Graduate School Form
30 Updated 12/26/2015

PURDUE UNIVERSITY
GRADUATE SCHOOL
Thesis/Dissertation Acceptance

This is to certify that the thesis/dissertation prepared

By Aysegul Dundar

Entitled

Learning from Minimally Labeled Data with Accelerated Convolutional Neural Networks

For the degree of Doctor of Philosophy

Is approved by the final examining committee:

Eugenio Culurciello
Chair

Anand Raghunathan

Bradley S. Duerstock

Edward L. Bartlett

To the best of my knowledge and as understood by the student in the Thesis/Dissertation Agreement, Publication Delay, and Certification Disclaimer (Graduate School Form 32), this thesis/dissertation adheres to the provisions of Purdue University's "Policy of Integrity in Research" and the use of copyright material.

Approved by Major Professor(s): Eugenio Culurciello

Approved by: George R. Wodicka 04/22/2016
Head of the Departmental Graduate Program Date

LEARNING FROM MINIMALLY LABELED DATA WITH ACCELERATED CONVOLUTIONAL NEURAL NETWORKS

A Dissertation

Submitted to the Faculty

of

Purdue University

by

Aysegul Dundar

In Partial Fulfillment of the

Requirements for the Degree

of

Doctor of Philosophy

May 2016

Purdue University

West Lafayette, Indiana

To my family: Nezaket, Cengiz and Deniz.

ACKNOWLEDGMENTS

This journey would not have been possible if I did not have the amazing family, advisor and friends supporting me.

First, I would like to thank my family for always being there for me and making the distance between us unnoticeable. My parents, Nezaket and Cengiz and my sister Deniz, I am grateful to you for adapting to my time zone, being the shoulder when I needed support, and reminding me that life is so much more than just working. Also, I would like to extend my gratitude to my wonderful aunts, uncles and cousins.

I am very fortunate to have Eugenio Culurciello as my advisor. It goes without saying that this dissertation would not have been possible without his continuous support and guidance. He has been a great mentor and advisor. Being a member of his lab was very enriching and rewarding experience. His enthusiasm for research, wisdom and deep knowledge have been extremely inspirational for me. He helped me develop as a researcher and an engineer as well as prepared me for the life after grad school. I would also like to express my deepest gratitude to my advisory committee members, Professor Anand Raghunathan, Professor Brad Duerstock, and Professor Edward L Bartlett.

I owe a big thank to Jonghoon Jin, my lab-mate, a wonderful collaborator, and an incredible friend. I have benefited greatly from our countless discussions on technical and non-technical matters. I am also thankful to my collaborators and labmates, Abhishek Chaurasia, Alfredo Canziani, Andre Chang, Bharadwaj Coimbatore, and Vinayak Gokhale for their great support and for making the working environment such a fun place.

Life at Purdue would not have been half as good without my dearest friends. Special thanks to Jonathan Gortat who has been my great friend as well as my salsa partner for many years. I am also thankful to Swapan Dara, Bob Marzec, Christie Shee, Kathryn Burden and the whole Salsa family at Purdue for brightening my days. We shared many fun moments which helped me stay focused at work. I owe sincere thanks to my dear

friend, Janelle Tidd, for not only being a great support for me during my time at Purdue but also being so patient to proofread this thesis.

I am also thankful to my dearest friends, Ozge Inan and Cigdem Karaman, for the amazing time we spent during my undergraduate studies and for their constant support even I was miles away from them. I am also indebted to Olgun Adak who has been a great friend, as well as a Physics teacher to me and also guided me with my applications for Ph.D.

Lastly, I would like to thank Murat Anil Erdogdu for his great support, friendship, and for cheering me up when I was most stressful.

My thanks go to everyone who has contributed to my studies and my life in various ways.

TABLE OF CONTENTS

Page

LIST OF TABLES viii

LIST OF FIGURES ix

ABBREVIATIONS xiv

ABSTRACT xv

1 Introduction 1

1.1 Overview of Contributions 4

1.2 Outline 5

1.3 First Published Appearances 8

2 Background 9

2.1 Representation Learning 9

2.2 Convolutional Neural Networks (ConvNets) 10

2.3 Learning: Parameter Estimation 16

2.3.1 Supervised Learning 16

2.3.2 Unsupervised Learning 20

2.3.3 Semi-supervised Learning 25

2.4 Summary 25

3 Convolutional Clustering for Unsupervised Learning 27

3.1 Introduction 27

3.2 Related Work 29

3.3 Learning Filters 31

3.3.1 Learning Filters with k-means 31

3.3.2 Learning Convolutional Filters with k-means 32

3.3.3 Experimental Results of Single Layer Network 33

3.4 Learning Connections 35

Page

3.4.1 Experimental Results of Multi-layer Networks 38
3.5 Final Classification Results 41
3.5.1 STL-10 42
3.5.2 MNIST 42
3.6 Summary 45
3.7 Initialization on ImageNet 47
3.7.1 Initialization Methods 48
3.7.2 Results 49
3.8 Discussion 51
4 Context Augmentation for Learning 53
4.1 Introduction 53
4.2 Related Work 55
4.3 Segmentation-based Augmentation 57
4.3.1 Image Context 57
4.3.2 Segmentation 58
4.3.3 Segmentation Augmentation 59
4.4 Set-up with Backgrounds 60
4.5 Experimental Setup 63
4.6 Results on the Background 63
4.7 Augmentation Techniques 66
4.8 Summary 68
5 Embedded Streaming ConvNets Accelerator 70
5.1 Introduction 70
5.2 Related Work 72
5.2.1 Acceleration through reducing ConvNet size 72
5.2.2 Acceleration through parallelizations 75
5.3 Embedded Streaming ConvNets Hardware Accelerator 81
5.3.1 Architecture 81

Page

5.3.2 High-throughput Data Ports 82
5.3.3 Computational Resources 83
5.4 Control Method 85
5.4.1 Network Parser 86
5.4.2 Resource Allocator 86
5.4.3 Hardware Configurator 92
5.5 Experimental Results 93
5.6 Summary 95
6 Applications of the ConvNets Accelerator 96
6.1 Object Classification and Detection with ConvNets 96
6.1.1 Network Architecture and Training 96
6.1.2 Object Classification 98
6.1.3 Object Detection 99
6.2 Tracking with SMR 102
6.2.1 Introduction 104
6.2.2 Previous Work 107
6.2.3 Similarity Matching Ratio (SMR) Tracker 108
6.2.4 Accuracy Results 110
6.2.5 SMR Accelerator 113
6.3 Summary 115
7 Conclusion 116
REFERENCES 118
VITA 130

