' represents higher similarity.

We discuss this in detail for the task of object recognition. In the ideal case, where both objects (x, x') appear the same, it is relatively easy to develop the representation

Φ. For this simple case, the Euclidean distance measured by the trained classifier between identical images is an ideal zero, as shown in Fig. 1.1.

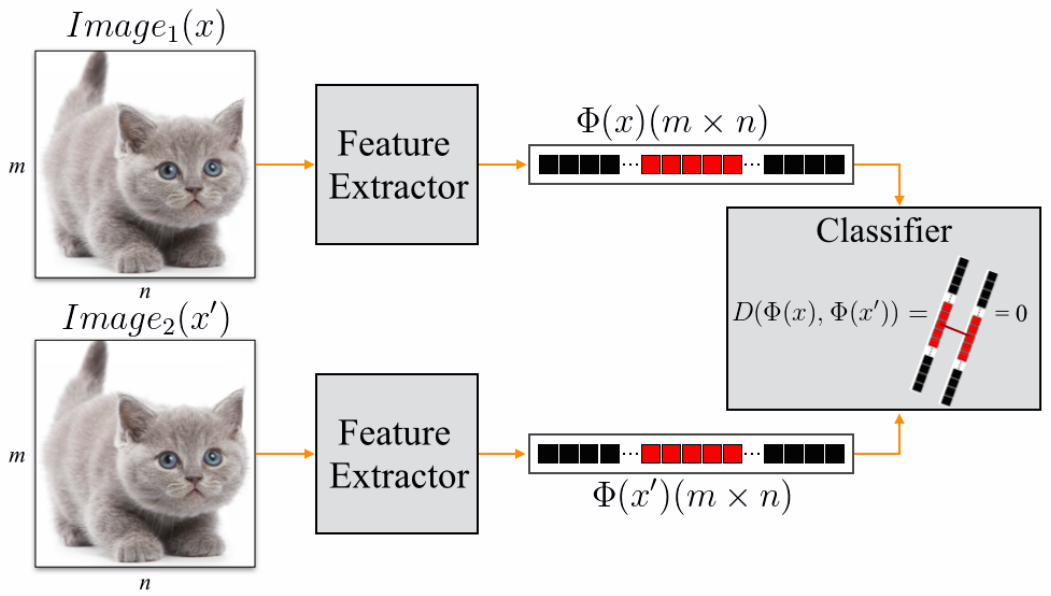


Fig. 1.1 Illustration of the ideal case in which the distance measured by a trained classifier between feature vectors extracted from both same images is zero. Black blocks in the feature vector correspond to features from the constant background while red blocks are features derived from the cat.

However, objects belonging to the same class can appear very different in complex real-world images [40]. The variations introduced due to the position, orientation, and scale of an object may change which can alter its appearance. The appearance can also vary due to the lighting conditions or if the object undergoes any physical deformations. Besides this, there can be different types of objects that represent a particular object class which further can make it difficult to recognize an object [40]. One such example is shown in Fig. 1.2, in which the airplanes have different decks and tails. This further can limit the ability of the detector to recognize the object of interest accurately.

The appearance altering variations can co-occur in the real-world images as shown in Fig. 1.3. The joint effect of these variations may result in a large distance between the feature vectors of both objects. This will result in the classifier assigning different class labels to both objects. An efficient image understanding system should be *invariant* to these appearance altering transformations while being *selective* to the aspects of the image that are important for discrimination. For example, an image understanding system should extract features representative of the parts of the car

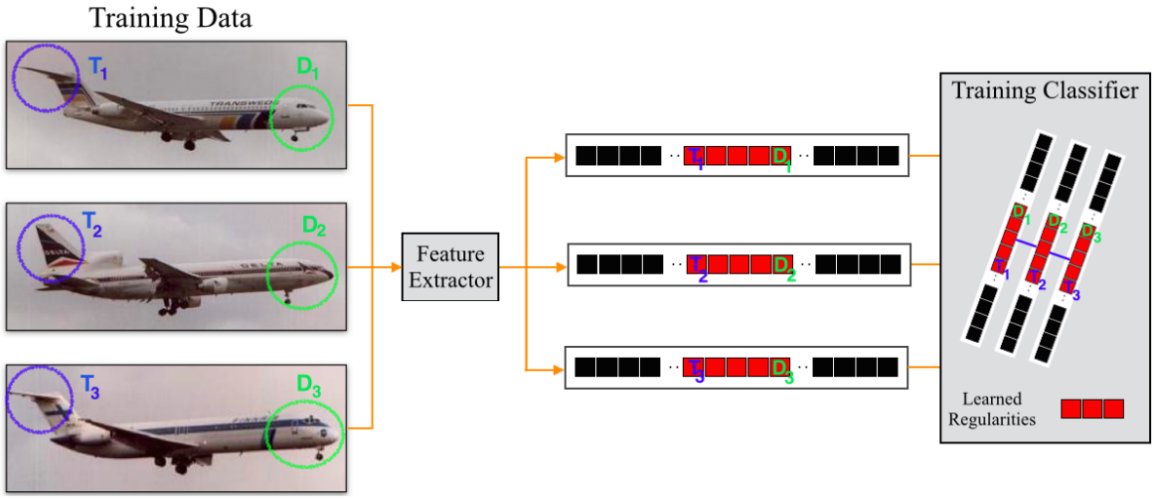


Fig. 1.2 Illustration presents the classifier which learns local and global regularities from the training set. The figure depicts that the classifier fails to learn the whole plane as a regularity due to the design variation in tail and deck of the planes.

(selective) irrespective of the position, orientation or scale (transformations) of the car. The next section details several such systems.

1.2 Image Understanding Systems

Numerous Image Understanding systems have been proposed over the years that capture the invariant and selective image representations to solve the individual image understanding tasks of object recognition and semantic image segmentation. These methods include architectures that: (i) encode *handcrafted* features extracted from the input images into rich non-hierarchical [19, 32, 116, 118] and hierarchical representations [3, 22, 27, 28, 86]; (ii) learn (*end-to-end*) feature hierarchies, directly from the input data [24, 35, 36, 48, 107, 109]; (iii) make use of the ideas from both the above-mentioned categories to develop *Hybrid Networks* that extract feature hierarchies from hand-crafted features [31, 40].

Hand-crafted features such as SIFT [19], HOG [116] or ScatterNet [3, 26–28, 86] represent the first class of architectures that incorporate the structural knowledge of images to produce discriminative and invariant (translation, rotation or even scale) representations. SIFT and HOG belong to the non-hierarchical class of architectures while the ScatterNet is a multi-layer architecture. This class of methods are elementary to design and cheap to evaluate but achieve only marginally good classification performance on different benchmarks [117]. The single layer architectures achieve an object

classification accuracy of around 70% [115] while the multi-layer ScatterNets perform with roughly 80% [22] accuracy on Cifar-10 and Caltech-101 datasets. Semantic texton forests were used to achieve a semantic segmentation accuracy of 67% on the MSRC 21 object class dataset [121].

End-to-end learned architectures represent the second category of networks which can learn the hierarchy of invariant features directly from the input images. Many such architectures have been proposed over the years including VGG [36], NIN [35], ResNet [104, 107], LeNet [17], Deep Belief Networks [24], etc. These networks have achieved the state-of-the-art performance on both object classification (95% [104]) and semantic segmentation (85% [69]) on various datasets.

These models produce the state-of-the-art performance only for applications with sizeable labeled training datasets and tend to overfit [97] on many other applications where large datasets are not available. Also, their design and optimal configuration are not well understood which it makes difficult to develop them. The design of these networks requires expertise, and often a designer with limited experience may produce a sizeable over-expressive network. The optimization of these large networks involves the learning of a large number of redundant hyperparameters which can lead to more optimal computationally efficient networks [145, 146]. However, the optimization requires considerable computational resources and energy which can be another challenge [110].

Hybrid models are the third class of networks that are formulated by combining the concepts from both earlier explained techniques [19, 31, 39, 40, 117, 122]. These models use the handcrafted descriptors extracted by the earlier layers to learn hierarchical features rapidly in a computationally efficient manner [31, 40]. These hierarchical representations are then used by a Support Vector Machine (SVM) or a Conditional Random Field (CRF) to perform the classification [40] or segmentation tasks [122]. The architecture of these models is inspired by the different regions of the visual cortex [40] which similarly learn hierarchical (from the primary visual cortex (V1) to inferior temporal (IT)) invariant and discriminative representations [153, 154]. The primary visual cortex (V1) learns [119, 155] sparse overcomplete representation of the visual input, with receptive fields of neurons that are most responsive to low-level features at different scales and orientations [147]. The handcrafted descriptors extracted by the hybrid models in the earlier layers are similar low-level features. Similar to the hierarchical features learned by the unsupervised learning section of the hybrid models, the extrastriate visual areas such as V2 or V4 contains neurons that are tuned to be selective for a multitude of complex shapes and are believed to be tuned in an

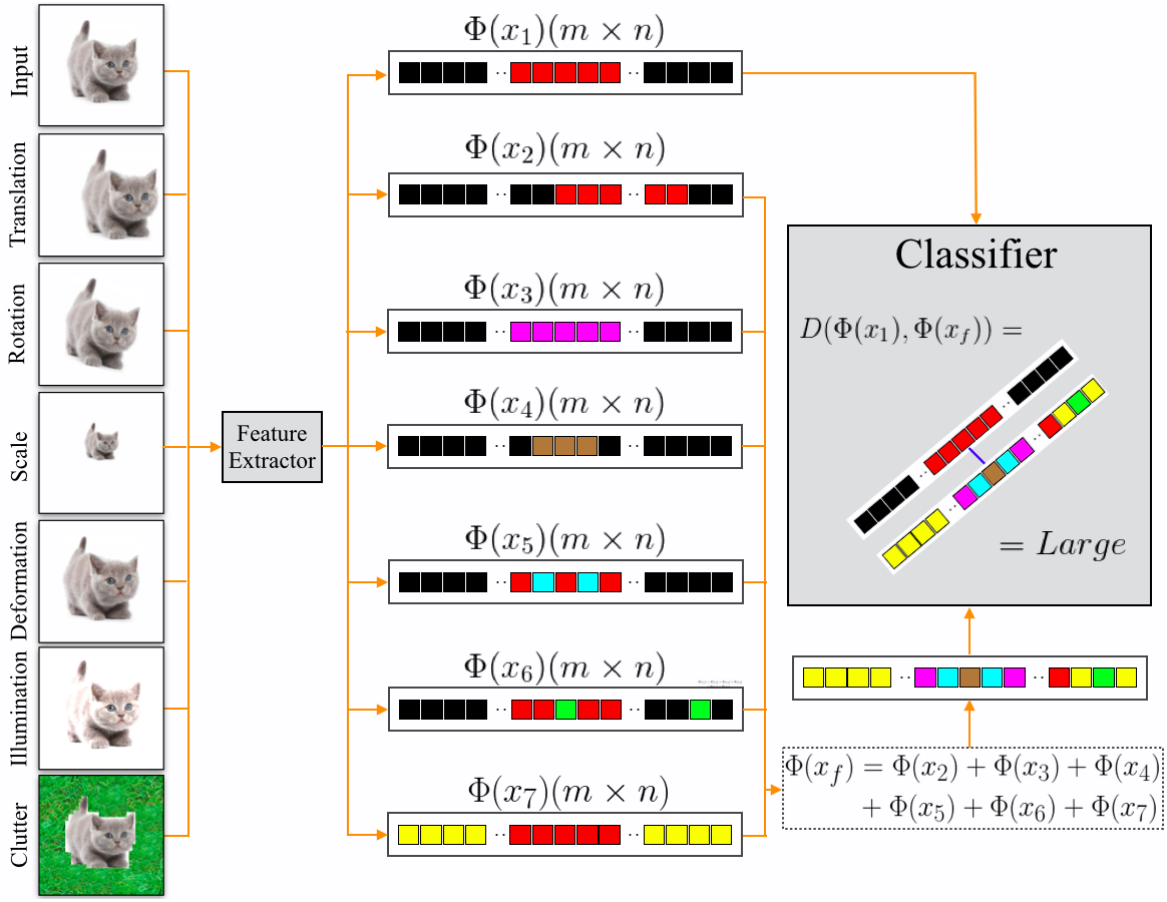


Fig. 1.3 Illustration of real-world scenario in which the distance measured by a trained classifier between feature vectors extracted from the image (x_1) and it's altered version (x_f), is significant. The changes in position, orientation and scale can considerably alter the object's appearance. The variation in the lighting conditions and background clutter can also change the object's appearance. In order to present the effects of the above-mentioned variations on object recognition, the original cat image (x_1) (as shown in Fig. 1.1) is translated (x_2), rotated (x_3), scaled (x_4), deformed (x_5) as well as captured under, different illumination (x_6) and complex background (x_7). The effect on the features corresponding to each variation is presented with a different color block as follows: Magenta Blocks (x_3): features extracted from the rotated object, Brown Blocks (x_4): features extracted from the rescaled object, Cyan Blocks (x_5): features extracted from the deformed parts of the object, Green Blocks (x_6): features extracted from the illuminated parts of the object, Yellow Blocks (x_7): features extracted from the background (i.e. Grass).

unsupervised manner as it is consistent with the physiological data [149–152]. Finally, the neurons in inferior temporal (IT) are tuned to form semantic representations corresponding to different object classes [120, 148]. This part of the visual cortex

is believed to assign labels to the learned features. These models have produced a promising performance on both object recognition (80% [19, 31, 41, 117]) and semantic segmentation (75% [122]) tasks.

This dissertation proposes the ScatterNet Hybrid Deep Learning (SHDL) framework that rapidly learns hierarchical features mainly from unlabelled data, in a computationally efficient manner. These features are similar to the representations encoded by different regions of the visual cortex. The framework is composed of a hand-crafted front-end, an unsupervised learning based middle-section, and a supervised back-end. The handcrafted front-end is analogous to the primary visual cortex that extracts low-level features at different scales and orientations [40, 119, 155]. The unsupervised learning based middle-section learns hierarchical features similar to the extrastriate visual areas of the visual cortex [149–152] while the supervised back-end assigns labels to the hierarchical features same as the inferior temporal (IT) region of the visual cortex [148]. The term 'Hybrid' is coined because the framework uses both unsupervised as well as supervised learning. Each layer in the proposed framework is automatically optimized to produce the desired computationally efficient architecture.

Two ScatterNet Hybrid Deep Learning (SHDL) networks have been constructed using the ScatterNet hybrid framework. We proposed two hand-crafted front-ends for the SHDL networks which extract invariant features from the input signals. Next, the proposed front-ends are combined with two unsupervised learning modules to learn hierarchical features. These hierarchical features were finally used either by a Support Vector Machine (SVM) or a Conditional Random Field (CRF) to perform the classification or segmentation tasks. The proposed front-ends have also been shown to improve the performance and learning of current Deep Supervised Learning Networks (VGG, NIN, ResNet) as shown in Chapter 5.

The SHDL networks derived from the proposed ScatterNet Hybrid framework have the following attractive properties: (i) SHDL networks outperform the unsupervised and semi-supervised architectures on object recognition and semantic image segmentation tasks. The system also exceeds supervised deep networks on relatively small labeled datasets but fails to outperform them on large datasets. This property can be beneficial for applications where the amount of labeled data is limited. (ii) The SHDL network can rapidly learn intricate structure from the unlabelled hand-crafted descriptors. The speedy optimization of the reconstruction loss function for the unsupervised learning mid-section module is obtained using structural priors or eigen-decomposition based approximation. (iii) Each layer of the SHDL network is automatically designed as

a specific optimization problem leading to a computationally efficient deep learning architecture as compared to the more usual deep network architectures.

1.3 Contributions

The contributions made in this dissertation are mentioned below:

- **SHDL Framework (chapter 3)**: This chapter proposes the ScatterNet Hybrid Framework for Deep Learning (SHDL) which is inspired by the circuitry of the visual cortex as detailed in the previous section. This framework is able to rapidly learn hierarchical features from mainly unlabeled data in an computationally efficient manner. These advantages make the proposed framework an attractive choice over the standard deep learning architectures for a range of application areas, where labeled data is difficult or expensive to generate.

- **Hand-crafted front-end module (chapter 4)**: This chapter introduces two hand-crafted scattering (shallow) networks that form the front-end of the ScatterNet Hybrid Deep Learning (SHDL) networks. These front-ends extract invariant edge representations (due to the shallow architecture) that are densely spaced over scale.

- **Efficient learning of Deep Supervised Networks using the hand-crafted front-end Module (chapter 5)**: The proposed hand-crafted front-end module is used to accelerate the learning of different deep supervised convolutional networks. Also, these networks are shown to improve the generalization of the pruned deep convolutional networks for limited labeled datasets.

- **SHDL Networks (chapter 6)**: This chapter introduces two ScatterNet Hybrid Deep Learning (SHDL) Networks (generative and deterministic) to solve the individual image understanding tasks of object recognition and semantic image segmentation. These networks can rapidly learn the hierarchical features from mostly unlabelled input signals and perform classification or segmentation with only a few labeled examples. Both the networks are more computationally efficient than other deep learning networks, making them a more desirable choice.

In order to avoid conflating the methods, we have presented each of these in isolation in this dissertation. However, the techniques address different aspects of the SHDL framework and complement each other.

1.4 Thesis overview

Chapter 2 is on the prior work and briefly describes the relevant networks that have been proposed over the years to solve the image understanding tasks of object recognition and semantic image segmentation. These methods include the single layer or multi-layer hand-crafted networks, end-to-end learned networks, and hybrid networks formulated by borrowing ideas from the above two categories. The chapter presents the features encoded by the methods from each class along with their performance on both the tasks on various datasets.

Chapter 3 presents the proposed ScatterNet Hybrid Framework for Deep Learning which is inspired by the circuitry of the visual cortex (as explained in Section 1.2). This chapter details the different modules of the proposed framework and how they relate to the various circuits of the visual cortex. The features encoded by each module are also presented along with their advantages (rapid learning from unlabelled data in a computationally efficient manner) over the conventional deep networks.

Chapter 4 presents two hand-crafted networks that form the front-end of the ScatterNet Hybrid framework. These networks extract low-level representations at different scales and orientations similar to the primary visual cortex (V1) [40, 119, 155]. This chapter introduces two variants of Mallat's Scattering network [3] that extract translation invariant low-level edge representations using the Dual-Tree Complex Wavelet Transform (DTCWT) [15]. These representations are densely spaced over scale as they are obtained from the multi-resolution signal derived from the input signal. These variants have been shown to outperform Mallat's Scattering network [3, 22, 27, 28, 86] on various classification tasks.

Chapter 5 proposes the use of hand-crafted front-end to improve the learning and generalization of deep supervised networks. This chapter introduces the DTCWT ScatterNet Convolutional Neural Network (DTSCNN) formed by replacing the first convolutional, rectified linear unit (ReLu), and pooling layers of a Convolutional Neural Network (CNN) with the proposed hand-crafted front-end. This formulation accelerates the convergence of the deep supervised convolutional networks as it has fewer filter weights to learn as compared to its corresponding CNN architecture. Besides, the CNN layers can learn more complex patterns from the start of learning as it is not necessary to wait for the first layer to learn edges because the ScatterNet already extracts them. This is also shown to improve the generalization of the CNN for limited labeled datasets.

Chapter 6: The handcrafted front-end module is shown to improve the learning and generalization of the deep supervised networks, as shown in Chapter 5. Despite

this, the training of deep networks is still slow and requires more than the desired labeled training examples. This chapter proposes two ScatterNet Hybrid Deep Learning (SHDL) Networks that are constructed from the proposed ScatterNet Hybrid framework. These networks learn hierarchical representations rapidly from mainly unlabelled input signals and few labeled examples to solve the individual image understanding tasks of object recognition and semantic segmentation. The proposed SHDL networks are also computationally efficient as opposed to other deep learning networks, making them a more desirable choice.

Chapter 7: summarizes the contributions made in this dissertation, reviews the results, and looks at research questions that arise from the work presented in this dissertation. In particular, it shows the effectiveness of hybrid networks which extract hierarchical features using computationally efficient modules that can be trained rapidly utilizing mainly unlabelled and small labeled datasets. Proposals are made for future research directions which are pertinent given our results.

Chapter 2

Literature Review

This chapter presents the relevant approaches that have been proposed in the past to solve the image understanding tasks of object recognition and semantic segmentation. These techniques are divided into three primary categories: (i) hand-crafted; (ii) end-to-end learned, and; (iii) hybrid, networks. The networks for each category extract features with different degree of invariance and discrimination using the elementary operations of filtering, non-linearity, and pooling to achieve the image understanding tasks. These operations are wrapped together as the feature extraction block which is used by the architectures either once (shallow) or multiple times (deep).

The subsequent sections present the networks corresponding to each category along with their performance on the image understanding tasks.

2.1 Handcrafted Architectures

This category of architectures use hand-encoded filters, non-linear modulus, and pooling operators to extract invariant descriptors. Many such single-layer and multi-layer networks have been proposed over the years including SIFT [19], HOG [116], SURF [118] etc.

Scale-invariant feature transform or SIFT [19] is a single-layer hand-crafted network that extracts a local feature descriptor from the important regions on the input image. This descriptor is approximately invariant against rotation, scale, illumination changes, etc. Speeded up robust feature (SURF) [118] is another local descriptor which is the faster version of SIFT and can be more robust against certain image transformations. The histogram of oriented gradients (HOG) [116] is another feature descriptor which is similar to that of SIFT but differs in that it is computed on a dense grid of uniformly spaced cells and uses overlapping local contrast normalization for improved accuracy.

In addition to the single-layer descriptors described above, there have been numerous other descriptors that have been proposed.

More recently, multi-layer scattering networks were introduced by Mallat and his colleagues [22, 27, 28, 86] which use a battery of complex wavelet filters, non-linear modules, and pooling operations to extract invariant descriptors. The information lost by the averaging is recovered by computing the next layer of invariant coefficients, with the same wavelet convolutions, non-linear modulus, and poolings operations. A wavelet scattering is thus a multi-layer convolutional network which cascades wavelet transforms and modulus operators.

Since SIFT [19] (single layer) is the most popular and ScatterNet [86] (multi-layer) is the most recent architecture, both of them are explained in detail below.

2.1.1 Scale-Invariant Feature Transform (SIFT)

Scale-invariant feature transform or SIFT [19] extracts local invariant descriptor by first locating the key-points on the image. These potential interest points are invariant to scale changes as they are identified by scanning the image over a range of scales. This is implemented efficiently by constructing a Gaussian pyramid and searching for local peaks (termed key points) in a series of difference-of-Gaussian (DoG) images. Candidate keypoints are localized to sub-pixel accuracy and eliminated if found to be unstable. Next, a neighborhood is selected around the key-point which is divided into blocks of a certain size. For each sub-block, eight bin orientation histograms are created which are further represented as a vector to form the descriptor. Several measures are taken to achieve robustness against illumination changes, rotation, etc.

SIFT keypoint detector

A SIFT key-point is a circular image region, described by a geometric frame of four parameters: the key-point center coordinates x and y, it's scale (the radius of the region), and its orientation (an angle expressed in radians). The SIFT detector searches for "blobs" key-points at multiple scales and positions that are invariant (or, more accurately, covariant) to translation, rotations, and rescaling of the image. The scale space is just a collection of images obtained by progressively smoothing the input image, which is analogous to gradually reducing the image resolution. Conventionally, the smoothing level is called scale. Increasing the scale by an octave means doubling the size of the smoothing kernel, whose effect is roughly equivalent to reducing the image resolution by half.

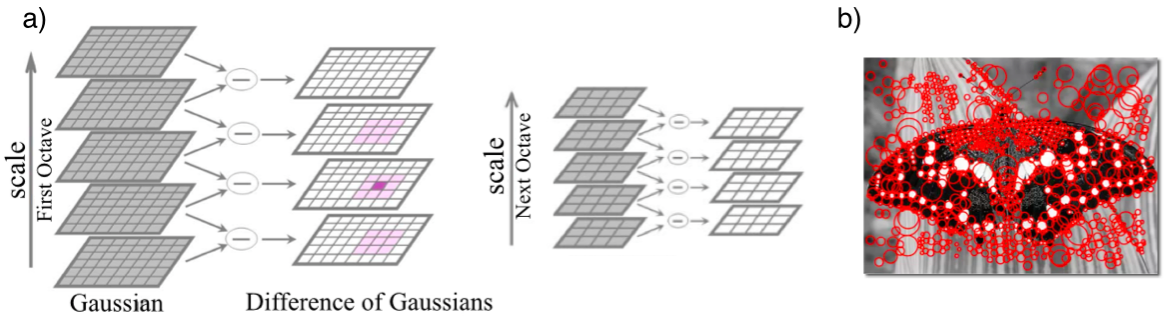


Fig. 2.1 Illustration of: a) Difference-of-Gaussians (DoG) procedure used to generate approximations to the Laplacian-of-Gaussian at multiple scales and octaves. b) Keypoints detected from an image at multiple locations and scales using the SIFT keypoint detector. Adapted from [19].

To search for image blobs at multiple scales, the SIFT detector construct a scale space as shown in Fig. 2.1(a). Blobs are detected as local extrema of the Difference of Gaussians (DoG) scale space, obtained by subtracting successive scales of the Gaussian scale space (as shown in Fig. 2.1(a)). Once extracted, local extrema are obtained, they are filtered to eliminate low-contrast responses, or responses close to edges and the orientation(s) are assigned, as explained next. The keypoint detector, applied to an image, detects keypoints at different locations and scales as shown in Fig. 2.1(b).

SIFT Descriptor

The final keypoint descriptor stage of the SIFT algorithm builds a representation for each key point based on a patch of pixels in its local neighborhood. The goal is to create a descriptor for the patch that is compact, highly distinctive and yet robust to changes in illumination and camera viewpoint. The SIFT keypoint descriptor is generated by sampling the magnitudes and orientations of the image gradient in the patch around the keypoint, and building smoothed orientation histograms to capture the essential aspects of the patch as shown in Fig. 2.2. The SIFT descriptor is suboptimal for the object detection task due to its inability to accurately capture the shape of the objects at the boundaries. The descriptor is created in a neighborhood around the detected keypoint as explained above. The descriptor represents the properties of the object well if the keypoint is within the object but for the keypoints which are on the boundary, the descriptor is constructed by including parts of the object as well as the background. Since SIFT fails to capture the boundary feature of the objects, this limits the performance of the descriptor.

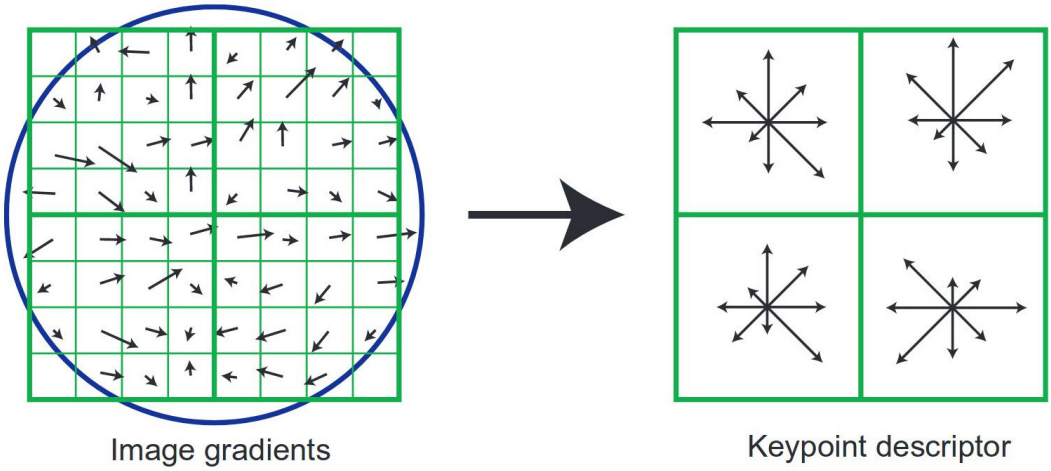


Fig. 2.2 The example illustrates the SIFT descriptor design. Left: Gradients within each of the sub-arrays are pre-rotated by the dominant orientation of the keypoint and weighted by a 2-D Gaussian weighting function (represented by the overlaid circle). A 16 × 16 array of samples is used to generate 16 8-bin histograms. Reproduced from [19].

2.1.2 Scatter Net

The Scattering network or ScatterNet, introduced by S. Mallat and Bruna in [3, 86], is a multilayer handcrafted convolutional networks that build locally invariant, stable, and informative signal representations by cascading wavelet modulus decompositions followed by a pointwise non-linearity and local averaging. Wavelets are stable to deformations and provide sparse image representations which makes them a good filtering choice [20].

The scattering network extracts the multi-layer translation invariant descriptors from the input signal x with a wavelet transform. Two-dimensional directional wavelets are obtained by dilating and rotating a single band-pass filter ψ [20]. Multiscale directional wavelet filters are defined for any $j \in Z$ and rotation $r \in G$ by

$$\begin{aligned} \psi_\lambda(t) = 2^{-j}\psi(2^{-j}rt) \quad where \quad \lambda = (2^{-j}, r) \\ and \quad \psi(t) = \psi^a(t) + \iota\psi^b(t) \quad t = (t_1, t_2) \end{aligned} \tag{2.1}$$

The real and imaginary parts of the complex Morlet wavelets at multiple scales and orientations is shown in Fig. 2.3. A wavelet transform filters signal x using a family of

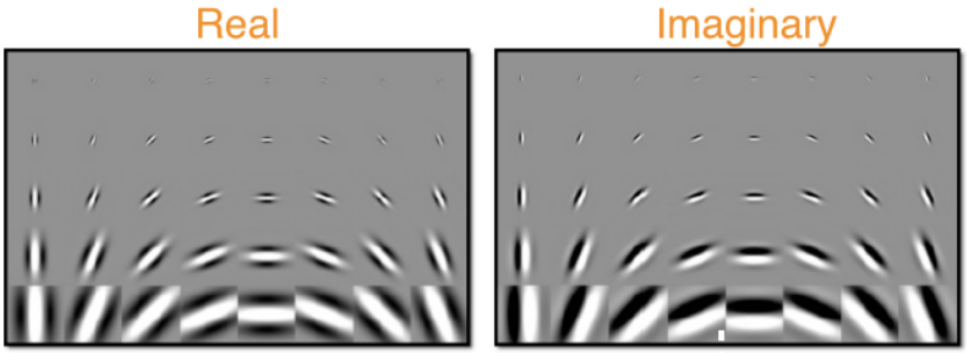


Fig. 2.3 Illustration of the real and imaginary parts of the complex Morlet wavelets at multiple scales and orientations.

wavelets $\psi_{\lambda_1}(t)$ (as shown in Fig. 2.4(a)) formulated as:

$$x \star \psi_{\lambda_1}(t) = x \star \psi^a_{\lambda_1}(t) + \iota x \star \psi^b_{\lambda_1}(t) \tag{2.2}$$

A wavelet transform response commutes with translations and is therefore not translation invariant. To build a translation invariant representation, the wavelet coefficient is first pooled using L_2 non-linearity (as shown in Fig. 2.4(b)) as shown below:

$$|x \star \psi_{\lambda_1}(t)| = \sqrt{|x \star \psi^a_{\lambda_1}(t)|^2 + |x \star \psi^b_{\lambda_1}(t)|^2} \tag{2.3}$$

Next, a local average is computed on the pooled output $|x \star \psi_{\lambda_1}| \star \phi(t)$ to get the desired locally translation invariant representation as shown in Fig. 2.4(c). The averaging by ψ_{2^J} at the output is an averaging operator which aggregates coefficients to build an invariant. It has been argued [2] that an average pooling results in the loss of higher frequency components. The high frequencies lost by the averaging are recovered as wavelet coefficients in the next layers, which explains the importance of using a multilayer network structure. As a result, it only loses the phase of these wavelet coefficients. This phase may however be recovered from the modulus thanks to the wavelet transform redundancy. These higher frequencies can be recovered by calculating the wavelet coefficients $\left\{|x \star \psi_{\lambda_1}| \star \psi_{\lambda_2}(u)\right\}_{\lambda_2}$ of $|x \star \psi_{\lambda_1}|$, as shown in Fig. 2.4(d). The recovered frequencies are converted into a translation invariant representation by again taking a local average $||x \star \psi_{\lambda_1}| \star \psi_{\lambda_2}| \star \phi(t) \vee \lambda_1, \lambda_2$. Their L_1 norms define a much larger family of invariants, for all λ_1 and λ_2.

For classification, it is often better to compute localized descriptors which are invariant to translations smaller than a predefined scale 2^J, while keeping the spatial variability at scales larger than 2^J. A scattering transform computes higher-order

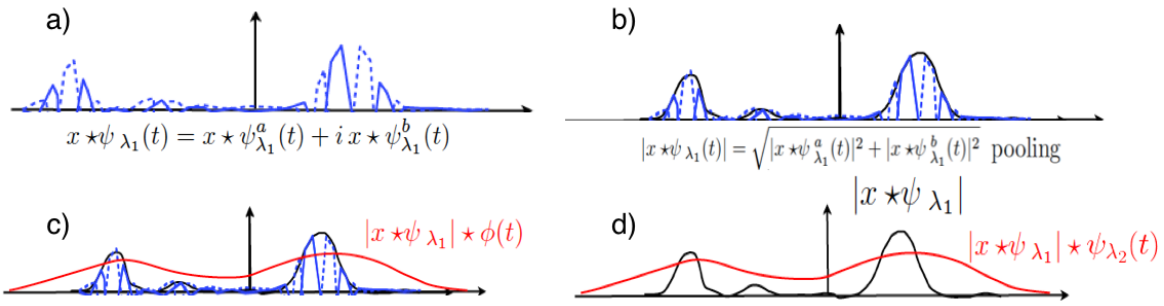


Fig. 2.4 Illustration presents a) signal x convoluted with family of wavelets $\psi_{\lambda_1}(t)$ b) The regular envelop $|x \star \psi_{\lambda_1}(t)|$ of the filtered response is obtained using pooling with L_2 non-linearity c) A wavelet transform response commutes with translations, and is therefore not translation invariant. To build a translation invariant representation, the pooled response $|x \star \psi_{\lambda_1}(t)|$ is averaged as $|x \star \lambda_1| \star \phi(t)$ to the support of ϕ d) Average pooling results in the loss of non-zero frequency components. These non-zero frequencies can be recovered by calculating the wavelet coefficients $\left\{|x \star \psi_{\lambda_1}| \star \psi_{\lambda_2}(u)\right\}_{\lambda_2}$ of $|x \star \psi_{\lambda_1}|$.

coefficients by further iterating on wavelet transforms and modulus operators. At a maximum scale 2^J, wavelet coefficients are computed at frequencies $2^j \geq 2^{-J}$, and lower frequencies are filtered by $\psi_{2^J}(u) = 2^{-J/2}\psi(2^{-J}u)$. The scattering coefficients $S_J x$ for the network at different scales and orientation for multiple layers can be obtained using the following:

$$S_J x = \begin{pmatrix} x \star \phi_{2^J} \\ |x \star \psi_{\lambda_1}| \star \phi_{2^J} \\ ||x \star \psi_{\lambda_1}| \star \psi_{\lambda_2}| \star \phi_{2^J} \\ |||x \star \psi_{\lambda_1}| \star \psi_{\lambda_2}| \star \psi_{\lambda_3}| \star \phi_{2^J} \\ \dots \end{pmatrix}_{\lambda_1, \lambda_2, \lambda_3, \dots} \tag{2.4}$$

As opposed to most convolution networks, a scattering network outputs coefficients $S_J[p]x(u)$ at all layers $m \leq m_{max}$, and not just at the last layer m_{max}, as shown in Fig. 2.5. In the figure, the signal x is represented with f.

Trans-Roto-Scale Invariant ScatterNet

Sifre and Mallat [26–28] extended the scattering network to incorporate rotation and scale invariant in addition to translation invariance. The proposed network first processes the position (u), rotation (θ) and scale(j) variables separately to build first a representation that is invariant to translations but covariant to rotations and scale.

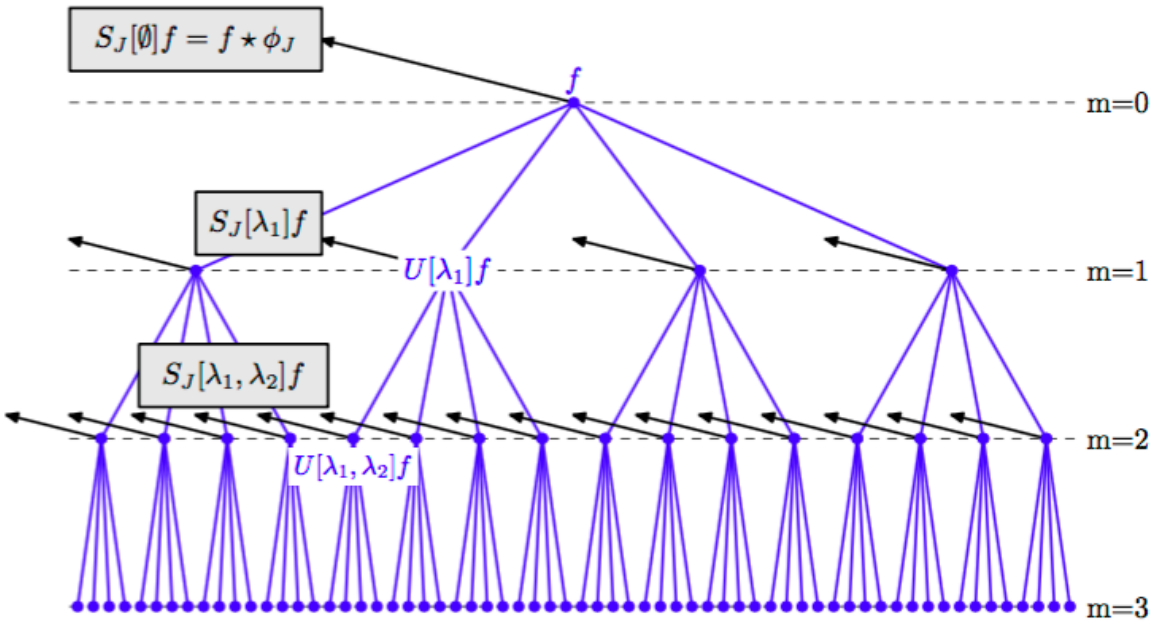


Fig. 2.5 A scattering propagator U_J applied to f computes each $U\left[\lambda_1\right] f = |f \star \psi_{\lambda_1}|$ and outputs $S_J\left[\oslash\right] f = f \star \phi_{2^J}$. Applying U_J to $U\left[\lambda_1\right] f$ computes all $U\left[\lambda_1, \lambda_2\right] f$ and outputs $S_J\left[\lambda_1\right] f = U\left[\lambda_1\right] \star \phi_{2^J}$. Applying U_J iteratively to $U\left[p\right] f$ outputs $S_J\left[p\right] f = U\left[p\right] \star \phi_{2^J}$. Reproduced from [86].

This representation is next appended on the rotations and scale dimensions to build invariance towards them.

The proposed scattering network that extracts descriptors that are invariant to translation, rotation and scale is detailed below. The wavelet-modulus operator applied on the signal x to achieve translation invariance (as explained in Section 2.1.2) is given by:

$$\widetilde{W_1} x = (x \star \phi_J, |x \star \psi_{\theta,j}|_{\theta,j}) = (S_0 x, U_1 x) \tag{2.5}$$

The locally invariant part of U_1 is computed with an averaging over the spatial and angle variables. It is implemented for each j_1 fixed, with a roto-translation convolution of $Y(h) = U_1 x(h, j_1)$ along the $h = (u', \theta')$ variable, with an averaging kernel $\Phi_J(h)$. For $p_1 = (g_1, j_1)$ and $g_1 = (u, \theta_1)$, this is written as:

$$S_1 x(p_1) = U_1 x(., j_1) \star \Phi_J(g_1) \tag{2.6}$$

We choose $\Phi_J(u', \theta') = (2\pi)^{-1} \Phi_j(u')$ to perform an averaging over all angles θ and over a spatial domain proportional to 2^J.

The high frequencies lost by this averaging are recovered through roto-translation convolutions with separable wavelets. Roto-translation wavelets are computed with three separable products. Complex quadrature phase spatial wavelets $\psi_{\theta_2,j_2}(u)$ or averaging filters $\Phi_J(u)$ are multiplied by complex 2π periodic wavelet $\bar{\psi}_k(\theta)$ or by $\bar{\phi}(\theta) = (2\pi)^{-1}$

$$\Psi_{\theta_2,j_2,k_2}(u,\theta) = \psi_{\theta_2,j_2}(u)\bar{\psi}_{k_2}(u) \tag{2.7}$$

$$\Psi_{0,J,k_2}(u,\theta) = \Phi_J(u)\bar{\psi}_{k_2}(u) \tag{2.8}$$

$$\Psi_{\theta_2,j_2,0}(u,\theta) = \psi_{\theta_2,j_2}(u)\bar{\phi}(\theta) \tag{2.9}$$

Finally, roto-translation wavelets for second layer are computed as $\widetilde{W_2}U_1x = (S_1x, U_2x)$ where S_1x is defined in (2.8) and

$$U_2x(p_2) = |U_1x(.,j_1) \star \Psi_{\theta_2,j_2,k_2}(g_1)| \tag{2.10}$$

with $g_1 = (u,\theta_1)$, $p_2 = (g_1,p_2)$, and $p_2 = (j_1,\theta_2-\theta_1,j_2,k_2)$. Since $U_2x(p_2)$ is computed with a roto-translation convolution, it remains covariant to the action of the roto-translation group. Fast computations of roto-translation convolutions with separable wavelet filters $\Psi_{\theta_2,j_2,k_2}(u,\theta) = \psi_{\theta_2,j_2}(u)\bar{\psi}_{k_2}(\theta)$ are performed by factorizing

$$Y \star \Psi_{\theta_2,j_2,k_2}(u,\theta) = \sum_{\theta'}(\sum_{u'} Y(u',\theta')\psi_{\theta_2,j_2}(r_{-\theta'}(u-u')))\bar{\psi}_{k_2}(\theta-\theta') \tag{2.11}$$

It is thus computed with a two-dimensional convolution of $Y(u,\theta')$ with $\psi_{\theta_2,j_2}(r_\theta u)$ along $u = (u_1,u_2)$, followed by a convolution of the output and a one-dimensional circular convolution of the result with k_2 along θ. Fig. 2.6 illustrates this convolution which rotates the spatial support $\psi_{\theta_2,j_2}(u)$ by θ while multiplying its amplitude by $\bar{\psi}_{k_2}(u)$.