LIST OF TABLES

Table		Page
3.1	Classification accuracy on STL-10 testing set with 2 layer networks.	37
3.2	Classification accuracy on STL-10. Algorithms that learn the filters unsupervised.	43
3.3	Classification accuracy on STL-10. Supervised and semi-supervised algorithms.	43
3.4	Classification test error on MNIST. Algorithms that learn the filters unsupervised.	45
3.5	Classification test error on MNIST. Supervised and semi-supervised algorithms.	45
4.1	Experiments with different combination of backgrounds (bg) with the foreground objects (fg).	64
4.2	Accuracies with 500 unique examples. Each row includes the augmentation from the previous rows.	69
6.1	The specifications of the network used for the applications. Input size of each layer is calculated based on the zero-padding size, parameters of convolution and pooling operations.	97
6.2	Performance per second and performance per second-watt numbers over classification and detection applications on different platforms.	100
6.3	Performance per second and performance per second-watt numbers for each layer for classification application with our implementation.	102
6.4	Properties of the video dataset used in this work [152].	105
6.5	Number of correctly tracked frames from the state-of-art trackers and the SMR tracker. Table is taken and modified from [153].	105

LIST OF FIGURES

Figure		Page
1.1	Schematic of a biological neuron. Figure is taken from [3]	2
1.2	Schematic of an artificial neuron. Figure is taken from [3]	2
2.1	Diagram of Deep Convolutional Neural Networks. They include n-layer (blocks) of three consecutive operations - convolution, max-pooling and non-linearity (ReLU). A final layer is usually a linear classifier that is followed by softmax regressor which translates the set of inputs into a set of probability distribution which can be the labels of objects contained in an input image. The mid-level feature maps are created by processing the presented input image with a pre-trained network.	11
2.2	Convolution operation. Convolution operation can be used for edge detection. The output maps have high values at places where there are edges that are in same orientation as the filters. Other values are surpassed.	12
2.3	A screenshot from the application which visualizes the features ConvNets extract to classify an object. A pre-trained network processes the images from a camera and displays the mid-level feature maps and the predicted categories of the images. By clicking on a mid-level feature map, you can zoom in to the feature map and by going backward from that feature map to the input, you can see what part of the input feature is extracted by the corresponding filter (grey image below the input). Code: https://github.com/Aysegul/torch-visbox . .	14
2.4	Simple examples to underfitting and overfitting to the data.	18
2.5	Demonstration of k-means clustering algorithm. 1) User chooses the number of centroids, and the centroids are randomly initialized. 2) The points are assigned to the label of the centroid that is closest to them. 3) The points with the same labels represent clusters. Mean values of the points from the same clusters become the new values of the centroids. 4) 2 and 3 repeat until convergence (points stop changing labels).	21
2.6	Randomly extracted patches of images and filters trained with k-means algorithm with the extracted patches. Filters are edge selective while the patches that are used to train the filters are not.	23

Figure Page

2.7 Demonstration of training set-up of autoencoders. The network learns to compress images and decompresses them by minimizing the distance between input and output pairs. Figure taken from [40]. 24

3.1 Filters trained on the STL-10 dataset with k-means and convolutional k-means. Filters are sorted by variance in descending order. While convolutional k-means learns unique features, the k-means algorithm introduces redundancy in filters. The duplicated features for horizontal edges are highlighted in red. 31

3.2 Comparisons of accuracy on the STL-10 dataset with filters that are trained by k-means and convolutional k-means. Tests use a single layer network and the sizes of filters are fixed to 11×11 for (a) while the number of filters is set to 96 for (b). 34

3.3 Learning Connections setup. The setup network includes a connection matrix and a convolutional layer with a predefined non-complete connection scheme. First, the network with randomly initialized connection matrix is trained with supervised learning to learn the correct connection weights. Second, using the trained matrix, the next-layer filters are learned with convolutional k-means, as performed in the previous layer. 36

3.4 Comparisons of accuracy on the STL-10 dataset with filters that are trained by k-means and convolutional k-means. Performance comparisons of two and three layer networks with different learning methods on the STL-10 dataset. Supervised denotes that the corresponding layer trained via backpropogation and unsupervised denotes that filters are trained with the convolutional k-means algorithm after the connections are learned (this work). 39

3.5 Examples of images from the STL-10 dataset. 41

3.6 Examples of images from the MNIST dataset. 44

3.7 Two distinct classes from the 1000 classes of the ImageNet dataset. Left: Siberian husky, Right: Eskimo dog [36]. 50

3.8 Testing accuracy on the ImageNet dataset with different initialization techniques. 50

3.9 Testing accuracy on the ImageNet dataset with different initialization techniques. 51

4.1 The foreground segmented images in the first row and the original image in second. All images belong to the cat category. 57

4.2 Examples of segmented images. 58

Figure Page

4.3 Segmented-based Augmentation Overview. a) The image is split into foreground and background. The background is filled in. b) Each foreground can now be combined with the infilled backgrounds which creates new examples for the training. 59

4.4 Images of extracted backgrounds. Gaps are automatically filled using *patchmatch* approach [102]. 60

4.5 Different set-up of context during training. First column has original images. Second column images are foreground objects with uniform gray color as background. Third column has mean value of the extracted background replaced with the background. Forth column has examples of foreground objects combined with different backgrounds from the same category as themselves. Fifth column images are foreground objects that are combined with backgrounds from other categories. 61

4.6 First row are images that standard data augmentations applied to the whole image. Second row are the images where the backgrounds stay the same and the augmentations are only applied to the foreground objects. They are referred to as segmented versions of the corresponding data augmentations. 66

4.7 Influence of augmentations on the STL-10 dataset when 500 images and 5000 images are used. Baseline accuracy value for the training with 500 images is 47.42% and with 5000 images, it is 69.8%. For 500 images, the segmented versions of the augmentations results (label 500 images segmented) are given as well as the standard augmentation techniques' (label 500 images). For 5000 images, only the standard augmentations are applied (label 500 images). . . 68

5.1 Testing accuracy on Cifar-10 dataset. 75

5.2 Figure taken from [82]. Illustration of how a simple convolution can be lowered to a matrix multiplication. N: Number of images in mini-batch, C: Number of input feature maps, H: Height of input image, W: Width of input image, K: Number of output feature maps, R: Height of filter kernel, S: Width of filter kernel. The colors in this illustration represent the input feature maps, and elements of D and F are uniquely labeled in the illustration so as to show how each participates in forming D_m and F_m. The filter matrix Fm has dimensions K $\times$ CRS $= 2 \times 12$, while the data matrix Dm has dimensions CRS $\times$ NPQ $= 12 \times 4$. Note that each element of D is duplicated up to RS = 4 times in D_m. The output matrix O_m has dimensions K $\times$ NPQ $= 2 \times 4$. 77

5.3 Demonstration of calculating an output feature at each layer. Each input is convolved with a different kernel and summed together. 87