Applying $\widetilde{W_3} = \widetilde{W_2}$ to U_2x computes second order scattering coefficients as a convolution of $Y(g) = U_2x(g,\bar{p}_2)$ with $\Phi_J(g)$, for $\bar{p}_2$ fixed is given as:

$$S_2x(p_2) = U_1x(.,\bar{p}_2) \star \Phi_J(g) \tag{2.12}$$

The output roto-translation of a second order scattering representation is a vector of coefficients:

$$Sx = (S_0x(u), S_1x(p_1), S_2x(p_2)) \tag{2.13}$$

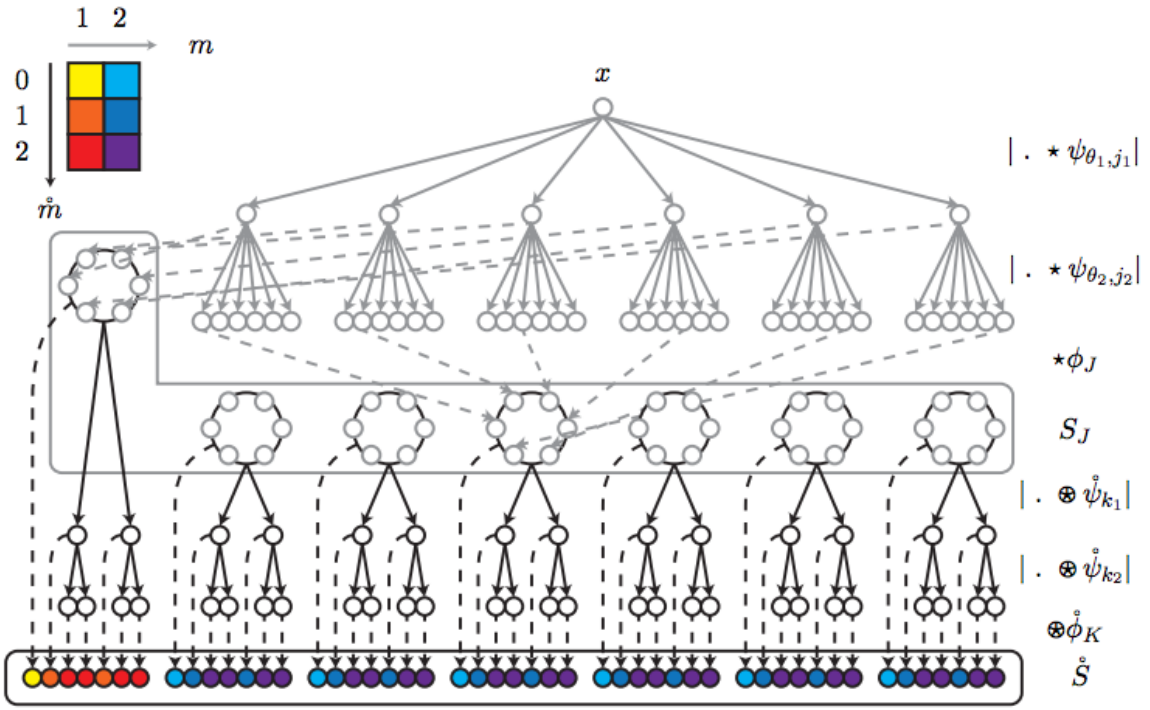


Fig. 2.6 Separable Scattering architecture. First spatial scattering layers in grey, second scattering layers in black. Spatial wavelet-modulus operators (grey arrows) are averaged (doted grey arrows), as in [86]. Outputs of the first scattering are reorganized in different orbits (large black circles) of the action of the rotation on the representation. The second cascade of wavelet-modulus operators along the orbits (black arrows) splits the angular information in several paths that are averaged (doted black arrows) along the rotation to achieve rotation invariance. Output nodes are colored concerning the order m, $\mathring{m}$ of their corresponding paths. Reproduced from [26].

with $p_1 = (u, \theta_1, j_1)$ and $p_2 = (u, \theta_1, j_1, \theta_2, j_2, k_2)$. The scale invariance is similarly obtained by filtering jointly across the position (u), rotation (θ) and scale(j) variables as detailed in [28].

This class of methods are elementary to design and cheap to evaluate but achieve only marginally good performance on different benchmarks. SIFT is the best single layer local descriptor which only reached a classification accuracy of approximately 73% and 65% on Caltech-101 and CIFAR-10 respectively [115] due to the reasons detailed in Section 2.1.1. Multi-layer ScatterNets [22] descriptors have performed much better than SIFT and even unsupervised as well as semi-supervised learning techniques, achieving the classification accuracy of roughly 79% and 82% on Caltech-101 and CIFAR-10 respectively. An ensemble of hand-crafted features was used with a random

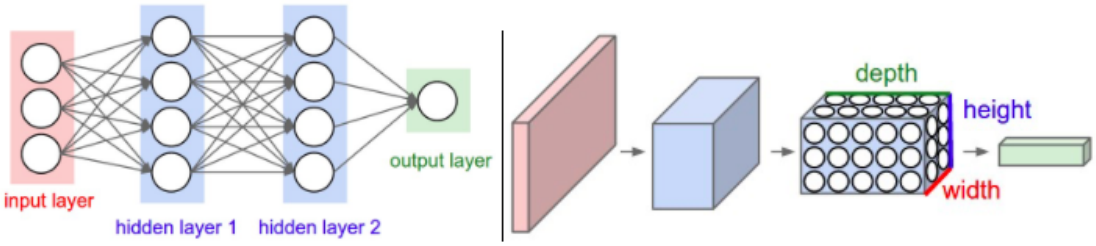


Fig. 2.7 Illustration of Left: An earlier 3-layer Neural Network with two hidden layers. Right: A convolutional neural network architecture arranges its neurons in three dimensions (width, height, depth), as visualized in one of the layers. In this architecture, the weights of the filters are locally connected within a layer and shared between different locations, instead of full connectivity. Adapted from [18].

forest to produce a semantic segmentation accuracy of 67% on the MSRC 21 object class dataset [121].

The next section presents the end-to-end learned networks which perform considerably better as compared to the hand-crafted systems.

2.2 End-to-end Learned Networks

This class of networks has made significant advances towards the tasks of object recognition and semantic segmentation. These networks learn invariant features by training these network using backpropagation, in an end-to-end manner using supervised or unsupervised learning directly from the input images. The features learned in the earlier layers of these networks are responsive to the presence of edges at particular orientations and scales. The middle layers usually detect motifs by spotting specific arrangements of edges, regardless of small variations in the edge positions. The later layer may assemble motifs into larger combinations that correspond to parts of familiar objects, and subsequent layers would detect objects as combinations of these parts.

Several such supervised and unsupervised architectures have been proposed over the years. The earlier architectures connected the neurons in a layer to every other neuron in the subsequent layer. The optimization of these connection weights was slow and required sizeable labelled training examples due to the large number of connections. As an alternative, convolutional architectures were introduced that exploited weight-sharing to accelerate the network training by significantly reducing the number of free parameters to being learned. These constraints on the model also enabled the

convolutional networks to achieve better generalization. An example of the earlier and the convolutional networks is shown in Fig. 2.7.

An exhaustive list of every new deep learning architecture would be infeasible, and outside the scope of this dissertation. However, in the next sections, we have made an effort to cover the elementary and modern convolutional architectures which have both inspired our work and formed the basis of many of our results.

2.2.1 Convolutional Neural Network

Convolutional Neural Network (CNN) is a subclass of neural networks introduced by Yann LeCun et al. [17] that cascades filter banks with pooling and nonlinearities. These filters are learned for each specific task using the backpropagation algorithm with the gradient descent strategy. The network is designed to exploit spatially-local correlations in images (weight sharing) which requires a much lower number of parameters to be learned.

These networks learn the desired features from the input image by first performing convolutions that compute the output of neurons connected to local regions in the input images. During the forward pass, the dot product between the entries of the filter and the input is computed. Intuitively, the network learns filters that activate for a specific type of feature at some spatial position in the input. An activation function thresholding at zero is applied in the Rectified Linear Units (ReLu) layer to increase the nonlinear properties of the decision function. Next, the pooling layer performs a max operation along the spatial dimensions to reduce translation dependence. Finally, one or more fully connected layers compute the class scores resulting in a vector whose length corresponds to some classes used. This type of layer is different from previous layers in the sense that this layer is connected to all the neurons in the last layer.

A Convolutional Neural Network (CNN) architecture constructed using alternating convolutional and max-pooling layers is shown in Fig. 2.8. The individual layers of the network are described below.

- **Input Layer:** The input consists of images that are arranged in 3 dimensions: width, height, depth (note that the word depth here refers to the third dimension of an activation volume, not to the depth of a full Neural Network, which can relate to the total number of layers in a network.) For example, the input images in CIFAR-10 [16] are an input volume of activations, and the volume has dimensions 32x32x3 (width, height, depth respectively). During the training, a batch of images is often given as input to the network. In that case, the input

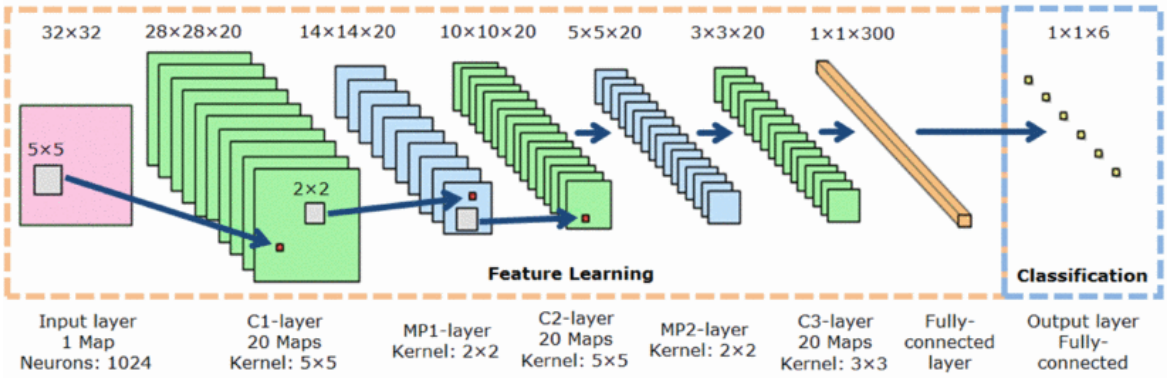


Fig. 2.8 Convolutional Neural Network (CNN) architecture constructed using alternating convolutional and max-pooling layers. Adapted from [21].

would be a 4D array where the fourth dimension will correspond to the number of images in the batch.

- **Filtering (Convolutional Layer):** The end-to-end trained architectures learn the weights of the filters to extract hierarchical (low, mid and high, level) features. As explained earlier, the earlier layers learn low-level functions while the middle and layer layers learn more complex features.

  Each layer of these networks computes the output of neurons that are connected to local regions in the input, by computing a dot product between their weights and the area they are connected to in the input volume. Therefore, the convolutional network can be thought of as the composition of the number of functions.

$$f(x) = f_L(f_2(f_1(x; w_1); w_2), w_L) \tag{2.14}$$

  Each function f_l takes as input, data x_l and a parameter vector, w_l and produces as output a data x_{l+1}. While the type and sequence of functions is usually handcrafted, the parameters $w = (w_1, ..., w_L)$ are learned from data. Each x_i is a $M \times N \times K$ real array of $M \times N$ pixels and K channels per pixel. Hence, the first two dimensions of the array span space, while the last one spans channels. Note that only the input $x = x_1$ of the network is an actual image, while the remaining data are intermediate feature maps.

  Few filters learned by the first layer of the network detailed in Fig. 2.8, on cifar-10 dataset [16], are shown in Fig. 2.9. An example image (class: frog) from the dataset is chosen to show the activations obtained after applying few of the first layer filters on that particular image.

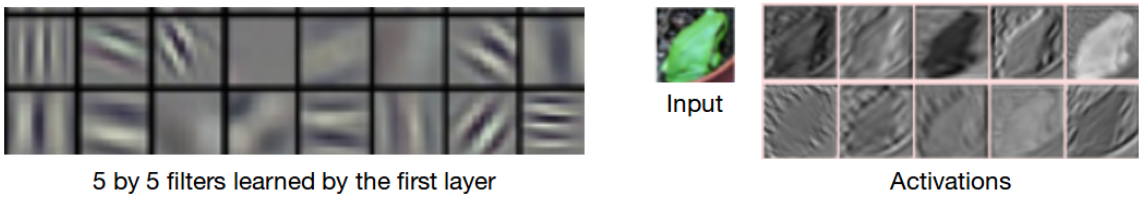


Fig. 2.9 Illustration shows a set of filters learned by the first layer of the network on cifar-10 dataset [16]. An example image (class: frog) from the dataset is chosen to show the activations obtained after applying few of the first layer filters on that particular image.

- **Activation functions (Non Linearity Layer):** Activation functions are a critical component of the deep networks. They introduce non-linearity in the model that allows it to learn complex decision boundaries. Several activation functions have been proposed in the past including RELU [126], P-RELU [127], etc. These functions are designed to overcome the problem of exploding/vanishing gradients that is commonly encountered with traditional functions such as the sigmoid [128]. This function is applied to increase the nonlinear properties of the output obtained from the convolution layer.

  The simplest non-linearity is obtained by following a linear filter by a non-linear gating function, applied identically to each component (i.e. point-wise) of a feature map. This is shown for the Rectified Linear Unit (ReLU), formulated as:

$$y_{ijk} = max\left\{0, x_{ijk}\right\} \tag{2.15}$$

- **Pooling Layer:** Pooling is applied on the feature representation extracted using filtering to introduce translation invariance. This is achieved by first partitioning the feature maps into non-overlapping rectangular regions. Next, every rectangular area is replaced by the pooled value using one of the several pooling techniques [123–125] that have been proposed over the years. In addition to pooling within each feature map, the same operation can be applied to feature maps of different orientation and scale to achieve rotation and scale invariance respectively. An example of pooling performed within and across feature maps is shown in Fig. 2.10.

  The max-pooling operator is given by:

$$y_{ijk} = max\left\{y_{i'j'k} : i \leq i' < i+p, j \leq j' < j+p\right\} \tag{2.16}$$

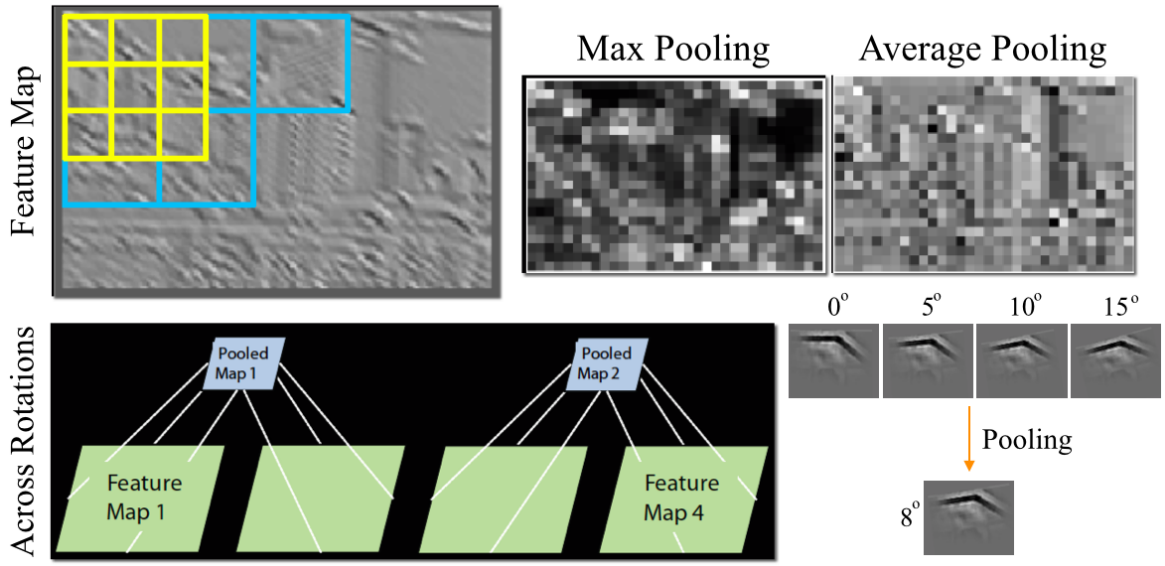


Fig. 2.10 Illustration of pooling operation performed (top) within a feature map is presented. The max and averaging pooling produce the resultant translation invariant feature map, as shown on the top right. Next, pooling across feature maps at different orientations is presented (below). An example of the rotation invariance achieved by this cross-channel pooling is shown (lower left) by pooling a mid-level feature captured at different orientations.

This operation also reduces the computation load for later layers as the spatial size of the representations is reduced.

- **Normalization Layer:** Another important CNN building block is channel-wise normalization [96, 144]. This operator normalizes the vector of feature channels at each spatial location. The normalization operator is given as:

$$y_{ijk} = \frac{x_{ijk}}{(\kappa + \alpha \sum_{k \in G(k')} x_{ijk}^2)^{\beta}} \tag{2.17}$$

where y is the output; κ, α, β are normalization parameters, G is a group of consecutive feature channels in the input map x_{ijk}. This normalization layer helps to ignore the effects of illumination.

Normalization can also be produced over a batch of images, termed as the batch normalization. Ioffe and Szegedy [129] proposed that batch normalization can also reduce the impacts of earlier layers by keeping the mean and variance fixed, which makes the layers independent of each other, finally producing faster convergence.

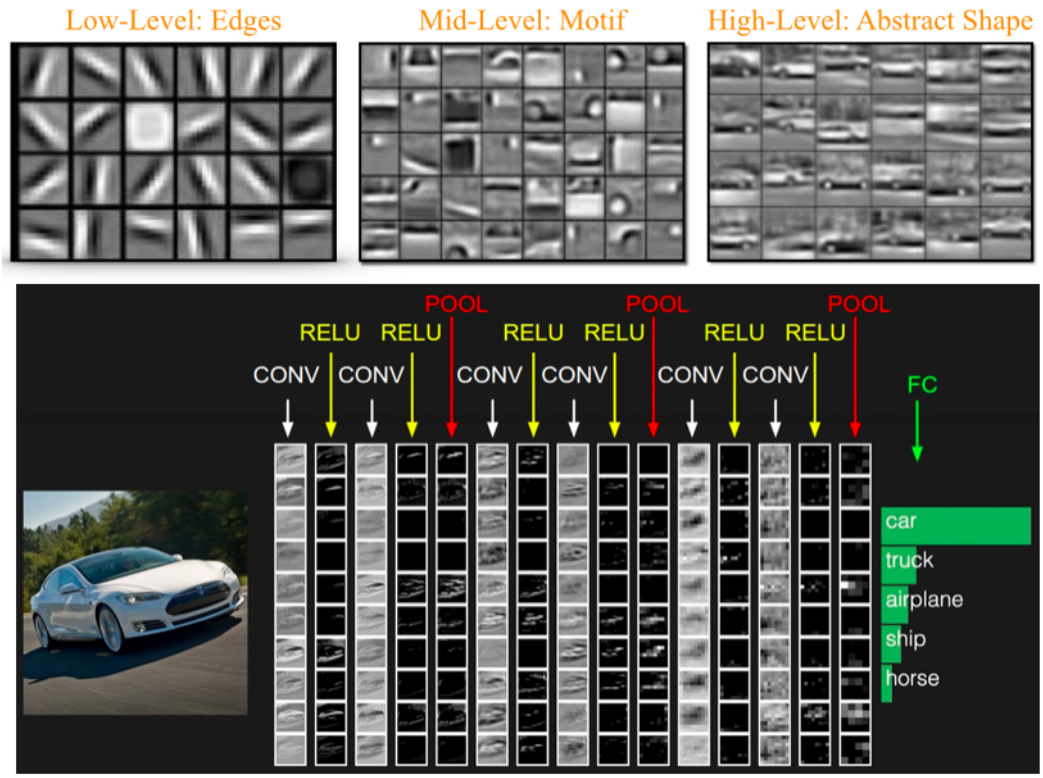


Fig. 2.11 Illustration of Top: Image representations at low (edges), mid (motif) and high (abstract shape) level learned by the filters during training on car class in the ImageNet dataset [9]. Below: The trained Convolutional Neural Network is applied to an input image of the car. The original volume stores the raw image pixels and the last volume stores the class scores. Each volume of activations along the processing path is shown as a column. Since it's difficult to visualize 3D volumes, we lay out each volume's slices in rows. The last layer volume holds the scores for each class, but here we only visualize the sorted top 5 scores and print the labels of each one. Adapted from [18].

- **Fully-connected Layer:** Finally, after several convolutional, non-linearity, and pooling layers, the high-level reasoning in the convolutional network is performed using the fully connected layers. Neurons in a fully connected layer have connections to all activations in the previous layer, as seen in regular neural networks. Their activations can hence be computed with a matrix multiplication followed by a bias offset.

Example of filter weights corresponding to different layers of a convolutional neural network trained using back-propagation (see Appendix A) on the ImageNet dataset [9] for the car class are shown in Fig. 2.11.

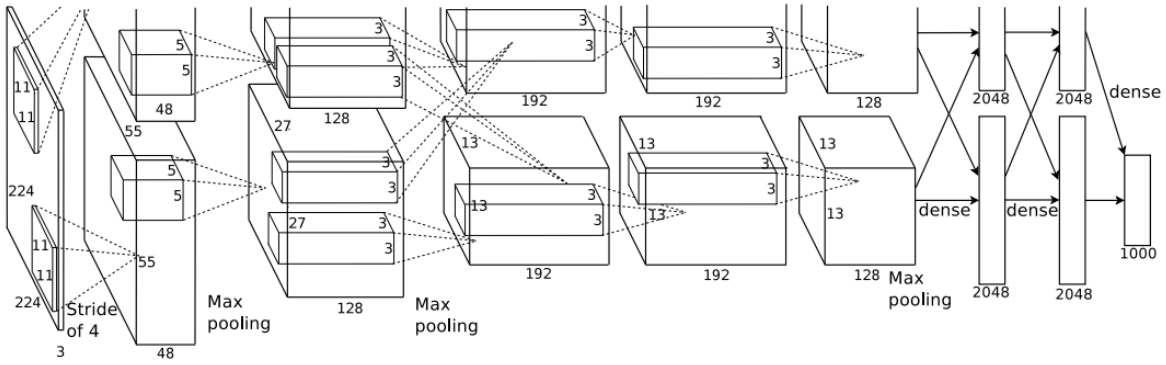


Fig. 2.12 An illustration of the architecture of our CNN, explicitly showing the delineation of responsibilities between the two GPUs. One GPU runs the layer-parts at the top of the figure while the other runs the layer-parts at the bottom. The GPUs communicate only at specific layers. The network's input is 150, 528-dimensional, and the number of neurons in the network's remaining layers is given by 253,440–186,624–64,896–64,896–43,264–4096–4096–1000. Adapted from [96].

The next sections present the modern architectures which have achieved the state-of-the-art performance in different tasks.

2.2.2 AlexNet

Training DNNs, that is neural networks with many (i.e., two or more) hidden layers, had proven difficult due to the high computational complexity, and the so-called 'vanishing gradient' problem [128]. Krizhevsky et al. [132] showed that a deep CNN (the specific architecture since referred to as AlexNet) trained on a very large dataset [9], with the appropriate initialization [130], weight decay ReLU activation functions [131] and dropout [132] could beat state-of-the-art methods on large scale object class recognition methods, based on hand-crafted features, by a large margin. This single paper introduced or motivated many of the recent advances in training neural networks such as Visual Geometry Group (VGG) [36], GoogLeNet [108], Residual Network (ResNet) [107], etc.

AlexNet notably used training-time and test-time augmentation to achieve its state-of-the-art accuracy. During training, random 224 × 224 crops of a 256 × 256 image are used, along with random mirroring of these crops. Also relighting augmentation is used, where the PCA component's overall RGB pixels in the image are used to perturb the "brightness" of the image and give some robustness to photometric variations in the test images. At test time "10× oversampling" is used, that is for each 256 × 256 test image, and its mirrored image, four corners, and one center crop are pushed through

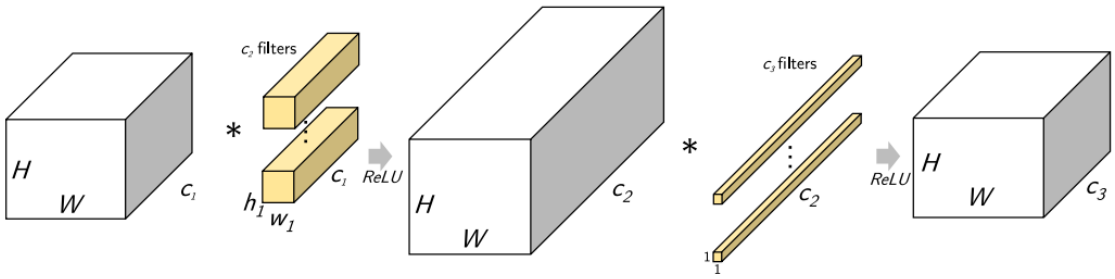


Fig. 2.13 Introduced in the Network in Network (NIN) architecture, an LDE consists of learning a 1 × 1 convolutional layer after a normal convolutional layer. The pairing of 1×1 filters and a non-linearity (i.e., ReLU) can effectively learn a non-linear transformation into a diferent space. If c3 <c2, then a transformation into a lower-dimensional space is learned as a more compact embedding of the learned representation. Adapted from [138].

the network, and the prediction is simply the average over these ten crops. AlexNet produced the Top-5 classification error of 15.4% on the ImageNet dataset.

AlexNet uses two filter groups throughout most of the layers of the model to split computation and model parameters across two GPUs, the motivation being that at the time GPUs did not have enough memory to fit such a large model. The authors observed that the filters on each GPU appeared to specialize in learning fundamentally different features regardless of initialization [132]. This exciting observation has mostly been ignored in subsequent networks where GPU memory has increased enough that such a split of the network is not require. The architecture of AlexNet is presented in Fig. 2.12.

2.2.3 Network in Network (NIN)

Lin et al. introduced NIN (Network in Network) [35], in which the main contribution was the use of so-called 'micro-networks,' consisting of increased non-linearity between convolutions using 1×1 convolutions. The authors claimed the extra non-linearities allowed the network to capture more complex functions. These 1×1 convolutions, illustrated in Fig. 2.13, have since been referred to as Low-Dimensional Embeddings (LDE). If the number of 1×1 filters is lower than the number of regular convolutional filters, then the 1×1 layer learns a non-linear transformation of the input feature map into a smaller space, i.e., a reduction in the number of filters by a mapping of a high-dimensional feature map onto a lower-dimensional feature map. This reduces

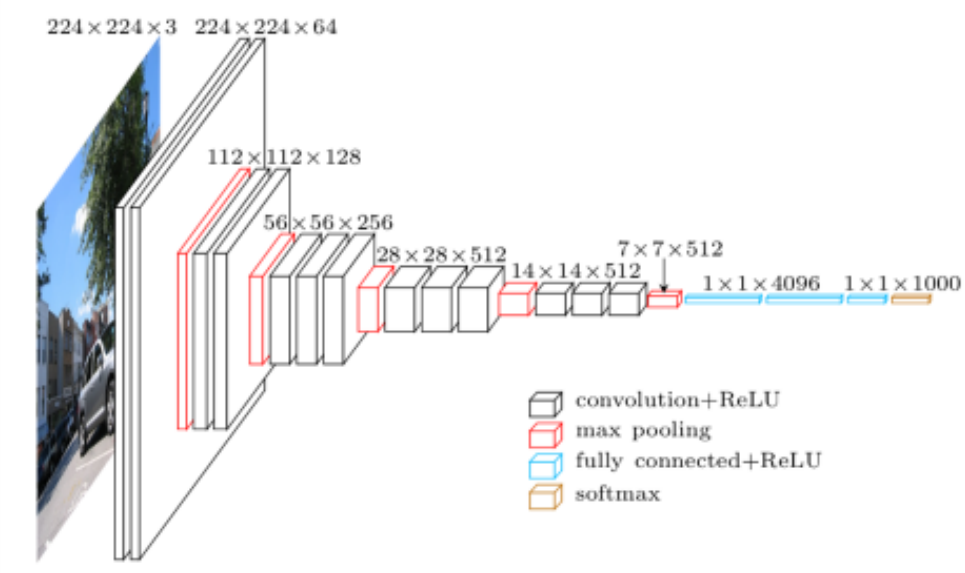


Fig. 2.14 The illustration presents the architecture of the Visual Geometry Group (VGG) used to perform classifcation on the 1000 class ImageNet dataset. Reproduced from [143].

the computation and parameters of convolutional layers significantly, while potentially learning a more compact and efficient representation.

2.2.4 Visual Geometry Group (VGG)

Simonyan and Zisserman of the Visual Geometry Group (VGG) [36] at Oxford proposed the VGG network [36] which is a 16 or 19 layer network. VGG is an evolution of the AlexNet models, with the same number of max-pooling layers. However, this network uses very small convolutional filters (3×3) in the convolutional layers with numerous convolutional layers between pooling, instead of the relatively large single-layers of convolutional filters in AlexNet (7×7). Also, VGG uses small non-overlapping max-pooling (2×2), and the fully convolutional trick introduced by Sermanet et al. [133]. In the fully convolutional trick, the fully connected layers in a stage are transformed into convolutional layers which makes it possible to efficiently run the CNN on images of any size. VGG also uses extensive training augmentation, extending the augmentation used in AlexNet [96] by adding scale augmentation, where crops are taken from images of different rescaled sizes. The architecture of the VGG network is shown in Fig. 2.14.

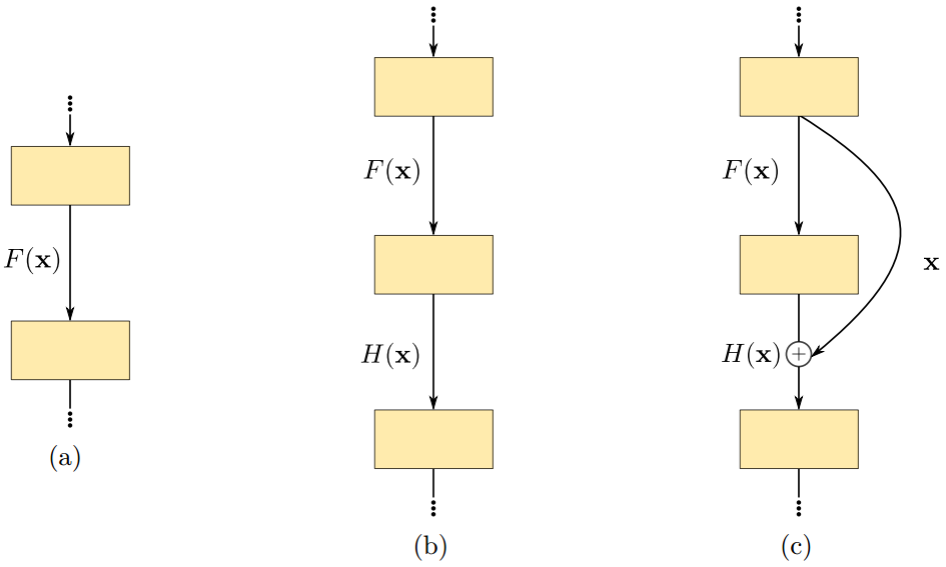


Fig. 2.15 Residual networks. (a) A convolutional network, where the mapping between the final two layers is $F(x)$, (b) learning an additional layer with the mapping $H(x)$, and (c) learning an additional residual layer with the mapping $H(x)+x$. Adapted from [138].

2.2.5 Residual Networks

He et al. [107] introduced residual networks, which provide an important insight on a problem with the training of very deep networks. While deeper networks have been found to improve generalization, especially with large datasets; at a sufficiently large depth training becomes difficult, even with batch normalization and the correct initialization, and generalization begins to level off, or even decline.

The important insight of He et al. [107] into this problem can be summarized in Fig. 2.15. Having trained a deep network with good generalization (i.e., Fig. 2.15(a)), with N layers, a training loss of L_1 is observed. On adding a single-layer to the otherwise identical deep network architecture (i.e., Fig. 2.15(b)), and re-training from random initialization, the new training loss of the network with $N+1$ layers is found to be $L_2 > L_1$, i.e., the training loss has increased. From an optimization standpoint, it is not clear why this should be so. We can observe that there is a trivial set of parameters defining a transformation that will maintain the training loss of the shallower network, i.e., $L_2 = L_1$ that is the identity transformation $H(x) = x$.

He et al. [107] proposed that to aid the optimization, a residual connection (as in Fig. 2.15(c)) is added to the convolutional layers, allowing the trivial identity solution to be easily learned. This residual connection can be thought of as a shortcut,

bypassing the previous layer. Assuming our desired, but difficult to optimize, mapping from one layer to the next is $H(x)$, the residual function learned is simply:

$$H(x) + x \tag{2.18}$$

In practice, these residual layers greatly help the training of very deep networks and have pushed state-of-the-art accuracy in many datasets. All current state-of-the-art models for image classification use residual layers.

This class of architectures achieved the state-of-the-art performance for both the object recognition [107] and semantic segmentation [192] tasks (in 2016). However, due to the limited space, the performance of these networks is only discussed for the ImageNet object recognition task.

AlexNet [96] is the pioneering deep convolutional network that won the ImageNet (ILSVRC) competition in 2012 with a TOP-5 test accuracy of 84.6% while the closest competitor, which made use of traditional techniques instead of deep architectures, achieved a 73.8% accuracy in the same challenge. VGG [36] is a CNN model that was introduced by the Visual Geometry Group (VGG) from the University of Oxford. This model is composed of 16 layers and achieved TOP-5 test accuracy of 92.7% on the ImageNet competition. Finally, Microsoft introduced the ResNet [107] architecture that is composed of 152 layers. This network presented the residual blocks which address the problem of training a deep architecture by introducing identity skip connections so that the layers can copy their inputs to the next layer. This network defeated all the other networks in the ImageNet competition with an accuracy of 96.4% in 2016.

2.2.6 Restricted Boltzmann Machine

Restricted Boltzmann Machine or RBM [24] is a generative stochastic artificial neural network that is used to learn a probability distribution over the input images. These RBMs can be stacked together to learn hierarchical features in an unsupervised manner.

The RBM model the probability distributions on high-dimensional binary vectors $v \in \{0,1\}^D$ that can model distributions of exciting features in the training data using symmetrically connected hidden stochastic binary units $h \in \{0,1\}^H$, as shown in Fig. 2.16(a). The hidden units act as latent variables (features) that allow the Boltzmann machine to model distributions over visible state vectors that cannot be modeled by direct pairwise interactions between the visible units. The input vector layer v is connected to the hidden layer h using weights w as shown in Fig. 2.16(a). The aim of

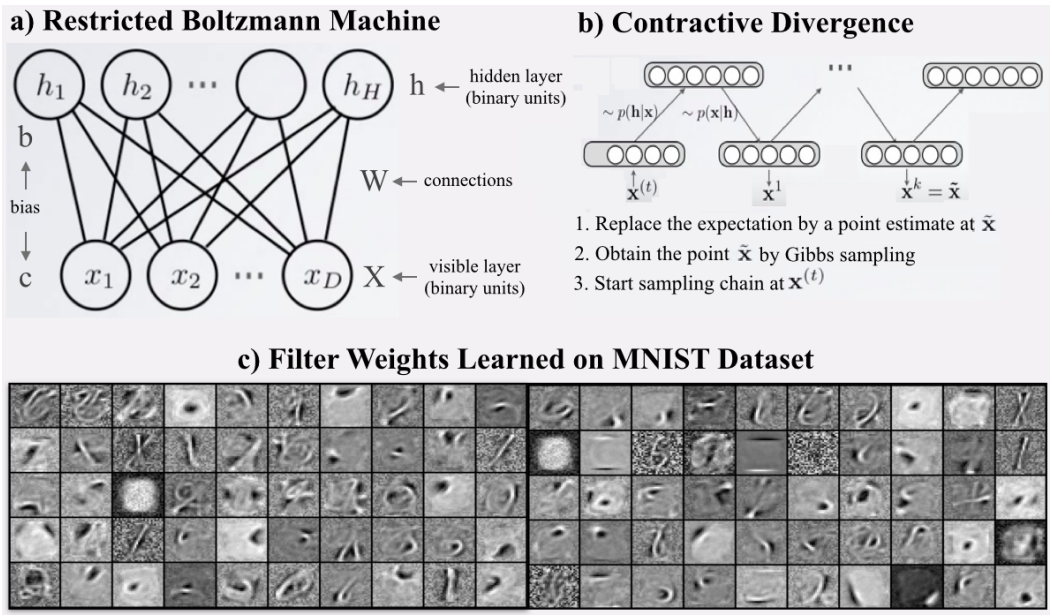


Fig. 2.16 Illustration of: a) The graphical model of Restricted Boltzmann as a fully-connected bipartite graph. b) The contractive algorithm used to train an RBM with three visible units and two hidden units. c) Visualization of the second layer bases (filters) learned from MNIST dataset [17].

the algorithm is to learn the optimal weights w that can regenerate the training data vectors with the highest probability.

The binary output z_i of a hidden unit i is given by the sum of its own bias, b_i , and the weights on connections coming from other active units:

$$z_i = b_i + \sum_j s_j w_{ij} \tag{2.19}$$

where w_{ij} is the weight on the connection between i and j , and s_j is 1 if unit j is on and 0 otherwise. Unit i then turns on with a probability given by the logistic function:

$$prob(s_i = 1) = \frac{1}{1 + e^{-z_i}} \tag{2.20}$$

The network eventually reaches a Boltzmann distribution when updated sequentially in any order that does not depend on their total inputs. The probability of a state vector, v, is determined solely by the "energy" of that state vector relative to the energies of all possible binary state vectors:

$$P(v; w) = \frac{e^{-E(v;w)}}{\sum_u e^{-E(u;w)}} \tag{2.21}$$

Given a training set of state vectors (the data), the aim is to find weights and biases that define a Boltzmann distribution in which the training vectors have high probability. This is done by minimizing the energy of the model given as:

$$E(v,h;w) = -v^{\intercal}wh - c^{\intercal}v - b^{\intercal}h \tag{2.22}$$

By differentiating Eq. 2.22 and using the fact that $\frac{\partial E(v)}{\partial w_{ij}} = -s_i^v s_j^v$ it can be shown that

$$\left\langle \frac{\partial logP(v)}{\partial w_{ij}} \right\rangle_{data} = \langle s_i s_i \rangle_{data} - \langle s_i s_j \rangle_{model} \tag{2.23}$$

where $\langle . \rangle_{data}$ is an expected value in the data distribution and $\langle . \rangle_{model}$ is an expected value when the Boltzmann machine is sampling state vectors from its equilibrium distribution at a temperature of 1. Exact maximum likelihood learning in this model is intractable because the exact computation of both data-dependent expectations and the model's expectations takes a time that is exponential in the number of hidden terms. The expectations are computed using Contractive Divergence that uses Gibbs sampling to approximate both expectations. For each iteration, a separate Markov chain is run for each training data vector to approximate $\langle s_i s_j \rangle_{data}$ and a separate Markov chain is run to approximate $\langle s_i s_j \rangle_{model}$ as shown in Fig. 2.16(b). The w_{ij} parameters are updated using.

$$\Delta w_{ij} = \alpha \left(\langle s_i s_j \rangle_{data} - \langle s_i s_j \rangle_{model} \right) \tag{2.24}$$

$$W_{ij}^{new} = W_{ij} + \Delta w_{ij} \tag{2.25}$$

while the learning rule for the bias, b_i, is the same as Eq. 2.25, but with s_j omitted. An example of weights learned by performing training for 15 iterations on the MNIST dataset [17] is shown in Fig. 2.16(c).

The unsupervised learning systems proposed over the years to solve the image classification task on the ImageNet dataset have only attained a marginal accuracy of around 65% [142]. However, the ability to learn hierachies features by using purely unsupervised examples is of great interest to researchers due to the limited availbility of large labelled datasets for many applications.

The next section presents the hybrid networks that are constructed by borrowing ideas from the hand-crafted, end-to-end supervised and unsupervised categories.

2.3 Hybrid Architectures

Hybrid networks are single or multi layer architectures that combine the ideas from both the above-explained approaches to obtain the low-level features using hand-crafted filters while learning the filters at the later layers to obtain more complex features. Many hybrid networks have been proposed in the past including the Bag of Features [33], HMAX model [40], hierarchical LDA [39], deep sparse coding [31], etc.

Bag of Words (BoW) [33] represent the first class of single layer architectures that encode handcrafted bag-of-visual-words (BOV) descriptors into rich feature representations using unsupervised coding and pooling [32]. The pipeline was improved by encoding the local descriptor patches by a set of visual codewords with sparse coding with a linear SPM kernel [117]. Hierarchical max (HMAX) [40] are the multi-layer hybrid networks that use an RBF kernel to learn a single layer of high-level features from descriptors captured with a battery of Gabor filters. He et al. [31] learned three layers of sparse hierarchical features from SIFT descriptors using unsupervised learning. Sivic et al. [39], discovered object class hierarchies from visual codewords using hierarchical LDA.

These models have several advantages over the hand-crafted and end-to-end learned networks. The hybrid networks can learn useful hierarchical features using mainly unlabelled data. Also, each layer of these models can be posed as an optimization problem resulting in computationally efficient architectures [31]. This can be very useful for applications with limited labelled training examples and for areas where abundant computational resources are not available.

A few architectures from this category that extracts single layer (Bag of features model [8]) and multi-layer hierarchical (DeepSC network [31] and HMAX network [40]) features are detailed below.