Figure Page

5.4 Routing schemes for 3D convolutions. The naive approach is to process the operations to calculate one output map, as shown in the Figure 5.4(a). This approach streams 3 inputs and an intermediate value (generally) and reaches a 38% utilization of computational resources. Our approach, 3(b), improves upon the naive approaches and reaches 100% utilization, using data concatenation and input reuse. 89

5.5 The flowchart of the control method. Each instruction is scheduled as long as resources are available. Otherwise the instructions that are scheduled are processed. If there are not enough resources to complete the calculations for one output, the partial output from the calculations is saved as an intermediate value which comes in the later transaction for further processing. 92

5.6 Performance per second test over one layer network on different platforms. The input is an 500×500 RGB image. The network has 6×6 128 filters, 4×4 max-pooling and non-linearity operation. We also compare the hardware accelerator that is configured with the naive implementation described above and our implementation. 93

5.7 Performance per watt test over one layer network on different platforms. The input is an 500×500 RGB image. The network has 6×6 128 filters, 4×4 max-pooling and a ReLU non-linearity operation. We also compare the hardware accelerator that is configured with the naive implementation described above and our implementation. 94

6.1 Object classification: ConvNet takes each frame and produces a probability distribution among the classes the network is trained with. The top 2 classes with the highest probabilities are displayed on the left corner of images with the probabilities. The network is able to recognize two objects at the same time. When there is only one object in the image and the second probability is low, it can be considered noise. 101

6.2 Object detection: ConvNet scans the frame in a sliding window fashion. A simple post-processing is applied to show ConvNets capabilities. Detections above a threshold are displayed after pruning the detections in close regions. Detections can be improved with more sophisticated post-processing or by using multiple scales. 103

6.3 (Top) The red box is the SMR tracker's output, the blue box is the SAD tracker's output. The ground-truth from the first frame is used as a template which is shown on the left top corner of the frame. (Bottom) The absolute differences for each pixel between the template and result from the SMR tracker are mapped on the left and from the SAD tracker on the right. Dark values (close to zero) report a better match. Note that even though there are higher differences, the SMR tracker is able to find the correct patch. 108

Figure Page

6.4 Histogram of the pixel differences that were mapped in Figure 6.3. (a) Map between the template and result from the SMR tracker and (b) result from the SAD tracker. The SAD tracker minimizes the number pixels with large differences, whereas the SMR tracker maximizes the number of pixels that have small differences. 109

6.5 The red boxes are the SMR tracker's outputs. The video frame is extended and padded by zeroes. The SMR tracker is able to track when the target is going out of the frame. The template update is ceased in these situations which prevents the drifting from the object. 111

6.6 (Top) The red boxes are the SMR tracker's outputs. (Bottom) The blue boxes are the SAD tracker's outputs. Outlying pixels cause the SAD tracker to drift, whereas the SMR tracker is not affected by them. 111

6.7 Red boxes are the SMR tracker's results. The every-frame template update causes the outlying pixels to propagate to the templates. When outlying pixels dominate the template, the SMR tracker fails. 113

6.8 Tracking results obtained on the hardware accelerator. The application takes video sequences from the TLD dataset, Pedestrian 1 [152]. 114

ABBREVIATIONS

ConvNet	Convolutional Neural Network
SGD	Stochastic Gradient Descent
PCA	Principal Component Analysis
ReLU	Rectified Linear Unit
RGB	Red Green Blue
OCR	Optical Character Recognition
CPU	Central Processing Unit
GPU	Graphics Processing Unit
FGPA	Field-programmable Gate Array
BLAS	Basic Linear Algebra Subprograms
DMA	Direct Memory Access
TDP	Thermal Design Power
SAD	Sum of Absolute Differences
SSD	Sum of Square Differences
SMR	Similarity Matching Ratio

ABSTRACT

Dundar, Aysegul Ph.D., Purdue University, May 2016. Learning from Minimally Labeled Data with Accelerated Convolutional Neural Networks. Major Professor: Eugenio Culurciello.

The main objective of an Artificial Vision Algorithm is to design a mapping function that takes an image as an input and correctly classifies it into one of the user-determined categories. There are several important properties to be satisfied by the mapping function for visual understanding. First, the function should produce good representations of the visual world which will be able to recognize images independently of pose, scale and illumination. Furthermore, the designed artificial vision system has to learn these representations by itself. Recent studies on Convolutional Neural Networks (ConvNets) produced promising advancements in visual understanding. These networks attain significant performance upgrades by relying on hierarchical structures inspired by biological vision systems. In my research, I work mainly in two areas: 1) how ConvNets can be programmed to learn the optimal mapping function using the minimum amount of labeled data, and 2) how these networks can be accelerated for practical purposes.

1) The task of labeling data for training deep neural networks is daunting and tedious work. It requires millions of labels to achieve the current state-of-the-art accuracy. The dependency on large amounts of labeled data can be mitigated by exploiting hierarchical features via unsupervised learning techniques. Unsupervised learning algorithms explore patterns in datasets consisting only of input data without labeled targets. It is believed that this is akin to how humans learn to recognize objects, because unlike traditional computer learning algorithms, humans do not need thousands of examples of an object to learn it. In this thesis, algorithms that learn from unlabeled data are studied, and state-of-the-art results on common benchmarks are achieved.

2) The applications of ConvNets exist in autonomous robots, security systems, mobile phones, and automobiles where high throughput of the feed-forward evaluation phase and power efficiency are important. As a result of their wide applicability, many custom hardware accelerators for ConvNets have been proposed. In this thesis, an optimized streaming method for ConvNets' hardware accelerator on an embedded platform is presented. The proposed method utilizes maximum computational resources available based on a novel scheduled routing topology. This system is tested on object classification and detection applications using ConvNets as well as a template matching algorithm for tracking objects in real-world scenarios. Experimental results indicate high computational efficiency, and significant performance upgrades over all other existing platforms.

1. INTRODUCTION

Neural computation is a science that aims at modeling the brain and studying the design and construction of neurally-inspired information processing systems. This area of study is interesting because it can help us understand how the brain works as well as aid in solving practical problems by using algorithms inspired by the brain. This thesis aims at the latter: it advances image classification algorithms by taking inspiration from biology - specifically the human virtual cortex.

Neural computation is an interdisciplinary field which utilizes biology, machine learning, statistics, optimization and probability theory to create high-functioning systems. Machine learning, also known as artificial intelligence is "a field of study that gives computers the ability to learn without being explicitly programmed" as defined by research pioneer, Arthur Samuel [1]. Such learning is designed to mimic the way in which mammals learn via rewards and punishments as opposed to traditional explicit programming, which relies on the programmer to come up with the algorithm. Society already widely benefits from machine learning algorithms through spam filters, auto-corrections of search engines and voice recognition, to name a few. Solving such problems requires mapping an input pattern to an output value which then predicts the category of the input. For example, for visual understanding, an algorithm should find a mapping function which translates image pixels into a label such as the category of the object the pixels belong to.