2.3.1 Bag of Features

Bag of features (BoF) model [8] has been a popular model that is frequently used for object recognition. The BOF is extracted from an image first by selecting two-dimensional interest points using the Harris or SIFT key point detector. After feature detection, each image is abstracted by several local patches. Next, SIFT descriptor is extracted for each key point due to its ability to handle the intensity, rotation, scale and affine variations to some extent. The final step for the BoF model is to convert vector represented patches to 'codewords', which also produces a 'codebook'. A codeword can be considered as a representative of several similar patches. One simple method is

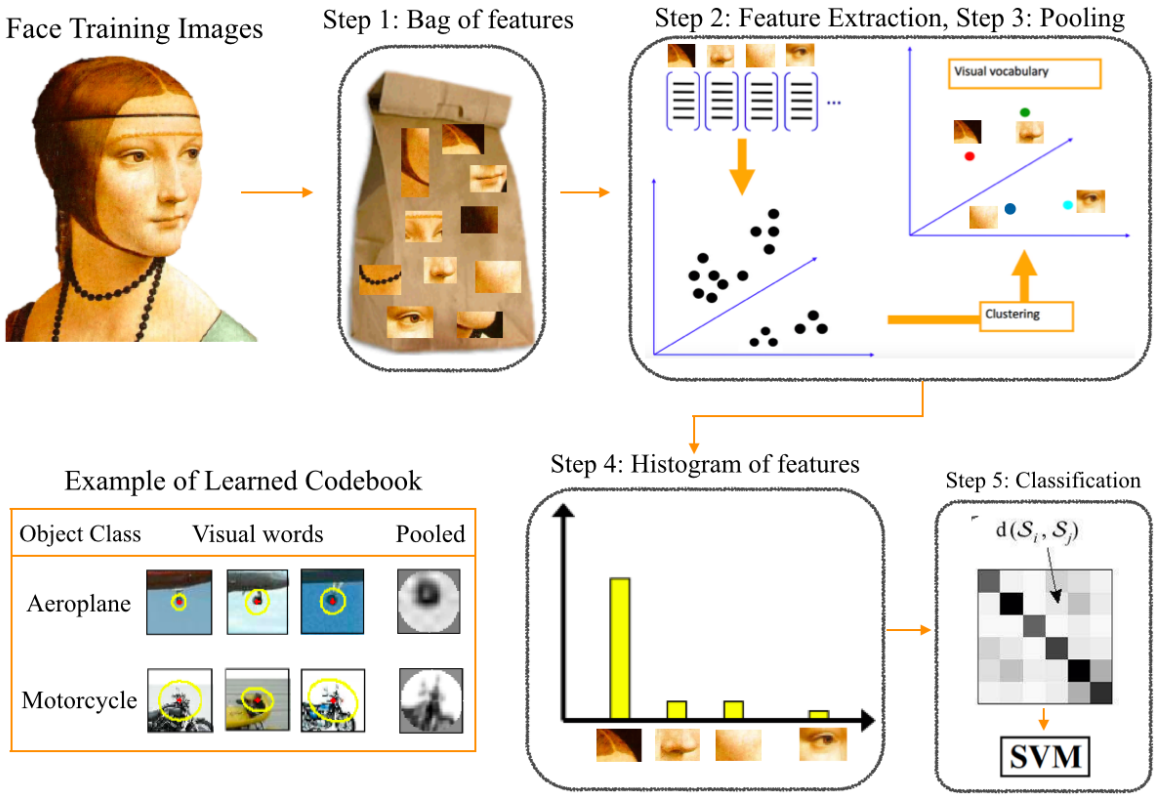


Fig. 2.17 Illustration of different steps involved in the bag of features model. Step 1: The features are extracted from 2-D image patches. Step: 2, 3 A visual codebook is learned by clustering features and further replacing each cluster with a pooled value. An example of the learned codebook is also presented. Step 4: At test time, each feature extracted from the input test image patches are replaced by the most similar codebook element. This steps reduced the design variance as mentioned in Section. 1.1. Step 5: The feature vector generated is further given to the Support vector machine for classification. Adapted from [42]

performing k-means clustering over all the vectors [23]. Codewords are then defined as the centers of the learned clusters. The number of the clusters is the codebook size. Thus, each patch in an image is mapped to a certain codeword through the clustering process, and the histogram of the codewords can represent the image. The whole process is described in Fig. 2.17. This model can eliminate exterior as well as internal variabilities but still is unable to create discrimination between the object and background.

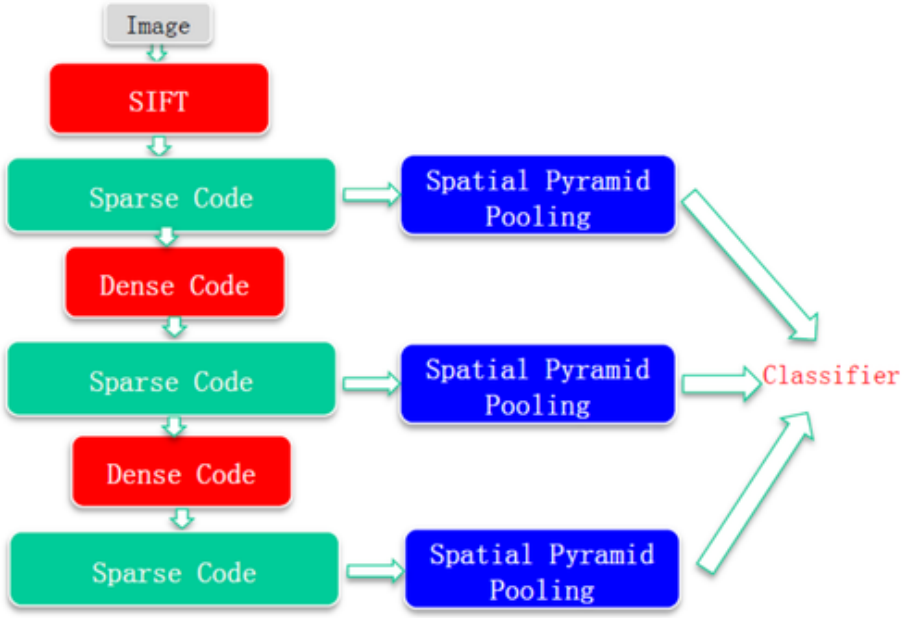


Fig. 2.18 The illustration shows the three-layer deep sparse coding framework. Each of the three layers contains three modules. The first module converts the input (image patches at the first layer and sparse codes at other layers) to dense codes. The second module is a sparse encoder converting the dense codes to sparse codes. The sparse codes are then sent to the next layer, and simultaneously to a spatial pyramid pooling module. The outputs of the spatial pyramid pooling modules can be used for further tasks such as classification. Adapted from [31].

2.3.2 Deep Sparse Coding

This section presents the deep sparse coding (DeepSC) network [31] that learns a hierarchy of rich feature representations using unsupervised learning to achieve better performance on object recognition tasks. The DeepSc network combines the rich bag of features representations with multiple layers of unsupervised sparse coding to learn the feature hierarchies.

The first layer of the DeepSC framework is the same as the bag-of-visual-features as explained in the Section 2.3.1. The subsequent layers of the framework use sparse-to-dense module which converts the sparse codes obtained from the last layer to dense codes, which is then followed by a sparse coding module. The sparse output code of the sparse coding module is the input of the next layer. The sparse-to-dense module is the key innovation of the DeepSC framework which uses a pooling function which is the composition of a local spatial pooling step and a low-dimensional embedding step [31]. Furthermore, the spatial pyramid pooling step is conducted at every layer

such that the sparse codes of current layer are converted to a single feature vector for that layer. Finally, the feature vectors from all layers are concatenated and given as the input to the classifier. We summarize the DeepSC framework in Fig. 2.18.

The proposed network learns feature hierarchies purely from unlabelled examples. Also, each module of the architecture has a sound explanation and can be formulated as explicit optimization problems with promising computational performance.

2.3.3 HMAX Model

Hierarchical Max or HMAX model [40] presents an object recognition architecture inspired by the general organization of visual cortex in a series of layers from primary visual cortex (V1) [5] to inferior temporal cortex (IT) [6] to prefrontal cortex (PFC) [7], as detailed in section 1.2. The model achieves an excellent trade-off between invariance and selectivity at the level of shape-tuned (S) and invariant cells (C) from which many recognition tasks can be readily accomplished. The model uses the two key modules to extracts visual features with the properties as mentioned earlier.

The first module extracts position and scale invariant features using simple units (S1) tuned in a Gaussian-like way to a bar of a particular orientation. Next, pooling is performed by complex cells (C1) over a group of simple cells (S1) through a maximum operation (max). The pooling operation performed at the same preferred orientation and position in space but at slightly different spatial frequency leads to scale invariance as shown in Fig. 2.19(b). However, pooling over a group of simple cells at the same preferred orientation and spatial frequency but at a slightly different position in space would provide position invariance as shown in Fig. 2.19(a).

The second module learns selectivity to more complex patterns such as combinations of oriented lines with different selectivities by pooling the activities of several complexes (C1) using weights obtained using an unsupervised learning algorithm. Thus learning at the S2 and the C2 level is efficiently learning correlations present in the training data.The connectivity of complex units arises from learning associations over time, e.g., that simple unit with the same orientation and nearby locations should be wired together in a complex group because often such a pattern changes smoothly in time (e.g., under translation).

The model alternates layers of units combining simple filters into more complex ones to increase pattern selectivity followed by a max operation to build invariance to position and scale as shown in Fig. 2.20. The layers of the network are explained in detail below.

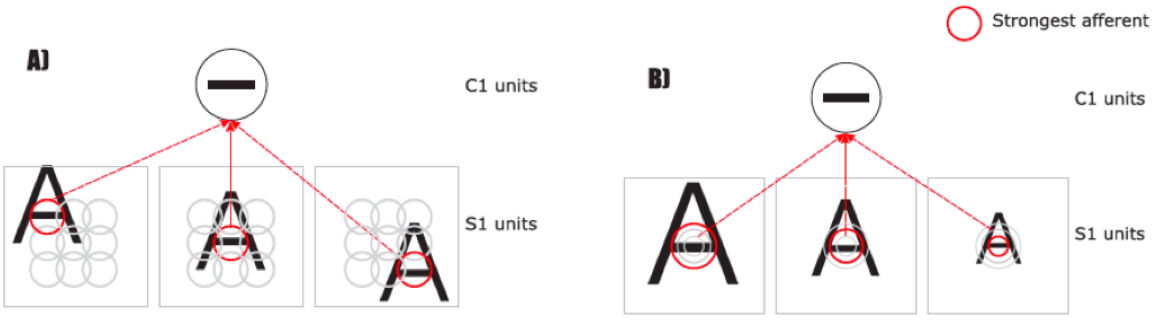


Fig. 2.19 Illustration of schematic that shows how size and shift tolerances are increased at the (C1) level: A complex (C1) cell pools over S1 cells at the same orientation, however, (left) centered at different location thus providing some translation invariance and (right) at different scales providing some scale invariance to the complex cell. Reproduced from [25]

.

- **S1 Layer:** In the model, the input image is first densely sampled by arrays of two-dimensional Gaussian filters, called S1 units given by

$$F(x,y) = exp\left(-\frac{(x_0^2+\gamma^2 y_0^2)}{2\sigma^2}\right) \times cos\left(\frac{2\pi}{\lambda}x_0\right) \quad st.$$
$$x_0 = xcos\theta + ysin\theta; \quad y_0 = -xsin\theta + ycos\theta \tag{2.26}$$

  For the filters, γ represents the aspect ratio, θ is the orientation, and σ decides the effective width. Further, the wavelength is chosen with the parameter λ while the size of the filter is selected by s. The filters respond to bars of different orientations, thus roughly resembling properties of simple cells in the striate cortex.

- **C1 Layer:** The next, C1, stage corresponds to cortical complex cells which show some tolerance to shift and size. C1 units pool over the S1 units from the previous layer with the same orientation and the same scale band. This pooling increases the tolerance to 2D transformations from layer S1 to C1. The corresponding pooling operation is a *max* operation. That is, the response r of a complex unit corresponds to the response of the strongest of its m afferents $x_1, ..., x_m$ from the previous S1 layer such that:

$$r = \max_{j=1,....,m} x_j \tag{2.27}$$

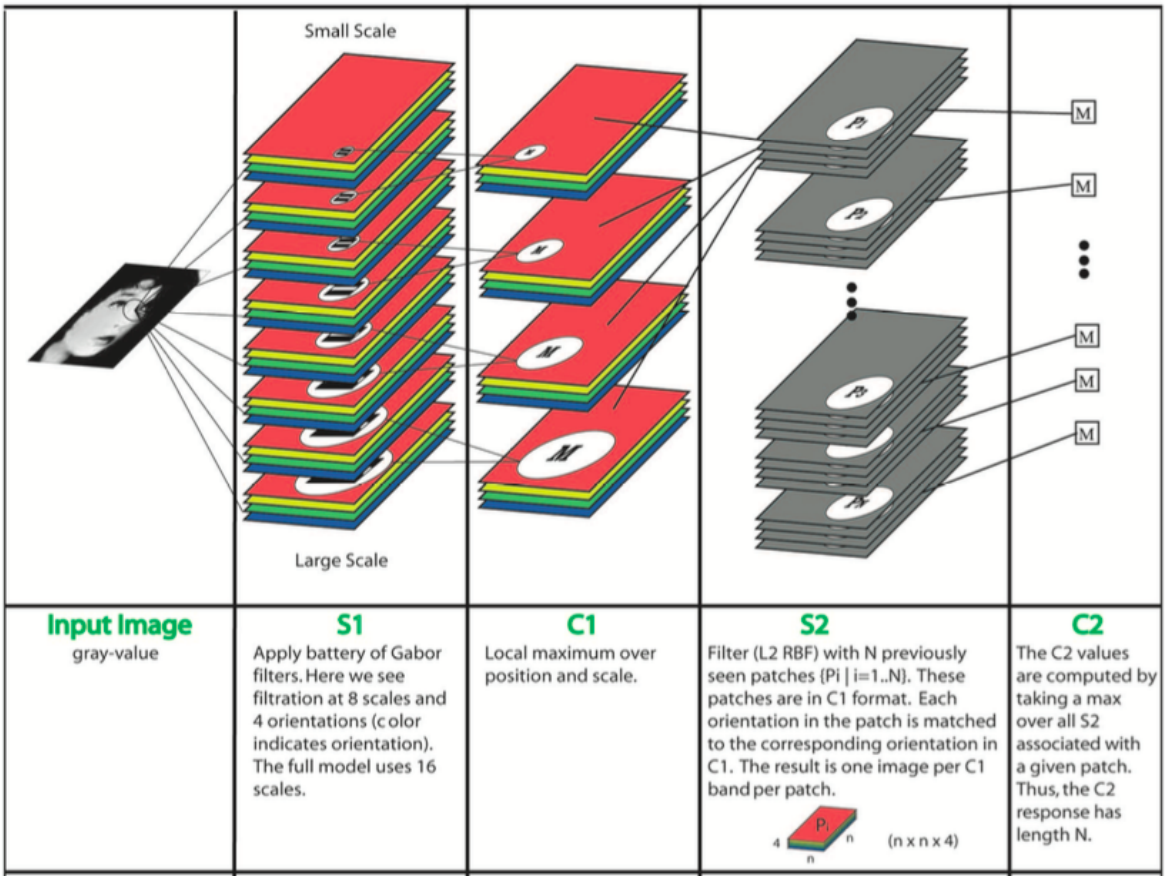


Fig. 2.20 System overview: An array of S1 units first analyzes a gray-value input image at four different orientations and 16 scales. At the next C1 layer, the image is subsampled through a local MAX pooling operation over a neighborhood of S1 units in both space and scale, but with the same preferred orientation. In the next stage, S2 units are essentially RBF units, each having a differently preferred stimulus. Note that S2 units are tiled across all positions and scales. A MAX pooling operation is performed over S2 units with the same selectivity to yield the C2 unit responses. Adapted from [25]

From each grid cell, one single measurement is obtained by taking the maximum of all the afferents. As the last stage, we take a *max* over the two scales from within the same spatial neighborhood, by recording only the maximum value from the two maps.

- **S2 Layer:** In the $S2$ layer, units pool over afferent $C1$ units from a local spatial neighborhood across four orientations. Each $S2$ unit response depends in a Gaussian-like way on the Euclidean distance between a new input and a stored prototype. A set of N prototypes P_i (or features) for the $S2$ units are learned

in an unsupervised way from a target set of images using a simple sampling process. This is roughly equivalent to learning a dictionary of patterns that appear with high probability. For an image patch, X from the previous $C1$ layer at a particular scale S, the response r of the corresponding $S2$ unit is given by:

$$r = exp(-\beta \left\| X - P_i \right\|^2) \tag{2.28}$$

where β defines the sharpness of the TUNING and P_i is one of the N features learned during training.

- **C2 Layer:** Our final set of shift and scale-invariant $C2$ responses is computed by taking a global maximum max operation over all scales and positions for each $S2$ type over the entire $S2$ lattice, i.e., the $S2$ measures the match between a stored prototype P_i and the input image at every position and scale; we only keep the value of the best match and discard the rest. The result is a vector of N $C2$ values, where N corresponds to the number of prototypes extracted during the learning stage.

This is an efficient model as it is capable of eliminating external variabilities and create little discrimination between the object and background by connecting the edges of the object using S2 (all the edges are still in the same manifold). The model is even to an extent able to eliminate internal variabilities.

This class of models has produced promising performance (around 80%) on various classification tasks [41, 42]. These models form the basis for the SHDL framework proposed in this thesis due to the ability of these networks to learn hierarchical features mainly from unlabelled examples in a computationally efficient way. The few labeled training examples used by these models to solve the classification task was also a significant motivator on which to base the proposed SHDL framework.

Chapter 3

ScatterNet Hybrid Framework for Deep Learning

Deep Convolutional Neural Networks (DCNNs) were first proposed by Yann LeCun et al. [17] that cascaded filter banks with pooling and nonlinearities to learn hierarchical representations. The success of these models was limited until 2006 due to the lack of available training sets and the shallower depth of the networks. The breakthrough by Krizhevsky et al. [96] was made due to the supervised training of a vast network with eight layers and millions of parameters on the ImageNet dataset with 1 million training images. Since then, even larger and deeper networks have been trained [35, 36]. These networks have now made significant advances at numerous classification [96] and regression [102] tasks in computer vision and speech applications. In addition to their superior performance, these models are good candidates for models of the visual ventral pathways as the hierarchical representations learned by them resemble the representations learned by different regions of the visual cortex [157], as shown in Fig. 3.1. Therefore, these models have been used to make strides in modeling neural single-unit and population responses in higher visual cortical areas [157].

These models produce state-of-the-art results only for applications with sizeable labeled training datasets and tend to overfit [97] on many other applications where large datasets are not available such as the analysis of hyperspectral images [99], stock market prediction [37], medical data analysis [38] etc. Obtaining labelled data for most real-world applications is often difficult, expensive, and time-consuming, as tagging the training data may require human annotators with specific domain experience and training. Two methods have been suggested to aid the learning of these models for applications with small training datasets: (i) Data augmentation and synthetic data generation, and (ii) Transfer Learning. Training CNNs on synthetic datasets may

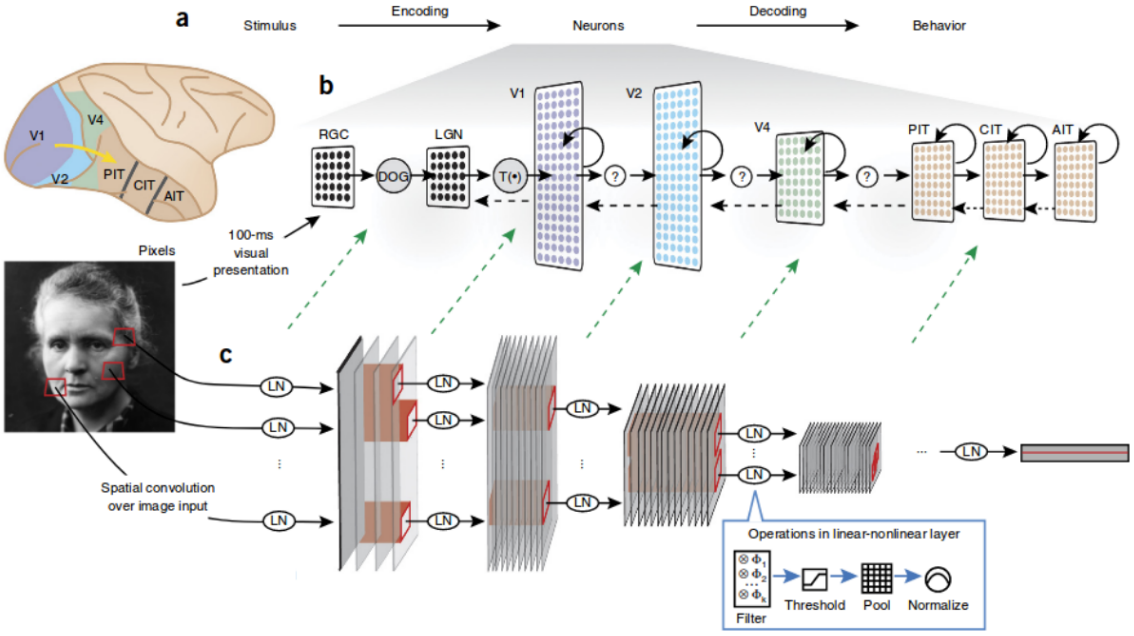


Fig. 3.1 Deep convolutional neural networks (DCNNs) as models of sensory cortex. (a) The basic framework in which sensory cortex is studied is one of encoding—the process by which stimuli are transformed into patterns of neural activity—and decoding, the process by which neural activity generates behavior. DCNNs have been used to make models of the encoding step; that is, they describe the mapping of stimuli to neural responses as measured in the brain. (b) The ventral visual pathway is the most comprehensively studied sensory cascade. It consists of a series of connected cortical brain areas (macaque brain shown). PIT, posterior inferior temporal cortex; CIT, central; AIT, anterior; RGC, retinal ganglion cell; LGN, lateral geniculate nucleus. DoG, difference of Gaussians model; T(•), transformation. (c) DCNNs are multilayer neural networks, each of whose layers are made up of a linear-nonlinear (LN) combination of simple operations such as filtering, thresholding, pooling, and normalization. The filter bank in each layer consists of a set of weights analogous to synaptic strengths. Each filter in the filter bank corresponds to a distinct template, analogous to Gabor wavelets with different frequencies and orientations; the image shows a model with four filters in layer 1, eight in layer 2, and so on. The operations within a layer are applied locally to spatial patches within the input, corresponding to simple, limited-size receptive fields (red boxes). The composition of multiple layers leads to a complex nonlinear transform of the original input stimulus. At each layer, retinopy decreases and effective receptive field size increases. DCNNs are therefore good candidates for models of the ventral visual pathway. Reproduced from [157].

not learn potentially useful patterns of real data as often the feature distribution of synthetically generated data shifts away from the real data [106]. On the other hand, transfer Learning aims to extract the knowledge from one or more source tasks and

applies the experience to a target task. The weights of the CNN are initialized with those from a network trained for related tasks before fine-tuning them using the target dataset [100]. These networks have been quite successful and generalize well to new categories [101]. However, they fail to generalize to new domains which are significantly different from the source [156].

Early Deep Learning architectures such as NIN [35] and VGG [36] net only used convolution, fully connected, or pooling operations but still provided large improvements over classical vision approaches. Recent advances in the field have improved performance further by using several new and more complex building blocks that involve operations such as branching (e.g. inception [108], ResNeXt [109] blocks) and skip connections (e.g. residual [107], ResNeXt [109]). Despite the superior performance of these networks, the design and optimal configuration of these networks are not well understood which it makes difficult to develop these networks. Large over-expressive non-optimal network architectures require optimization of several redundant hyperparameters which can lead to computational and memory intensive networks. Training these large networks also requires considerable energy to fetch the weights along with numerous dot products computations [110].

Evolutionary [111, 112] and reinforcement learning [113, 114] algorithms have been proposed over the past few years that aim to automate the architecture discovery process. The objective of both methods is to search for a particular building block which is then repeated many times to create the deep architecture. This increase in the search space means that in addition to finding the optimal traditional Deep network hyperparameters such as layer size and the number of filters, training a model now includes searching for the various combinations involved in constructing a useful building block. This increased complexity corresponds to increased training time and often means that the process of finding the right architecture or configuration remains the result of an extensive search [110].

Despite the proposed techniques to design optimal deep networks and reduce their dependence on sizeable labeled training datasets, the current deep networks still mostly require numerous labeled training examples along with considerable computational resources and time to achieve effective learning. The next section introduces the ScatterNet Hybrid Deep Learning (SHDL) framework which can be used to construct networks which don't suffer from the above-mentioned disadvantages.

Fig. 3.2 ScatterNet Hybrid Deep Learning (SHDL) framework inspired by the circuitry of the visual pathway is presented. The front-end of the SHDL network is a hand-crafted ScatterNet similar to the V1 of the pathway, which decomposes the input signal into features at different scales and orientations using complex wavelets [119, 155]. The mid-section then uses unsupervised learning on these hand-crafted features to rapidly encode invariant hierarchal feature similar to what is believed to be the functions of V2 and V4 regions [149–152]. Finally, the back-end of the framework uses supervised learning to assign class specific labels to the features obtained from the last layer of the unsupervised module using few labeled examples [120, 148].

3.1 ScatterNet Hybrid Deep Learning Framework

This section proposes the ScatterNet Hybrid Deep Learning (SHDL) framework that is inspired by the pathways of the visual cortex. This framework can be used to design SHDL networks that learn hierarchical representations that resemble the features learned by different cortical areas of the ventral visual pathways [153, 154, 157]. The SHDL framework is motivated from the visual cortex similarly to the Deep Convolutional Neural Networks (DCNNs). However, the proposed framework (unlike the DCNNs) is able to rapidly learn meaningful representations mainly from unlabelled data, in a computationally efficient manner.

The SHDL framework is composed of a hand-crafted front-end that extracts low-level invariant features, an unsupervised learning based middle-section that extracts hierarchical features, and a supervised back-end that assigns labels to the features, as shown in Fig. 3.2. The handcrafted front-end captures low-level features similar to that of the primary visual cortex as shown by the groundbreaking work of Hubel and Wiesel [119] along with Blakemore and Cooper [155] in cats. The unsupervised middle section captures hierarchical representations which are similar to the neurons along the ventral stream in the extrastriate visual areas of V2 and V4. These neurons show an increase in receptive field size as well as in the complexity of their preferred stimuli [40]. The V2 and V4 visual regions are believed to encode a hierarchy of increasingly complex features rapidly with unsupervised learning [149–152]. Finally, the inferior temporal (IT), cells are tuned to complex object specific stimuli such as faces [120, 148] similar to the supervised back-end of the SHDL framework. Each

layer of the SHDL network is created and automatically optimized to produce the desired computationally efficient architectures. The term 'Hybrid' is coined because the framework uses both unsupervised as well as supervised learning.

3.1.1 ScatterNet Front-end

The SHDL front-end module is required to decompose the input image into local low-level edge feature representations using a hand-crafted network. Several single and multi, layer networks have been proposed to extract these feature representations as detailed in section 2.2.

Mallat and his collaborators [22, 27, 28, 86] has shown that ScatterNets incorporate geometric knowledge of images to produce discriminative and locally invariant (translation and rotation) representations which can give superior performance to other hand-crafted network and accuracy comparable to trained networks. The invariants at the first layer of the system are obtained by filtering the image with multi-scale and multi-directional complex Morlet wavelets followed by a point-wise nonlinearity and local smoothing. The high frequencies lost due to smoothing are recovered at the later layers using cascaded wavelet transformations, justifying the need for a multilayer network.

Chapter 4 proposes two improved versions of Mallat's scatternets. These enhanced networks have also been shown to improve the performance and learning of current Deep Supervised Learning Networks (VGG, NIN, ResNet) in chapter 5.

3.1.2 Unsupervised Learning Mid-section Module

The neurons in the V2 and V4 regions of the visual cortex are thought to learn high-level features rapidly with minimal supervision [135]. The mid-section of the SHDL framework is designed on the same principle as it applies unsupervised learning to discover exciting structure from the input hand-crafted feature representations. This is particularly useful as the labeled data can be expensive to obtain while unlabelled data is cheap and available in abundance. If the unsupervised algorithms can exploit this unlabelled data efficiently, then it might be possible to achieve better performance using these unsupervised models than the pure hand-engineered or hand-labeling approaches.

Many deterministic and generative unsupervised learning approaches have been proposed over the years detailed in section 2.3.5. These techniques have been successfully used to achieve moderately good image understanding performance.

In chapter 6, a deterministic and a generative, unsupervised technique are used to construct two proposed SHDL networks which can learn invariant hierarchical mid-level features rapidly from unlabelled data.

3.1.3 Supervised Learning Back-end

The supervised backend of the SHDL network assigns labels to the features obtained from the unsupervised mid-section similar to the IT region of the visual cortex [148]. In this thesis, we used the orthogonal least squares (OLS) method, detailed in section 2.2.2 to select the object class specific features which are then used by the SVM to perform object classification. A conditional random field is used to process the features extracted by the last layer of the unsupervised learning module to achieve semantic image segmentation. This is shown by the two proposed SHDL networks presented in chapter 6.

Next, the proposed hand-crafted networks are presented in chapter 4, the deep learning networks improved with the hand-crafted networks in chapter 5, and the two SHDL networks with unsupervised learning modules which can learn an invariant hierarchy of features rapidly from unlabelled hand-crafted feature representations, in chapter 6.

Chapter 4

Hand-crafted Front-end

This chapter introduces the front-ends of the of the ScatterNet Hybrid Deep Learning (SHDL) networks. These front-ends correspond to the V1 of the visual cortex as detailed by the experiments of Hubel and Wiesel [119] along with Blakemore and Cooper [155] in cats and should capture low-level edge representations. Therefore the proposed front-ends are composed of architectures (shallower that standard deep architectures) which extract the desired descriptors. The proposed hand-crafted networks are the improved versions of the scattering networks proposed by Mallat and his colleagues [20, 26–28, 86]. The performance of the features extracted by these networks is evaluated on numerous classification tasks using the Support Vector Machine (SVM) [87]. The SVM creates discrimination between different signal classes by learning weights that best summarize the regularities (common coefficients) in the training data and simultaneously ignore the coefficients arising due to the irregularities.

For both the networks, the input signal is first obtained at multi-resolution signal representations which are then decomposed with the Dual-Tree Complex Wavelet Transform (DTCWT) [15] to extract the feature representations that are dense over the scale. The extracted feature representations are further improved by the proposed networks by introducing the property of translational invariance, contrast normalization, and symmetry to the distribution, of the feature representation.

The multi-resolution signal representations and DTCWT filtering operations are first explained after which both the hand-crafted networks are presented.

The primary reasons for both the operations are explained below:

- *Multi-scale Input Signal*: The input signal is decomposed into representations that are densely spaced on the scale using a DTCWT based decimated pyramid of complex values [53]. The dense representations allow the classifier to learn additional discriminatory features which can aid the classification.

The multi-resolution representation of the signal x is obtained at interleaved scale (s) [53] using a DTCWT based decimated pyramid of complex values as shown below:

$$[x, x'] = decimated(x) \tag{4.1}$$

where x is the input signal while x' is the rescaled signal by approx $\sqrt{0.5}$.

- *Multi-Resolution DTCWT Filtering*: The proposed network uses the DTCWT bank for filtering as opposed to Morlet wavelets [86] due to its perfect reconstruction and computational efficiency [15]. Perfect reconstruction property allows the DTCWT filter to extract features without any aliasing. They provide similar rich features to Morlet wavelets but with less computation and somewhat lower redundancy in the output vectors. Besides, dual-tree wavelets can be efficiently implemented in the spatial domain, rather than requiring the complexities and constraints of Fourier domain filtering.

  The DTCWT filter bank is applied at each resolution (x, x') to extract the feature representations. For the sake of simplicity, the derivation is presented only for the signal x.

  The filtering is realized by arranging the DTCWT filters with numerous scales (and orientations (for only 2D signal)) to extract stable and informative signal representations. DTCWT wavelets are represented by ψ (with a real ψ^a and imaginary ψ^b group). The complex band-pass filter ψ is decomposed into real and imaginary groups as shown:

$$\psi(t) = \psi^a(t) + \iota\psi^b(t) \quad t = (t_1, t_2) \tag{4.2}$$

  The signal x is filtered using a family of DTCWT wavelets $\psi_{\lambda_1}(t)$ at different scales and orientations (λ_1), formulated as:

$$x \star \psi_\lambda(t) = x \star \psi^a_\lambda(t) + \iota x \star \psi^b_\lambda(t) \tag{4.3}$$

These convolutions are implemented efficiently via standard wavelet multi-resolution filtering techniques [136].

4.1 Multi-Resolution Region Pooling ScatterNet

This section presents the first out of the two proposed hand-crafted scattering network, coined as, Multi-Resolution Region Pooling ScatterNet, that decomposes the input signal with DTCWT filters to extract locally translation invariant multi-resolution feature representations. Translation invariance is next introduced within each feature representation by applying a non-linearity over a region followed by local averaging. The region non-linearity selects the dominant feature within the region while simultaneously suppressing features with lower magnitudes leading to invariance similar to the max operator in CNNs. Finally, a log non-linearity is used to separate the multiplicative low-frequency illumination components within the representations.

The formulation of the two-layer Multi-Resolution Region Pooling ScatterNet is presented for a two-dimensional image signal, as shown in Fig. 4.1. However, this formulation can be easily extended to process signals of other dimensions and design ScatterNets with deeper layers.

The input image x is first decimated into different multi-resolution images which are then filtered using the dual-tree complex wavelets as explained in the previous section. The filtered feature representations extracted using the DTCWT transform commute with translations and are therefore not translation invariant. To build a translation invariant representation, a $L2$ smooth non-linearity is first applied over all overlapping regions of size R ($R \times R$ for 2D signal) in feature representations, obtained at a particular scale (and six orientation (θ) (for 2D signal)). The non-linearity applied to one of the above-mentioned regions is shown below:

$$|x \star \psi_{\lambda_1}|_R = \sqrt{|G_{real} \star \psi^a_{\lambda_1}(t)|^2 + |G_{imag} \star \psi^b_{\lambda_1}(t)|^2} \tag{4.4}$$

where R is the size of the region and G is a group of R ($R \times R$ for 2D signal) complex scattering coefficients. $L2$ is a non-expansive non-linearity that makes it stable to additive noise and deformations [86]. The region non-linearity selects the dominant feature in the region while simultaneously suppressing features with lower magnitudes. This creates translation invariance in a larger region similar to the max operator in CNNs. The scattering coefficients obtained after applying the region non-linearity to the outputs of every wavelet scales is given by $|x \star \psi_{\lambda_1}|_R$.

Next, the desired translation invariant representation are obtained at the first layer (L_1) by applying a local average on $|x \star \psi_{\lambda_1}|_R$, as shown below:

$$(L_1)_R = \left(|x \star \psi_{\lambda_1}|_R \right) \star \phi_{2^J} \tag{4.5}$$

The high frequencies coefficients lost by the averaging operator are recovered at the second layer (L_2) by calculating the wavelet coefficients of $|x \star \psi_{\lambda_1}|_R$ by the wavelet at scale and orientation, λ_2, given as $(|x \star \psi_{\lambda_1}|_R) \star \psi_{\lambda_2}(t)$ [86]. The features extracted at the first layer (L_1) are filtered with the DTCWT filter at coarser scales (λ_2) to recover the high frequency components at the second layer (L_2). The recovered frequencies are converted into translation invariant representations by again taking a local average over a larger locality as shown:

$$(L_2)_R = \left(|\Big(|x \star \psi_{\lambda_1}|_R \Big) \star \psi_{\lambda_2}|_R \right) \star \phi_{2^J} \tag{4.6}$$

The scattering coefficients $S_J x$ for the network at different scales and orientations for two layers can be obtained using the following:

$$S_J x[p] = \begin{pmatrix} x \star \phi_{2^J}(L0) \\ \left(|x \star \psi_{\lambda_1}|_R \right) \star \phi_{2^J}(L1) \\ \left(|\Big(|x \star \psi_{\lambda_1}|_R \Big) \star \psi_{\lambda_2}|_R \right) \star \phi_{2^J}(L2) \end{pmatrix}_{\lambda=(2,3,4)} \tag{4.7}$$

Next, the low (L0) and the high (L1, L2), frequency scattering features are concatenated. A logarithm non-linearity proposed by Oyallon et al. [22], is applied to the concatenated scattering coefficients to transform the low-frequency multiplicative components that arise due to illuminations into additive components. The logarithm applied to the scattering coefficients (Sx) extracted from a dataset with M training signals, where the coefficients computed from a single signal has N dimensions, is given by:

$$\Phi^{M \times N} = \log(Sx^{M \times N} + p) \tag{4.8}$$

where Sx are the scattering coefficients, and p is (small) constant added to reduce the effect of noise magnification at small signal levels. The transformation of multiplicative components to the additive components allows the classifier to ignore the coefficients arising due to illumination as noise. A p value of 1×10^{-6} is used for all the experiments

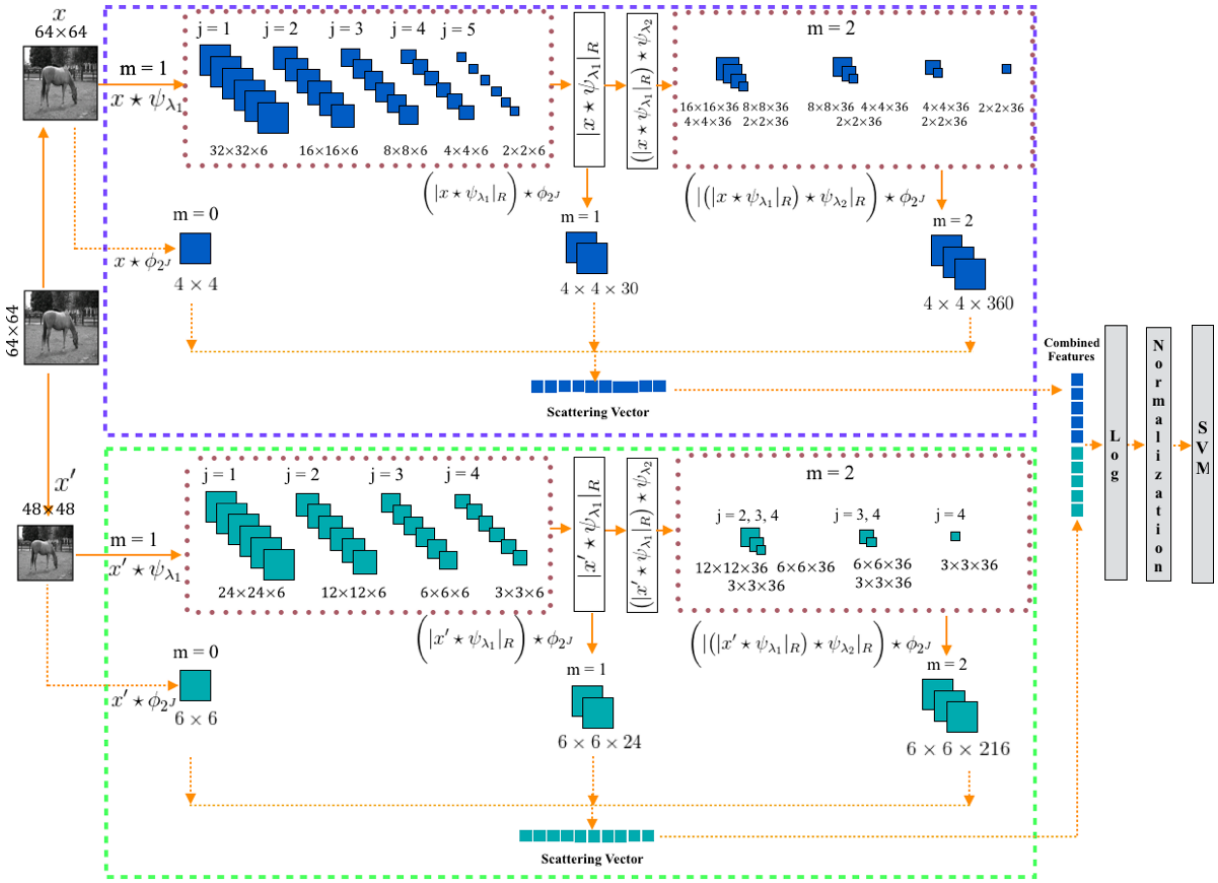


Fig. 4.1 Illustration shows the input image (x) of size 64×64 is first resized to images of resolution, x (64×64) and x' (48×48) respectively. Image representations at the first layer ($m = 1$) are obtained using dual-tree wavelets at different scales and 6 orientations. Then, L2 region non-linearity is applied on the representations to obtain the regular envelope followed by local averaging to extract the local translation invariant coefficients. Cascaded wavelet filtering recovers the information lost due to averaging at the second layer. Translation invariance is introduced in the recovered frequencies using L2 region non-linearity and local averaging. The high-frequency features extracted at both layers ($m = 1, 2$) and the low-frequency features at $m = 0$ are concatenated. A log non-linearity with a small constant ($p = 1 \times 10^{-6}$) is applied to the concatenated features to transform the low-frequency multiplicative components that arise due to illuminations into additive components. Next, the features are standardized before given as input to the Gaussian SVM for classification.

related to the proposed ScatterNet. $\Phi^{M \times N}$ obtained in Eq. 4.8 is similar to the metric defined in Eq. 1.1.

The features obtained after applying the log non-linearity are standardized before being given as input to the classifier. Standardization removes the scaling effects caused by the use of features with different amplitude measurement scales [137]. The

standardization step transforms the raw scattering features into z-scores using the mean and standard deviation of feature values over all the input samples, given by the relationship:

$$z_{ij} = \frac{x_{ij} - \mu_j}{\sigma_j} \tag{4.9}$$

where x_{ij} is the value for the ith sample and jth feature. μ_j is the average of all x_{ij} for feature j, σ_j is the standard deviation of all x_{ij} over all the input samples for the jth feature. The range and scale of the z-scores after standardization is similar, which can aid the learning of the classifier.

Support vector machine with the Gaussian kernel is used for classification. The optimization function of the SVM is parameterized with a cost parameter while the kernel contains the hyperparameter gamma.