To learn such mappings, there have been a number of machine learning algorithms proposed. These algorithms are intended to function by identifying the relations between an input and an output via analyzing a large number of examples of correct input and output pairs. Such algorithms are expected to understand the relationship between input and output pairs and to be able to predict the correct output value for a novel input that they have not seen yet. The initial conception of machine learning algorithms goes back to

1940s, during which research was conducted to create a 'Perceptron' [2], which was a type of a linear classifier inspired by biological neurons.

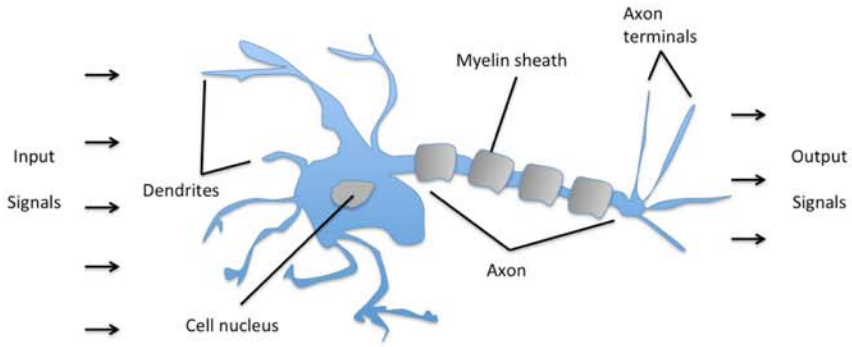


Fig. 1.1. Schematic of a biological neuron. Figure is taken from [3]

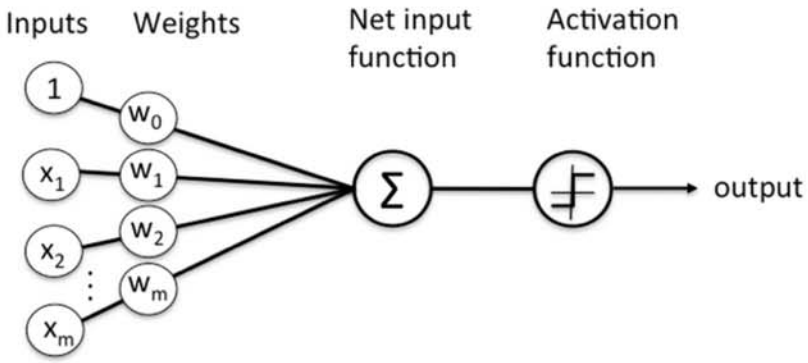


Fig. 1.2. Schematic of an artificial neuron. Figure is taken from [3]

Biological neurons, as shown in Figure 1.1, receive signals from other neurons via their dendrites. The signals that arrive at dendrites are then accumulated in the cell bodies of neurons. If the accumulated signals exceed a certain threshold, the neuron generates an output signal and passes it to its axon. In other words, a neuron uses its connections to look for a pattern. The pattern is found after the neuron accumulates a certain amount

of signal from its neighbors. Similarly the first machine learning algorithm, 'Perceptron', was designed to look for a pattern through its connections (Figure 1.2). Perceptrons are trained with specific machine learning algorithms to predict the values of weights (w). These weights are multiplied with input features in order to make a decision whether a neuron will fire or not. Similarity between the biological and artificial neuron (perceptron) is obvious when Figure 1.1 and Figure 1.2 are compared.

The perceptron algorithm is limited to learning linear functions. It was designed to solve the binary task of discriminating between two linearly separable classes. Currently, machine learning algorithms are utilized to understand images, segment objects, generate captions that summarize image contents, play atari games, and to recognize speech. For such complex tasks, many attempts have been made to design hand-crafted feature extraction before the features are fed to a machine learning algorithm. Designing feature extraction by hand is arduous and must be redone for each new problem, but better recognition systems can be built by relying more on the data to come up with an appropriate set of features rather than on the designer. For complex tasks like image understanding, building features by hand is particularly difficult.

Today, the human task of designing hand-crafted features has been superseded with the manual task of labeling data. To create current state-of-the-art systems, enormous amounts of labeled data are needed. The task of labeling data for training a machine learning algorithm is still daunting and tedious. The reliance on large amounts of labeled data can be mitigated by exploiting hierarchical features via unsupervised learning techniques. Unsupervised learning algorithms explore patterns from datasets consisting only of input data without labeled targets. It is believed that this is akin to how humans learn to recognize objects, because unlike traditional computer learning algorithms, humans do not need thousands of examples of an object to learn it. It is now commonly believed that an advanced ability to mimic human visual systems through Artificial Intelligence will result from furthering research on unsupervised learning algorithms. In this thesis, we will study algorithms that learn from unlabeled data.

Another research topic designed to advance the ability of machine learning applications is acceleration of the speed of comprehension. Recent findings reveal that a key factor of successful algorithms is their scalability. Algorithms that can learn large number of features are significantly better performers. Currently, state-of-the-art object recognition networks have around 140 million neurons [4] which is similar to the number of neurons we have in our visual cortex [5]. However, these networks are extremely computationally heavy for real-time processing of images on general purpose processors. Replication of biological parallel processing systems needs to be explored in order to create networks that can run in real-time at speeds similar to biological neural networks. In order to provide real-time applications, researchers work extensively to reduce the size of the mapping functions, and to implement custom hardware that are specialized to run these specific functions. In this thesis, we will investigate different approaches for accelerating the run time of ConvNets. The result of this work provides a system that combines a custom hardware and a custom compiler for ConvNets that can run applications of ConvNets in real-time.

1.1 Overview of Contributions

There has been extensive research on learning feature hierarchies in supervised, semi-supervised and unsupervised manners for image understanding tasks. Improvements can be made to unsupervised learning algorithms that can scale up the number of categories that a machine can learn to the hundreds of thousands, with the goal of making the output of unsupervised learning comparable and eventually superior to that of supervised learning algorithms. Currently, researchers generally address machine learning problems by collecting large amounts of data for each task which results in a high level of ConvNets performance. However, the research community recognizes that a large breakthrough lies in the ability to use unlabeled data, which is freely available in abundant quantities.

In this thesis, we investigate the use of unlabeled data in the context of ConvNets. We show that the accuracies obtained with unsupervised learning algorithms can be greatly improved by modifying them to work better with ConvNets. Furthermore, some of the

connections in the networks can be removed which enables unsupervised learning algorithms to better scale-up. This also allows ConvNets to conduct significantly faster image processing.