The cost parameter of the SVM represents the price of misclassification on the training examples [140]. A large value of the parameter indicated a hefty price for misclassification which forces the optimization to choose a hyperplane with a smaller margin which should lead to lower classification error on the dataset. Such a model tends to overfits the training set and exhibits poor generalization performance. Conversely, a minimal value of the parameter will cause the optimizer to look for a larger-margin separating hyperplane, even if that hyperplane misclassifies more training example. This can result in numerous misclassified training examples, even if the data is linearly separable.

The gamma hyperparameter controls the margin that produces the classification [140]. For a substantial value of gamma, the SVM learns a model that captures the complexity and shape of the data. This can lead to overfitting of the training data and result in poor generalization. On the other hand, a small value for gamma implies the SVM learns a simpler model which is less prone to overfitting but risks not learning a decision boundary that appropriately captures the shape and complexity of the training dataset.

The value of both the parameters that produced the optimal classification is obtained using 5-fold cross-validation for all the experiments.

4.2 Overview of Results

Experiments were conducted on four real-world datasets selected from the image, audio, biology and material modalities to evaluate the performance of the proposed DTCWT multi-resolution scattering network. To test the generalization to different problems,

Table 4.1 Classification error (%) on different datasets for each part of the proposed network. DTCWT ScatNet: DSCAT, DTCWT ScatNet + Pooling: DSCATP, Multi-Resolution DTCWT ScatNet: MDSCAT, Multi-Resolution DTCWT ScatNet + Pooling: MDSCATP. The left result in / represent the without (left) and with log (right) non-linearity classification erros (No Log/Log).

Dataset	DSCAT	DSCATP	MDSCAT	MDSCATP
USPS	3.31 / 3.38	3.24 / 3.33	2.89 / 3.11	2.54 / **2.49**
Isolet	5.14 / 5.3	5.10 / 5.36	4.75 / 5.02	4.14 / **4.04**
Yeast	41.65 / 41.17	45.86 / 45.85	37.04 / **34.62**	39.77 / 39.04
Glass	31.78 / 29.16	31.77 / 33.68	27.82 / **24.32**	30.05 / 26.06

a number of data sets from different domains with various sizes and dimensionality are used for experimentation. Please see Table 4.1 for a detailed description of the datasets used in our experiments.

The proposed multi-resolution DTCWT scattering network is applied to each dataset to extract locally translation invariant multi-scale representations that are further used for classification. Scattering coefficients for two-dimensional signals are computed at six orientations (15°, 45°, 75°, 105°, 135°, 165°). The discrimination between the signal classes is achieved using a Gaussian SVM. Before the SVM is trained on the training set, each feature is standardized by the mean and standard deviation of the training dataset. The advantages of the standardization are detailed in the previous section.

The test set generalization error of each part the proposed scattering network, as detailed in Table 4.1, is reported on each dataset. The performance of the proposed method is compared with the scattering network proposed by Mallat et al [86] as well as with other (shallow) machine learning approaches on the datasets presented in Table 4.1.

4.2.1 US Postal Service Dataset

The US postal service dataset consists of two-dimensional grayscale images with 7291 training observations and 2007 test observations [55]. This dataset was generated by scanning the handwritten digits from envelopes by the U.S. Postal Service. The recorded images are de-slanted, and size normalized to 16 x 16 (256) pixels images in the dataset. The objective is to differentiate between the 10 digits, 0 to 9.

The proposed scattering network extracts the input signal at 6 resolution (s) (1, 0.85, 0.70, 0.6, 0.5, 0.35) and then extracts scattering coefficients from each resolution using the dual-tree wavelets at 3 scales (J) and 6 orientations (θ). The region non-linearity

is applied to overlapping regions (R) of size 2× 2. The cost value of 5 and gamma value of 0.00002 was selected for the SVM using 5-fold cross-validation on the training dataset.

As noted from Table 4.1, the proposed network with region non-linearity and log non-linearity (MDSCATP) results in the lowest classification error of 2.49%. The

Table 4.2 Classification error (%) comparison on USPS Dataset

Dataset	Proposed	ScatNet [86]	FIC [56]
USPS	**2.49**	2.6	5.43

proposed network produces an improvement of around 3% over the FIC [56] method. However, this improvement is only marginally better (<0.2%) than Mallat's ScatNet [86] as shown in Table 4.2.

4.2.2 The UCI Isolet Dataset

The Isolet dataset comprises of one-dimensional audio signals collected from 150 speakers uttering all characters in the English alphabet twice. Each speaker contributed 52 training examples with a total of 7797 recordings [88]. The recordings are represented with 617 attributes such as spectral coefficients, contour, sonorant and post-sonorant are provided to classify letter utterance.

The proposed scattering network decomposes the input signal, which is arranged as a 1× 617 length vector to the network, at 4 resolution (s) (1, 0.70, 0.5, 0.35). The translation invariant features are extracted the dual-tree wavelets at 6 scales (J) for the input signal at every resolution. Regions (R) of size 1×4 is chosen for the application of the region non-linearity. The cost value of 15 and a gamma value of 0.00015 was chosen for the SVM. Again, the parameters are selected using 5-fold cross-validation. The generalization error is reported on 10-fold cross validation for this dataset.

Table 4.1 shows that the multi-resolution scattering network with region non-linearity and log non-linearity (MDSCATP) produces the lowest classification error of 4.04%. The proposed method outperformed ScatterNet [86] but was unable to surpass

Table 4.3 Classification error (%) comparison on Isolet Dataset

Dataset	Proposed	ScatNet [86]	EEM [57]
Isolet	4.04	5.78	**2.70**

the performance of Extreme entropy machines [57] as shown in Table 4.3.

4.2.3 The UCI Yeast Dataset

This is a highly imbalanced one-dimensional signal dataset that consists of 1484 yeast proteins with 10 cellular binding sites [88]. Each binding site is described with 8 attributes. The aim is to classify the most probable (1 among the 10) cellular localization site of the proteins.

Each sample from the 1484 samples dataset is arranged as a 1×8 length vector which is decomposed by the proposed scattering network at two resolution (s) (1, 0.70). The translation invariant features are extracted the dual-tree wavelets at 2 scales (J) for the input signal at every resolution. The Region (R) size of 1×2, a cost value of 15 and a gamma value of is chosen using 5-fold cross-validation. The generalization error was reported on 10-fold cross validation for this dataset.

It is interesting to note that for this dataset the multi-resolution scattering network with log non-linearity and no region non-linearity (MDSCAT) produces the lowest classification error of 34.62% as shown in Table 4.1. The proposed method outperformed

Table 4.4 Classification error (%) comparison on Yeast Dataset

Dataset	Proposed	ScatNet [86]	IS [58]
Yeast	34.62	35.89	**33.0**

ScatterNet [86] by more than 1% but was unable to outrank the instance selection genetic algorithm [58] by the similar amount, as shown in Table 4.4.

4.2.4 The UCI Glass Dataset

This dataset consists of 214 one-dimensional signals that describe six types of glass based on nine chemical fractions of the oxide content [88]. This dataset was motivated by a criminological investigation where the correct classification of glass left on the crime scene could be used for evidence. Hence, the aim is to classify between different types of glass.

The proposed scattering network uses the same parameters as mentioned in Section 4.2.3 for feature extraction. The generalization error was reported on 10-fold cross validation for this dataset.

Similar to the yeast dataset, MDSCAT network achieves the lowest classification error of 24.32% as mentioned in Table 4.1. It was able to outperform the ScatterNet [86] and the Kernelized Vector Quantization [59] by a large margin as shown in Table 4.5.

Table 4.5 Classification error (%) comparison on Glass Dataset

Dataset	Proposed	ScatNet [86]	KVQ [59]
Glass	**24.32**	28.86	31.6

4.3 Computational Complexity

This section presents the computational time required to extract the features from the one dimensional (Isolet, Yeast, Glass) and two dimensional (USPS), datasets using the proposed ScatterNet with the NVIDIA GeForce 7800 GTX GPU. The proposed ScatterNet extracts the features for a batch of 128 images, at 16 $\times$ 16 resolution, (from the USPS dataset) in 0.01 seconds. The features for the whole dataset, at the resolutions and scales mentioned in Section 4.2.1, are extracted in 2.8 minutes. This computational time is less than 1 minute for the one-dimensional datasets.

4.4 Discussions

The proposed Multi-Resolution Region Pooling ScatterNet extracts locally translation invariant features from an input signal that are equally spaced over the scale space. The proposed algorithm was tested on four datasets and is shown to outperform Mallat's ScatterNet on all the datasets. The proposed network was able to outperform the learning based algorithms only on two datasets. Hence, it is necessary to take learning into account. The proposed scattering network can then provide the first two layers of such learning networks. It eliminates local translation variability, which can help in learning the next layers. Also, this network can replace simpler low-level features such as SIFT vectors and can compute the features very quickly.

4.5 Multi-resolution Parametric Log ScatterNet

This section details the second hand-crafted scattering network or ScatterNet termed as, the Multi-resolution Parametric Log Scattering Network that extracts translation invariant multi-resolution feature representations, which possess approximately symmetric amplitude distributions, from the input signal. The scatternet first decomposes the input signal using the *dual-tree complex wavelet transform* (DTCWT) [15]. The parametric log transformation is applied to the extracted representations to introduce relative symmetry to the distributions of the coefficient magnitudes. The transformation also de-correlates the multiplicative low-frequency components (illumination) while simultaneously creating a form of contrast normalization which enhances weaker features. Translation invariance is then introduced in the features by applying local smoothing. The information that would have been lost due to local smoothing is recovered at the second layer by filtering with coarse scale DTCWT wavelets, as well as the lowpass smoothing filter. The OLS layer next selects a subset of signal-specific components without undesired bias from outliers due to the introduced symmetry. The formulation of the two-layer Multi-resolution Parametric Log Scattering Network is presented for a two-dimensional image signal, as shown in Fig. 4.2. This formulation can be easily extended to signal of other dimensions. The introduced network can also be easily modified to design ScatterNets with deeper layers.

The input image x is first decimated into different multi-resolution images which are then filtered using the dual-tree complex wavelets as detailed in the introduction of this chapter. The filtered feature representations commute with translations and are covariant with translations. To build a more translation invariant representation, a point-wise $L2$ non-linearity is applied to the filtered signal, as described below:

$$U[\lambda_{m=1}] = |x \star \psi_{\lambda_1}| = \sqrt{|x \star \psi^a_{\lambda_1}|^2 + |x \star \psi^b_{\lambda_1}|^2} \tag{4.10}$$

This step produces the regular envelope of the filtered signal and reduces the redundancy of each representation to 2:1. $L2$ is a good non-linearity as it is stable to deformations and additive noise [86]. However, the representations may also contain outliers that can hinder the performance of the orthogonal least squares based feature selection layer (see Appendix B). Hence, the parametric log transformation layer is applied to all the oriented representations ($U[j]$) extracted at a particular scale j with a parameter k_j, to reduce the effect of outliers by introducing relative symmetry to the feature

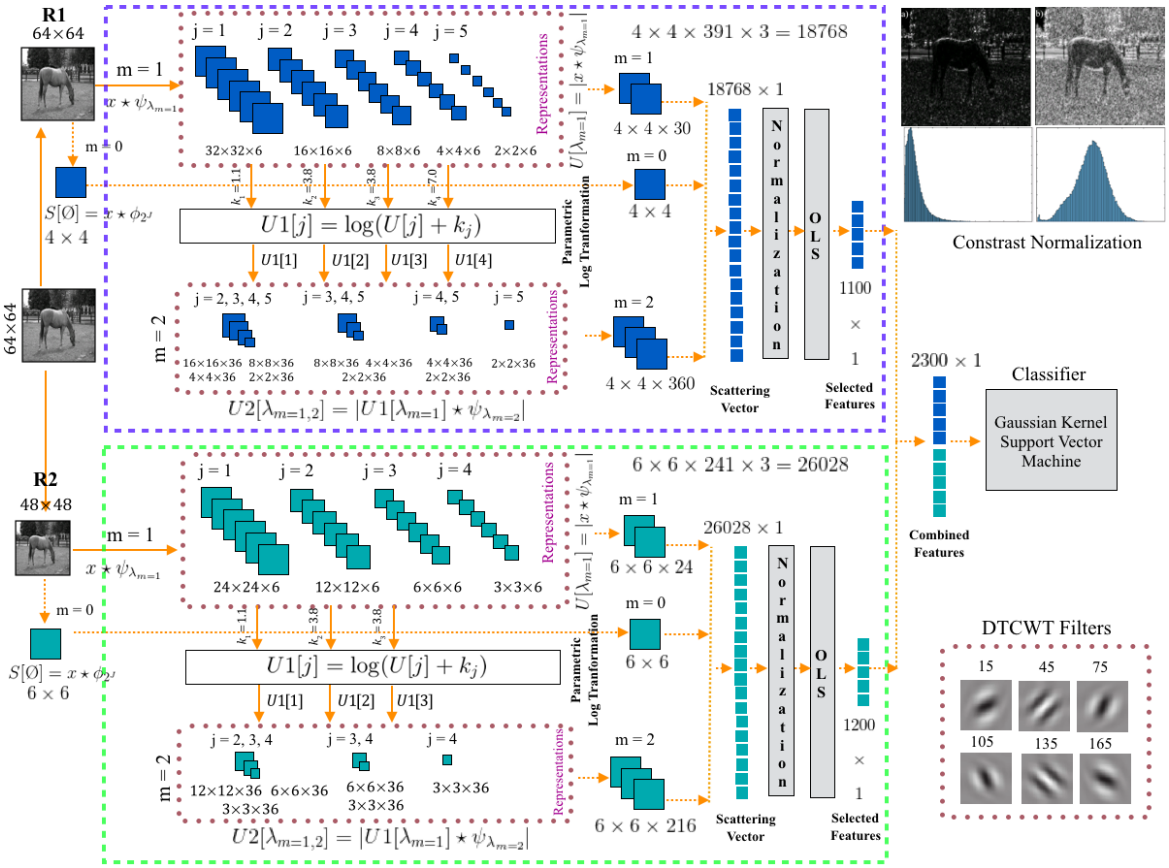


Fig. 4.2 Illustration shows the input image (x) of size 64×64 resized to images of resolution, R1 (64×64 (x)) and R2 (48×48 (x_1) respectively. Image representations at m $= 1$ are obtained using DTCWT filters at 5 scales for R1, 4 scales for R2 and 6 orientations ($x \star \psi_{\lambda_{m=1}}$). Next, L2 non-linearity (complex modulus) is applied on the representations to obtain the regular envelope $|x \star \psi_{\lambda_{m=1}}|$. Log transformation $U1[j] = \log(U[j] + k_j)$ with parameters k_j is applied on the envelope for all scales j except the coarsest scale. Next, local smoothing is applied to extract the translation invariant coefficients $U1[\lambda_{m=1}] \star \phi_{2^J}$. The information lost due to smoothing are recovered by cascaded wavelet filtering at the second layer $|U1[\lambda_{m=1}] \star \psi_{\lambda_{m=2}}|$. Translation invariance is introduced in the recovered frequencies using L2 non-linearity and local smoothing $U2[\lambda_{m=1}, \lambda_{m=2}] \star \phi_{2^J}$. The contrast normalization effect of the parametric log transformation is shown in the top right while the DTCWT filters at six fixed orientations are shown in the bottom.

distributions, as shown below:

$$U1[j] = \log(U[j] + k_j), \quad U[j] = |x \star \psi_j|, \tag{4.11}$$

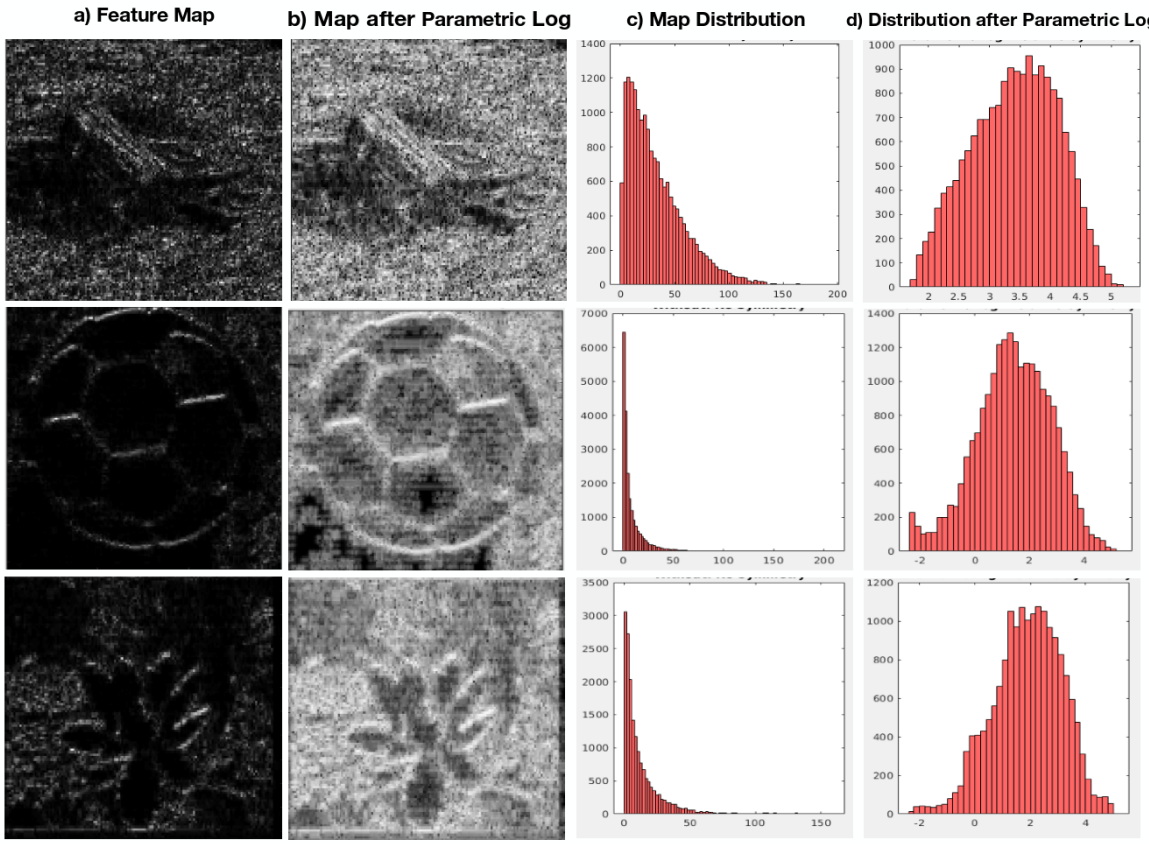


Fig. 4.3 The illustration shows the feature representations extracted using the dual-tree wavelets at the first scale ($j = 1$) and 6 orientations from images randomly chosen from the Caltech-101 dataset. Log transformation $U1[j = 1] = \log(U[j = 1] + k_{j=1})$ with parameters k_j is next applied on the combined representation for all 6 orientations with $k_{j=1}$. $k_{j=1}$ is obtained by minimizing the difference between the mean and median of the combined representation. The illustration shows the feature representations extracted at $j = 1$ and 15° orientation along their histogram distribution before (a, c) and after (b, d) the application of the parametric log non-linearity. The non-linearity also introduces contrast normalization as seen from above figures. It is also interesting to note that after the application of the log non-linearity the amplitude range of the features shrinks from around 0 to 100 to around -2 to 5. This can be seen from the horizontal axis of the histograms.

Good symmetry is achieved for the distribution of oriented representations obtained by selecting the parameter k_j that minimizes the difference between the mean and median of the distribution as shown in Fig. 4.3. The parametric log transformation also de-correlates the low-frequency multiplicative components arising due to illumination variation and noise [22] as well as *normalizing the contrast* of the representations by elevating the weak features and suppressing the stronger as shown in Fig. 4.3.

Next, a local average is computed on the envelope $|U1[\lambda_{m=1}]|$ that aggregates the coefficients to build the desired local translation-invariant representation. This local average is obtained using the scaling function lowpass filter as shown below:

$$S[\lambda_{m=1}] = |U1[\lambda_{m=1}]| \star \phi_{2^J} \tag{4.12}$$

The high frequency components lost due to smoothing are retrieved by cascaded wavelet filtering performed at the second layer. The retrieved components are again not translation invariant. Translation invariance (over a relatively larger local region) is achieved by first applying the L2 non-linearity of eq(2) to obtain the regular envelope:

$$U2[\lambda_{m=1}, \lambda_{m=2}] = |U1[\lambda_{m=1}] \star \psi_{\lambda_{m=2}}| \tag{4.13}$$

A local-smoothing operator is then applied to the regular envelope ($U2[\lambda_{m=1}, \lambda_{m=2}]$) to extract the desired second layer ($m = 2$) translation invariant coefficients:

$$S[\lambda_{m=1}, \lambda_{m=2}] = U2[\lambda_{m=1}, \lambda_{m=2}] \star \phi_{2^J} \tag{4.14}$$

The scattering coefficients obtained at each layer are:

$$S = \begin{pmatrix} x \star \phi_{2^J} \\ U1[\lambda_{m=1}] \star \phi_{2^J} \\ U2[\lambda_{m=1}, \lambda_{m=2}] \star \phi_{2^J} \end{pmatrix}_{j=(2,3,4,5...)} \tag{4.15}$$

The coefficients extracted from each layer are concatenated to generate a feature vector for each of the images in the training dataset as shown in Fig. 4.2. The scattering feature vectors are then normalized across each dimension and given as input to the feature selection layer.

The feature selection layer is implemented using a supervised orthogonal least square (OLS) regression [1] (see Appendix B) that greedily selects discriminative features specific to class C with a one-versus-all linear regression using the following indicator function:

$$f_C(x) = \begin{cases} 1 & \text{if x belongs to class C} \\ 0 & \text{otherwise} \end{cases} \tag{4.16}$$

The regression is applied to a training set of scattering feature vectors where each vector of N dimensions is reduced to N' selected dimensions ($N' << N$) that belong to a specific class C. Let $(\Phi_t^{M \times N})_C$ be the dictionary at the t^{th} iteration for a specific class C. The t^{th} feature x is selected such that the linear regression of $f_C(x)$ has a

minimum mean-squared error, computed on the training set corresponding to class C. The reduced scattering features dataset represents the metric defined in Eq. 1.1. The reduced training feature dataset is given as input to the G-SVM that learns the weights that best discriminate the classes in the dataset. Feature selection makes training and applying a classifier more efficient due to the decreased vector size. It also tends to improve performance by eliminating unnecessary components of the input and their associated noise.

4.6 Overview of Results

The performance of the proposed network is evaluated on CIFAR-10 and CIFAR-100 datasets with 10 and 100 classes respectively. Each dataset contains a total of 50000 training and 10000 test images of size 32×32 equally divided among the classes. The evaluation is performed on the classification accuracy, computational efficiency, and feature richness. A comparison with Mallat's ScatterNet [22], unsupervised [62], [60] and supervised methods [35, 36, 107] is also performed.

To extract the scattering representations, every 32×32 image is first upsampled into two images of resolution 64×64 (R1) and 48×48 (R2). R1 and R2 are decomposed using DTCWT filters with six fixed orientations at 5 and 4 scales respectively, followed by L2 non-linearity, as shown in Fig. 4.2. Next, the log transformation is applied to the representations (except the ones obtained at the coarsest scale) obtained from both R1 and R2 pipeline with parameters $k_1 = 1.1$, $k_2 = 3.8$, $k_3 = 3.8$ and $k_4 = 7$ chosen for scale j = 1, 2, 3 and 4 respectively. The transformation parameters for each scale are obtained by averaging the parameter obtained for each image of the dataset using the scheme explained in Section 4.4. A smoothing operator is then applied to introduce translation invariance in the representations. The classification accuracy for representations obtained at various scales (J), with and without the use of parametric log transformation and the concatenated coefficients at $m = 1$ with G-SVM, are shown for both R1 and R2 pipelines in Table 4.6. The G-SVM parameter (c) is selected as 14 while gamma parameter is set to 0.00002 using 5-fold cross-validation on the training feature set. We see that the parametric log transformation results in a small improvement in classification accuracy. The information lost due to smoothing at the first layer is retrieved at the next layer using cascaded filtering as shown in Fig. 4.2. The retrieved information is made translation invariant by local smoothing. Representations for the three color channels at $m = 0, 1, 2$ are concatenated to produce a 18768 (6256×3) dimensional vector for R1 image and a vector of length 26028 (8676×3) for R2 as

Table 4.6 Accuracy (%) on CIFAR-10 for both R1 and R2 for each scale (J) and coefficients at $m = 1$, with and without applying log transformation. The accuracy for features selected from the final scattering vector at $m_{1,2}$ using OLS is presented in the last column.

	$J = 1$	$J = 2$	$J = 3$	$J = 4$	m_1	$m_{1,2}$
R1: No-log	62.7	66.9	69.0	70.2	70.4	80.7
R1: log	**65.6**	**69.9**	**71.5**	**72.4**	**72.5**	**81.6**
R2: No-log	65.9	70.0	71.2	–	71.7	80.9
R2: log	**68.0**	**71.5**	**72.6**	–	**73.4**	**81.8**

shown in Fig. 4.2. OLS is then applied to the training dataset (50000×18768) to select 108 dimensions per class resulting in a total of 1080 discriminative dimensions for every R1 image (50000×1080). Similarly, 1200 dimensions per image are chosen for the R2 image. This reduced feature dataset results in a classification accuracy of 81.6% (80.7% without log transformation) for R1 images while accuracy of 81.8% (80.9% without log transformation) is recorded for R2 images, using the above-mentioned SVM for the CIFAR-10 datasets as shown in Table 4.6. Classification accuracy of 82.4% is obtained by concatenating the selected dimensions of R1 and R2. A decrease in classification accuracy is recorded on selecting more than the above-mentioned feature dimensions.

Next, scattering coefficients extracted using DTCWT ScatterNet with the above-mentioned parameters result in a classification accuracy of 56.7% for the CIFAR-100 dataset, as shown in Table 4.7. The translation invariant coefficients extracted using the proposed network outperform the translation as well as Roto-translation invariant features of Mallat's ScatterNet [22], on both datasets. The system also outperformed state-of-the-art unsupervised methods [62], [60] but underperformed by nearly 10% against supervised deep learning models [158], as shown in Table 4.7. The proposed

Table 4.7 Accuracy (%) and comparison on both datasets. Pro.: Proposed, Sup: Supervised and Unsup: Unsupervised, learning.

Dataset	Pro.	ScatNet [22]	Unsup	Sup
CIFAR-10	82.4	81.6	82.2 [62]	**93.56** [158]
CIFAR-100	56.7	55.8	54.2 [60]	**70.48** [158]

network can be an attractive choice over Mallat's ScatterNet due to its computational efficiency and gain in classification accuracy. The proposed system extracts the coefficients from both R1 and R2 images in almost three-quarters (0.78 (s)) of the time needed by Mallat's network (0.98 (s)) to decompose only the R1 image, as shown in Table 4.8. This marginal difference is significant for large image datasets such as

Table 4.8 Arc.: Architectures, Pro.: Proposed, R1, R2: Resolution - 1,2 pipeline, FVL: Feature vector length, SD: Selected dimensions using OLS, FR: Feature richness (%), TS (s): Scattering time an image in seconds, T-OLS: Feature selection time using OLS in hours.

Arch.	FVL	SD	FR (%)	TS (s)	T-OLS (h)
ScatNet [22]	113712	2000	1.75	0.98	3.22
R1	18762	1100	**5.86**	**0.46**	**1.07**
R2	26028	1200	**4.61**	**0.32**	**1.14**
Pro. (R1+R2)	44796	2300	**5.13**	**0.78**	**2.21**

CIFAR. In addition, since the scattering vector produced by the proposed network is smaller (44796) as compared to Mallat's network (113712) (three-layer network) [22], the OLS layer can select the desired feature dimensions (1080 and 1200) in almost 3/4 of the time (2.21(h) vs. 3.22(h)). The dimensions chosen with OLS from the scattering vector are more for the proposed network ($1080 + 1200 = 2300$) as compared to Mallat's network (1080). This suggests that the features extracted by the proposed network are significantly more abundant in information as compared to Mallat's network as feature richness is defined as the number of dimensions selected with OLS divided by the total feature dimensions. The simulations are computed on a server with 32 Gb RAM per node in normal conditions.

However, supervised models require large training datasets to learn which may not exist for most application. Table 4.9 shows that DTCWT ScatterNet outperformed LeNet [61] and Network in Network (NIN) [35] supervised learning networks on the CIFAR-10 datasets with less than 10,000 images. The experiments were performed by dividing the training dataset of 50,000 images into eight datasets of different sizes. The images for each dataset are obtained by randomly selecting the required number of images from the full 50,000 training dataset. It is made sure that an equal number of images per object class are sampled from the training dataset. The whole test set of 10000 images is used for all the experiments. Deeper models like NIN [35] and VGG [36] result in low classification accuracy due to their inability to train on the small training dataset.

4.7 Computational Complexity

This section presents the computational time required to extract the features from the CIFAR-10 and CIFAR-100 images of size $64 \times 64 \times 3$ (up-sampled from $32 \times 32 \times 3$) using the proposed ScatterNet (dual-tree wavelets at 5 scales and 6 orientations) with

Table 4.9 Comparison of Proposed (Pro.) network on accuracy (%) with two supervised learning methods (LeNet [61], NIN: Network in Network [35] and VGG [36] against different training dataset sizes on CIFAR-10.

Arch.	300	500	1K	2K	5K	10K	20K	50K
Pro.	**39.3**	**48.8**	**55.9**	**61.8**	**67.0**	72.9	76.8	82.4
LN	34.9	44.7	53.1	57.9	63.0	69.0	74.0	77.6
NIN	10.1	10.3	10.9	40.4	63.4	72.0	83.1	89.6
VGG	9.6	10.3	10.7	43.4	53.5	**77.8**	**88.3**	**93.56**

the NVIDIA GeForce 7800 GTX GPU. The proposed ScatterNet extracts the features for a batch of 128 images in 0.41 seconds. The features for the whole dataset, at both resolutions mentioned in Section 4.2.1, are extracted in 6 minutes. Another 2 hours are taken by the orthogonal least squres algorithm to select the relevant features as shown in Table 4.8. Finally, the support vector machine (SVM) with the Gaussian kernel learns the discriminatory features in 36 minutes.

4.8 Discussions

The proposed Multi-resolution Parametric Log Scattering Network is an improved version of Mallat's Scatter-Net which uses dual-tree wavelets and parametric log non-linearity to extract relatively symmetric translation invariant multi-resolution representations from the input signal. The ScatterNet gives an enhanced performance on classification accuracy and computational efficiency as compared to Mallat's ScatterNet on two image classification datasets. The network has also been shown to outperform unsupervised learning methods while evidence of the advantage of DTCWT ScatterNet over supervised learning (CNNs) methods is presented for applications with small training datasets. In addition, these networks can extract features rapidly which is advantageous as the time needed to train the unsupervised and supervised networks is considerable.

4.9 Comparison between the Proposed ScatterNets

This section presents the comparison between the Multi-Resolution Region Pooling ScatterNet and Multi-resolution Parametric Log ScatterNet introduced in Section 4.1 and Section 4.4. The performance is evaluated on the CIFAR-10 and Cifar-100 image datasets.

Table 4.10 Accuracy (%) and comparison on both datasets for the the Multi-Resolution Region Pooling ScatterNet and Multi-resolution Parametric Log ScatterNet.

Dataset	Region Pooling ScatterNet	Parametric Log ScatterNet
CIFAR-10	81.9	**82.4**
CIFAR-100	56.1	**56.7**

For both the CIFAR datasets, every 32×32 images is upsampled into two images of resolution 64×64 (R1) and 48×48 (R2) which are decomposed using dual-tree wavelets with six fixed orientations at 5 and 4 scales respectively. For the Region Pooling ScatterNet, the region non-linearity is applied to overlapping regions (R) of size 2× 2. As for the Parametric Log ScatterNet, the log transformation is applied to the representations (except the ones obtained at the coarsest scale) obtained from both R1 and R2 pipeline with parameters $k_1 = 1.1$, $k_2 = 3.8$, $k_3 = 3.8$ and $k_4 = 7$ chosen for scale j = 1, 2, 3 and 4 respectively. The Parametric Log ScatterNet marginally outperforms the Region Pooling ScatterNet on both the datasets as shown in Table 4.10. It is therefore used as the front-end for the networks proposed in the rest of the thesis.

Chapter 5

Efficient Learning using ScatterNets

The training of deep convolutional neural networks (DCNNs) requires large training datasets along with considerable computational resources and time. Data augmentation and transfer learning have been proposed to improve the training of deep networks for applications where sufficient labeled training data is not available. However, these methods have their limitations as detailed in the introduction of chapter 3.

The hand-crafted descriptors extracted using the proposed hand-crafted scattering networks layers can improve the learning and classification performance of the standard deep networks (VGG, ResNet, NIN). This is shown by the proposed DTCWT ScatterNet Convolutional Neural Network (DTSCNN) formed by replacing the first convolutional, Relu and pooling layers of the CNN with the parametric log based DTCWT ScatterNet proposed in Section 4.4. The translation invariant features with relatively symmetric probability density functions (pdfs) are extracted by the hand-crafted network that incorporate edge information similar to the first layers of the DCNNs trained on ImageNet [105], [95]. These features are used by the middle and later CNN layers to learn high-level features. This helps the proposed DTSCNN architecture to converge faster as it has fewer filter weights to learn compared to its corresponding CNN architecture (Section 5.2.3). Also, the CNN layers can learn more complex patterns from the start of learning as it is not necessary to wait for the first layer to learn low-level features as they are already extracted by the ScatterNet.

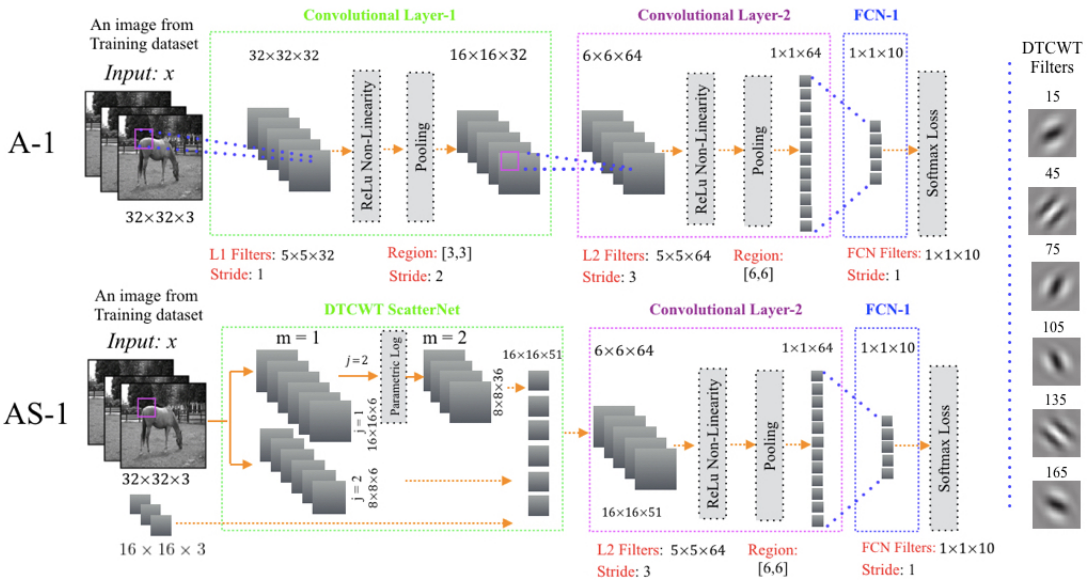


Fig. 5.1 The proposed DTSCNN architecture, termed as AS-1, formed by replacing the first convolutional, ReLu, and pooling layer of A-1 (Table. 1) CNN architecture with the two-layer parametric log based DTCWT ScatterNet. The ScatterNet extracts relatively symmetric translation invariant representations from a multi-resolution image that are processed by the CNN architecture to learn complex representations. However, the illustration shows the feature extraction only for a single image due to space constraints. The invariant information ($U[\lambda_{m=1}]$) obtained for each R, G and B channel of an image is combined into a single invariant feature by taking an L2 norm of them. Log transformation is applied with parameter $k_1 = 1.1$ for scale j = 1. The representations at all the layers ($m = 0(3)$, $m = 1(12)$ and $m = 2(36)$) are concatenated to produce 51*2 (two resolutions) = 102 image representations that are given as input to the mid and back layers of the CNN.

5.1 DTCWT ScatterNet Convolutional Neural Network (DTSCNN)

This section details the proposed DTCWT ScatterNet Convolutional Neural Network (DTSCNN) composed by combining the two-layer parametric log based DTCWT ScatterNet, as described in Section 4.4, with the following layers (middle and back-end) of the CNN to perform object classification. The parametric log based DTCWT ScatterNet extracts feature maps that are denser over the scale from multi-resolution images at 1.5 times and twice the size of the input image as detailed in Section 4.4.

Next, the proposed DTSCNN architectures (AS1 to AS4) were realized by replacing the first convolutional layer of the A-1 to A-4 CNN architectures with the hand-crafted multi-resolution parametric log ScatterNet, as shown in Fig. 5.1. The four CNN

Table 5.1 Experiments are performed with CNN architectures (derived from LeNet [17]) designed for CIFAR-10 dataset that contain convolutional (CV) layers (L1 to L5) with b number of filters of size $a \times a$, denoted as L-F: a, b. The max pooling is performed for a layer within a region of size $c \times c$, denoted as PL-R: $[c, c]$. The network also contains fully connected layers (FCN) that feed the final CNN outputs to a softmax loss function. The architectures are: (i) A-1: 2CV-1FCN (ii) A-2: 3CV-2FCN (iii) A-3: 4CV-3FCN (iv) A-4: 5CV-3FCN.

Arch.	Layers										
	L1-F	PL1-R	L2-F	PL2-R	L3-F	PL3-R	L4-F	L5-F	FCN1	FCN2	FCN3
	a,b	[c,c]	a,b	[c,c]	a,b	[c,c]	a,b	a,b	a,b	a,b	a,b
A-1	5,32	[3,3]	5,64	[6,6]	–	–	–	–	1,10	–	–
A-2	5,32	[3,3]	5,32	[6,6]	5,64	[4,4]	–	–	1,32	1,10	–
A-3	5,32	[3,3]	5,32	[3,3]	5,64	[3,3]	4,64	–	1,32	1,16	1,10
A-4	5,32	[3,3]	5,32	[3,3]	5,64	–	4,64	4,64	1,32	1,16	1,10

Table 5.2 Parameter values used by the architectures mentioned in Table. 1 for training are: Learning rate = 0.001, Number of Epochs = 300, Weight Decay = 0.0005 and Momentum = 0.9. The batch size is changed according to the number of training samples as mentioned below.

Training Data Sample Size	300	500	1000	2000	5000	10000	25000	50000
Batch Size	5	5	10	20	50	100	100	100

architectures (A-1 to A-4, shown in Table 5.1) are derived from the LeNet [17] network because they are relatively easy to design and train due to its small memory footprint. Also, the LeNet [17] architecture is shallow which makes it easier to observe the benefits (faster learning and better generalization) of replacing the earlier layers with the ScatterNets. This effect of removing earlier layer is not easily visible in deep networks. In addition to the derived architectures, the DTSCNN is also realized by using ScatterNet as the front-end of three standard deep networks namely; Network in Network (NIN) [35] (A-5), VGG [36] (A-6), and wide ResNet [104] (WResNet) (A-7). The DTSCNN architectures (AS-5, AS-6, AS-7) for the standard architectures (NIN (A-5), VGG (A-6), WResNet (A-7)) are again obtained by removing the first convolutional layer of each network and replacing it with the ScatterNet. The architectures are trained in an end-to-end manner by Stochastic Gradient Descent with softmax loss until convergence. The trained network learns a metric similar to the one defined in Eq. 1.1.

5.2 Experimental Results

The performance of the DTSCNN architecture is demonstrated on CIFAR-10 and Caltech-101 datasets with over 50 experiments performed with 14 CNN architectures of increasing depth on (i) Classification error and, (ii) Computational efficiency and the rate of learning. The generic nature of the features extracted by the DTCWT ScatterNet is shown by an equivalent performance to the pre-trained CNN front-ends. The details of the datasets and the results are presented below. The performance of the DTSCNN architecture is evaluated on (i) Classification error and, (ii) Computational efficiency and the rate of learning with over 50 experiments performed with 14 CNN architectures. The efficient learning of the DSTCNN architectures is demonstrated by their ability to train faster and with lower classification error on small as well as large training datasets, generated from the CIFAR-10 dataset. The DTCWT ScatterNet front-end is also shown to give similar performance to the first convolutional pre-trained layers of CNNs which capture problem specific filter representations, on Caltech-101 as well as CIFAR-10 datasets. A comparison with the state-of-the-art is also presented on both datasets.

5.2.1 Datasets

The CIFAR-10 [16] dataset contains a total of 50000 training and 10000 test images of size 32×32. The efficient learning of the proposed DTSCNN network is measured on eight training datasets of sizes: 300, 500, 1000, 2000, 5000, 10000, 25000 and 50000 images generated randomly by selecting the required number of images from the full 50000 training dataset. In each case, we made sure that an equal number of images per object class were sampled from the full training dataset. For example, a training dataset sample size of 300 included 30 images per class. The full test set of 10000 images was used for all the experiments. The ScatterNet extracts 102 features maps of size 16×16 from the cifar images of both resolutions which are used by the convolutional layers of the CNN to learn high-level features (Fig. 5.1).

Caltech-101 [94] dataset contains 9K images each of size 224×224 labeled into 101 object categories and a background class. The classification error on this dataset is measured on three randomly generated splits of training and test data so that each split contains 30 training images per class and up to 50 test images per class. In each split, 20% of training images were used as a validation set for hyper-parameter selection. Networks trained on ImageNet dataset are used to initialize the filter weights for the DTSCNN or other deep networks used to perform classification (Table 5.4 and 5.6) on

Table 5.3 Classification error (%) on the CIFAR-10 dataset for the original CNN architectures and their corresponding DTSCNN architectures.