Another important issue is the need for fast algorithms that can be used in real-time applications. When an input is sent to a system for categorization and detection, users often may need the output immediately. In this thesis, we investigate different approaches to the acceleration of ConvNets. We propose a custom library for a hardware accelerator that can provide significant acceleration and energy efficiency during ConvNet image processing. We showcase the proposal by building applications that process videos and detect objects in real-time for each frame. Our results indicate high performance efficiency, and outperform all other existing platforms while running these applications.

1.2 Outline

This thesis will proceed as follows:

Chapter 2: Background. This chapter covers background material on learning representations for images. We discuss established ConvNets which have shown promising results in the field of visual understanding. There are many components of ConvNets that provide networks with scale and distortion invariance. Such invariances are essential for general object recognition tasks. In this chapter, we highlight these components as well as the inspiration taken from biological systems on which these components are modeled. We present different learning algorithms and place an emphasis on the investigation of supervised and unsupervised learning algorithms. The first essential component of our work will be to utilize unsupervised learning algorithms in ConvNets and the second will be to accelerate ConvNets by studying their components.

Chapter 3: Convolutional Clustering for Unsupervised Learning. This chapter proposes a pipeline for learning neural networks when there is very little labeled data. This work is built on the unsupervised learning algorithm known as k-means clustering. Through extensive analysis, we propose a refined version of an unsupervised clustering algorithm.

Another major contribution of this work is to learn sparse connection matrices between ConvNet layers by forcing sparser groups of features to map into the feature of next layer. In other words, while scaling up a network, the number of parameters can be exploited as the number of layers increases. We can attack this problem by limiting the connections between layers, e.g. by removing some connections. In such a scenario, it is important to determine which connections are the best to remove. We introduce a learning set-up which learns these connections with a backpropagation algorithm. Our experiments show that the proposed algorithm outperforms other techniques that learn filters unsupervised. Specifically, we obtained a test accuracy of 74.1% on STL-10 and a test error of 0.5% on MNIST. Furthermore, we test different initialization methods for ConvNets including the unsupervised pre-training technique. We run out experiments on a large dataset. Our results show that the initialization weights wash out with large datasets, and all the networks that are initialized with different methods converge to a similar training speed and overall performance.

Chapter 4: Context Augmentation for Learning. While aiming to learn mapping functions from images to create low-level predictions with small amounts of labeled data, a question that emerges is whether learning algorithms fully use the information that are available in the images. For instance, can we increase the dataset by modifying the images while keeping the identity of the objects the same? In this chapter, we study the effect of background in the task of image classification. Our results show that changing the background in training datasets can have drastic effects on testing accuracies. We also enhance existing augmentation techniques by facilitating augmentation through the segmentation of foreground objects. The findings of this work are important for increasing accuracies when only a small dataset is available, for creating datasets, and also for creating synthetic images.

Chapter 5: Embedded Streaming ConvNets Accelerator. Developments in ConvNet optimizations and increased efficiency in the implementation of ConvNets have resulted in an increased accuracy of many tasks, including object classification and detection. As a result, ConvNets have become particularly useful for applications in autonomous robots,

security systems, mobile phones, and automobiles, all of which require high-accuracy real-time execution. ConvNets are, however, very computationally expensive due to the fact that the convolution operations they perform on images spend over ninety percent of their time being processed by the filters. There are two methods to accelerating ConvNets. The first is to reduce their size, thus removing unnecessary parameters which results in a significant reduction in computational calculations. The second is to exploit parallelisms in different hardware architectures such as CPUs, GPUs, and custom hardware. We review several such algorithms that were created to speed up ConvNets. We implement some of these algorithms for generic datasets and explain the results in this chapter.

Furthermore, this chapter explores custom hardware for accelerating ConvNets. Our work combines a streaming method with the hardware implementation to uncover different levels of parallelisms within ConvNets. The streaming method acts as a compiler, transforming high-level representation of ConvNets into operation codes to execute applications in a hardware accelerator. The proposed system fully explores weight level and node level parallelizations of ConvNets and achieves a peak performance of 247 G-ops while consuming less than 4 watts of power. In this chapter, we present the hardware architecture and our proposed streaming method. Our results indicate high performance efficiency, and superior performance to all other presented platforms.

Chapter 6: Applications of the ConvNets Accelerator. This chapter showcases the presented hardware from the last chapter with different applications. We test our system with ConvNet applications on object classification and object detection in real-world scenarios. Furthermore, our hardware system can be used for generic image processing applications which use convolution-like data flow. For example, convolution operations can be replaced with sum-of-absolute- differences (SAD) or sum-of-square-differences (SSD) operations, widely used in tracking and motion estimation algorithms. In this chapter, we replace the convolution operation with similarity-matching-ratio (SMR) and showcase the flexibility of our proposed system. SMR is an algorithm we propose which achieves state-of-the-art results on challenging video sequences. We introduce the SMR algorithm, run it on our hardware system, and achieve real-time tracking in videos. Our results indicate high

performance efficiency, and superior performance to all other presented platforms which run these applications.

Chapter 7: Conclusion. This chapter concludes the dissertation with a summary of the main contributions.

1.3 First Published Appearances

Much of the work presented here has appeared first in other publications. The results of Chapter 3 appeared in [6], though here a new section with experiments on a large dataset is added. Chapter 5 includes the hardware accelerator that first appeared in [7] and extended in [8]. Finally, our work in Chapter 6 which showcases the hardware accelerator with different applications has been first demonstrated in [9] and [8]. Furthermore, the tracker that is accelerated through the same architecture has appeared in [10].

2. BACKGROUND

In this chapter, we will cover the building blocks of Convolutional Neural Networks (ConvNets) and important components of learning algorithms that we will re-use frequently in this thesis.

2.1 Representation Learning

Machine learning algorithms are designed to learn mapping functions which can predict the correct output of given input patterns. For each task, specific algorithms and specific mapping functions are utilized based on the prior knowledge we have about the given task. In this thesis, we will concentrate on the problem of visual understanding. Prominent researchers in the field of visual understanding have concluded that objects do not change identity based on their pose, scale, and illumination. Therefore, while designing the mapping function, the first problem to be addressed is how we can produce good representations of the visual world and create mapping functions which will be able to recognize images independently of pose, scale and illumination. Furthermore, we must determine how an artificial vision system can learn these representations by itself. While considerable progress has been achieved, these questions are still open for more research and discussion.

It is commonly accepted that good internal representations for artificial vision problems are hierarchical. Images are composed of objects and these objects are composed of parts. The parts are composed of motifs and finally, the motifs are composed of edges. Therefore, artificial vision systems should have multiple stages of representations in a hierarchical manner. Interestingly, such a hierarchy also exists in the mammalian visual cortex [11–14].