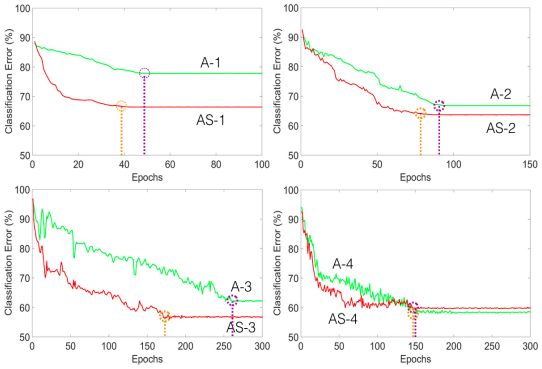
Architectures	Classification Error							
Variants of LeNet [17]	Training Data Sample Size							
	300	500	1000	2000	5000	10000	25000	50000
A-1: 2Conv-1FCon	77.8	73.2	70.3	66.7	61.3	54.9	45.3	**38.1**
AS-1: DTS-1Conv-1FCon	**69.4**	**65.8**	**60.1**	**58.9**	**52.7**	**54.7**	**40.4**	38.7
A-2: 3Conv-2FCon	66.8	62.0	57.5	52.8	46.1	40.1	**32.7**	**27.3**
AS-2: DTS-2Conv-2FCon	**63.7**	**55.1**	**49.5**	**43.7**	**39.1**	**40.0**	33.8	28.3
A-3: 4Conv-3FCon	62.2	57.4	51.0	46.8	40.1	35.1	29.2	24.2
AS-3:DTS-3Conv-3FCon	**56.8**	**54.9**	**50.9**	**45.3**	**39.7**	**34.9**	**28.7**	**24.1**
A-4: 5Conv-3FCon	**58.4**	54.4	47.4	41.8	**35.0**	32.2	25.7	22.1
AS-4: DTS-4Conv-3FCon	59.8	**54.0**	**47.3**	**41.3**	38.4	**31.8**	**25.2**	**22.0**
Deep Architectures	Training Data Sample Size							
A-5: NIN [35]	89.2	84.4	45.5	34.9	27.1	18.8	**13.3**	**8.1**
AS-5: DTS-NIN	**83.2**	**80.1**	**41.0**	**32.2**	**25.3**	**18.4**	13.4	8.2
A-6: VGG [36]	89.9	89.7	89.1	59.6	36.6	28	16.9	**7.5**
AS-6: DTS-VGG	**83.5**	**82.8**	**81.6**	**56.7**	**34.9**	**27.2**	**16.9**	7.6
A-7: WResNet [104]	87.2	53.2	43.2	31.1	18.8	13.6	10.1	3.6
AS-7: DTS-WResNet	**81.2**	**49.8**	**41.2**	**30.1**	**18.6**	**13.5**	**9.9**	**3.6**

this dataset as the number of training samples is not sufficient to train the networks from a random start. The ScatterNet extracts 102 features maps of size 128×128 from the cifar images of both resolutions which are used by the convolutional layers of the CNN to learn high-level features.

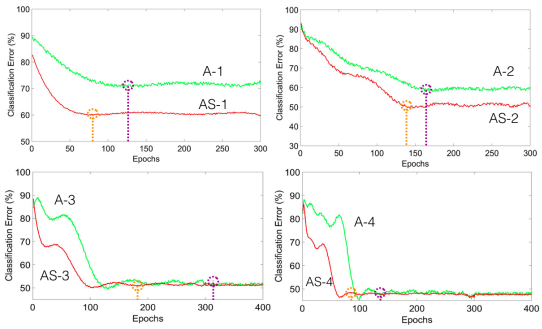
5.2.2 Evaluation and Comparison on Classification Error

The classification error is recorded for the proposed DTSCNN architectures and compared with the derived (A-1 to A-4) as well as the standard (A-5 to A-7) CNN architectures, for different training sample sizes, as shown in Table 5.3. The parameters used to train the CNN architectures are shown in Table 5.2. The classification error corresponds to the average error computed for five repetitions.

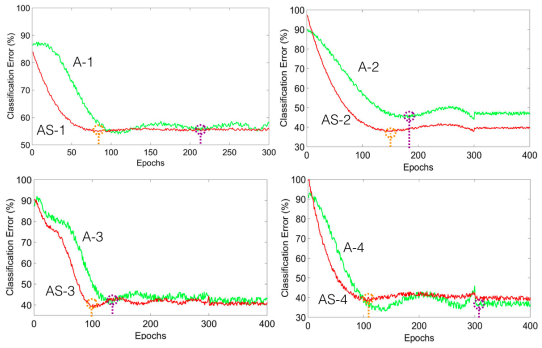
It can be observed from Table 5.3 that the difference in classification error for the derived CNN architectures (A-1 to A-4) is from 3% to 10% for the small training datasets with ≤ 1000 training images. This difference in error reduces with the increase in the size of the training dataset and with an increase in the depth of the architectures as shown in Table 5.3. In fact, the more deep CNN architectures such 5CV-3FCN (A-4) outperformed their corresponding DTSCNN architectures by a small margin.



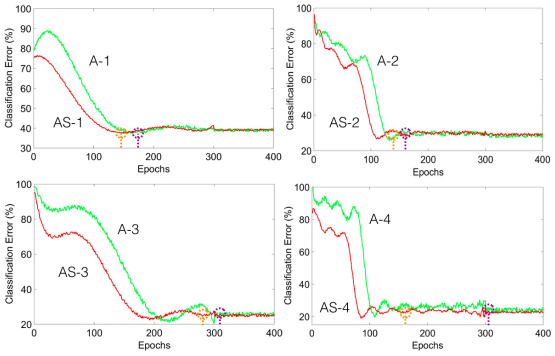
(a) Training dataset sample size: 300



(b) Training dataset sample size: 1000



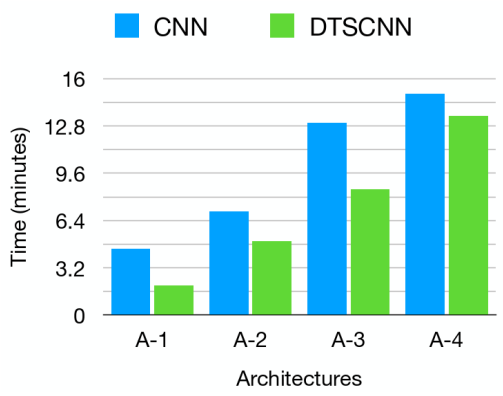
(c) Training dataset sample size: 10000



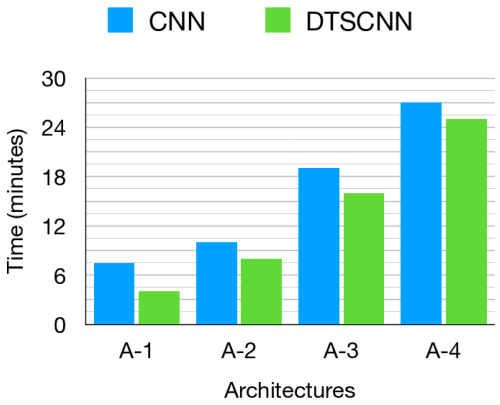
(d) Training dataset sample size: 50000

Fig. 5.2 Graphs show the faster convergence and rate of learning of the DTSCNN derived architectures (AS-1 to AS-4) compared to the CNN (A-1 to A-4) architectures for a range of small and large training data sizes. A network is considered to have converged at a specific epoch when the error value for the subsequent epochs lies within 2% of the error value at that specific epoch. The convergence is marked on the epoch axis using an (vertical) orange dotted line for the DTSCNN architecture and a (vertical) purple dotted line for the CNN architectures. The orange line has a lower epochs value as compared to the purple line indicating the faster convergence.

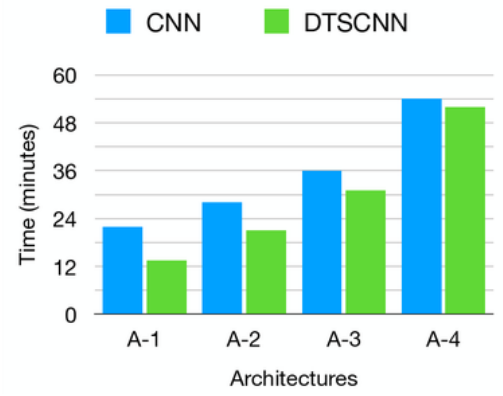
A similar trend in classification error is also observed for the standard more deep CNN architectures (A-5 to A-7). The difference in classification error is large between the DTSCNN (AS-5 to AS-7) and the original (A-5 to A-7), architectures for small training datasets while both types of networks produce a similar classification error for datasets with large training size. In fact, the wide ResNet (WResNet) (A-7) and its corresponding DTSCNN architecture (AS-7) result in the same classification error of 3.6%.



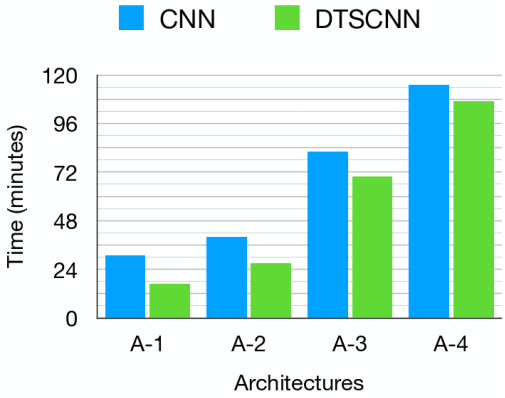
(a) Computational time for convergence for 300 training size



(b) Computational time for convergence for 1000 training size



(c) Computational time for convergence for 10000 training size



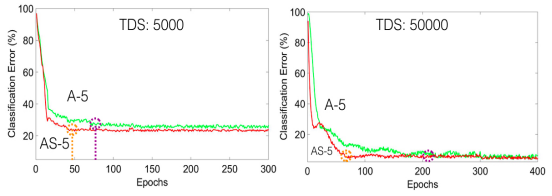
(d) Computational time for convergence for 50000 training size

Fig. 5.3 Computational time the DTSCNN derived architectures (AS-1 to AS-4) compared to the CNN (A-1 to A-4) architectures for a range of small and large training data sizes. A network is considered to have converged at a specific epoch when the error value for the subsequent epochs lies within 2% of the error value at that specific epoch.

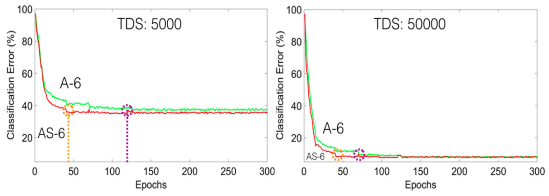
5.2.3 Analysis on Computational Efficiency and Learning

This section compares the speed of learning of the proposed DTSCNN architectures against the CNN architectures (A-1 to A-4) derived from LeNet [17] as well as the standard (A-5 to A-7) deep learning architectures for a range of small and large training dataset sizes.

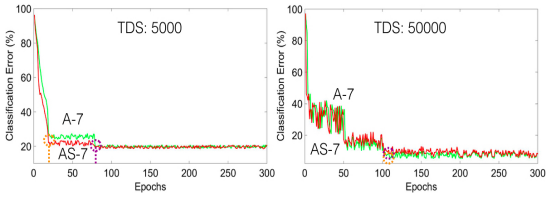
The DTSCNN architectures have a higher rate of learning or faster convergence than the original CNN architectures because the numbers of filter weights required to be learned are smaller. Also, the ScatterNet extracts edge representations that allow the next CNN layers to learn high-level features from the first epoch onwards. The



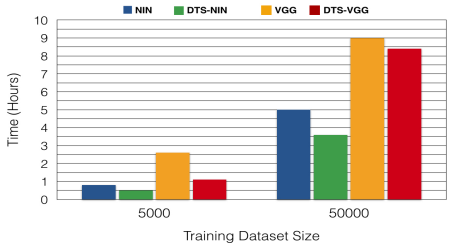
(a) Network in Network (NIN)



(b) Visual Geometry Group (VGG) Network



(c) Wide Residual Network (WResNet)



(d) Computational time for convergence.

Fig. 5.4 Graphs show the faster convergence and rate of learning of the DTSCNN standard deep architectures (AS-5 to AS-7) compared to the CNN (A-5 to A-7) architectures for a small (5000) and large (50000) training dataset. A network is considered to have converged at a specific epoch when the error value for the subsequent epochs changes within 2% of the error value at that specific epoch. The convergence is marked on the epoch axis using an orange dotted line for the DTSCNN architecture and a purple dotted line for the CNN architectures. The orange line has a lower epochs value as compared to the purple line indicating the faster convergence. Computational time for convergence (hours) for NIN (A-5) and VGG (A-6) standard deep architectures and corresponding DTSCNN architectures for a small (5000) and a large (50000) training dataset is also presented.

faster convergence is shown for the derived (A-1 to A-4) for different training dataset sizes in Fig. 5.2.

A network is considered to have converged at a specific epoch when the error value for the subsequent epochs changes within 2% of the error value at that particular epoch. The convergence is marked on the epoch axis (x-axis) using an orange vertical dotted line for the DTSCNN architecture and a purple dotted line for the CNN architectures. As observed from Fig. 5.2, the orange line has a lower epoch value as compared to the purple line indicating the faster convergence.

The time required for training the original and their corresponding DSTCNN architectures is presented for training datasets of different sizes for the derived (A-1 to A-4) in Fig. 5.3. The time for convergence is again measured to within 2% of the final converged error value. As observed from the figure, the training time is higher for the original networks than the DTSCNN networks because of the reasons mentioned above.

The faster convergence with lower training time is also presented for the standard deep architectures (A-5 to A-7) for a small (5000) and a larger (50000) training dataset sizes in Fig. 5.4.

The networks are trained using the MatConvNet [103] (see Appendix D) package on a server with an NVIDIA GeForce 7800 GTX card.

5.2.4 Comparison with Pre-trained CNN First Layers

The classification performance of the DTCWT ScatterNet front-end is compared with the first pre-trained convolutional layer for the Network in Network (NIN) [35] and the Visual Geometry Group convolutional (VGG) [36] architectures, on Caltech-101 and CIFAR-10 datasets. The filter weights for both the NIN and the VGG networks are initialized with the weights obtained from their models pre-trained on ImageNet (found here [89]). The first layers for both the architectures are fixed to be the ScatterNet and the pre-trained convolutional layer, while the filter weights only in later layers are fine-tuned using the training images of CIFAR-10 and Caltech-101. The ScatterNet front-end gives similar performance to the pre-trained first convolutional layer on classification error for both datasets as shown in Table 5.4. For this experiment, dropout, batch normalization and data augmentation with crops and horizontal flips were utilized [90]. The use of the NIN network is preferred as it gives similar performance to the VGG network while being four times faster. The classification error for the VGG network is lower as compared to the NIN network due to its significantly deeper architecture which allows it to learn more complex features. However, the wide ResNet network achieves the state-of-the-art accuracy as shown in [104].

Table 5.4 Table shows the comparison on classification error (%) between the DTCWT ScatterNet (DTS) front-end and the first convolutional layer pre-trained on ImageNet for NIN [35] and VGG [36] architectures for Caltech-101 and CIFAR-10 datasets.T-NIN: Transfer-NIN, T-VGG: Transfer-VGG

Dataset	State-of-the-art Architectures			
	T-NIN	DTS-NIN	T-VGG	DTS-VGG
Caltech-101	12.3	**12.26**	**8.78**	9.23
CIFAR-10	**8.25**	8.34	**8.31**	9.02

5.2.5 Comparison with the state-of-the-art

This section compares the architectures that produced the best classification performance with the state-of-the-art on CIFAR-10 and Caltech-101. The DTS-WResNet

(AS-7) and DTS-VGG (AS-6) resulted in the best classification performance on CIFAR-10 and Caltech-101 with 3.6% and 8.08% classification error, respectively.

The DTS-WResNet (AS-7) architecture is compared with various state-of-the-art CNN architectures on CIFAR-10. DTS-WResNet (DW) outperformed these architectures as shown in Table 5.5.

Table 5.5 Table shows the comparison on classification error (%) between the DTCWT ScatterNet ResNet (DTS-WResNet) Architecture with the state of the art architectures on the CIFAR-10 dataset. DW: DTS-WResNet, NIN: Network in Network [35], VGG [36], DSN: Deeply Supervised Networks [61], MON: Max-Out Networks [93], E-CNN: Exemplar CNN [92]

Dataset	State-of-the-art Architectures					
	DW	VGG	E-CNN	NIN	DSN	MON
Cifar-10	**3.6**	7.5	8.0	8.1	8.2	9.3

Next, the DTS-VGG (AS-6) architecture is compared against the state-of-the-art CNN architectures for the Caltech-101 dataset. On this dataset, the DTS-VGG outperformed some of the networks while produced a marginally lower classification performance for others (Table 5.6).

Table 5.6 Table shows the comparison on classification error (%) between the DTCWT ScatterNet VGG (DTS-VGG) Architecture with the state of the art architectures on the Caltech-101 dataset. DV: DTS-VGG, SPP: Spatial Pyramid Pooling [123], VGG [36], E-CNN: Exemplar CNN [92], EP: Epitomic Networks [98], ZF: Zieler and Fergus [105]

Dataset	State-of-the-art Architectures					
	DV	SPP	VGG	E-CNN	EP	ZF
Caltech-101	8.78	**6.6**	7.3	8.5	12.2	13.5

5.3 Discussions

The proposed DTSCNN architectures, when trained from scratch, outperforms the corresponding original CNN architectures on small datasets by a useful margin. For larger training datasets, the proposed networks give similar error compared to the original architectures. Faster rate of convergence is observed in both cases for shallow as well as deep architectures.

The DTCWT scattering front-end is mathematically designed to deal with all boundary orientations equally and with two or more scales, as required. The generic

nature of the DTCWT scattering front-end is shown by its similar classification performance to the front-end of learned networks, on two different datasets. The generic features are likely to give it wide applicability to both small and large image datasets as it provides lower (small dataset) or similar classification error (large dataset), with faster rates of convergence.

Future work includes extending the DTCWT Scattering Network front-end for other learning frameworks, with the aim of improving the learning rates further.

Chapter 6

ScatterNet Hybrid Deep Learning (SHDL) Networks

We have shown that the hand-crafted ScatterNet features improve the learning and generalization of several deep networks in the previous chapter. Despite this, the training of deep networks is still slow and requires more training samples than are available or desired. This chapter proposes the ScatterNet Hybrid Deep Learning (SHDL) Networks that are constructed using the ScatterNet Hybrid framework presented in Chapter 3. These SHDL networks rapidly learn hierarchical features from mainly unlabelled input signals. Also, the SHDL networks are computationally efficient as compared with other deep learning networks, making them an attractive choice. We now propose two SHDL networks to solve the individual image understanding tasks of object recognition and semantic segmentation.

SHDL networks are composed of a hand-crafted front-end, an unsupervised learning mid-section module, and a supervised learning based back-end as explained in Chapter 3. Both the proposed networks use the hand-crafted features extracted by the Multi-resolution Parametric Log Scattering Network (as presented in Section 4.4) to learn the mid-level features.

The first network, termed as the Deterministic ScatterNet Hybrid Deep Learning (D-SHDL) network uses the hand-crafted features with the *deterministic unsupervised learning* module to learn symmetrically distributed hierarchical mid-level features. The unsupervised learning module is composed of two stacked PCA layers with parametric log non-linearity. A supervised orthogonal least squares (OLS) layer [1, 43] is then applied to the concatenated features obtained from the unsupervised layers to select a subset of object-class-specific features. The features are chosen without undesired bias from the outliers due to the introduced symmetry. The selected features are

finally fed into a Gaussian-kernel support vector machine (G-SVM) to perform object classification.

This network is fast to train as compared to other unsupervised learning modules (autoencoders or RBMs) as the minimization of the loss function with the PCA layers can be obtained in its simplistic form as the Eigen decomposition. Despite the favorable increase in the rate of learning, we have found that the approximate solution of PCA loss function may produce undesired checkerboard filters which can limit the performance of these networks (section 6.1).

The second proposed network, coined as the Generative ScatterNet Hybrid Deep Learning (G-SHDL) network uses the *generative unsupervised learning* module to learn the features from the hand-crafted descriptors, and it turns out that these do not produce checkerboard patterns (section 6.4). A conditional random field is applied to the features obtained from the hierarchical features to accomplish the second image understanding task of semantic segmentation.

Sparse coding and RBMs are the most effective unsupervised learning algorithms that have been successfully used in developing powerful image representations. For example, Zeiler et al. [166] and Kavukcuoglu et al. [165] developed algorithms for convolutional sparse coding, which approximately solves the optimization problem to minimize the reconstruction error between the data and the higher layer features convolved with the filters. Compared to sparse coding, RBMs can compute posterior probabilities in a feedforward way, which is usually orders of magnitude faster. This computational efficiency provides a significant advantage over sparse coding since it scales up to a much larger number of codes [167].

This motivated us to use the Restricted Boltzmann Machines (RBMs) as the mid-level unsupervised learning module. RBMs have been successfully used to learn high-level structure in a wide variety of domains, including hand- written digits [163] and human motion capture data [164]. While RBMs were successful in controlled domains, scaling them to realistic-sized images was challenging. Lee et al. [161] proposed stacked convolutional RBMs which were demonstrated to learn hierarchical representations for realistic-sized images. Therefore we use stacked layers of convolutional Restricted Boltzmann Machine (cRBMs) as the unsupervised learning module to learn the desired hierarchical features from the handcrafted descriptors.

Training of convolutional RBMs is slow as the partition function is approximated by sampling using MCMC [161]. To accelerate the training, the filters in each RBM layer are initialized with structural priors (filters) learned using PCA as opposed to random initialization. We have shown that the structural priors accelerate the training

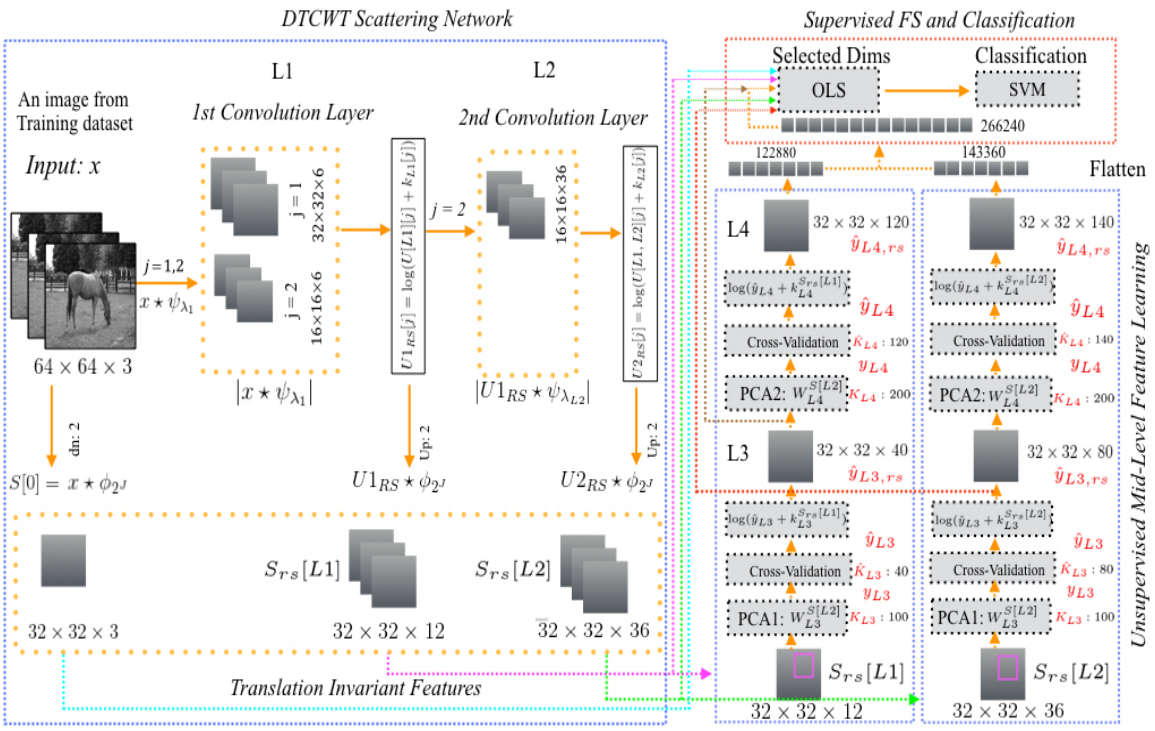


Fig. 6.1 D-SHDL: The illustration shows the input image (64×64 (x)) from the CIFAR-10 dataset at resolution R1 decomposed to extract the translation invariant relatively symmetric coefficients at L0 ($S_{rs}[L0]$), L1 ($S_{rs}[L1]$) and L2 ($S_{rs}[L2]$). Features at the higher level of abstraction are captured at L3 and L4 layers of the PCA-Net using unsupervised learning. Parametric log transformation is applied on the output of each PCA stage to introduce relative symmetry. The representations extracted at each stage (L0, L1, L2, L3, L4) are concatenated and given to the supervised OLS layer that select the object-specific features finally used for classification using the Gaussian SVM (G-SVM).

of RBMs. Since it is extremely fast to learn the filters or the structural priors using PCA (eigen-decomposition), the whole process is much quicker than training RBMs with random weight initialization.

The number of filters in each layer of both of these unsupervised learning modules are optimized as part of the automated design process. The optimization of the number of filters in a layer leads to more efficient learning of the subsequent layer as the filters are now learned from a smaller feature space. This results in improved computational performance as compared to the more usual deep network architectures.

The next sections present the deterministic and generative SHDL networks. The details of the experimentation performed with each network are also given along with the discussions of their performance.

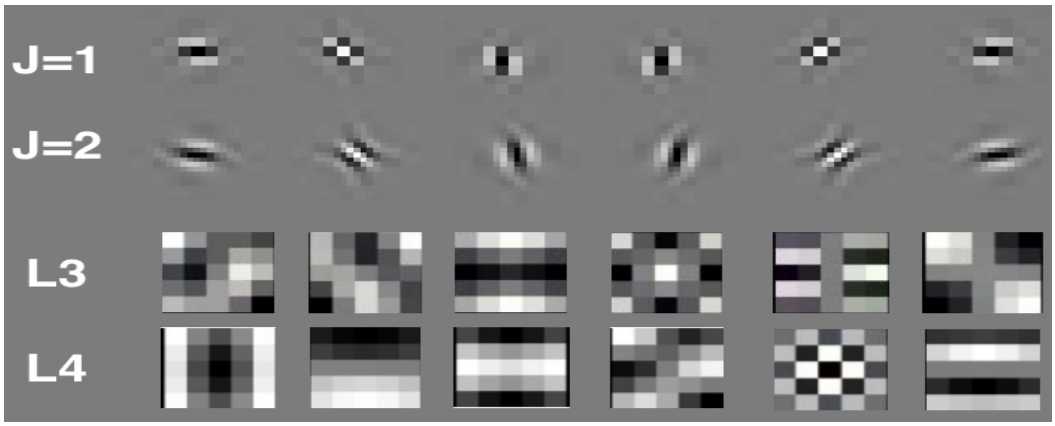


Fig. 6.2 Illustration shows the DTCWT real filters at two scales used at Layer L1 and L2. The filters learned by the PCA-Net at L3, and L4 stage are also shown.

6.1 Deterministic ScatterNet Hybrid Deep Learning (D-SHDL) network

The section introduces the Deterministic ScatterNet Hybrid Deep Learning (D-SHDL) network for object classification. This framework first extracts the hand-crafted feature descriptors from the input signal using the Multi-resolution Parametric Log Scattering Network. These features are used by the layers of the PCA based unsupervised learning module to learn hierarchical features that capture intricate structure between different object classes. Next, from the concatenated unsupervised feature hierarchies, an Orthogonal least squares (OLS) based supervised learning layer selects the features specific to each object class which are finally used for classification. Each layer of the network is designed and optimized using cross-validation so that it produces the desired computationally efficient architectures. The architecture of the Deterministic ScatterNet Hybrid Deep Learning (D-SHDL) network is shown in Fig. 6.1.

6.1.1 ScatterNet Hand-crafted Descriptors

The Multi-resolution Parametric Log Scattering Network extracts feature coefficients from the input signal (x) which are typically formed from three layers: $x \star \phi$ (Layer 0), $S_{rs}[L1]$ (Layer 1) and $S_{rs}[L2]$ (Layer 2) for each of the two image resolutions R1 and R2, as detailed in Section 4.4.

The scattering coefficients obtained at each layer are:

$$S = \begin{pmatrix} x \star \phi_{2^J}(L0) \\ U1[\lambda_{m=1}] \star \phi_{2^J}(L1) \\ |U1[\lambda_{m=1}]| \star \psi_{\lambda_2}| \star \phi_{2^J}(L2) \end{pmatrix} \tag{6.1}$$

6.1.2 Unsupervised Learning Module: PCA-Net Layers

The hand-crafted descriptors extracted at $L1$ or $L2$ as explained in the previous section are used by the stacked PCA [44] (see Appendix C) layers based unsupervised learning module to learn symmetrically distributed hierarchical mid-level features at $L3$ and $L4$ of the D-SHDL network, as shown in Fig. 6.1.

The formulation is presented for the $S_{rs}[R1, L1]$ invariant features obtained for R1 resolution image at layer L1. The same formulation can be similarly applied to $S_{rs}[R1, L2]$ (features for R1 resolution at layer L2) as well as features extracted at the R2 resolution at both layers ($S_{rs}[R2, L1]$ and $S_{rs}[R2, L2]$).

The objective of the PCA layer is to minimize the reconstruction error by learning a family of multi-channel orthonormal filters. In order to learn the filters, M overlapping patches (which capture object sub-parts) of size $z_1 \times z_2$ are collected from each channel of the input $S[R1, L1]$ i.e., $x_1, x_2, ..., x_M \in R^{z_1 z_2 \times P}$ where x is the sampled patch, M represents the number of patches and P (12 and 36, Fig. 6.1) represents the number of channels of the input. After this, the patch mean is subtracted to obtain $\tilde{X} = [\tilde{x}_1, \tilde{x}_2, ..., \tilde{x}_M]$, where $\tilde{x}$ is a mean-removed patch. Given N training images, we get the unified matrix:

$$X = [\tilde{X}_1, \tilde{X}_2, ..., \tilde{X}_N] \in R^{z_1 z_2 M \times PN}. \tag{6.2}$$

This filters are learned by minimizing the following equation,

$$\min_{W_{L3} \in R^{sL_3 sL_3 \times P \times K_{L3}}} \|X - W_{L3} W_{L3}^T \, X\|_F^2, \text{ s.t. } W_{L3}^T W_{L3} = I_{K_{L3}}, \tag{6.3}$$

where W_{L3} represents the learned filters at layer L_3 with size $sL_3 \times sL_3 \times P \times K_{L3}$, where K_{L3} represents the number of filters. These K_{L3} filters are learned by simply by obtaining the top K_{L3} principal eigenvectors of XX^T. These learned filters (Fig. 6.2) capture the variance of the training dataset.

The output responses of the $L3$ layer can be obtained as:

$$y_{L3} = S_{rs}[R1, L1] \star W_{L3}^{sL_3 \times sL_3 \times P \times K_{L3}}, i = 1, 2, 3, ..., N \tag{6.4}$$

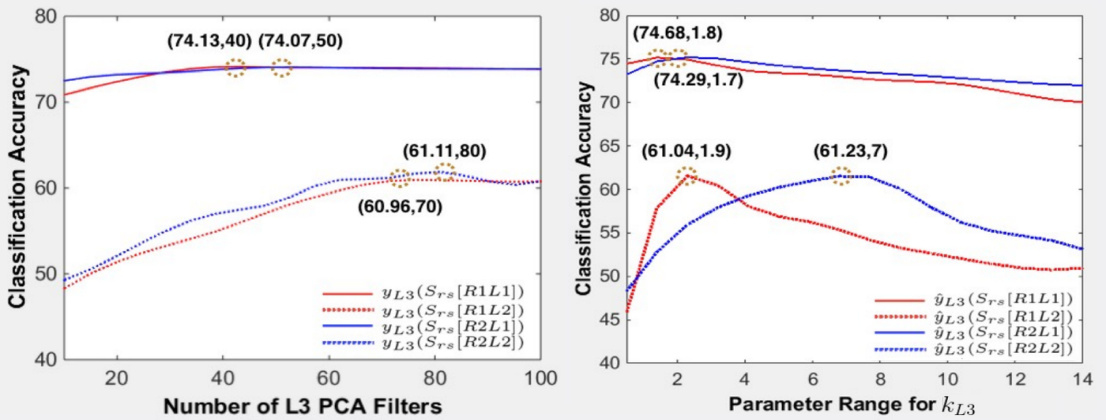


Fig. 6.3 Illustration presents the optimization for the number of filters (K_{L3}) learned at L3 layer as well as the log non-linearity parameter k_{L3}, applied on the L3 layer feature representations, using 5-CV classification. The graph that shows the optimization of the filters is shown in (a) while the graph that presents the selection of the optimal non-linearity parameter is shown in (b). The number of filters ($\hat{K}_{L3}$) and the chosen parameter value ($\hat{k}_{L3}$) that result in the highest accuracy are marked with dotted yellow circles on the graphs.

$S_{rs}[R1, L1]$ is zero-padded before convolving with W_{L3} so as to make y_{L3} have the same size as $S_{rs}[R1, L1]$.

Next, G-SVM is used at the output of the L3 layer, with the varying number of filters (10,20,..,K_{L3}), to select the optimal number ($\hat{K}_{L3}$) of learned filters that result in the highest five-fold cross-validation accuracy (5-CV) on the training dataset (Fig. 6.3(a)). The optimum output at L3 layer ($\hat{y}_{L3}$) is computed using the optimum number of filters ($\hat{K}_{L3}$). Next, parametric log transformation is applied on $\hat{y}_{L3}$ to introduce relative symmetry to its amplitude distribution:

$$\hat{y}_{L3,rs} = \log(\hat{y}_{L3} + \hat{k}_{L3}) \tag{6.5}$$

The optimal parameter ($\hat{k}_{L3}$) for the log non-lineairty is obtained by again computed using 5 fold cross validation (5-CV) as shown in Fig. 6.3(b).

Next, K_{L4} filters with weights W_4 at layer $L4$ can be learned similarly:

$$\min_{W_{L4} \in R^{sL_4 sL_4 \times K_3 \times K_{L4}}} \|X^{L3} - W_{L4} W_{L4}^T X^{L3}\|_F^2, \text{ s.t.} \quad W_{L4}^T W_{L4} = I_{K_{L4}}, \tag{6.6}$$

where X^{L3} represents the matrix computed by extracting patches from $\hat{y}_{L3,rs}$ (L3 output (relatively symmetric (rs)) obtained using the optimal ($\hat{K}_{L3}$) number of filters).

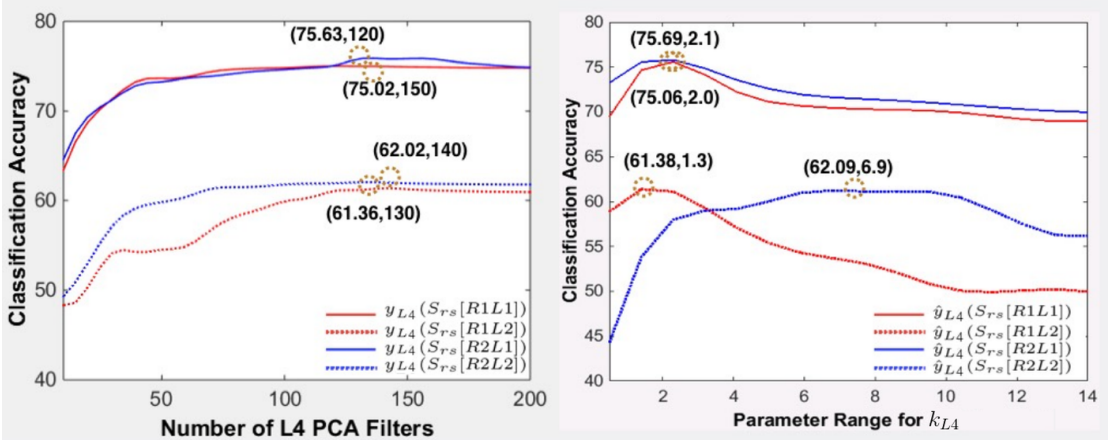


Fig. 6.4 Illustration presents the optimization for the number of filters (K_{L4}) learned at L4 layer as well as the log non-linearity parameter k_{L4}, applied on the L4 layer feature representations, using 5-CV classification. The graph that shows the optimization of the filters is shown in (a) while the graph that presents the selection of the optimal non-linearity parameter is shown in (b). The number of filters ($\hat{K}_{L4}$) and the chosen parameter value ($\hat{k}_{L4}$) that result in the highest accuracy are marked with dotted yellow circles on the graphs.

The output response at Layer $L4$ can be computed as shown:

$$y_{L4} = \hat{y}_{L3,rs} \star W_{L4}^{sL_4 \times sL_4 \times K_{L3} \times K_{L4}}, i = 1,2,3,.....,N \tag{6.7}$$

Here, $\hat{y}_{L3,rs}$ is also zero padded before applying the convolutions as described above. The optimal L4 output ($\hat{y}_{L4}$) is computed using $\hat{K}_{L4}$ filters, obtained using five-fold cross-validation (5-CV) as shown in Fig. 6.4 (a). Parametric log transformation is finally applied on $\hat{y}_{L4}$ to introduce relative symmetry:

$$\hat{y}_{L4,rs} = \log(\hat{y}_{L4} + \hat{k}_{L4}) \tag{6.8}$$

$\hat{k}_{L4}$ is obtained using five-fold cross-validation (5-CV) (similar to $\hat{k}_{L3}$) as shown in Fig. 6.4(b).

The effect of parametric log transformation applied to the features extracted at layers, L3 and L4 of the D-SHDL network are presented in Fig. 6.5.

6.1.3 Supervised Learning Module: OLS and G-SVM

The features obtained from each layer of the network (L0, L1, L2, L3, L4) for both R1 and R2 images are concatenated, normalized across each dimension and fed to

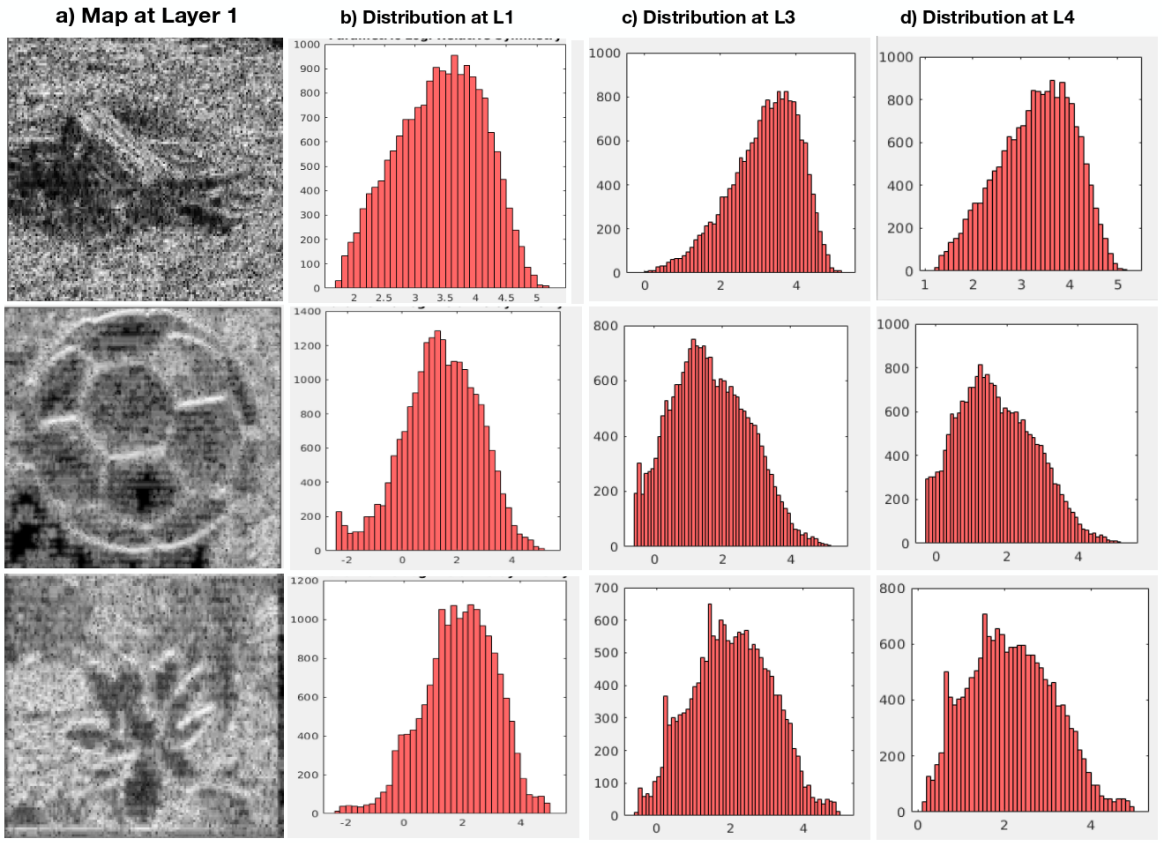


Fig. 6.5 Illustration shows the impact of the parametric log transformation used to the features extracted at layers, L3 and L4 of the D-SHDL network. The feature map extracted at $j = 1$ and 15° orientation at R1 resolution (with relatively symmetric feature amplitudes) from L1 of the D-SHDL network is used to produce the L3 features. The parametric log non-linearity is applied to the L3 features to introduce the symmetry in the amplitudes as explained in Section 6.1.2. Similarly, the parametric log non-linearity is applied to the L4 features to introduce the symmetry in the amplitudes. The effect of the non-linearity is shown using three randomly selected images taken from the Caltech-101 Dataset, similar to Fig. 4.3.

the OLS as shown in Fig. 6.1. Orthogonal least square (OLS) regression [**?**]selects discriminative features specific to class C in a supervised way using one-versus-all linear regression. The regression is applied to the training set of scattering features where each vector of N (Cifar: N $\approx$ 176000, Caltech: N $\approx$ 474000) dimensions is reduced to N' (Cifar: N' $\approx$ 10300, Caltech: N' $\approx$ 21000) selected dimensions. The dimensions are selected using 5-fold cross validation on the training dataset. The reduced scattering features dataset represents the metric defined in Eq. 1.1. The reduced training feature dataset is utilized by the G-SVM to learn weights that best discriminate the classes in

the dataset. Feature selection results in limited dimensions that lead to more efficient training of the G-SVM and improves generalization.

6.2 Overview of the D-SHDL Results

The performance of the SHDL network is evaluated on CIFAR-10 and Caltech-101 datasets. CIFAR-10 contains a total of 50000 training and 10000 test images each of size 32×32. The Caltech-101 dataset is an unbalanced image dataset with images of different sizes. In these experiments, 30 images (resized to 128×128) per class (clutter class removed) are used for training, 10 for validation and the rest of the images in each class are used for testing. Average per class classification results is reported with an averaging over five random splits. A detailed comparison with unsupervised, semi-supervised, and supervised methods is also presented.

6.2.1 ScatterNet feature extraction

The scattering representations are extracted by first obtaining multi-resolution images of size (64 × 64 (R1) and 48 × 48 (R2)) for CIFAR-10 and (256 × 256 (R1) and 192 × 192 (R2)) for Caltech-101, as described in Section. 2.1. The images in the CIFAR dataset are decomposed for each color channel separately using DTCWT filters at 5 (for R1), and 4 (for R2) scales respectively, while the images in the Caltech dataset are decomposed at 6 and 5 scales for R1 and R2 resolutions respectively. Next, log transformations are applied to the representations obtained (except at the coarsest scale) for both the R1 and R2 pipeline with parameters $k_{j=1} = 1.1$, $k_{j=2} = 3.8$, $k_{j=3} = 3.8$, $k_{j=4} = 7$ and $k_{j=5} = 6.8$ for CIFAR-10 images. The parameters $k_{j=1} = 1.05$, $k_{j=2} = 1.34$, $k_{j=3} = 1.94$, $k_{j=4} = 2.79$ and $k_{j=5} = 3.3$, are used for the Caltech-101 images. These parameters are obtained by averaging the individual k value for the particular scale, for all the images in the training dataset as described in Section 4.4. The classification accuracies for each layer (L0, L1, L2) and the concatenated features (HC = S[L0, L1, L2]) are presented for both resolutions, using G-SVM in Table 6.1. L2 features give a less good performance on their own than L1, probably due to their lower energies, but still, give a useful improvement when combined with L1.

6.2.2 PCA Layers: features and layer optimization

The L3 PCA layer of the network is trained on $S_{rs}[R1L1]$, $S_{rs}[R1L2]$, $S_{rs}[R2L1]$ and $S_{rs}[R2L2]$, to learn K_{L3}=100 filters of size 5×5 ($s_{L3} \times s_{L3}$) for the CIFAR-10 dataset.

Table 6.1 Accuracy (%) on CIFAR-10 for features extracted at different layers and resolutions. $S_{rs}[Layer]$, HC = S[L0, L1, L2]

	S[L0]	S [L1]	$S_{rs}[L1]$	S[L2]	$S_{rs}[L2]$	HC	HC_{rs}
R1	53.26	71.48	**72.58**	60.34	**60.51**	80.7	**81.7**
R2	55.14	72.04	**73.39**	60.12	**60.39**	80.9	**81.9**

The filter size (5×5) is same for the Caltech-101 dataset. Cross-validation is used (as explained in Section 6.1.2) on the L3 layer output (y_{L3}) to select the 40, 70, 50 and 80 optimal filters ($\hat{K}_{L3}$) for the four cases (selected from 100 learned filters), as shown in Fig. 6.4(a). Next, the relatively symmetric L3 output ($\hat{y}_{L3,rs}$) is obtained by applying a log transformation with $\hat{k}_{L3} = 1.8$, 1.9, 1.7 and 7.0 on $\hat{y}_{L3}$, respectively (Fig. 6.4(b)). The L3 PCA layer filters for the Caltech-101 dataset are 115, 175, 125, and 190 optimal filters ($\hat{K}_{L3}$), selected from 200 learned filters, for $S_{rs}[R1L1]$, $S_{rs}[R1L2]$, $S_{rs}[R2L1]$ and $S_{rs}[R2L2]$ respectively. The log non-linearity parameters are for the four cases are ($\hat{k}_{L3}$) 0.9, 1.1, 1.34 and 2.9 respectively.

L4 PCA layer is trained on L3 layer outputs ($\hat{y}_{L3,rs}$) correspondingly to learn 200 (K_{L4}) filters, of size 5×5 ($s_{L4} \times s_{L4}$). The filter size (5×5) is same for the Caltech-101 dataset. Similarly, 150, 140, 120 and 130 optimal filters ($\hat{K}_{L4}$) are selected for the four cases ($S_{rs}[R1L1]$, $S_{rs}[R1L2]$, $S_{rs}[R2L1]$ and $S_{rs}[R2L2]$), as shown in Fig. 6.5(a). Next, the relatively symmetric L4 outputs ($\hat{y}_{L4,rs}$) are obtained by applying a log transformation with $\hat{k}_{L4} = 2.0$, 1.3, 2.1 and 7.2 on $\hat{y}_{L4,rs}$, respectively, for the four cases, as shown in Fig. 6.5(b). The L4 PCA layer filters for the Caltech-101 dataset are 210, 235, 310, and 335 optimal filters ($\hat{K}_{L4}$), selected from 400 learned filters, for $S_{rs}[R1L1]$, $S_{rs}[R1L2]$, $S_{rs}[R2L1]$ and $S_{rs}[R2L2]$ respectively. The log non-linearity parameters are for the four cases are ($\hat{k}_{L4}$) 1.2, 1.38, 1.95 and 3.3 respectively. The PCA layers are trained using the software presented in Appendix D.3.

The five-fold cross-validation (5-CV) classification accuracies on CIFAR-10, obtained using G-SVM, at different stages of Layers L3 and L4 are presented in Table 6.2. There

Table 6.2 5-CV Accuracy (%) on CIFAR-10 at L3 and L4. y_{L3}, y_{L4} output, $\hat{y}_{L3}$, $\hat{y}_{L4}$ optimal output and $\hat{y}_{L3,rs}$, $y_{L4,rs}$ relatively symmetric output, at L3 and L4.

	y_{L3}	$\hat{y}_{L3}$	$\hat{y}_{L3,rs}$	y_{L4}	$\hat{y}_{L4}$	$\hat{y}_{L4,rs}$
$S_{rs}[R1L1]$	73.83	74.13	74.68	74.81	75.02	75.06
$S_{rs}[R1L2]$	60.78	60.96	61.04	60.93	61.36	61.38
$S_{rs}[R2L1]$	73.86	74.07	74.29	74.88	75.63	75.69
$S_{rs}[R2L2]$	60.81	61.11	61.23	61.78	62.02	62.67

Table 6.3 Object classification accuracy (%) on CIFAR-10 and Caltech-101 for each module computed with OLS and G-SVM. The increase in accuracy with the addition of each layer is also shown. HC: Hand-crafted, PCA features ($(Layer)_{filter-size}$): eg $(L3)_{s_{L3}=5}$

Accuracy	HC	$HC,(L3)_5$	$HC,(L3,L4)_5$	$HC,(L3,L4)_{3,5}$
CIFAR-10	82.40	82.80	83.50	83.90
Caltech-101	76.30	78.10	80.88	81.46

are fewer optimal filters in ($\hat{K}_{L3}$, $\hat{K}_{L4}$) than the initially learned filters (K_{L3}, K_{L4}) but produce an equal or higher cross-validation accuracy. This suggests that some of the filters learn redundant information which can be removed. This results in efficient learning of L4 layer (subsequently for OLS as well as SVM) as the L4 filters are learned from a smaller feature space $\hat{y}_{L3,rs}$ (obtained with $\hat{K}_{L3}(40,70,50,80) << K_{L3}(100)$).

6.2.3 Classification performance

This section evaluates the classification performance of each module of the D-SHDL network. The classification accuracy of each module is presented by applying the supervised OLS layer on the features to select the relevant features which are then fed to the G-SVM to compute the accuracy. The accuracy of the handcrafted module (HC) is computed on the concatenated symmetric features extracted at L0, L1, L2, for both resolutions (R1, R2) using OLS for feature selection and then G-SVM for classification.

The hand-crafted module produced a classification accuracy of 82.4% (HC) on CIFAR-10 as shown in Table 6.3. An increase of 0.4% is observed when the mid-level features, learned at L3 with s_{L3}=5 are concatenated with the features of the hand-crafted module (HC,$(L3)_{s_{L3}=5}$), again for both R1 and R2. A further increase of 0.7% (HC,$(L3,L4)_{s_{L3,L4}=5}$) is noticed when mid-level features from the L4 layer learned with s_{L4}=5 are concatenated to (HC,$(L3)_{s_{L3}=5}$) features. This suggests that the PCA layers (L3 and L4) learn useful image representations as they improve the classification performance. Finally, to test the optimality of the filter sizes, the L3 and L4 layers were also trained with 3×3 ($s_{L3}\times s_{L3}$) and 3×3 ($s_{L4}\times s_{L4}$). A further increase of around 0.4% ($HC,(L3,L4)_{3,5}$) is observed by concatenating the features obtained at L3 and L4 layers, with filters trained with the kernel s_{L3}, s_{L4} of size 3 and 5, with the hand-crafted module (HC). This suggests that filters of different sizes learn unique and useful image representations.

Next, the classification accuracy is computed for the Caltech-101 dataset. As seen from the Table 6.3, the accuracy increases from (76.30% to 81.46%) with the use of representations obtained from the deeper layers of the D-SHDL network. The network

results in an accuracy of 81.46% by using the concatenated features obtained at L3 and L4 layers, with filters trained with the kernel s_{L3}, s_{L4} of size 3 and 5, with the hand-crafted module (HC), as shown in Table 6.4.

6.2.4 Comparison with the state-of-the-art

This section presents the comparison of the D-SHDL network with the state-of-the-art methods from the unsupervised, semi-supervised and supervised domains as shown in Table 6.4.

Table 6.4 Object classification accuracy (%) and comparison with other approaches on both datasets. Unsup: Unsupervised, Semi: Semi-supervised and Sup: Supervised.

Dataset	SHDL	Semi	Unsup	Sup
CIFAR-10	83.90	83.3 [47]	82.9 [41]	**96.2 [48]**
Caltech-101	81.46	81.5 [49]	81.0 [42]	**92.7 [36]**

The methods used for comparison are briefly described below.

Unsupervised Methods: Lin et al. [41] proposed an unsupervised method that used multi-layer deep architecture for representation learning. The deep network made use of K-means clustering for feature learning and the nonnegative variant of Orthogonal matching pursuit (OMP) for representation encoding. These representations were finally used to perform classification and produced a classification accuracy of 82%. The unsupervised learning model proposed in [42] encoded the local visual features in a codebook which were then pooled into histograms at several spatial granularities. These histogram features were utilized for classification using a support vector machine (SVM) and performed with an accuracy of 81% on Caltech-101.

Semi-Supervised Methods: Dei et al. [49] proposed a simple, yet effective feature learning method to exploit the available, unlabeled data. By using two consistency assumptions, they generated a diverse set of training data for surrogate classes to learn visual attributes in a discriminative way. By doing so, images were classified and linked to the surrogate classes. The images were represented with their affinities to a rich set of discovered image attributes which finally resulted in 81% classification performance. Salimans et al. [47] trained the Generative adversarial networks (GANs) in a semi-supervised setting to learn rich features which were used for classification. This method produced a classification accuracy of 83%.

Supervised Methods: Two very deep networks [36, 48] were trained end-to-end using supervised learning to learn hierarchical features and produced classification

accuracy of more than 90% on both datasets. However, these networks require sizeable computational resources to train along with large labeled training examples which may not be available for most applications.

The D-SHDL outperformed the semi-supervised and unsupervised learning methods on both datasets. However the network underperformed by nearly 13% against supervised deep learning models. However, the advantages over the end-to-end supervised methods are detailed in the next section.

6.2.5 Advantage over supervised learning

The end-to-end training of supervised models requires large training datasets which may not exist for most applications. Table 6.5 shows that D-SHDL network outperformed VGG [36] and Network in Network (NIN) [35] on the CIFAR-10 datasets with less than 2k images. The experiments were performed by dividing the training dataset of 50000 images into eight datasets of different sizes. The images for each dataset are obtained randomly from the full 50000 training dataset. We ensure that an equal number of images per object class are sampled from the training dataset. The whole test set of 10000 images is used for all the experiments. Deeper models like NIN [35] and VGG [36] result in low classification accuracy due to their inability to train on the small training dataset.

Table 6.5 Comparison of SHDL network on accuracy (%) with two supervised learning methods (VGG [36] and NIN [35] against different training dataset sizes on CIFAR-10.

Arch.	500	1K	2K	5K	10K	20K	50K
SHDL	**50.3**	**57.9**	**63.4**	68.6	72.3	78.4	83.9
NIN	15.6	54.5	61.1	**72.9**	**81.2**	**86.7**	89.6
VGG	10.3	10.7	43.4	63.4	72.0	83.1	**92.7**

6.2.6 Computational Complexity

This section presents the computational complexity of different modules of the D-SHDL network.

As mentioned in Section 4.7, the ScatterNet front-end extracts the features for a batch of 128 images of size $64 \times 64 \times 3$ and $48 \times 48 \times 3$ of the CIFAR dataset (using dual-tree wavelets at 5 scales and 6 orientations) with the Titan X Pascal GPU, in 0.41 seconds. The features for the complete CIFAR dataset of 60,000 images are extracted in 6 minutes.

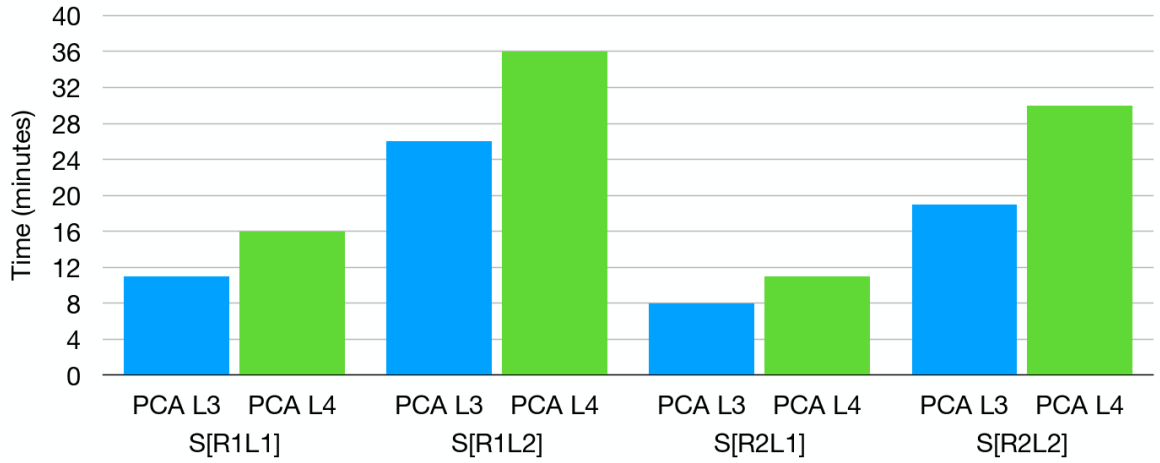


Fig. 6.6 The illustration presents the computational time required to learn the filters at Layers, L3 and L4, for the following four cases: $S_{rs}[R1L1]$, $S_{rs}[R1L2]$, $S_{rs}[R2L1]$ and $S_{rs}[R2L2]$.

Next, we present the computational time required to learn the filters at Layers, L3 and L4, for the following four cases: $S_{rs}[R1L1]$, $S_{rs}[R1L2]$, $S_{rs}[R2L1]$ and $S_{rs}[R2L2]$. The major time is spent on extracting the patches (at each pixel location) from the hand-crafted feature representations obtained at layers L1 or L2. To speed up this process, we used multiple cores to extract the patches in parallel which are then used to learn the desired L3 and L4 filters using Eigen decomposition.

As observed from Fig. 6.6, the time required to learn the filters for the L4 layer is higher for both resolutions (R1, R2) as the number of filters to be determined is more than for layer L3. Also, the time required to learn the filters at resolution R2 is lower than the R1 resolution as the number of patches to be extracted are fewer due to the smaller size of the feature representations obtained for the R2 resolution at layers, L1 or L2.

The features extracted from the layers L0, L1, L2, L3, and L4 are concatenated for both resolutions. These concatenated features are used by the orthogonal least squares (OLS) for feature selection. The OLS module reduces the dimensions of the feature vector from 176000 to 10300 in 33 minutes. The OLS performs feature selection for class separately by using one vs. all regression. Multiple cores are used to assign individual works to each regressor which significantly accelerates the feature selection process. The selected features are given as input to the Gaussian SVM which takes another 30 minutes to learn the desired classification model. The classifier is also trained in a one vs. all manner to achieve faster training.

The total taken by the D-SHDL network to learn the desired representations for classification is around 1 hour on the CIFAR-10 dataset as opposed to the CNNs which takes significantly larger [162] (3 hours (NIN), 4 hours (VGG), 9 hours (ResNet (110 layers)) to train for the reasons detailed in section 6.1. The time required to train our (D-SHDL) model on the Caltech-101 dataset is around 3 hours because of the large size of the input images.

6.3 Discussions

The D-SHDL network proposed in this work is inspired from the circuitry of the visual cortex. The proposed network uses a hand-crafted front-end to extract invariant edge features similar to the V1 of the cortex. These hand-crafted features are used by a PCA-based unsupervised learning module to learn mid-level features while OLS-based supervised learning is used to select features that aid the discriminative SVM learning. It is shown that a straightforward PCA based network can learn useful features that can be useful in improving the classification performance. The network has been shown to outperform unsupervised and semi-supervised learning methods while evidence of the advantages of D-SHDL network over supervised learning (CNNs) methods are presented for small training datasets. The D-SHDL network is also significantly faster (3x to 8x depending on the network) to train as the PCA layer filters can be obtained simply by computing the Eigen decomposition of the patches extracted from the hand-crafted features which is advantageous as training of deep CNNs requires extensive computational resources.

6.4 Generative ScatterNet Hybrid Deep Learning (G-SHDL) network

This section presents the Generative ScatterNet Hybrid Deep Learning (G-SHDL) network which is derived from the ScatterNet Hybrid framework presented in Chapter 3, similar to the D-SHDL network. This network is utilized to solve the task of semantic image segmentation which is the second image understanding task. G-SHDL network can also be used to address the task of object classification. However, G-SHDL is used to solve the semantic segmentation task as it is more complicated which requires the system to assign a label to each pixel of the images as opposed a label to the whole image in case of object recognition.

The G-SHDL is constructed by using the ScatterNet front-end to extract the translation invariant hand-crated features which are used by the four stacked layers of convolutional Restricted Boltzmann Machine (RBM) with PCA structural priors to rapidly learn an invariant hierarchy of features. The features obtained from the last RBM layer are then used by a supervised conditional random field (CRF) to achieve the semantic segmentation. The convolutional Restricted Boltzmann Machine (RBM) is selected as the unsupervised learning module as it is a generative model.

The G-SHDL network can learn meaningful hierarchical representations rapidly using mainly unlabelled data and produce the desired semantic segmentation using smaller labeled datasets. This is especially advantageous for this task as it can be expensive and time-consuming to generate pixel-wise annotations. The proposed network is also computationally efficient as the number of filters in each RBM layer are optimized using cross-validation. The filters in the subsequent layer are then also learned from a smaller feature space. The above-mentioned advantages may make the G-SHDL an attractive choice over the standard deep networks.

The Generative ScatterNet Hybrid Deep Learning (G-SHDL) network is shown in Fig. 6.7 and detailed in the following section.

6.4.1 ScatterNet Hand-crafted Descriptors

The parametric log based DTCWT ScatterNet is used to extract the relatively symmetric translation invariant hand-crafted features from the input signal x. This ScatterNet is identical to hand-crafted network used for the D-SHDL network as presented in Section 6.1.1 but extracts features only at the input resolution.

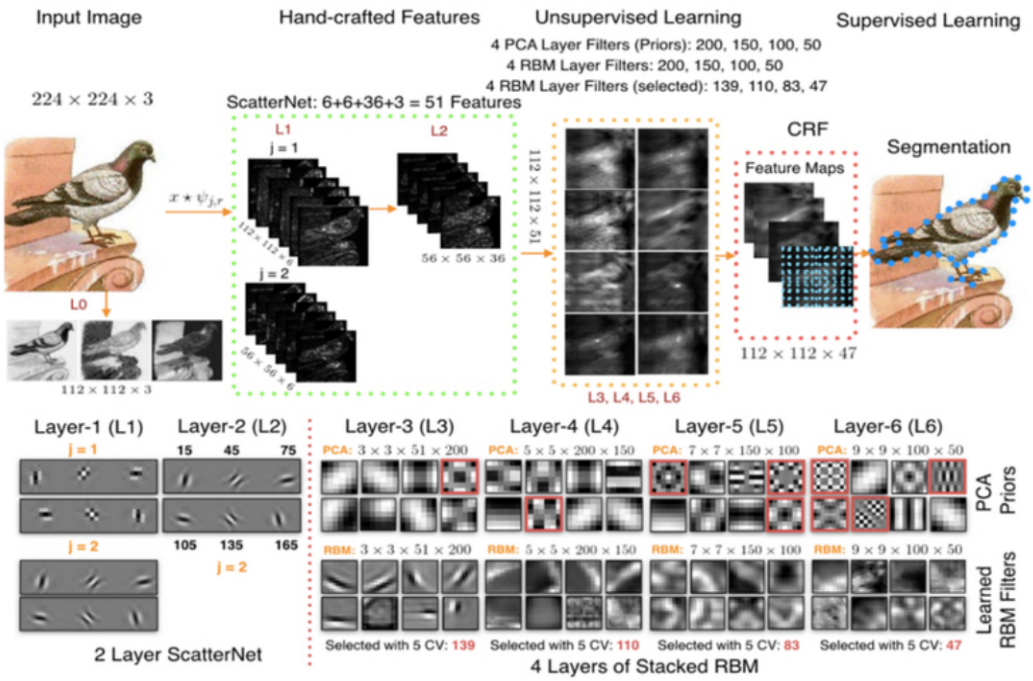


Fig. 6.7 The proposed G-SHDL network uses the ScatterNet front-end to extract hand-crafted scatternet features from the input image at L0, L1 and L2 using DTCWT filters at two scales and six fixed orientations (filters shown). The handcrafted features extracted from the three layers are concatenated and given as input to the four stacked convolutional RBM layers (L3, L4, L5, L6) with 200, 150, 100 and 50 filters to learn a hierarchy of features. Each RBM layer is initialized with PCA based structural priors with the same number of filters which improves their training as shown by L3 to L6 convergence graphs. The RBM layers are trained in a layer by layer greedy type fashion. Once an RBM layer is trained the optimal number of filters are selected using five-fold cross-validation that results in a computationally efficient architecture (Table. 1) as the later layers can feature from a smaller feature space. The features learned from the last RBM layer (L6) are used by the CRF for semantic image segmentation. PCA layers can learn the undesired checkerboard filters (shown in red) which are avoided and not used as the prior for the RBMs. To detect and remove the checkerboard filters from the learned filter set, we used the method defined in [71].

6.4.2 Unsupervised Learning: RBM with Priors

The Scattering features extracted at (L0, L1, L2) are concatenated and given as input to 4 stacked convolutional restricted Boltzmann machine (CRBM) [161] layers that learn 200, 150, 100 and 50 filters respectively.

The Convolutional RBM (CRBM) [161] is similar to the regular RBM (presented in section 2.2.6), but the weights between the hidden and visible layers are shared among all locations in an image. The basic CRBM consists of two layers: an input

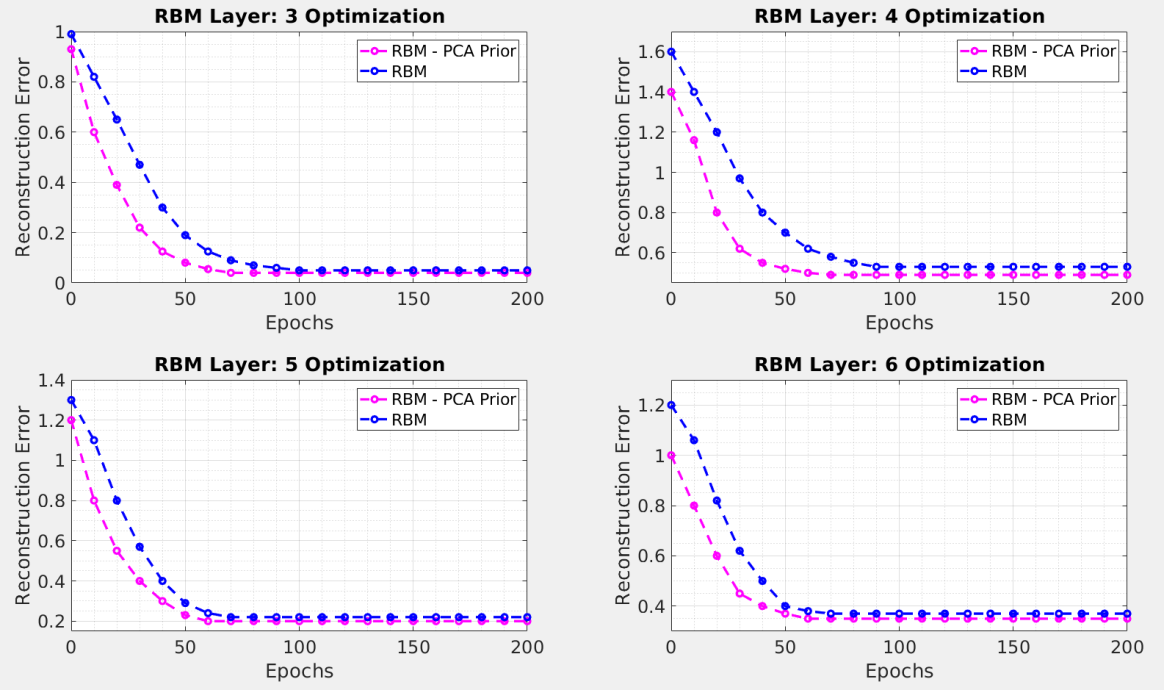


Fig. 6.8 The illustration presents the improvement in rate of learning for the RBM with the use of PCA priors as compared to using random initialisation.

.

layer V and a hidden layer H (corresponding to the lower two layers in Figure 1). The input layer comprises of $N_V \times N_V$ array of Gaussian units. The hidden layer consists of M "groups (No. of filters)," where each group is an $N_H \times N_H$ array of Gaussian units, resulting in N_H^2 M hidden units. Each of the M groups is associated with an $N_W \times N_W$ filter; the filter weights are shared across all the hidden units within the group. Also, each hidden group has a bias b_k and all visible units share a single bias c. Markov chain Monte Carlo (MCMC) sampling in the form of Gibbs sampling is used to approximate the likelihood and its gradient. The estimation of the likelihood of the RBM or its gradient for inference is computationally intensive [76]. However, initializing RBMs with priors on the hidden layer instead of a random initialization has been shown to improve the training [76].

We propose structural priors for each convolutional RBM layer (L3 to L6) which is shown to improve the training of the RBMs (Fig. 6.8 Graphs). The Structural priors are obtained using the PCA [44] layer that learns a family of orthonormal filters by minimizing the following reconstruction error:

$$\min_{V \epsilon \ R^{z_1 z_2 \times K}} \left\| X - VV^T X \right\|_F^2, \ s.t. \ VV^T = I_K \tag{6.9}$$

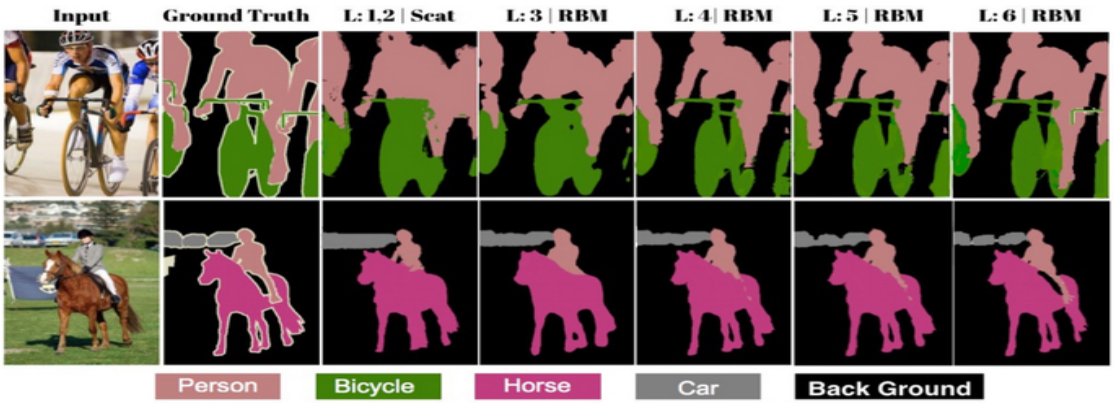


Fig. 6.9 Figure shows two images from MSRC dataset with their ground truth and segmentation obtained at L2 to L6 of G-SHDL.

where X are patches sampled from N training images (concatenated handcrafted features), I_K is an identity matrix of size $K \times K$. The solution of Eq. 6.9 in its simplified form represents K leading principal eigenvectors of XX^T obtained using Eigen decomposition. The PCA layers may learn undesired checkerboard filters. To detect the checker-board filters from the learned filter set, we use the method defined in [71]. These checkerboard filters are avoided as filter priors. Each RBM layer (L3, L4, L5, L6) of the G-SHDL is trained individually in a greedy fashion (with structural priors). Once the RBM layer is trained the filters that learn redundant information are removed using five-fold cross-validation. (Table 6.6 and section 6.4.2).

The CRBM's hidden units extract features from overlapping patches of visible units, and features of neighboring patches complement each other and cooperate to model the input. CRBM is trained on complete images or large regions of them to learn local features and exploit spatial relationship of overlapping patches. Because the hidden units of overlapping patches co-operate in a CRBM, an image pattern that is explained by one hidden unit in one neighborhood does not demand to be explained again in another overlapping patch. This reduces the redundancy of features.

6.4.3 Supervised CRF Segmentation

Conditional Random Field (CRF) is a probabilistic graphical model that uses the features obtained from the L6 RBM along with edge potentials computed on four pairwise connected grids [77] to perform the desired segmentation. L6 RBM features are similar to the metric defined in Eq. 1.1. The segmentation is obtained by minimizing the clique loss function with Tree-Reweighted [77] inference that uses the LBFGS optimization algorithm. The pairwise potentials are computed on four adjacency grid

Fig. 6.10 The illustration shows the L6 RBM features thresholded to the top 10, 20 and 30 activations and back-projected [105] to the input image selected from the Caltech-101 dataset. The L6 RBM features are most responsive to the beaks of the birds, then feet and wings.

because it is faster to perform the inference. However, this grid has limited expressive power as it only models local interactions. Larger grids can be considered for more accurate segmentation, yet, it may make the inference significantly slower.

6.5 Overview of the G-SHDL Results

G-SHDL was evaluated and compared with other segmentation frameworks on both MSRC [73] and Stanford Background (SB) [74] datasets. The MSRC dataset contains 572 images with 21 classes while the SB dataset is formed of 715 images with eight classes, where each image in both datasets has a resolution of 320×240. The quantitative results are presented with the class pixel accuracy which represents the ratio of correct pixels computed in a per-class (PA) [69] basis and then averaged over the total number of classes. The results are presented for 5-fold cross-validation for both datasets randomly split into 45% training, 15% validation and 40% test images for each fold. We provide a quantitative comparison against the state-of-the-art to evaluate the performance of G-SHDL.

6.5.1 Handcrafted Front-end: ScatterNet

ScatterNet features are extracted from the input RGB image using DTCWT filters at two scales and six fixed orientations. Next, log transformation with parameter $k_{j=1} =$

1.1 is applied to the representations obtained at the finer scale to introduce relative symmetry (Section 4.4). These parameters are used for both the datasets.

6.5.2 Unsupervised Mid-section: RBM with PCA priors

The four stacked convolutional RBM layers learn 200, 150, 100 and 50 filters respectively with PCA structural priors (obtained by training on the handcrafted features) in a greedy layer-wise fashion (Section 6.4.2). The convolutional RBM for each layer is trained using the following parameters: (i) Batch Size: 10 images (ii) Learning Rate: 0.05 which is dropped to 0.001 (iii) Standard deviation of the Gaussians at the visible units: 0.01 (iv) Number of Gibbs updates per weights update: 1. Once, each RBM layer is trained, five-fold cross-validation (5-CV) is computed with filters randomly selected from the trained filter set to evaluate the segmentation accuracies using CRF. We were able to achieve similar PA accuracy on the 5-CV with the fewer number of filters than the complete learned filter set. This suggests that some of the filters learn redundant information which can be removed. This results in efficient learning of subsequent layers as the filters are determined from a smaller feature space. The numbers of selected filters are shown in Table 6.6. The PCA and RBM layers are trained using the softwares presented in Appendix D.3 and D.4 respectively.

Table 6.6 5 fold cross validation performed on the training dataset of Stanford background (SB) ansd MSRC dataset to select the optimal numbers of filters for L3 to L6 RBM layers. L(size) = No. of Filters (a, a represents $a \times a$)

Filters	L3 (size)	43 (size)	L5 (size)	L6 (size)
PCA	200 (3,3)	150 (5,5)	100 (7,7)	50 (9,9)
RBM	200 (3,3)	150 (5,5)	100 (7,7)	50 (9,9)
Selected-SB	139	110	83	47
Selected-MSRC	163	136	92	48

6.5.3 Classification performance

This section presents the classification performance for each module of the G-SHDL network. The accuracy of the hand-crafted module (HC) is computed on the concatenated features relatively with symmetric amplitudes extracted at L0, L1, L2, for both resolutions (R1, R2) using CRF for segmentation on Stanford background (SB) dataset. The hand-crafted module produced a classification accuracy of 68.4% (HC) as shown in Table 6.7. An increase of approximate 4%, 2%, 2% and 2% is observed

when the mid-level features, learned at L3, L4, L5, and L6 are used by the CRF. This suggests that the RBM layers learn useful image representations as they improve the segmentation performance finally producing an accuracy of 78.21%.

Table 6.7 PA (%) on both dataset for each module computed with CRF. The increase in accuracy with the addition of each layer is also shown. HC: Hand-crafted. RBM Layers: L3, L4, L5 and L6.

Accuracy	HC	L3	L4	L5	L6
SB [73]	68.43	72.38	74.82	76.79	**78.21**
MSRC [74]	73.12	77.95	80.4	81.95	**83.90**

Next, the performance of the G-SHDL network is evaluated on the MSRC dataset. As seen from the Table 6.7, the accuracy increases from (73.12% to 83.90%) with the use of representations obtained from the deeper layers of the network. The network results in a segmentation accuracy of 83.90%. The segmentation results for two images from the MSRC dataset are shown in Fig. 6.9. The network can also be used to understand the features which are most important for the segmentation as shown in Fig. 6.10.

6.5.4 Comparison with the state-of-the-art

This section presents the comparison of the G-SHDL network with the state-of-the-art methods from the unsupervised, semi-supervised and supervised domains as shown in Table 6.8.

Table 6.8 PA (%) and comparison on both datasets. Unsup: Unsupervised, Semi: Semi-supervised and Sup: Supervised.

Dataset	G-SHDL	Semi	Unsup	Sup	
SB [73]	78.21	77.76 [80]	68.1 [81]	80.2 [82]	**87.2** [192]
MSRC [74]	83.90	83.6 [83]	74.7 [84]	89.0 [79]	**91.4** [159]

The methods used for comparison are briefly described below.

Unsupervised Methods: Coates et al. [81] used K-means clustering for learning large-scale representations from images which were then used to produce semantic segmentation with a pixel accuracy of 68.1% on the SB dataset. Rubinstein et al. [84] proposed a model which used dense correspondences between images to capture the sparsity and visual variability of the common object over the entire image database. The features learned by both the networks were finally used to perform 74.7% pixel accuracy on the MSRC dataset.

Semi-Supervised Methods: Souly et al. [80] proposed a semi-supervised Generative Adversarial Networks (GANs) framework which generated extra weakly labeled examples to train a multi-class classifier, acting as the discriminator. The features learned by the framework were later used to produce the desired segmentation and produced a pixel accuracy of 77% on the SB dataset. Liu et al. [83] proposed a non-parametric Random Forest framework to obtain an accurate quality measure of splitting by incorporating abundant unlabeled data. This framework was then used to produce the desired semantic segmentation with 83% pixel accuracy on the MSRC dataset.

Supervised Methods: Four supervised deep networks [79, 82, 159, 192] were trained end-to-end using supervised learning to learn hierarchical features which were used for semantic segmentation. Two earlier networks produced a pixel accuracy (PA) of greater than 80% on SB and MSRC datasets. The networks outperformed the proposed G-SHDL by a narrow margin on the SB dataset [79] while the performance difference was significant on the MSRC dataset [82]. The recent deep networks, DeepLab-CRF [192] and Deep Fully Connected Continuous CRFs method [159] outperformed the G-SHDL network by more than 10% as compared both datasets. Despite their superior performance, supervised networks require sizeable computational resources to train along with large labeled training examples which may not be available for most applications.

The G-SHDL outperformed the semi-supervised and unsupervised learning methods on both datasets; however, the network underperformed against the end-to-end deep learning models [79, 82, 159, 192], as shown in Table 6.8.

6.5.5 Advantage over Deep Supervised Networks

Deep Supervised models need large labeled datasets for training which may not exist for many application. The proposed G-SHDL network is compared with DeepLab-CRF [192] that combines ideas from deep convolutional neural networks and fully-connected conditional random fields, yielding a novel method able to produce semantically accurate predictions and detailed segmentation maps, while being computationally efficient. The comparison is performed between a DeepLab-CRF model whose deep network is pretrained on the ImageNet [96] dataset and one with randomly initialized weights. The training of the DeepLab-CRF method involves learning the weights of both the deep network and the CRF. Table 6.9 shows that our G-SHDL network outperformed the (pre-trained) DeepLab-CRF [192] on the SB dataset with 200 images or less due to the poor ability of this method to finetune the weights of the deep

network and learn the CRF parameters for limited training examples. The model with random weights initialization underperforms several as merely 500 images are insufficient to learn meaningful parameters for this model. The experiments were performed by dividing the training dataset into eight datasets of different sizes. For each size of the dataset, we ensured that an equal number of images per object class were sampled from the training dataset. The full test set was used for all experiment.

Table 6.9 Comparison of G-SHDL on PA (%) with DeepLab-CRF [192] against different training dataset sizes on SB dataset.

Arch.	50	100	200	300	400	500	572
G-SHDL	**40.3**	**59.9**	**66.4**	72.6	75.7	78.20	78.21
DeepLab-CRF	8.9	14.7	24.4	34.9	40.9	46.4	59.2
DeepLab-CRF (pre-trained)	17.9	44.7	60.4	**77.9**	**80.9**	**86.4**	**87.2**

6.5.6 Computational Complexity

This section presents the computational complexity of different modules of the G-SHDL network.

The ScatterNet front-end extracts the features for a batch of 128 images of size $320 \times 240 \times 3$ for both the Stanford Background (SB) and the MSRC datasets (using dual-tree wavelets at 2 scales and 6 orientations) with the Titan X Pascal GPU, in 0.71 seconds. The features for the complete Stanford Background (SB) dataset of 572 images and the MSRC dataset of 715 images are extracted in less than 5 minutes.

Next, the features from the all the layers of the ScatterNet are combined and used to learn the PCA priors at four layers. The PCA priors are learned for the L3, L4, L5 and L6 layers in approximately one and a half hours as shown in Fig. 6.11 top row.

The four CRBM layers at L3, L4, L5 and L6 layers are then learned using PCA priors in around 3 hours as shown in Fig. 6.11 middle and lower rows. The CRF takes another 13 minutes to compute the segmentation using the LBFGS algorithm (Titan X Pascal GPU).

The total taken by the G-SHDL network to learn the desired representations for segmentation is around 5 hours on both the Stanford Background (SB) dataset and the MSRC dataset. This is relatively lower than recurrent CNN's which require approximately 11 hours of training to learn meaningful representations that produce the desired segmentation.