Deep neural networks are functions that are particularly well suited to represent hierarchical signals, as they naturally decompose into a hierarchy of simpler linear functions. These neural networks range from fully trained ConvNets [15] to SIFT and SURF fea-

ture extractors [16, 17] and hierarchical models of the visual cortex (HMAX) [18–20]. We elected to use existing ConvNet architecture because their greater similarity to biological systems make them more accurate than previously existing models. In recent years, ConvNets have significantly advanced the execution of many tasks including object classification [15], object detection [21], scene segmentation [22], and action recognition [23]. This thesis shows how ConvNets can be programmed to learn from unlabeled data and also how they can be accelerated. This chapter introduces ConvNets. Section 2.2 explains the architecture of ConvNets. Section 2.3 describes different kinds of learning algorithms to predict the weights of ConvNets.

2.2 Convolutional Neural Networks (ConvNets)

Because biological visual systems are so advanced, there has been extensive effort to reverse engineer them. In recent years, ConvNets that have hierarchical structures inspired by biological vision systems have produced promising advancements in visual understanding. The building blocks of ConvNets are complex. There are many components of ConvNets that provide the networks with scale and distortion invariance. Such invariances are essential for general object recognition tasks. We will highlight these components, as well as the inspirations taken from biological systems.

ConvNets are powerful tools that use hierarchical layers and filters to extract meaningful features from objects in images. ConvNets consist of multiple layers of convolutions, each comprised of ten to a hundred or more filters. Figure 2.1 describes the most common ConvNet structure in which each convolution is followed by a pooling (typically max-pooling), and a non-linearity operation (usually a rectified linear unit - ReLU [15]). The last layer is usually a linear classifier followed by softmax regressor. Softmax regressors translate the set of inputs into a set of probability distributions which predict the object labels contained in an input image. Each of these operations are explained more in depth herein. Next we will elaborate on each of these operations and reveal how each component functions in relation to the greater ConvNet system.

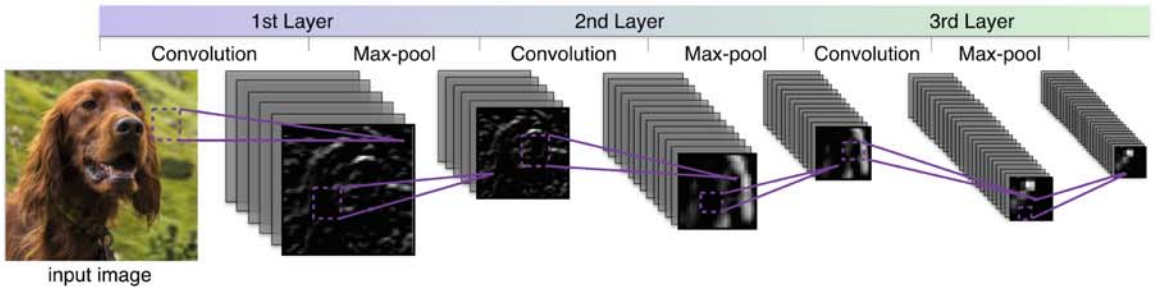


Fig. 2.1. Diagram of Deep Convolutional Neural Networks. They include n-layer (blocks) of three consecutive operations - convolution, max-pooling and non-linearity (ReLU). A final layer is usually a linear classifier that is followed by softmax regressor which translates the set of inputs into a set of probability distribution which can be the labels of objects contained in an input image. The mid-level feature maps are created by processing the presented input image with a pre-trained network.

Convolution Layer

The convolution operation takes two inputs; an image ($D \in \mathbf{R}^{C\times H\times W}$) and a filter ($F \in \mathbf{R}^{K\times C\times R\times S}$). The image ranges over; C input feature maps (e.g. $C = 3$ in RGB images), H height of the image, W width of the image. The filters range over K number of filters, C input feature maps, R height of each filter, S width of each filter. The output is also three-dimensional tensor $O \in \mathbf{R}^{K\times P\times Q}$, where K defined as previously, and $P = H - R + 1, Q = W - S + 1$. The convolution operation is as follows:

$$O(k,p,q) = \sum_{c=1}^{C}\sum_{r'=1}^{R}\sum_{s'=1}^{S} D(c, r-r', s-s')F(k,c,r',s') \tag{2.1}$$

Convolution operation is very important in image processing and computer vision algorithms, particularly in the areas of feature detection and feature extraction. For example, in computer vision, it is broadly used to detect edges. Figure 2.2 shows an example of the use of convolution operation to detect different orientations in images. The filters in Figure 2.2 are Gabor filters, designed to mimic edge sensitive simple cells found in the visual cortex of mammalian brains [24]. Therefore, Gabor filter image analysis is considered to be quite

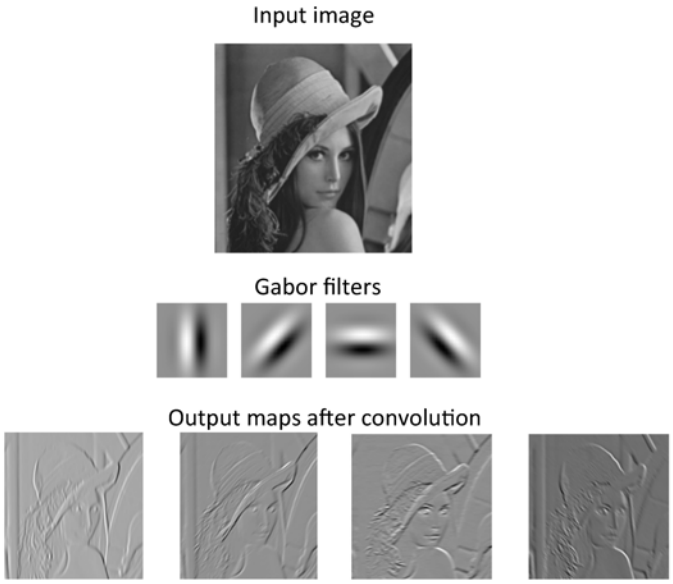


Fig. 2.2. Convolution operation. Convolution operation can be used for edge detection. The output maps have high values at places where there are edges that are in same orientation as the filters. Other values are surpassed.

similar to perceptions experienced by the human visual system. [18]. As can be seen in Figure 2.2, when an image is convolved with edge detectors, the output maps have high values at places where the edges have the same orientations as the filters.

As can be seen from Figure 2.2 and Equation 2.1, filters have smaller sizes than images and they step around the images to extract features. This structure is referred to as local connection because filters look at only a local area to produce each output. Local connections are important ingredients of ConvNets because elementary features that are useful on part of the image, are likely to be useful anywhere in the image and should be detected independent of the location. Therefore, such connections provide the ability to detect features regardless of location, which is known as translation invariance. This idea of local connections goes back to very early work on machine learning conducted in the 1960's, which coincided with Hubel and Wiesel's discovery of locally-sensitive orientation selective neurons in cats' visual systems [11]. Because of this, the filters in ConvNets are

sometimes referred to as simple cells of the mammalian visual system. Another important characteristic of local connections is that they decrease the number of parameters that are needed to be trained. It forces weight sharing, which increases the generalization ability of ConvNets. When there is freedom in the parameters, the network can learn noises in the images or the details that do not matter, but when the parameters are limited to local connections, the network is forced to summarize the images within the confines of a few parameters.