6.5.7 Discussions

The G-SHDL network is a generative variant of the D-SHDL network proposed in Section 6.1. The G-SHDL network similar to the D-SHDL network uses the ScatterNet front-end to extract the low-level edge features which are used by the PCA layers to learn filters which capture hierarchical features. These filters are used as priors for the CRBM to learn refined filters using iterative contractive divergence learning. It is shown that simple PCA based priors can be used to improve the learning of CRBMs. The networks also outperform unsupervised and semi-supervised learning methods while they fall short compared to the performance of the supervised networks. However, this model is advantageous when the amount of labeled data is not available

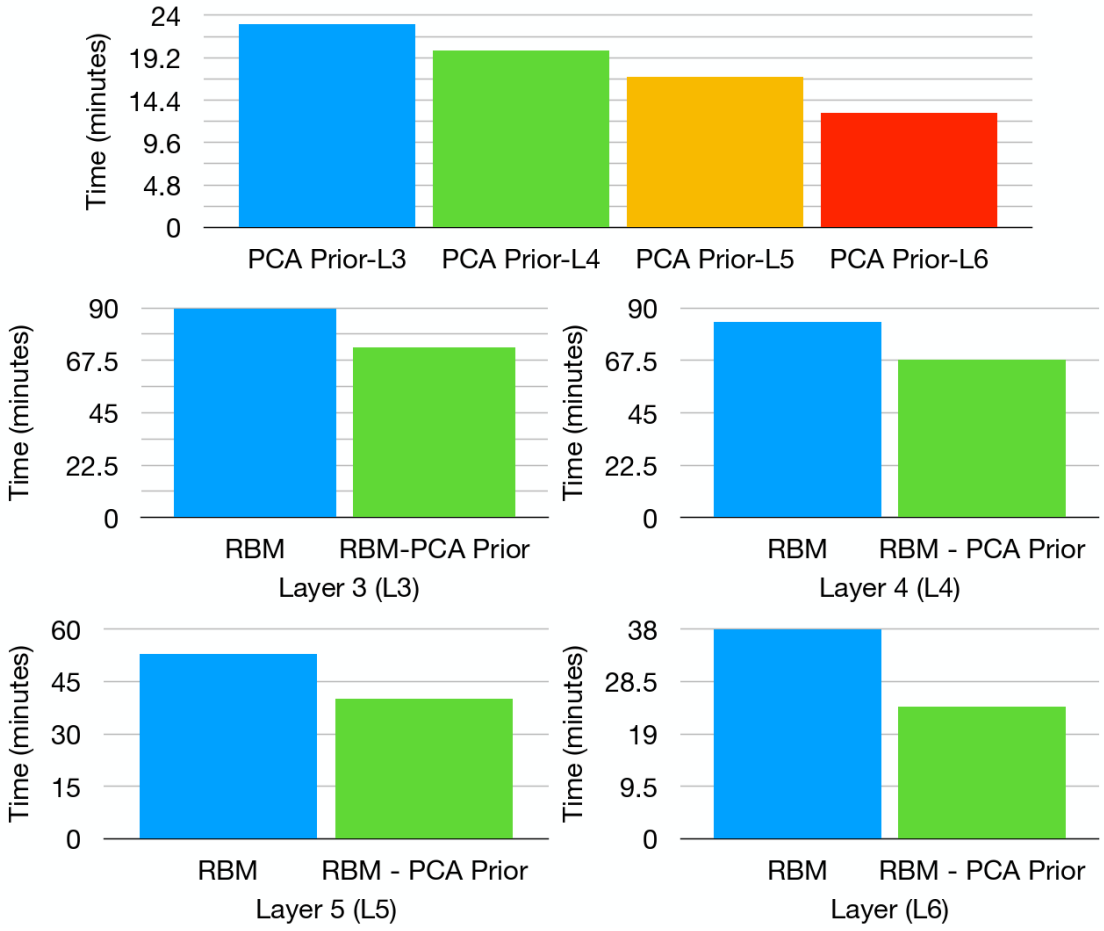


Fig. 6.11 The illustration presents the computational time required to learn the PCA priors and RBM filters at different Layers of the G-SHDL network. The figure also presents the comparison between the time required to learn the RBM filters with and without the use of PCA priors.

in abundance which is necessary to train the supervised networks. Overall, the G-SHDL can be an attractive choice due to its computationally efficient architecture and its ability to learn meaningful representations quickly using mainly unlabelled examples and a small set of labeled examples.

Chapter 7

Conclusions

In this dissertation, we proposed that deep networks that are carefully designed by taking inspiration from the principles of visual cortex can learn invariant and discriminative representations rapidly using mainly unlabeled data and limited labeled examples with reduced computational complexity. The proposed ScatterNet hybrid framework was used to derive such networks, termed as, the ScatterNet Hybrid Deep Learning Networks. These networks have also been shown to improve the learning and generalization of the deep end-to-end supervised networks.

7.1 Summary of Key Results

The circuitry of the visual cortex inspires the design of the different parts of the deep network. While this philosophy defines our approach, there are many facets related to the design of the different parts of the deep neural networks that warrant such analysis. We have attempted to present each of these in detail below:

Chapter 3 proposed the ScatterNet Hybrid Framework for Deep Learning that was inspired by different regions of the visual pathways. The ScatterNet Hybrid Framework was formed of the hand-crafted front-end that captures low-level features similar to the V1 of the cortex [40, 119, 155]. The mid-section of the framework encodes mid-level features which are believed to be similar to the representations captured by the V2-V4 regions of the cortex [149–152]. This process seems to take place in an unsupervised manner [149–152]. The supervised back-end of the framework assigns a label to the features learned by the mid-section. The IT region is believed to be involved in a similar process [148]. This framework can be used to derive networks which can learn meaningful representations rapidly using mainly unlabelled dataset and a few labeled examples. The derived networks are computationally efficient as each layer of the

network is optimized by minimizing an objective function. The ability of this framework to learn informative representations rapidly using mainly unlabelled data and limited labeled examples in a computationally efficient way makes this framework an attractive choice over the standard deep learning networks.

Chapter 4 proposed two hand-crafted architectures that captures low-level representations similar to the V1 of the visual cortex [40, 119, 155] and forms the front-end of the ScatterNet Hybrid framework. This chapter proposed the *Multi-Resolution Region Pooling ScatterNet* and the *Multi-resolution Parametric Log ScatterNet* which are improved variants of Mallat's Scattering networks [22, 27, 28, 86]. Both the networks decompose the input signal to extract translation invariant low-level edge representations that are densely spaced over the scale.

The proposed *Multi-resolution Region Pooling ScatterNet* was tested on four real-world datasets selected from the image, audio, biology and material modalities and compared with Mallat's handcrafted ScatterNet [86] as well as the state-of-the-art learned methods. The proposed network was able to outperform Mallat's handcrafted ScatterNet [86] on all four datasets by 2-3%. However, the proposed network was only able to surpass the performance of learned algorithms on only two out of the four datasets. The performance gain over the two learned algorithms was around 2-3% while for the remaining two algorithms the network underperformed by about 5% (Table 4.2-4.5).

The proposed *Multi-resolution Parametric Log ScatterNet* was tested on more complex CIFAR-10 and CIFAR-100 image datasets. The performance of the network was compared with a later version of the ScatterNet proposed by Oyallon and Mallat [22] as well as the state-of-the-art unsupervised and supervised methods. The proposed network outperformed this variant of the ScatterNet and the state-of-the-art unsupervised methods on both datasets by around 1-2% (Table 4.7) but underperformed by nearly 10% (Table 4.7) against supervised deep learning models. However, the proposed network was shown to outperform the supervised networks for the cases with small training datasets.

Hence, it is necessary to take learning into account. The proposed scattering network can then provide the first two layers of such learning networks. It eliminates local translation variability, which can help in learning the next layers. Also, this network can replace simpler low-level features such as SIFT vectors and can compute the features very quickly. Both the proposed networks can extract the feature relatively quickly which also makes these networks attractive as the time needed to train the unsupervised and supervised networks is considerable.

Chapter 5 proposed the DTCWT ScatterNet Convolutional Neural Network (DTSCNN) formed by replacing the first convolutional, Relu and pooling layers of the CNN with the parametric log based DTCWT ScatterNet. The DTCWT ScatterNet captures low-level translation invariant edge representations similar to the representations learned by these earlier layers. This resulted in the faster convergence of the DTSCNN network as it had fewer filter weights to learn compared to its corresponding CNN architecture. Also, the middle and later CNN layers were able to learn more complex patterns from the start of learning as it was not necessary to wait for the first layer to learn edges as they were already extracted by the ScatterNet. The DTSCNN was also able to produce better generalization as compared to the original CNN architectures, for datasets with limited labeled examples because the number of filter weights required to be learned were lower as explained above. The generic nature of the DTCWT scattering front-end was also shown by its similar classification performance to the earlier layers of the learned networks, on two different datasets. The generic features are likely to give it wide applicability to both small and large image datasets as it provides lower (small dataset) or similar classification error (large dataset), with faster rates of convergence.

Chapter 6 introduced two ScatterNet Hybrid Deep Learning (SHDL) Networks (*deterministic* and *generative*) that were derived from the ScatterNet Hybrid framework proposed in Chapter 3. These SHDL networks were utilized to learn the hierarchical representations from mostly unlabelled input signals rapidly. These representations were used to solve the individual image understanding tasks of object recognition and semantic segmentation with only a few labeled examples. Both the derived networks are also computationally efficient due to their optimal architecture.

Both the deterministic and the generative SHDL networks employed the scatternet as the front-end. The *deterministic SHDL network* or the *D-SHDL network* used PCA layers based unsupervised learning module to learn hierarchical features from the hand-crafted ScatterNet features. The hierarchical features were used by the OLS based supervised learning layers to select the features that aided the discriminative SVM learning. The D-SHDL network was used to perform classification on CIFAR-10 and Caltech-101 datasets. The network has been shown to outperform the unsupervised and the semi-supervised learning methods by around 1% (Table 6.4) on both datasets while underperforming by approximately 13% (Table 6.4) when compared to the supervised methods. However, the proposed network has been shown to be more accurate by 45% to 3% (Table 6.5) over supervised learning (CNNs) methods for the cases with small training datasets. The D-SHDL network is also significantly

faster to train as the approximate minimization of the reconstruction loss function of the PCA layers can be obtained simply by computing the Eigen decomposition of the patches extracted from the hand-crafted features. It has been shown that the D-SHDL network which used a straightforward PCA based network was able to learn useful hierarchical representations rapidly using mainly unlabelled dataset and produce reasonable classification performance using the learned hierarchical features with a small labeled dataset.

Limitation of D-SHDL: The D-SHDL network is interesting due to its ability to learn hierarchical representations rapidly in a computationally efficient manner. However, the complexity of the representations that can be learned by the network is restricted due to the limitation of the depth of the network (2 layers). The network learns the representations by selecting feature dimensions (eigenvectors) with maximum variations which represent the regularities of the training dataset. Due to this sampling, the energy of the features learned after every layer reduces. This limits the depth of this network and in effect, the complexity of the representations that can be learned as after a certain number of layers the representations learned wouldn't be meaningful. This is responsible for the underperformance of this network as compared with very deep supervised networks [35, 36]. This is also shown by Singh et al. [64] as they extended the D-SHDL network to 4 layers and observed that the features learned after the 4th layer were not sensible for the desired task of brain matter segmentation.

In order to improve performance further, the proposed *generative SHDL network* or the *G-SHDL network* used stacked Restricted Boltzmann Machines (RBMs) with PCA structural priors as the mid-section to learn hierarchical features. These features were used by the conditional random field (CRF) with a small labeled dataset to produce the semantic segmentation. The semantic segmentation performance of the G-SHDL network was tested on the 9 class Stanford background and the 21 class MSRC datasets. The network has been shown to outperform unsupervised methods by around 10% and semi-supervised learning methods by approximately 2% (Table 6.8). The performance of the network (78%) was comparable to the supervised methods (80%) and produced significant gains (25% to 6%) for small training datasets (Table 6.9). The G-SHDL network similar to the D-SHDL network extracts hierarchical features mainly from unlabelled datasets and produces semantic segmentation with performance similar to the state-of-the-art. The network used PCA based structural priors that accelerated the training of (otherwise slow) RBMs. Each layer of the network is trained greedily and optimized using 5-fold cross-validation which results in an optimal computationally efficient architecture. G-SHDL network doesn't suffer from the depth limitations as

detailed above for the D-SHDL network as the features are learned using gradient descent instead of PCA.

Both the deterministic and generative SHDL networks can be attractive choices over the deep supervised networks due to their ability to learn meaningful representation by utilizing mainly unlabelled and small labeled datasets. The computationally efficient architecture of the networks along with their faster convergence and reliance on the lower computational resources is also favourable.

7.2 Future Work

Research outcomes are often better evaluated by the questions born rather than the questions answered. In this section, we detail the future directions for the research presented in this dissertation which we are likely to have the most impact on the field.

7.2.1 SHDL Back-end

A single layer of the orthogonal least squares (OLS) [1, 43] has been used by the D-SHDL network (section 6.1) to select the relevant features using supervised learning. Active learning and Semi-supervised learning based models can also be used as they also leverage unlabelled data in addition to the labelled data to accomplish a pre-defined task (in this case feature selection). The advantage of such systems is that they often result in dramatic reductions in the amount of labeling required to train the system (and therefore cost and time).

Active Learning: Active learning is a framework where a model is trained on a small amount of data (the initial training set) and an acquisition function (often based on the model's uncertainty) uses to decide which data points to ask an external oracle for a label [177]. The acquisition function selects one or more points from a pool of unlabeled data points, with the pool points lying outside of the training set. An oracle (often a human expert) labels the selected data points which are added to the training set and a new model is trained on the updated training set. This process is then repeated, with the training set increasing in size over time. This results in fewer labeled examples, cost, and time, to train the system.

Past attempts at active learning of image data have concentrated on kernel based methods. Using ideas from previous research in active learning of low dimensional data [178, 179] used "margin-based uncertainty" and extracted probabilistic outputs from support vector machines (SVM) [180]. They used linear, polynomial, and Radial

Basis Function (RBF) kernels on the raw images, picking the kernel that gave best classification accuracy. Analogously to SVM approaches, Li and Guo [181] used Gaussian processes (GPs) with RBF kernels to get model uncertainty. Even though existing techniques for active learning have proven themselves useful in a variety of tasks, a major remaining challenge in active learning is its lack of scalability to high-dimensional data [178], with only a handful of exceptions [179, 182, 183] relying on kernel or graph-based approaches to handle high-dimensional data.

Semi-supervised Learning (SSL): Semi-supervised learning provide a powerful framework for leveraging unlabeled data when labels are limited or expensive to obtain [168]. In SSL, a model is given a fixed set of labelled data, and a fixed set of unlabelled data. The model can use the unlabelled data to learn about the distribution of the inputs, in the hopes that this information will aid in learning from the small labelled set as well. SSL algorithms based on deep neural networks have recently proven successful on standard benchmark tasks. Some recent results have shown that in certain cases, SSL approaches the performance of purely supervised learning, even when a substantial portion of the labels in a given dataset have been discarded [168].

There have been a wide variety of methods proposed to achieve this goal, including "transductive" [169] variants of k-nearest neighbors [171] and support vector machines [170], graph-based methods [172, 173] and algorithms based on learning features (frequently via generative modeling) from unlabeled data [174–176].

7.2.2 Artificial General Intelligence: Sequential Learning

SHDL networks like other deep learning algorithms can learn one or more tasks (multi-task learning) efficiently and produce the state-of-the-art performance. However, when trained on multiple tasks sequentially, the performance of the SHDL models (and other deep networks) degrades on earlier learned tasks. This phenomenon is referred to as catastrophic forgetting [184].

McClelland et al. [184] suggested that the human brain has evolved to learn the information related to numerous sequential tasks through the interactions of the neocortex (learner: deep network) with the hippocampus (Memory). A number of networks motivated by this idea have been proposed over the past few year [185–187] that augmented the neural network with an external memory to achieve reasonable performance on multiple sequential tasks. All the approaches follow the common theme of training the deep network for task 1 while simultaneously storing the data or a part of the data corresponding to the task in the augmented memory. This data is recalled from the memory when the data from task 2 is presented to the neural network to

perform joint multi-task training. The idea to store the data for tasks being presented to the networks sequentially can be inefficient due to the limited space in the memory module.

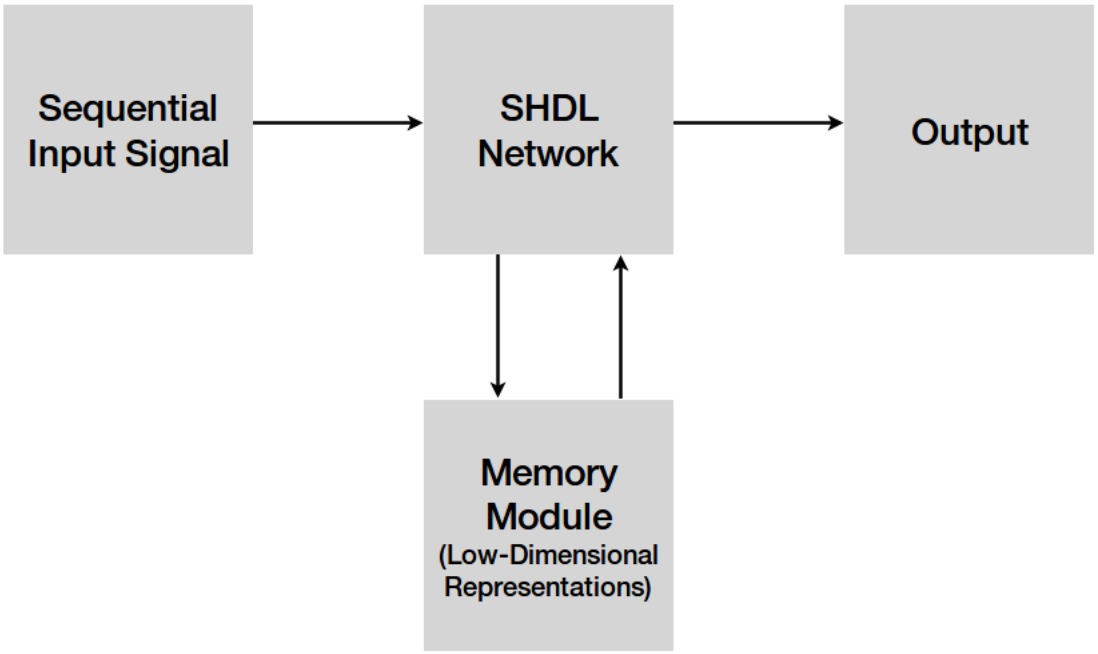


Fig. 7.1 The illustration presents the memory augmented ScatterNet Hybrid Deep Learning Network for sequential learning. The memory module stores low-dimensional representations corresponding to the earlier learned tasks which should consume less space in the memory module. These stored representations can also be used as priors for the SHDL networks for joint training with new tasks which can lead to accelerated training and can also reduce the dependence on large training datasets.

The aim is to store the low-dimensional representations corresponding to the earlier learned tasks rather than the raw data which should consume less space in the memory module. These stored representations can also be used as priors for the SHDL networks for joint training with new tasks which can lead to accelerated training and can also reduce the dependence on large training datasets. This idea is illustrated in Fig. 7.2.

7.3 Parting Note

In my doctorate, I have focused on understanding the functions of different regions of the visual cortex to design optimal deep architectures which can learn meaningful representations using principles of human learning. The proposed ScatterNet hybrid

framework is inspired by the visual cortex from which I derived several such optimal architectures, termed as ScatterNet Hybrid Deep Learning (SHDL) networks. SHDL networks presented in this dissertation can learn meaningful representations rapidly from mainly unlabelled data and limited labeled examples with reduced computational complexity, similar to the visual cortex. Designing Artificial Intelligence systems that mimic human learning should lead to more powerful machines as well as more powerful theoretical paradigms for understanding human cognition, which is my long-term future goal.

Appendix A

Training of Convolutional Neural Networks

A.1 Back-propagation Algorithm

The parameters of a Training of Convolutional Neural Networks (CNN) $w = (w_1, .., w_L)$ should be learned in such a manner that the overall CNN function $z = f(x; w)$ achieves a desired goal. In some cases, the goal is to model the distribution of the data, which leads to a generative objective. Here, however, we will use f as a regressor and obtain it by minimizing a discriminative objective. In simple terms, we are given:

1. examples of the desired input-output relations $(x_1, z_1), ..., (x_n, z_n)$ where x_i are input data and z_i corresponding output values;
2. and a loss $l(z, \hat{z})$ that expresses the penalty for predicting $\hat{z}$ instead of z.

We use those to write the empirical loss of the CNN f by averaging over the examples:

$$L(w) = \frac{1}{n} \sum_{i=1}^{n} l(z_i, f(x_i; w)) \tag{A.1}$$

Note that the composition of the function f with the loss l can be though of as a CNN with one more layer (called a loss layer). Hence, with a slight abuse of notation, in the rest of this part we incorporate the loss in the function f (which therefore is a map $\chi \to R$) and do not talk about it explicitly anymore.

The simplest algorithm to minimise L, and in fact one that is used in practice, is gradient descent. The idea is simple: compute the gradient of the objective L at a

current solution w^t and then update the latter along the direction of fastest descent of L:

$$w_{t+1} = w_t - \eta_t \frac{\partial f}{\partial w}(w^t) \tag{A.2}$$

where $\eta_t \in R_+$ is the learning rate.

The basic computational problem to solve is the calculation of the gradient of the function with respect to the parameter w. Since f is the composition of several functions, the key ingredient is the chain rule:

$$\begin{aligned} \frac{\partial f}{\partial w_l} &= \frac{\partial}{\partial w_l} f_L(...f_2(f_1(x;w_1);w_2)...), w_L) \\ &= \frac{\partial vecf_L}{\partial vecx_L^T} \frac{\partial vecf_{L-1}}{\partial vecx_{L-1}^T} ... \frac{\partial vecf_{l+1}}{\partial vecx_{l+1}^T} \frac{\partial vecf_l}{\partial vecw_l^T} \end{aligned} \tag{A.3}$$

The notation requires some explanation. Recall that each function f_l is a map from a $M \times N \times K$ array to a $M' \times N' \times K'$ array. The operator vec vectorizes such arrays by stacking their elements in a column vector (the stacking order is arbitrary but conventionally column-major). The symbol $\frac{\partial vecf_l}{\partial vecx_l^T}$ then denotes the derivative of a column vector of output variables by a row vector of input variables. Note that w_l is already assumed to be a column vector so it does not require explicit vectorization.

To apply the chain rule we must be able to compute, for each function f_l, its derivative with respect to the parameters w_l as well as its input x_l. While this could be done naively, a problem is the very high dimensionality of the matrices involved in this calculation as these are $M'N'K' \times MNK$ arrays. We will now introduce a "trick" that allows this to be reduced to working with MNK numbers only and which will yield the back-propagation algorithm.

The key observation is that we are not after $\frac{\partial vecf_l}{\partial w_l^T}$ but after $\frac{\partial f}{\partial w_l^T}$:

Unsupervised Pre-training: Training deep feed-forward neural networks can be difficult because of their tendency to converge to local optimas. These complex models are also prone to overfitting. The unsupervised layer-wise pre-training overcomes these challenges by introducing a useful prior to the supervised fine-tuning training procedure [134]. These priors can aid the learning of the deep networks.

Appendix B

Feature Selection

B.1 Orthogonal Least Squares(OLS)

This section presents the feature selection algorithm that is used to select the relevant features from a high-dimensional feature set. The features which reside in a high dimensional space make it especially important to reduce the dimensions before feeding them to the support vector machine to avoid the curse of dimensionality [139]. Orthogonal Least Squares (OLS) [1, 4] algorithm is used to select these relevant features. OLS algorithm greedily selects coefficients with a regression algorithm. We are given a set of training images $\left\{x_i\right\}_i$ with their class label. The orthogonal least square selects a set of features adapted to each class C with a linear regression of the one-versus-all indicator function.

$$f_C(x) = \begin{cases} 1 & \text{if x belongs to class C} \\ 0 & \text{otherwise} \end{cases} \tag{B.1}$$

It iteratively selects a feature in the dictionary and updates the dictionary. Let $\Phi^k x = \left\{\Phi_p^k x\right\}_p$ be the dictionary at the k^{th} iteration. We select a feature $\phi_{p_k}^k x$, and we update the dictionary by decorrelating all dictionary vectors, relatively to this selected vector, over the training set $\left\{x_i\right\}_i$:

$$\widetilde{\phi}_p^{k+1} = \phi_p^k - (\sum_i \phi_{p_k}^k x_i \phi_p^k x)\phi_{p_k}^k x \tag{B.2}$$

Each vector is then normalized.

$$\phi_p^{k+1} = \widetilde{\phi}_p^{k+1}(\sum_i |\widetilde{\phi}_p^{k+1}(x_i)|^2)^{-1} \tag{B.3}$$

The k^{th} feature $\phi^k_{p_k} x$ is selected so that the linear regression of $f_C(x)$ on $\left\{\phi^r_{p_r} x\right\}_{1 \leq r \leq k}$ has a minimum meansquare error, computed on the training set. This is equivalent to finding $\phi^k_{p_k}$ in Φ^k which maximizes the correlation $\sum_i f_C(x_i) \phi^k_p x_i$.

The orthogonal least square regression thus selects and computes K scattering features $\left\{\phi_{p_k} x\right\}_{k<K}$ for each class C, which are linearly transformed into K decorrelated and normalized features $\left\{\phi^k_{p_k} x\right\}_{k<K}$. For a total of n_C classes, the union of all these feature defines a dictionary of size $M = K n_C$. They are linear combinations of the original log scattering coefficients $\left\{\phi_p x\right\}_p$. This dimension reduction can thus be interpreted as a last fully connected network layer, which outputs a vector of size M. The parameter M governs the bias versus variance trade-off. It can be adjusted from the decay of the regression error of each f_C or fixed a priori.

Appendix C

PCA Network (PCANet)

This section presents the PCA network which learns filters at multiple layers or stages, in an unsupervised manner, from an image dataset. Suppose that we are given N input training images $\{\mathcal{I}_i\}_{i=1}^N$ of size $m \times n$, and assume that the patch size (or 2D filter size) is $k_1 \times k_2$ at all stages. The PCA filters that are needed to be learned from the input images $\{\mathcal{I}_i\}_{i=1}^N$. The steps need to extract features at different stages of the hierarchy are detailed below.

The first stage: PCA

Around each pixel, we take a $k_1 \times k_2$ patch, and we collect all (overlapping) patches of the ith image; i.e., $\boldsymbol{x}_{i,1}, \boldsymbol{x}_{i,2}, ..., \boldsymbol{x}_{i,mn} \in \mathbb{R}^{k_1 k_2}$ where each $\boldsymbol{x}_{i,j}$ denotes the jth vectorized patch in $\mathcal{I}_i$. We then subtract patch mean from each patch and obtain $\bar{\boldsymbol{X}}_i = [\bar{\boldsymbol{x}}_{i,1}, \bar{\boldsymbol{x}}_{i,2}, ..., \bar{\boldsymbol{x}}_{i,mn}]$, where $\bar{\boldsymbol{x}}_{i,j}$ is a mean-removed patch. By constructing the same matrix for all input images and putting them together, we get

$$\boldsymbol{X} = [\bar{\boldsymbol{X}}_1, \bar{\boldsymbol{X}}_2, ..., \bar{\boldsymbol{X}}_N] \in \mathbb{R}^{k_1 k_2 \times Nmn}. \tag{C.1}$$

Assuming that the number of filters in layer i is L_i, PCA minimizes the reconstruction error within a family of orthonormal filters, i.e.,

$$\min_{\boldsymbol{V} \in \mathbb{R}^{k_1 k_2 \times L_1}} \|\boldsymbol{X} - \boldsymbol{V}\boldsymbol{V}^T\boldsymbol{X}\|_F^2, \text{ s.t. } \boldsymbol{V}^T\boldsymbol{V} = \boldsymbol{I}_{L_1}, \tag{C.2}$$

where $\boldsymbol{I}_{L_1}$ is identity matrix of size $L_1 \times L_1$. The solution is known as the L_1 principal eigenvectors of $\boldsymbol{X}\boldsymbol{X}^T$. The PCA filters are therefore expressed as

$$\boldsymbol{W}_l^1 \doteq \text{mat}_{k_1,k_2}(\boldsymbol{q}_l(\boldsymbol{X}\boldsymbol{X}^T)) \in \mathbb{R}^{k_1 \times k_2}, \; l = 1, 2, ..., L_1, \tag{C.3}$$

where $\text{mat}_{k_1,k_2}(\boldsymbol{v})$ is a function that maps $\boldsymbol{v} \in \mathbb{R}^{k_1 k_2}$ to a matrix $\boldsymbol{W} \in \mathbb{R}^{k_1 \times k_2}$, and $\boldsymbol{q}_l(\boldsymbol{X}\boldsymbol{X}^T)$ denotes the lth principal eigenvector of $\boldsymbol{X}\boldsymbol{X}^T$. The leading principal eigenvectors capture the main variation of all the mean-removed training patches. Of course, similar to DNN or ScatNet, we can stack multiple stages of PCA filters to extract higher level features.

The second stage: PCA

Almost repeating the same process as the first stage. Let the lth filter output of the first stage be

$$\mathcal{I}_i^l \doteq \mathcal{I}_i * \boldsymbol{W}_l^1, \ i = 1, 2, ..., N, \tag{C.4}$$

where $*$ denotes 2D convolution, and the boundary of $\mathcal{I}_i$ is zero-padded before convolving with $\boldsymbol{W}_l^1$ so as to make $\mathcal{I}_i^l$ having the same size of $\mathcal{I}_i$. Like the first stage, we can collect all the overlapping patches of $\mathcal{I}_i^l$, subtract patch mean from each patch, and form $\bar{\boldsymbol{Y}}_i^l = [\bar{\boldsymbol{y}}_{i,l,1}, \bar{\boldsymbol{y}}_{i,l,2}, ..., \bar{\boldsymbol{y}}_{i,l,mn}] \in \mathbb{R}^{k_1 k_2 \times mn}$, where $\bar{\boldsymbol{y}}_{i,l,j}$ is the jth mean-removed patch in $\mathcal{I}_i^l$. We further define $\boldsymbol{Y}^l = [\bar{\boldsymbol{Y}}_1^l, \bar{\boldsymbol{Y}}_2^1, ..., \bar{\boldsymbol{Y}}_N^l] \in \mathbb{R}^{k_1 k_2 \times Nmn}$ for the matrix collecting all mean-removed patches of the lth filter output, and concatenate $\boldsymbol{Y}^l$ for all the filter outputs as

$$\boldsymbol{Y} = [\boldsymbol{Y}^1, \boldsymbol{Y}^2, ..., \boldsymbol{Y}^{L_1}] \in \mathbb{R}^{k_1 k_2 \times L_1 Nmn}. \tag{C.5}$$

The PCA filters of the second stage are then obtained as

$$\boldsymbol{W}_\ell^2 \doteq \text{mat}_{k_1,k_2}(\boldsymbol{q}_\ell(\boldsymbol{Y}\boldsymbol{Y}^T)) \in \mathbb{R}^{k_1 \times k_2}, \ \ell = 1, 2, ..., L_2. \tag{C.6}$$

For each input $\mathcal{I}_i^l$ of the second stage, we will have L_2 outputs, each convolves $\mathcal{I}_i^l$ with $\boldsymbol{W}_\ell^2$ for $\ell = 1, 2, ..., L_2$:

$$\mathcal{O}_i^l \doteq \{\mathcal{I}_i^l * \boldsymbol{W}_\ell^2\}_{\ell=1}^{L_2}. \tag{C.7}$$

The number of outputs of the second stage is $L_1 L_2$. One can simply repeat the above process to build more (PCA) stages if a deeper architecture is found to be beneficial.

Appendix D

Software

This appendix presents the scripts written by other researchers which were used to formulate the networks presented in this dissertation. The scripts used are presented below:

D.1 ScatterNet

The ScatterNet code developed by Edouard Oyallon [22] was the motivation for the code written to realize the ScatterNets proposed in Chapter 4. The ScatterNet code can be found at this link: http://www.di.ens.fr/data/software/ [Last visited: 22nd of April, 2018]

D.2 Deep Convolutional Networks

The DTCWT ScatterNet Convolutional Neural Network (DTSCNN) proposed in Chapter 5 was realized by replacing the earlier layers of several deep networks with the ScatterNets proposed in Chapter 4. The shallower deep networks were designed using the MatConvNet framework which can be found here: www.vlfeat.org/matconvnet/ [Last visited: 22nd of April, 2018].

The deeper networks such as the Network in Network (NIN) and the Visual Geometry Group (VGG) Network were also designed using the MatConvNet framework. The Wide Residual Network (Wide ResNet) code used for our research was written by Zagoruyko and Nikos Komodakis for their Wide Residual Networks paper at BMVC 2016 [104]. The code can be found here: https://github.com/szagoruyko/wide-residual-networks [Last visited: 22nd of April, 2018]

D.3 PCA Network

The deterministic SHDL network proposed in Chapter 6 uses the PCA layers to learn hierarchical features. The code used to realize this was written by Tsung-Han Chan and his collaborators for their PCANet paper [44]. The code can be found here: http://mx.nthu.edu.tw/ tsunghan/Source.html [Last visited: 22nd of April, 2018]. We used the tensorfloe version of this code so that it can be trained quickly. The code can be found at this link: https://github.com/PeterMitrano/PCANet [Last visited: 22nd of April, 2018].

D.4 Convolutional RBM

The generative SHDL network proposed in Chapter 6 uses stacked convolutional RBMs to learn hierarchical features rapidly using structural priors. The convolutional RBM code used for this research can be found at this link: https://github.com/OFAI/lrn2 [Last visited: 22nd of April, 2018]. The code that was used to learn the PCA priors is presented in D.3.

Appendix E

Datasets

This appendix presents the one and two, dimensional datasets used to evaluate the performance of the proposed networks on the classification and segmentation tasks. The datasets are presented below:

- **One Dimensional Datasets**: The classification performance of the proposed handcrafted scattering networks is evaluated on Isolet, Yeast, and Glass one dimensional datasets.

  The Isolet dataset is composed of one-dimensional audio signals collected from 150 speakers uttering all characters in the English alphabet twice. Each speaker contributed 52 training examples with a total of 7797 recordings [88]. The recordings are represented with 617 attributes such as spectral coefficients, contour, sonorant, and post-sonorant are provided to classify letter utterance.

  The weblink to the dataset can be found here: https://archive.ics.uci.edu/ml/datasets/isolet

  The Yeast dataset is a highly imbalanced one-dimensional signal dataset that consists of 1484 yeast proteins with 10 cellular binding sites [88]. Each binding site is described with 8 attributes. The aim is to classify the most probable (1 among the 10) cellular localization site of the proteins.

  The weblink to the dataset can be found here: https://archive.ics.uci.edu/ml/datasets/Yeast

  The Glass dataset consists of 214 one-dimensional signals that describe six types of glass based on nine chemical fractions of the oxide content [88].

  The weblink to the dataset can be found here: https://archive.ics.uci.edu/ml/datasets/glass+identification

A visualization of a randomly selected training sample from each of the above-explained datasets is presented in Fig. E.1.

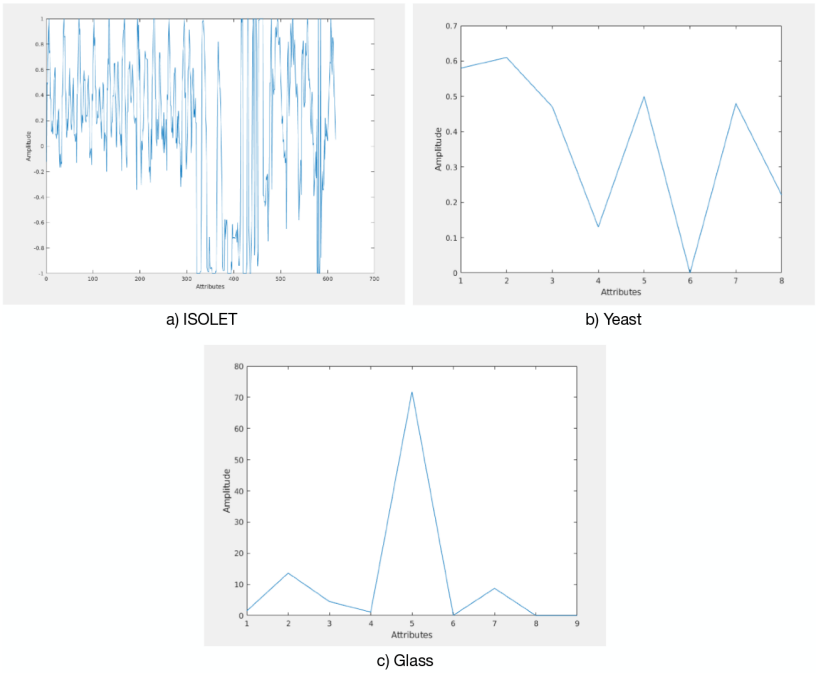


Fig. E.1 The illustration presents a visualization of a randomly selected training sample from each of the datasets.

- **Object Classification**: The CIFAR [16] and Caltech [94] datasets are used to evaluate the object classification performance. The CIFAR dataset has two variants with 10 and 100 classes while the Caltech dataset consists of two variants with 101 and 256 classes.

  Fig. E.2 presents examples of images with their classes for both CIFAR and Caltech datasets.

  The weblink to the datasets can be found here:

  - CIFAR: https://www.cs.toronto.edu/~kriz/cifar.html
  - Caltech: http://www.vision.caltech.edu/Image_Datasets/Caltech101/

- **Semantic Image Segmentation**: The Stanford Background [73] and MSRC [74] datasets are used to evaluate the semantic segmentation performance. The Stanford Background dataset has 9 classes while the MSRC dataset consists of 21 classes.

  The weblink to the datasets can be found here:

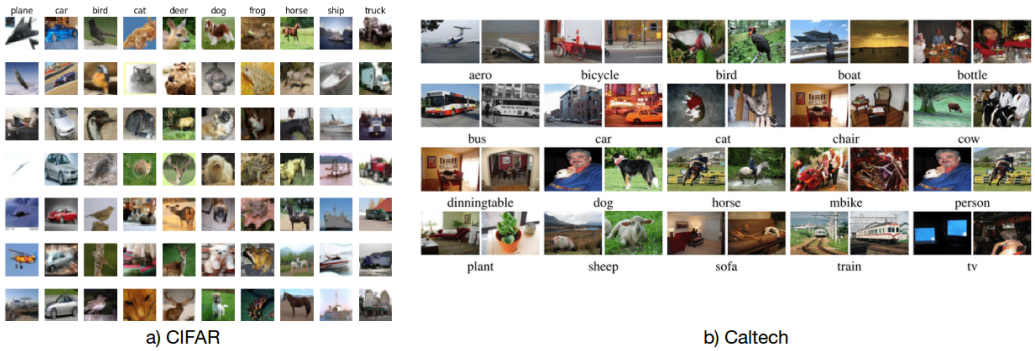


Fig. E.2 Examples of images with class labels from the CIFAR and Caltech datasets.

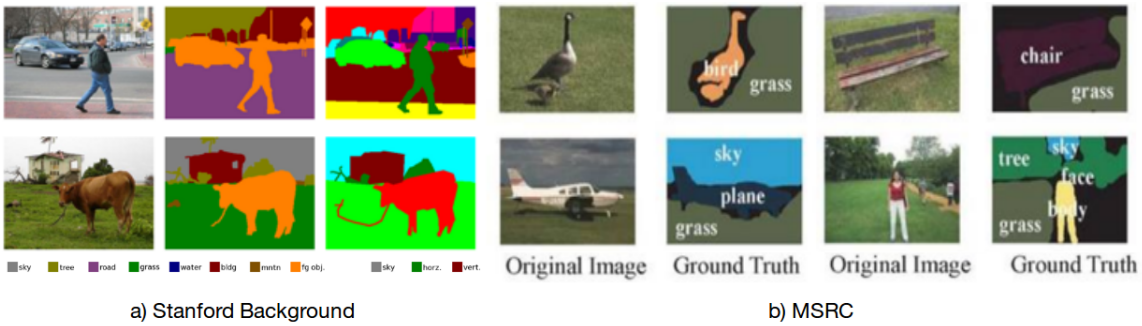


Fig. E.3 Example Images, Semantic Labels, and Regions from the a) Stanford Background b) MSRC dataset.

– Stanford Background: http://dags.stanford.edu/projects/scenedataset.html
– MSRC: https://jamie.shotton.org/work/data.html

Fig. E.3 presents examples of images with their semantic labels and regions for both Stanford Background and MSRC datasets.

Appendix F

Learned Filters

This appendix presents the filters learned by the different PCA layers and the convolutional RBM layers in the D-SHDL and G-SHDL networks introduced in Chapter 6.

Fig F.1 presents the filters learned by the PCA layers at different layers of the D-SHDL network presented in section 6.1. The filters that are visualized are selected randomly from the volume of filters learned at each layer of the network. It can be seen from the filter set that the complexity of the features captured by the filters increases for later layers of the D-SHDL network. Hence, the PCA unsupervised module captures hierarchical features.

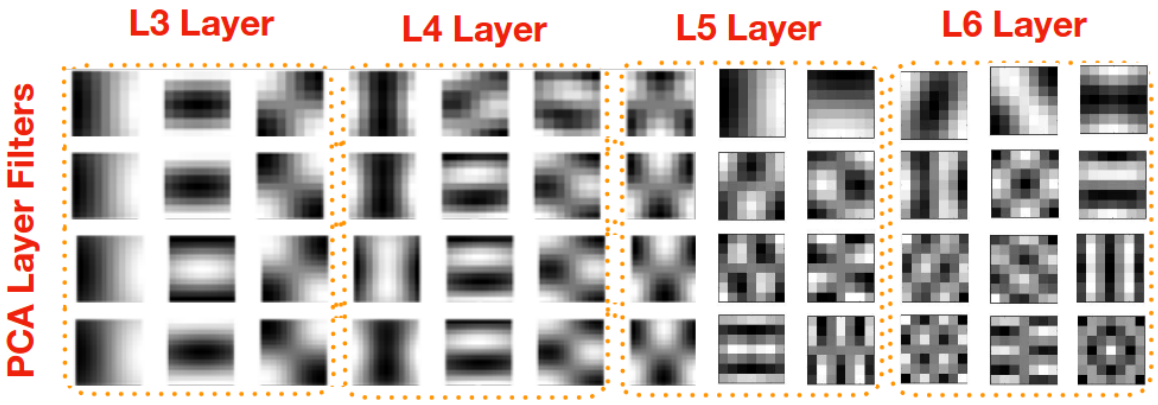


Fig. F.1 The illustration presents the filters learned at different layers of the D-SHDL network presented in section 6.1. The filters that are visualized are selected randomly from the volume of filters learned at each layer of the network.

The complexity of the features captured by the learned convolutional RBM filters of the G-SHDL network (presented in section 6.4) also increases as shown in Fig F.2.

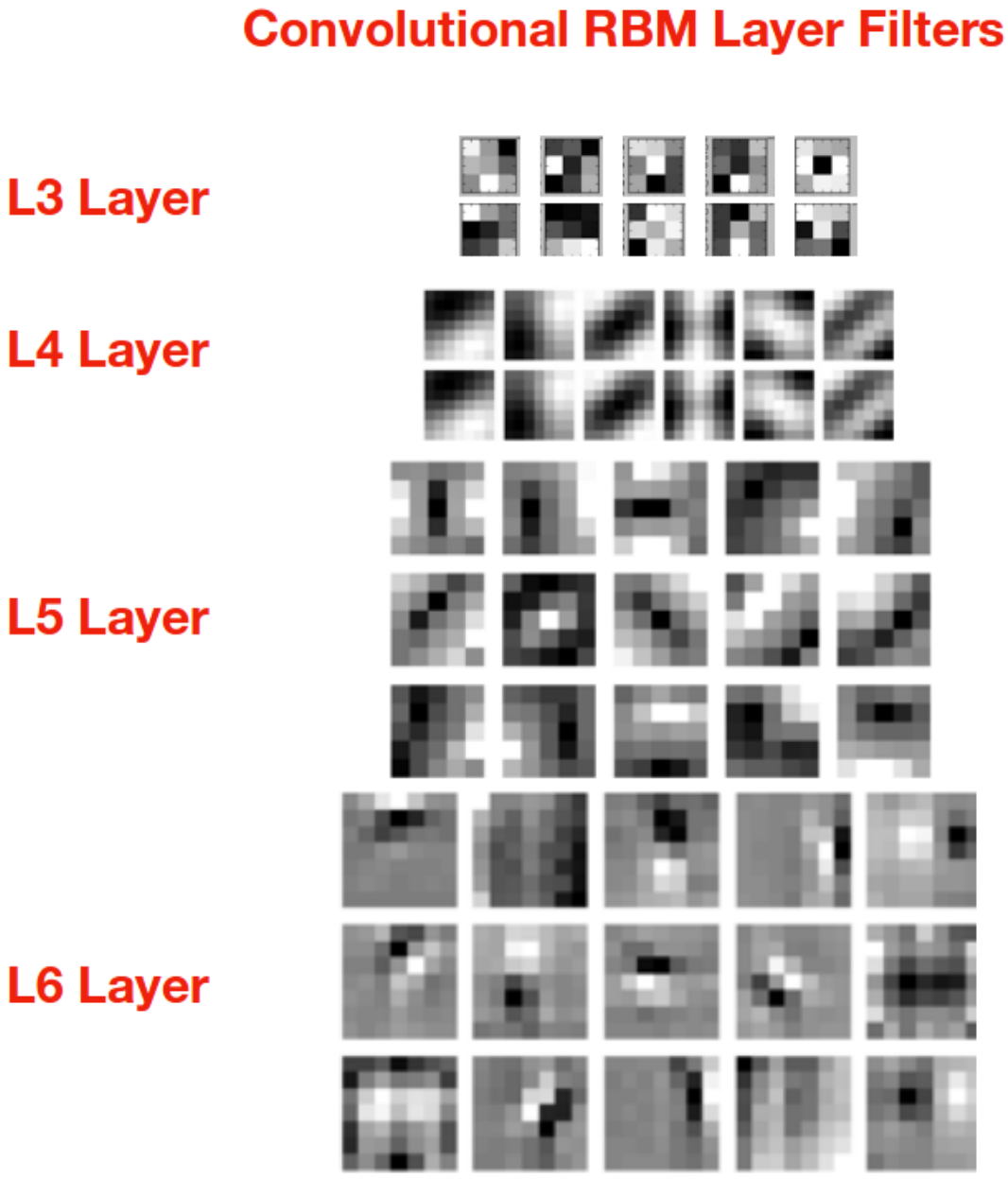


Fig. F.2 The illustration presents the RBM filters at different Layers of the G-SHDL network presented in section 6.4.

Appendix G

Publication List

We have explained the networks proposed in this dissertation in much detail. However, we are also providing the links to the publications corresponding to the proposed networks proposed if one intends to access them.

1. **Multi-resolution Region Pooling ScatterNet**: This network (proposed in Section 4.1) was published in the *International Conference on Mathematics in Signal Processing (IMA), 2016*, in Birmingham, UK, and the link for the publication can be found here: https://arxiv.org/pdf/1702.03345.pdf [Last visited: 22nd of April, 2018]

2. **Multi-resolution Parametric Log ScatterNet**: This network was proposed in Section 4.5 of this dissertation and was published in the *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2017*, in New Orleans, USA. The publication can be accessed at this link: https://ieeexplore.ieee.org/abstract/document/7952631/ [Last visited: 22nd of April, 2018]

3. **DTCWT ScatterNet Convolutional Neural Network (DTSCNN)**: The DTSCNN network, proposed in chapter 5, was published at the Compact and Efficient Feature Representation and Learning in Computer Vision (CEFRL) workshop at *IEEE International Conference on Computer Vision (ICCV), 2017*, in Venice, Italy. The link to the publication is here: http://openaccess.thecvf.com/content_ICCV_2017_workshops/papers/w18/Singh_Efficient_Convolutional_Network_ICCV_2017_paper.pdf [Last visited: 22nd of April, 2018]

4. **Deterministic ScatterNet Hybrid Deep Learning (D-SHDL)**: The D-SHDL network that was proposed in section 6.1 was published at the IEEE International Workshop on Machine Learning for Signal Processing (MLSP),

2017, in Tokyo, Japan. The link to the publication can be found here: https://ieeexplore.ieee.org/document/8168141/ [Last visited: 22nd of April, 2018]

5. **Generative ScatterNet Hybrid Deep Learning (G-SHDL)**: The G-SHDL work introduced in section was published at the *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2018*, in Calgary, Canada and the link to the publication can be accessed here: https://arxiv.org/abs/1802.03374 [Last visited: 22nd of April, 2018]

References

[1] Thomas Blumensath and Mike E. Davies (2007). "On the difference between orthogonal matching pursuit and orthogonal least squares". *Monograph (Project Report), University of Southampton.*

[2] Y-Lan Boureau, Francis Bach, Yann LeCun, and Jean Ponce (2010). "Learning mid-level features for recognition". *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2559-2566.

[3] Joan Bruna Estrach (2012). "Scattering representations for recognition". *Ph.D. dissertation, CMAP, Ecole Polytechnique.*

[4] S. Chen, C.F. Cowan, and P.M. Grant (1991). "Orthogonal least squares learning algorithm for radial basis function networks". *IEEE Transactions on Neural Networks*, Vol. 2(2), pp. 302-309.

[5] Brian Wandell (1995). "Foundations of Vision". *Wiley, Color, Research and Applications.*

[6] Keiji Tanaka (1996). "Inferotemporal cortex and object vision". *Annual Review of Neuroscience*, Vol. 19, pp. 109-139.

[7] Leonardo Bianchi (1895). "The functions of the frontal lobes". *Brain*, Vol. 18(4), pp. 497-522.

[8] Oana Cula and Kristan Dana (2001). "Compact representation of bidirectional texture functions". *Proc. of the Computer Vision and Pattern Recognition (CVPR).*

[9] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Fei-Fei Li (2009). "Imagenet: A large-scale hierarchical image database". *Proc. of the Computer Vision and Pattern Recognition (CVPR).*

[10] James J. Dicarlo and David Cox (2007). "Untangling invariant object recognition". *Trends in Cognitive Science*, Vol. 11, pp. 333-341.

[11] Bin Fan, Fuchao Wu, and Zhanyi Hu (2012). "Rotationally invariant descriptors using intensity order pooling". *IEEE Transaction on Pattern Analysis and Machine Intelligence*, Vol. 34(10), pp. 2031-2045.

[12] Josef Sivic (2009). "Efficient visual search of videos cast as text retrieval". *IEEE Transaction on Pattern Analysis and Machine Intelligence*, Vol. 31(4), pp. 591-605.

[13] Yunchao Gong, Liwei Wang, Ruiqi Guo, and Svetlana Lazebnik (2014). "Multi-scale orderless pooling of deep convolutional activation features". *Proc. of the European Conference on Computer Vision (ECCV)*, pp. 392-407.

[14] Geoffrey E. Hinton and Richard S. Zemel (1994). "Autoencoders, minimum description length, and helmholtz free energy". *Advances in Neural Information Processing Systems*, Vol. 175, pp. 1-9.

[15] Nick Kingsbury (1998). "The dual-tree complex wavelet transform: a new technique for shift invariance and directional filters". *IEEE Digital Signal Processing Workshop*, Vol. 86, pp. 120-131.

[16] Alex Krizhevsky (2009). "Learning multiple layers of features from tiny images". *Technical Report, University of Toronto.*

[17] Yan Lecun, Leon Bottou, Yoshua Bengio, and Patrick Haffner (1998). "Gradient-based learning applied to document recognition". *Proceedings of the IEEE*, Vol. 86(11), pp. 2278-2324.

[18] Fei-Fei Li and Andrej Karpathy (2015). "CS231n: Convolutional neural networks for visual recognition". *webpage: http://cs231n.stanford.edu/.*

[19] David Lowe (2004). "Distinctive image features from scale-invariant keypoints". *International Journal of Computer Vision*, Vol. 60(2), pp. 91-110.

[20] Stephane Mallat (2012). "Group invariant scattering". *Communication Pure and Applied Mathematics*, Vol. 65(10), pp. 1331-1398.

[21] Jawad Nagi, Frederick Ducatelle, Gianni A. Di Caro, Dan Cireşan, Ueli Meier, Alessandro Giusti, Farrukh Nagi, Jürgen Schmidhuber, and Luca Maria Gambardella (2011). "Max-pooling convolutional neural networks for vision-based hand gesture recognition". *IEEE International Conference on Signal and Image Processing Applications (ICSIPA)*, pp. 342-347.

[22] Edouard Oyallon and Stephane Mallat (2015). "Deep roto-translation scattering for object classification". *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2865-2873.

[23] James Philbin, Ondrej Chum, Michael Isard, Josef Sivic, and Andrew Zisserman (2008). "Lost in quantization: Improving particular object retrieval in large scale image databases". *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1-8.

[24] Ruslan Salakhutdinov and Geoffrey E. Hinton (2009). "Deep boltzmann machines". *Proc. of the 12th International Conference on Artificial Intelligence and Statistics (AISTATS).*

[25] Thomas Serre and Maximilian Riesenhuber (2004). "Realistic modeling of simple and complex cell tuning in the hmax model, and implications for invariant object recognition in cortex". *CBCL Memo 239, Massachusetts Institute of Technology, Cambridge.*

[26] Laurent Sifre (2014) "Rigid-Motion Scattering For Image Classification". *PhD thesis, ENS Paris.*

[27] Laurent Sifre and Stephane Mallat (2012) "Combined scattering for rotation invariant texture analysis". *European Symposium on Artificial Neural Networks (ESANN)*, pp. 127-132.

[28] Laurent Sifre and Stephane Mallat (2013) "Rotation, scaling and deformation invariant scattering for texture discrimination". *IEEE conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1233-1240.

[29] Yong Xu, Xiong Yang, Haibin Ling, and Hui Ji (2010). "A new texture descriptor using multifractal analysis in multi-orientation wavelet pyramid". *Proc. of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 161-168.

[30] Guoying Zhao, Timo Ahonen, Jiri Matas, and Matti Pietikainen (2012). "Rotation invariant image and video description with local binary pattern features". *IEEE Transactions on Image Processing*, Vol. 21(4), pp. 1465-1477.

[31] Yunlong He, Koray Kavukcuoglu, Yun Wang, Arthur Szlam, and Yanjun Qi (2014). "Unsupervised feature learning by deep sparse coding". *Proc. of International Conference on Data Mining (SDM)*, 2014.

[32] Svetlana Lazebnik, Cordelia Schmid, and Jean Ponce (2006) "Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories". *IEEE conference on Computer Vision and Pattern Recognition (CVPR).*

[33] Josef Sivic and Andrew Zisserman (2003). "Video google: A text retrieval approach to object matching in videos". *IEEE conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1-8.

[34] Jianchao Yang, Kai Yu, Yihong Gong, and Thomas Huang (2009). "Linear spatial pyramid matching using sparse coding for image classification". *IEEE conference on Computer Vision and Pattern Recognition (CVPR).*

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

[36] Karen Simonyan and Andrew Zisserman (2015). "Very deep convolutional networks for large-scale image recognition". *Prof. of the International Conference for Learning Representations (ICLR).*

[37] Sukriti Jain, Samarth Gupta, and Amarjot Singh (2013). "A novel method to improve model fitting for stock market prediction". *International Journal of Research in Business and Technology*, Vol. 3(1), pp. 78-83.

[38] Antonello Pasini (2015). "Artificial neural networks for small dataset analysis". *Journal of Thoracic Disease*, vol. 7(5), pp. 953–960.

[39] Josef Sivic, Bryan C. Russell, Andrew Zisserman, William T. Freeman, and Alexei A. Efros (2008). "Unsupervised discovery of visual object class hierarchies". *IEEE conference on Computer Vision and Pattern Recognition (CVPR).*

[40] Thomas Serre, Lior Wolf, Stanley Bileschi, Maximilian Riesenhuber, and Tomaso Poggio (2007). "Robust object recognition with cortex-like mechanisms". *IEEE Transaction on Pattern Analysis and Machine Intelligence*, Vol. 29(3), pp. 411-426.

[41] Tsung-Han Lin and H. T. Kung (2014). "Stable and efficient representation learning with non negativity constraints". *Proc. of the International Conference on Machine Learning*, Vol. 32(2), pp. 1323-1331.

[42] Sancho McCann and David G. Lowe (2012). "Spatially local coding for object recognition". *Asian Conference on Computer Vision*, pp. 204-217.

[43] Amarjot Singh and Nick Kingsbury (2017). "Dual-tree wavelet scattering network with parametric log transformation for object classification". *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 2622-2626.

[44] Tsung-Han Chan, Kui Jia, Shenghua Gao, Jiwen Lu, Zinan Zeng, and Yi Ma (2014). "Pcanet: A simple deep learning baseline for image classification?". *ArXiv:1404.3606*.

[45] Amarjot Singh and Nick Kingsbury (2016). "Multi-resolution dual-tree wavelet scattering network for signal classification". *International Conference on Mathematics in Signal Processing*.

[46] Sandeep Nadella, Amarjot Singh, and S.N. Omkar (2016). "Aerial scene understanding using deep wavelet scattering network and conditional random field". *European Conference on Computer Vision (ECCV) Workshops*, pp. 205-214.

[47] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen (2016). "Improved techniques for training gans". *Advances in Neural Information Processing Systems*, pp. 2234-2242.

[48] Sergey Zagoruyko and Nikos Komodakis (2016). "Wide residual networks". *arXiv:1605.07146*.

[49] Dengxin Dai and Luc Van Gool (2016). "Unsupervised high-level feature learning by ensemble projection for semi-supervised image classification and image clustering". *arXiv:1602.00955*.

[50] Bernhard Schölkopf and Alexander J. Smola (2002). "Learning with kernels". *MIT Press*.

[51] Joakim Anden, Vincent Lostanlen, and Stephane Mallat (2015). "Joint time-frequency scattering for audio classification". *Proceedings of IEEE International Workshop on Machine Learning and Signal Processing (MLSP) Workshop*, 2015.

[52] Vuk Milisic and Gilles Wainrib (2016). "Mathematical modeling of lymphocytes selection in the germinal center". *Journal of Mathematical Biology*, Vol. 74(4), pp. 933–979.

[53] Ryan Anderson, Nick Kingsbury, and Julien Fauqueur (2005). "Determining multi-scale image feature angles from complex wavelet phases". *In Proc. of the International Conference Image Analysis and Recognition*, pp. 490-498.

[54] Jeffery C. Chan, Hui Ma, and Tapan K. Saha (2013). "Partial discharge pattern recognition using multiscale feature extraction and support vector machine". *Proc. of the IEEE Power and Energy Society General Meeting.*

[55] J.J. Hull (1994), "A database for handwritten text recognition research". *Proc. of the IEEE Transaction on Pattern Analysis and Machine Intelligence*, Vol. 16(5), pp. 550-554.

[56] Bilal Hadjadji, Youcef Chibani, and Hassiba Nemmour (2014). "Fuzzy integral combination of one-class classifiers designed for multi-class classification". *International Conference Image Analysis and Recognition*, pp. 320–328.

[57] Wojciech Marian Czarnecki and Jacek Tabor (2015). "Extreme entropy machines: robust information theoretic classification". *Pattern Analysis and Applications*, Vol. 20(2), pp. 383-400.

[58] Zong-Yao Chen, Chih-Fong Tsai, William Eberle, Wei-Chao Lin, and Shih-Wen Ke (2015). "Instance selection by genetic-based biological algorithm". *Soft Computing*, Vol. 19(8), pp. 1269–1282.

[59] Thomas Villmann, Sven Haase, and Marika Kaden (2015). "Kernelized vector quantization in gradient-descent learning". *Neurocomputing*, Vol. 147, pp. 83–95.

[60] Yangqing Jia, Chang Huang, Trevor Darrell (2012). "Beyond spatial pyramids: Receptive field learning for pooled image features". *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR).*

[61] Chen-Yu Lee, Saining Xie, Patrick Gallagher, Zhengyou Zhang, and Zhuowen Tu (2014). "Deeply-supervised nets". *arXiv:1409.5185.*

[62] Kihyuk Sohn and Honglak Lee (2012). "Learning invariant representations with local transformations". *International Conference on Machine Learning*, pp. 1339-1346.

[63] Abhinav Valada, Gabriel L. Oliveira, Thomas Brox, and Wolfram Burgard (2016). "Deep multispectral semantic scene understanding of forested environments using multimodal fusions". *International Symposium on Experimental Robotics.*

[64] Amarjot Singh, Dev Hazarika, and A Bhattacharya (2017). "Texture and structure incorporated scatternet hybrid deep learning network (TS-SHDL) for brain matter segmentation". *IEEE International Conference on Computer Vision Workshop (ICCVW)*, 2017.

[65] Ondrej Miksik, Vibhav Vineet, Morten Lidegaard, Ram Prasaath, Matthias Nießner, Stuart Golodetz, Stephen L. Hicks, Patrick Pérez, Shahram Izadi, and Philip H.S. Torr (2015) "The semantic paintbrush: Interactive 3d mapping and recognition in large outdoor spaces". *Proc. of the Annual ACM Conference on Human Factors in Computing Systems*, pp. 3317-3326.

[66] Xuming He, R.S. Zemel, and M.A. Carreira-Perpinan (2004). "Multiscale conditional random fields for image labeling". *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR).*

[67] Evan Shelhamer, Jonathan Long, and Trevor Darrell (2015). "Fully convolutional networks for semantic segmentation". *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 39(4), pp. 640-651.

[68] Yong Li, Jing Liu, Yuhang Wang, Hanqing Lu, and Songde Ma (2015). "Weakly supervised RBM for semantic segmentation". *International Conference on Artificial Intelligence*, pp. 1888-1898.

[69] Alberto Garcia-Garcia, Sergio Orts-Escolano, Sergiu Oprea, Victor Villena-Martinez, and Jose Garcia-Rodriguez (2017). "A review on deep learning techniques applied to semantic segmentation". *arXiv:1704.06857*.

[70] Yu Jun, Huang Dia, and Wei Zhongliang (2018). "Unsupervised image segmentation via stacked denoising auto-encoder and hierarchical patch indexing". *Signal Processing*, Vol. 143, pp. 346-353.

[71] Andreas Geiger, Frank Moosmann, Ömer Car, and Bernhard Schuster (2012). "Automatic camera and range sensor calibration using a single shot". *IEEE International Conference on Robotics and Automation*.

[72] Amarjot Singh and Nick Kingsbury (2017). "Scatternet Hybrid Deep learning (SHDL) Network For Object Classification". *IEEE International Workshop on Machine Learning for Signal Processing (MLSP)*, 2017.

[73] Jamie ShottonJohn WinnCarsten RotherAntonio Criminisi (2006). "Textonboost: Joint appearance, shape and context modeling for multi-class object recognition and segmentation". *European Conference on Computer Vision*, pp 1-15.

[74] Stephen Gould, Richard Fulton, and Daphne Koller (2009). "Decomposing a scene into geometric and semantically consistent regions". *IEEE International Conference on Computer Vision (ICCV)*.

[75] N.G. Kingsbury (2001). "Complex wavelets for shift invariant analysis and filtering of signals". *Applied and computational harmonic analysis*, Vol. 10, pp. 234-253.

[76] Montavon, Grégoire, Geneviève, Müller, and Klaus-Robert (2012). "In neural networks: Tricks of the trade". *Springer*.

[77] Justin Domke (2013). "Learning graphical model parameters with approximate marginal inference". *IEEE Transaction on Pattern Analysis and Machine Intelligence*, Vol. 35(10),pp. 2454-2467.

[78] Xiaojie Jin, Yunpeng Chen, Jiashi Feng, Zequn Jie, and Shuicheng Yan (2017). "Multi-path feedback recurrent neural networks for scene parsing". *Proc. of the Association for the Advancement of Artificial Intelligence (AAAI)*.

[79] Fayao Liu, Guosheng Lin, and Chunhua Shen (2017). "Discriminative training of deep fully- connected continuous crfs with task-specific loss". *IEEE Transactions on Image Processing*, Vol. 26(5), pp. 2127-2136.

[80] Nasim Souly, Concetto Spampinato, and Mubarak Shah (2017). "Semi supervised semantic segmentation using generative adversarial network". *arXiv:1703.09695*.

[81] Adam Coates and Andrew Y. Ng (2012). "Learning feature representations with k-means". *Advances in Neural Information Processing Systems*, pp. 561-580.

[82] Pedro O. Pinheiro and Ronan Collobert (2013). "Recurrent convolutional neural networks for scene parsing". *International Conference on International Conference on Machine Learning*, Vol. 32, pp. 82-90.

[83] Xiao Liu, Mingli Song, Dacheng Tao, Zicheng Liu, Luming Zhang, Chun Chen, and Jiajun Bu (2013) "Semi-supervised node splitting for random forest construction". *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR).*

[84] Michael Rubinstein, Armand Joulin, Johannes Kopf, and Ce Liu (2013). "Unsupervised joint object discovery and segmentation in internet images" *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*

[85] Amarjot Singh and Nick Kingsbury (2017). "Efficient Convolutional Network Learning using Parametric Log based Dual-Tree Wavelet ScatterNet". *IEEE International Conference on Computer Vision Workshop (ICCVW).*

[86] Joan Bruna Estrach and Stéphane Mallat (2013). "Invariant scattering convolution networks". *IEEE Transaction on Pattern Analysis and Machine Intelligence*, Vol. 35, pp. 1872-1886.

[87] Thorsten Joachims (2002). "Optimizing search engines using clickthrough data". *8th ACM SIGKDD international conference on Knowledge discovery and data mining.*

[88] "UCI repository of machine learning databases". *webpage: https://archive.ics.uci.edu/ml/index.php.*

[89] "Tensorflow Model Zoo". *webpage: https://github.com/bvlc/caffe/wiki/model-zoo.*

[90] (2015). "Torch CIFAR classification blog". *webpage: http://torch.ch/blog/2015/07/30/cifar.html.*

[91] Fergal Cotter and Nick Kingsbury (2017). "Visualizing and improving scattering networks". *arXiv:1709.01355*, 2017.

[92] Alexey Dosovitskiy, Philipp Fischer, Jost Tobias Springenberg, Martin Riedmiller, and Thomas Brox (2016). "Discriminative unsupervised feature learning with exemplar convolutional neural networks". *IEEE Transactions on Pattern Analysis Machine Intelligence*, Vol. 38(9), pp. 1734-1747.

[93] Ian Goodfellow, David Warde-Farley, Mehdi Mirza, Aaron Courville, and Yoshua Bengio (2013). "Maxout networks". *Proc. of the International Conference on Machine Learning*, Vol. 28(3), pp. 1319-1327.

[94] Fei-Fei Li, Rob Fergus, and Peter Perona (2004). "Learning generative visual models from few training examples: an incremental bayesian approach tested on 101 object categories". *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshop on Generative-Model Based Vision*, 2004.

[95] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik (2014). "Rich feature hierarchies for accurate object detection and semantic segmentation". *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 580-587.

[96] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton (2012). "Imagenet classification with deep convolutional neural networks". *Advances in Neural Information Processing Systems*, pp. 1097-1105.

[97] Rongfu Mao, Haichao Zhu, Linke Zhang, and Aizhi Chen (2006). "A new method to assist small data set neural network learning". *Proceeding of the sixth International Conference on Intelligent Systems Design and Applications*, pp. 17-22.

[98] George Papandreou (2014). "Deep epitomic convolutional neural networks". *ArXiv:1406.6909*.

[99] Javier Plaza, Antonio Plaza, Rosa Perez, Pablo Martinez (2009). "On the use of small training sets for neural network-based characterization of mixed pixels in remotely sensed hyperspectral images". *Pattern Recognition*, Vol. 42(11), pp. 3032-3045.

[100] Rajat Raina, Alexis Battle, Honglak Lee, Benjamin Packer, and Andrew Y. Ng (2007). "Self-taught learning: Transfer learning from unlabeled data". *International Conference on Machine Learning*, pp. 759-766.

[101] Ali Sharif Razavian, Hossein Azizpour, Josephine Sullivan, and Stefan Carlsson (2014). "Cnn features off-the-shelf: an astounding baseline for recognition". *IEEE conference on Computer Vision and Pattern Recognition*, pp. 512-519.

[102] Alexander Toshev and Christian Szegedy (2013). "Deeppose: Human pose estimation via deep neural networks". *arXiv:1312.4659*.

[103] Andrea Vedaldi and Karel Lenc (2015). "Matconvnet". *University of Oxford*.

[104] Sergey Zagoruyko and Nikos Komodakis (2016). "Wide residual networks". *ArXiv:1605.07146*.

[105] Matthew D. Zeiler and Rob Fergus (2014). "Visualizing and understanding convolutional networks". *European Conference on Computer Vision*, pp. 818-833.

[106] Xi Zhang, Yanwei Fu, Shanshan Jiang, Leonid Sigal, and Gady Agam (2015). "Learning from synthetic data using a stacked multichannel autoencoder". *Axiv:1509.05463*.

[107] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun (2016). "Deep residual learning for image recognition". *In Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[108] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich (2015). "Going deeper with convolutions". *In Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[109] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He (2016). "Aggregated residual transformations for deep neural networks". *In Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*

[110] Jayanta K Dutta, Jiayi Liu, Unmesh Kurup, and Mohak Shah (2018). "Effective Building Block Design for Deep Convolutional Neural Networks using Search". *arXiv:1801.08577*

[111] Mohammad Javad Shafiee, Elnaz Barshan, Francis Li, Brendan Chwyl, Michelle Karg, Christian Scharfenberger, and Alexander Wong (2017). "Learning Efficient Deep Feature Representations via Transgenerational Genetic Transmission of Environmental Information during Evolutionary Synthesis of Deep Neural Networks". *IEEE International Conference on Computer Vision Workshop (ICCVW).*

[112] Audrey Chung, Mohammad Javad Shafiee, Paul Fieguth, and Alexander Wong (2017). "The Mating Rituals of Deep Neural Networks: Learning Compact Feature Representations through Sexual Evolutionary Synthesis". *IEEE International Conference on Computer Vision Workshop (ICCVW).*

[113] Barret Zoph and Quoc V. Le (2017). "Neural Architecture Search with Reinforcement Learning". *arXiv:1611.01578.*

[114] Bowen Baker, Otkrist Gupta, Nikhil Naik, and Ramesh Raskar (2017). "Designing Neural Network Architectures using Reinforcement Learning". *arXiv:1611.02167* .

[115] Liefeng Bo, Xiaofeng Ren, and Dieter Fox (2010). "Kernel Descriptors for Visual Recognition". *Advances in Neural Information Processing Systems.*

[116] Navneet Dalal and Bill Triggs (2005). "Histograms of oriented gradients for human detection". *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR).*

[117] Jianchao Yang, Kai Yu, Yihong Gong, and Thomas Huang (2009). "Linear spatial pyramid matching using sparse coding for image classification". *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*

[118] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool (2008). "Speeded Up Robust Features". *Computer Vision and Image Understanding*, Vol. 110(3), pp. 346-359.

[119] David H. Hubel and Torsten Wiesel (1962). "Receptive fields, binocular interaction and functional architecture in the cat's visual cortex". *Journal of Physiology*, Vol. 160(1), pp. 106–154.

[120] Robert Desimone, Thomas Albright, Charles G. Gross, and Charles Bruce (1984). "Stimulus-selective properties of inferior temporal neurons in the macaque". *Journal of Neuroscience*, Vol. 4(8), pp. 2051-62.

[121] Jamie Shotton, Matthew Johnson, and Roberto Cipolla (2008). "Semantic Texton Forests for Image Categorization and Segmentation". *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR).*

[122] Wenbin Zou, Kidiyo Kpalma, and Joseph Ronsin (2012). "Semantic segmentation via sparse coding over hierarchical regions". *IEEE International Conference on Image Processing (ICIP)*, pp. 2577-2580.

[123] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun (2014). "Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition". *European Conference on Computer Vision*, pp. 346-361.

[124] Benjamin Graham (2014). "Fractional Max-Pooling". *arXiv:1412.6071*.

[125] Matthew D. Zeiler and Rob Fergus (2013). "Stochastic Pooling for Regularization of Deep Convolutional Neural Network". *Prof. of the International Conference for Learning Representations (ICLR)*.

[126] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiterr (2016). "Fast and accurate deep network learning by exponential linear units (elus)". *Prof. of the International Conference for Learning Representations (ICLR)*.

[127] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun (2015). "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification". *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[128] Yoshua Bengio, Patrice Simard, and Paolo Frasconi (1994). "Learning Long-Term Dependencies With Gradient Descent Is Difficult". *In: IEEE Transactions on Neural Networks*, Vol. 5(2), pp. 157–166.

[129] Sergey Ioffe and Christian Szegedy (2015). "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". *In the Proceedings of the 32nd International Conference on Machine Learning*.

[130] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton (2013). "On the importance of initialization and momentum in deep learning". *In the Proc. of the 30th International Conference on Machine Learning*, Vol. 28(3), pp. 1139–1147.

[131] Vinod Nair and Geoffrey E. Hinton (2010). "Rectified Linear Units Improve Restricted Boltzmann Machines". *In the Proc. of the 27th International Conference on Machine Learning*, pp. 807–814.

[132] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R. Salakhutdinov (2012). "Improving neural networks by preventing co-adaptation of feature detectors". *arXiv:1207.0580* .

[133] Pierre Sermanet, David Eigen, Xiang Zhang, Michael Mathieu, Rob Fergus, and Yann LeCun (2014). "OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks". *Proc. of the International Conference for Learning Representations (ICLR)*.

[134] Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle (2006). newblock "Greedy Layer-Wise Training of Deep Networks". *Advances in Neural Information Processing Systems*.

[135] Anne S. Hsu and Peter Dayan (2007). "An unsupervised learning model of neural plasticity: Orientation selectivity in goggle-reared kittens". *Vision Research*, Vol. 47(22), pp. 2868-2877.

[136] Stéphane Mallat (1999). "A wavelet tour of signal processing". *Academic press.*

[137] Leif E. Peterson (2009). "K-nearest neighbor". *Scholarpedia*, Vol. 4(2), pp.1883-1889.

[138] Yani Andrew Ioannou (2017). "Structural Priors in Deep Neural Networks". *PhD Thesis, University of Cambridge.*

[139] Paul F. Evangelista, Mark J. Embrechts, and Boleslaw K. Szymanski (2006). "Taming the curse of dimensionality in kernels and novelty detection". *Applied Soft Computing Technologies: The Challenge of Complexity*, Vol. 34, pp. 425-438.

[140] Alexandros Karatzoglou, David Meyer, and Kurt Hornik (2005). "Support Vector Machines in R". *Report, Department of Statistics and Mathematics, WU Vienna University of Economics and Business, Vienna.*

[141] Piotr Bojanowski and Armand Joulin (2017). "Unsupervised Learning by Predicting Noise". *arXiv:1704.05310.*

[142] Jeff Donahue, Philipp Krähenbühl, and Trevor Darrell (2017). "Adversarial Feature Learning". *in Proc. of International Conference on Learning Representations (ICLR).*

[143] (2016) "Heuritech Blog". *weblink: https://blog.heuritech.com/2016/02/29/a-brief-report-of-the-heuritech-deep-learning-meetup-5/.*

[144] Tomas Uktveris and Vacius Jusas (2017). "Application of Convolutional Neural-Networks to Four-Class MotorImagery Classification Problem". *Proc. of the Journal of Information Technology and Control*, Vol. 46(2), pp. 260-273.

[145] Irwan Bello, Hieu Pham, Quoc V. Le, Mohammad Norouzi, and Samy Bengio (2017). "Neural Combinatorial Optimization with Reinforcement Learning". *Proc. of the 34th International Conference on Machine Learning*, Vol. 70, pp. 459-468.

[146] Ke Li and Jitendra Malik (2017). "Learning to Optimize". *Proc. of the International Conference on Learning Representations (ICLR)*, Vol. 70, pp. 459-468.

[147] Thomas Serre, Aude Oliva, and Tomaso Poggio (2007). "A feedforward theory of visual cortex accounts for human performance in rapid categorization". *Proc. of the National Academy of Sciences of the United States of America*, Vol. 104(15), pp. 6424-6429.

[148] Tolga Çukur, Shinji Nishimoto, Alexander G Huth, and Jack L Gallant (2013). "Attention During Natural Vision Warps Semantic Representation Across the Human Brain". *Nature Neuroscience*, Vol. 16(6), pp. 763-770.

[149] Thomas Serre, Minjoon Kouh, Charles Cadieu, Ulf Knoblich, Gabriel Kreiman, and Tomaso Poggio (2005). "A Theory of Object Recognition: Computations and Circuits in the Feedforward Path of the Ventral Stream in Primate Visual Cortex". *Computer Science and Artificial Intelligence Laboratory*, Technical Report, MIT-CSAIL-TR-2005-082.

[150] Anitha Pasupathy and Charles Edward Connor (2001). "Shape representation in area V4: position-specific tuning for boundary conformation". *Proc. of the Journal of Neurophysiology*, Vol. 86(5), pp. 2505-2519.

[151] John Reynolds, Leonardo Chelazzi, and Robert Desimone (1999). "Competitive mechanisms subserve attention in macaque areas V2 and V4". *Proc. of the Journal of Neuroscience*, Vol. 19, pp. 1736-1753.

[152] Jack L. Gallant, Charles Edward Connor, Subrata Rakshit, James Lewis, and David C. Van Essen (1996). "Neural responses to polar, hyperbolic, and cartesian gratings in area V4 of the macaque monkey". *Proc. of the Journal of Neurophysiology*, Vol. 76, pp. 2718-2739.

[153] Antonio J. Rodríguez-Sánchez, Mazyar Fallah, and Aleš Leonardis (2015). "Hierarchical Object Representations in the Visual Cortex and Computer Vision". *Proc. of the Frontiers in Computational Neuroscience*, Vol. 9(142).

[154] Stephan Tschechne and Heiko Neumann (2014). "Hierarchical representation of shapes in visual cortex—from localized features to figural shape segregation". *Proc. of the Frontiers in Computational Neuroscience*, Vol. 8(93).

[155] Colin Blakemore Grahame F. Cooper (1970). "Development of the brain depends on the visual environment". *Nature*, Vol. 228, pp. 477-478.

[156] Annegreet van Opbroek, M. Arfan Ikram, Meike W. Vernooij, and Marleen de Bruijne (2012). "Supervised Image Segmentation across Scanner Protocols: A Transfer Learning Approach". *Machine Learning in Medical Imaging, Lecture Notes in Computer Science, Springer, Berlin, Heidelberg*, pp. 160-167.

[157] Daniel Yamins and James J DiCarlo (2016). "Using goal-driven deep learning models to understand sensory cortex". *Nature Neuroscience*, Vol. 19(3), pp. 356-365.

[158] Shuying Liu and Weihong Deng (2015). "Very deep convolutional neural network based image classification using small training sample size". *IEEE IAPR Asian Conference on Pattern Recognition (ACPR)*, pp. 730-734.

[159] Fayao Liu, Guosheng Lin, and Chunhua Shen (2017). "Discriminative Training of Deep Fully Connected Continuous CRFs With Task-Specific Loss". *IEEE Transactions on Image Processing*, Vol. 26(5), pp. 2127-2136.

[160] Heng Fan, Xue Mei, Danil Prokhorov, and Haibin Ling (2018). "Multi-Level Contextual RNNs With Attention Model for Scene Labeling". *IEEE Transactions on Intelligent Transportation Systems*, Vol. 99, pp. 1-11.

[161] Honglak Lee, Roger Grosse, Rajesh Ranganath, and Andrew Y. Ng (2009). "Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations". *Proc. of the International Conference on Machine Learning*, pp. 609-616.

[162] "Learn Deep Learning with CIFAR Datasets". *weblink: https://github.com/BIGBALLON/cifar-10-cnn.*

[163] Geoffrey E. Hinton and Ruslan R. Salakhutdinov (2006). "Reducing the dimensionality of data with neural networks". *Science*, Vol. 313, pp. 504-507.

[164] Graham W. Taylor, Geoffrey E. Hinton, and Sam T. Roweis (2007). "Modeling human motion using binary latent variables". *Advances in Neural Information Processing Systems.*

[165] Koray Kavukcuoglu, Pierre Sermanet, Y-Lan Boureau, Karol Gregor, Michaël Mathieu, and Yann LeCun (2010). "Learning convolutional feature hierachies for visual recognition". *Advances in Neural Information Processing Systems.*

[166] Matthew D. Zeiler, Dilip Krishnan, Graham W. Taylor, and Rob Fergus (2010). "Deconvolutional networks". *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2528-2535.

[167] Kihyuk Sohn, Dae Yon Jung, Honglak Lee, and Alfred O. Hero III (2011). "Efficient learning of sparse, distributed, convolutional feature representations for object recognition". *IEEE International Conference on Computer Vision (ICCV)*, pp. 2643-2650.

[168] Avital Oliver, Augustus Odena, Colin Raffel, Ekin D. Cubuk, and Ian J. Goodfellow (2018). "Realistic Evaluation of Deep Semi-Supervised Learning Algorithms". *arXiv:1804.09170.*

[169] Alexander Gammerman, Volodya Vovk, and Vladimir Vapnik (1998). "Learning by transduction". *Proc. of the Fourteenth Conference on Uncertainty in Artificial Intelligence.*

[170] Thorsten Joachims (1999). "Transductive inference for text classification using support vector machines". *Proc. of the International Conference on Machine Learning (ICML).*

[171] Thorsten Joachims (2003). "Transductive learning via spectral graph partitioning". *Proc. of the International Conference on Machine Learning (ICML).*

[172] Yoshua Bengio, Olivier Delalleau, and Nicolas Le Roux (2006). "Label Propagation and Quadratic Criterion". *Chapter 11, MIT Press.*

[173] Xiaojin Zhu, Zoubin Ghahramani, and John Lafferty (2003). "Semi-supervised learning using gaussian fields and harmonic functions". *Proc. of the International Conference on Machine Learning (ICML).*

[174] Adam Coates and Andrew Y. Ng (2011). "The importance of encoding versus training with sparse coding and vector quantization". *Proc. of the International Conference on Machine Learning (ICML).*

[175] Ian J. Goodfellow, Aaron Courville, and Yoshua Bengio (2011). "Spike-and-slab sparse coding for unsupervised feature discovery". *Advances in Neural Information Processing Systems.*

[176] Diederik P. Kingma, Danilo J. Rezende, Shakir Mohamed, and Max Welling (2014). "Semi-supervised learning with deep generative models". *Advances in Neural Information Processing Systems.*

[177] David A. Cohn, David A. Cohn, and David A. Cohn (1996). "Active learning with statistical models". *Journal of artificial intelligence research.*

[178] Simon Tong (2001). "Active Learning: Theory and Applications". *PhD thesis, AAI3028187.*

[179] Ajay J. Joshi, Fatih Porikli, and Nikolaos Papanikolopoulos (2009). "Multi-class active learning for image classification". *In Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2372–2379.

[180] Corinna Cortes and Vladimir Vapnik (1995). "Support-vector networks". *Machine learning*, Vol. 20(3), pp. 273–297.

[181] Xin Li and Yuhong Guo (2013). "Adaptive active learning for image classification". *In Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 859-866.

[182] Alex Holub, Pietro Perona, and Michael C. Burl (2008). "Entropy-based active learning for object recognition". *In Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) workshops*, pp. 1-8.

[183] Xiaojin Zhu, John Lafferty, and Zoubin Ghahramani (2003). "Combining active learning and semi-supervised learning using Gaussian fields and harmonic functions". *Proc. of the International Conference on Machine Learning (ICML) workshops*, pp. 58-65.

[184] James L McClelland, Bruce L McNaughton, and Randall C O'reilly (1995). "Why there are complementary learning systems in the hippocampus and neocortex: insights from the successes and failures of connectionist models of learning and memory". *Psychological review*, Vol. 102(3).

[185] Pablo Sprechmann, Siddhant M. Jayakumar, Jack W. Rae, Alexander Pritzel, Adrià Puigdomènech Badia, Benigno Uria, Oriol Vinyals, Demis Hassabis, Razvan Pascanu, and Charles Blundell (2018). "Memory-based Parameter Adaptation". *Prof. of the International Conference for Learning Representations (ICLR).*

[186] Pablo Sprechmann, Siddhant M. Jayakumar, Jack W. Rae, Alexander Pritzel, Adrià Puigdomènech Badia, Benigno Uria, Oriol Vinyals, Demis Hassabis, Razvan Pascanu, and Charles Blundell (2017). "Overcoming catastrophic forgetting in neural networks". *Proceedings of the National Academy of Sciences.*

[187] Adam Santoro, Sergey Bartunov, Matthew Botvinick, Daan Wierstra, and Timothy Lillicrap (2016). "One-shot Learning with Memory-Augmented Neural Networks". *arXiv:1605.06065* .

[188] Uzma Sharif, Zahid Mehmood, Toqeer Mahmood, Muhammad Arshad Javid, Amjad Rehman, and Tanzila Saba (2018). "Scene analysis and search using local features and support vector machine for effective content-based image retrieval". *Artificial Intelligence Review*, pp. 1–25.

[189] Qing Wang, Chenren Xu, Supeng Leng,and Sofie Pollin (2018). "When Autonomous Drones Meet Driverless Cars". *Proc. of the 16th Annual International Conference on Mobile Systems, Applications, and Services*, pp. 514-514.

[190] Amarjot Singh, Devendra Patil, and SN Omkar (2018). "Eye in the Sky: Real-time Drone Surveillance System (DSS) for Violent Individuals Identification using ScatterNet Hybrid Deep Learning Network". *In Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) workshops*, pp. 1629-1637.

[191] Amarjot Singh, Srikrishna Karanam, and Devinder Kumar (2013). "Constructive learning for human-robot interaction". *IEEE Potentials*, Vol. 32(4), pp. 13-19.

[192] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L. Yuille (2016). "DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs". *arXiv:1606.00915*.