With the recent success of ConvNets, there has been great interest in exploring the kinds of features these architectures learn. It is possible to look at the first layer filters to understand what kind of selectivity such filters have. In fact visualizing the first layer filters, ConvNets usually learn edge detectors similar to Gabor filters [25]. On the other hand, for the higher layers, looking at the filters does not give any visual information. Because of this, different visualization algorithms are proposed [25–27]. Instead of looking at the filters, these algorithms look at the outputs of the convolution layers. For high values in the output maps, they go backward through the network and perform deconvolution operations with the filters. Figure 2.3 is a screenshot from an application we built which works with camera and displays which specific features activate each neuron. A pre-trained network processes the camera images and displays the mid-level feature maps (on the right), and the predicted categories of the images (below the image). By clicking on a mid-level feature map, you can zoom in to the feature map for better visualization. By going backward from that feature map to the input, you can see what part of the input feature is extracted by its corresponding filter (grey image below the input). For example, in this screenshot from Figure 2.3, a neuron in the fifth layer of ConvNet is displayed and this neuron's activation is back-propagated to the input image. This shows that this neuron is activated by the dog's face. From the analysis with visualization techniques [25], we also observe that while the first convolution layer extracts simple features like edges and curves from images, the convolution operations in the subsequent layers extract more complex shapes because they extract features from the output maps of previous layers.

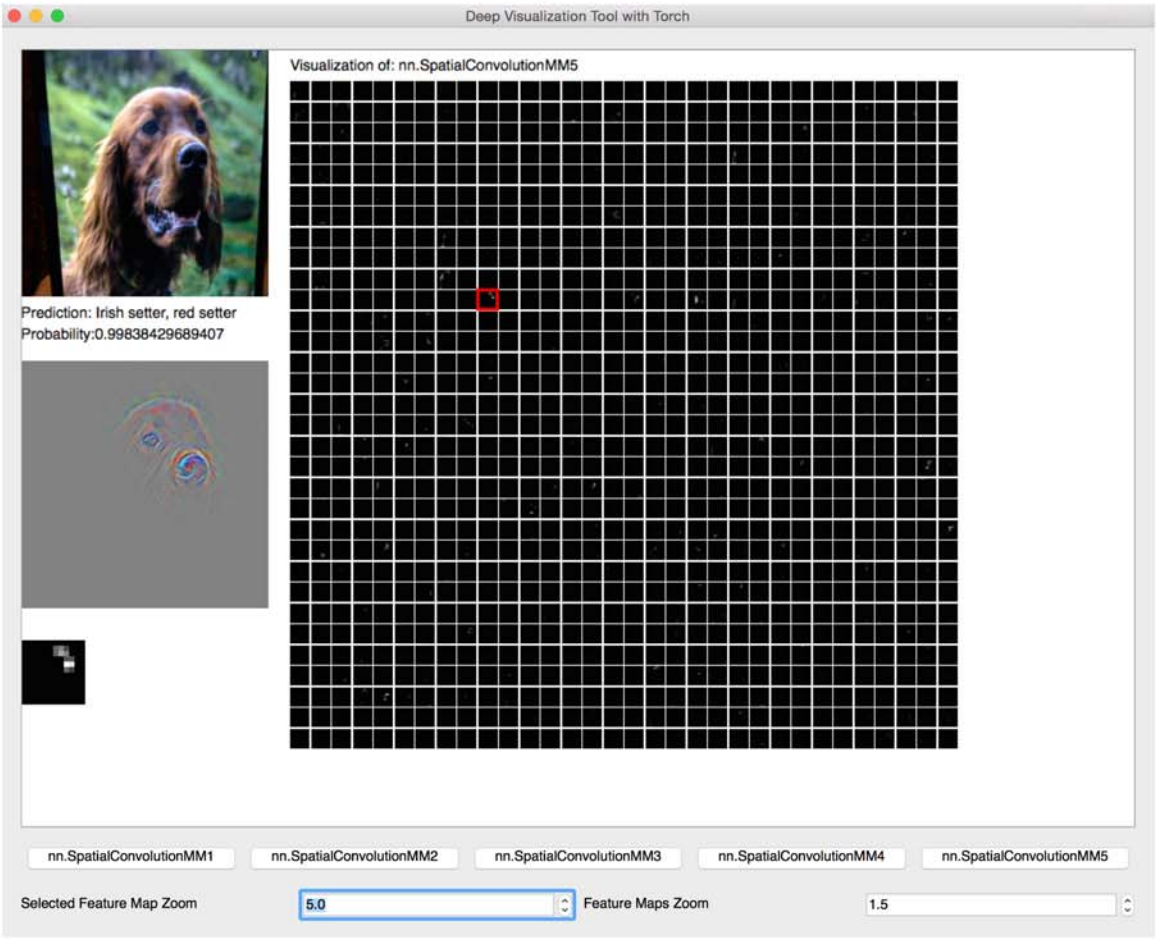


Fig. 2.3. A screenshot from the application which visualizes the features ConvNets extract to classify an object. A pre-trained network processes the images from a camera and displays the mid-level feature maps and the predicted categories of the images. By clicking on a mid-level feature map, you can zoom in to the feature map and by going backward from that feature map to the input, you can see what part of the input feature is extracted by the corresponding filter (grey image below the input). Code: https://github.com/Aysegul/torch-visbox

Pooling Layer

In ConvNets, a form of non-linear down-sampling called a pooling operation usually follows a convolutional layer. Max-pooling is the most widely used pooling operation in ConvNets. In a max-pooling operation, the images are divided into a set of non-overlapping

or a set of overlapping squares, and the maximum value is outputted for each square. This operation eliminates non-maximal values and reduces computation for upper layers. It gives robustness to ConvNets by providing a form of translation invariance, however the exact locations of the features get lost in the process.

The output maps of the pooling operations have inputs that are the direct outputs of convolutions, but cover a larger receptive field than that of simple cells. They are thus referred to as complex cells of the mammalian visual system because they are selective of a particular feature in a large receptive field.

Non-linearity Layer

A non-linear layer usually follows a pooling layer. Non-linearity operations increase the capability of a network to separate high dimensional inputs because without a non-linearity a composition of linear functions is merely a linear function, which has limited power to encode information.

Recently, the rectified linear unit became the most popular non-linearity operation because it provides fast convergence during training of ConvNets. This operation is basically a thresholding operation with a threshold equal to zero. This operation is similar to the activation mechanism in biological neurons. A biological neuron is activated if the electrical potential inside the neuron reaches a certain threshold which is the similar case in this layer.

Linear Layer

A linear layer (linear classifier) is usually used at the end of ConvNets after convolutional layers. Convolutional layers which include convolution, pooling, and non-linearity, are called feature extractors. In the end, these features are fed to a linear classifier in order to draw the classification decision boundaries. The linear classifier performs matrix multiplication operation. It does not use local connections as convolution operation. Therefore, it does not provide with translation invariance. Because a linear classifier is connected

to all activations in the previous layer rather than only to a local region in the input, it is also called a fully-connected layer. To summarize, linear classifiers make a classification decision based on the linear combinations of the features extracted from images.

Softmax Layer

A softmax operation is a generalization of the logistic function that translates the set of inputs into a set of probability distributions. This function is used in various probabilistic multiclass classification methods. It is appropriate to use softmax regression classifier when the classes are mutually exclusive. For example, when the task is classifying an image into three different classes: cat, dog and fish, the image can be belong to only one of these categories. The softmax operation, also called normalized exponential, squashes a N-dimensional vector to values in the range of (0, 1) that add up to 1. Each value is the predicted probability of the input belonging to the corresponding class in the N-dimensional vector.

2.3 Learning: Parameter Estimation

In the previous section, we examined the architecture of ConvNets, which use many layers and filters to extract meaningful features from objects in images in a hierarchical manner. Both these filters and the linear layer weights are trainable parameters and need to be learned for each task. There are three different ways of learning the parameters of a mapping function; supervised, semi-supervised and unsupervised learning algorithms. In this section, we will examine how ConvNets utilize these different methods.

2.3.1 Supervised Learning

Supervised learning is the task of learning a mapping function from labeled training data. The training data consist of a set of input and desired output pairs. A supervised learning algorithm infers a mapping function by analyzing the input-output pairs. It at-

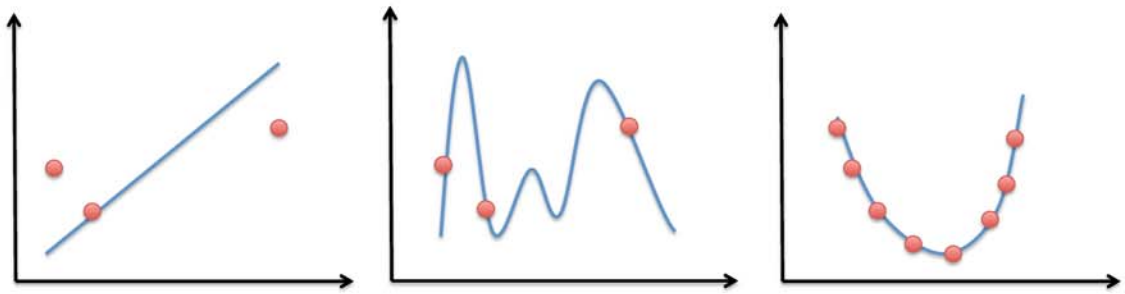
tempts to minimize the error between the desired output of an input and the output that a mapping function actually produces. The purpose of this algorithm is to be able to predict correct outputs of given novel inputs that are not included in the training data. In this setting, let's assume a training set with N training samples, $\{x^n, t^n\}$, where x^n is an input example and t^n is a correct output or target value, or a label if this is an object recognition task. $t^n \in \{1, ..., K\}$, while K is the number of categories in the task. We can write the loss function as:

$$\begin{aligned} y^n &= f(x^n; \theta) \quad \forall n \in \{1, \ldots, N\} \\ l(f; x^n, t^n, \theta) &= l(f(x^n; \theta), t^n) \quad \forall n \in \{1, \ldots, N\} \\ L(f; x, t, \theta) &= \sum_{n \in \{1,\ldots,N\}} l(f; x^n, t^n, \theta) \end{aligned} \tag{2.2}$$

where f is a model with trainable parameters θ, l is a loss function which captures the per-sample error to be minimized, and L is the global loss function which is the sum of per-sample error for each sample in the training set. L is the function that is try to minimized by supervised learning algorithms by updating the values of θ.

For example, with given input, x, and output, t, a simple function like linear function, $f(x) = ax + b = t$, can be learned. The parameters a and b are trainable parameters θ and can be predicted by minimizing the loss function such as the sum of differences between $f(x)$ and y. While this is a very simplified problem and θ that minimizes the loss function can be calculated by linear algebra, for more complicated functions of $f(x)$, optimization techniques are used.

By choosing a f and l differentiable, the optimization can be solved by gradient descent procedure. Since ConvNets have many layers, each layer is trained with a backpropogation algorithm in conjunction with Stochastic Gradient Descent (SGD) algorithm. Backpropogation algorithm is an abbreviation for backward propagation of errors. It calculates the gradient of the ConvNet in relation to the ConvNet's modifiable weights. After each training sample or a batch of samples, SGD algorithm immediately updates the parameters online, without going through all the examples. Although SGD approximates the



(a) Small model (underfitting) (b) Big model (overfitting) (c) Big model - more data

Fig. 2.4. Simple examples to underfitting and overfitting to the data.

minimization of the loss function, several studies [28–30] have shown that even when f is a non-convex function with respect to the trainable parameters, SGD yields significantly faster convergence when combined with a proper learning rate. It is computationally more efficient, especially for very large datasets. To learn the parameters of ConvNets, the entire hierarchy of features can be minimized directly with backpropogation algorithm and SGD [31–34].

Overfitting - Underfitting

In supervised learning settings, the mapping functions that are learned by analyzing the correct input-output pairs need to generalize to deal with unseen input data. Suppose we have a mapping function that exactly fits every data point we have now, can we know that it will give a correct output for a novel input? In machine learning tasks, to test the generalization error of a function the dataset is divided into two: training data and testing data. The function is learned from the training data. The accuracies of the training and testing give us information about the quality of the function and how we can improve the performance of the function.

This generalization ability is related to the concept of overfitting and underfitting phenomenon which we will cover in this section. Suppose the mapping function that is designed to be trained has very few parameters. It will be unable to learn the data points because of its limited capacity which results in an underfitting problem. Basically, when the mapping function is simple (e.g. liner function in Figure 2.4(a)), the function has an insufficient number of parameters to represent the data points. When there is the problem of underfitting, both the training and testing accuracies will be low. On the other hand, if the mapping function is complex (e.g. fifth order function in Figure 2.4(b)) and there is not enough labeled data, there can be many ways for the function to fit perfectly with the data points. However, learning the data points correctly does not mean that the function learned to generalize and can predict the correct output of a novel input. This means the mapping function overfits the data. The training error is zero, but the testing error will be high. For example, when more data points are added, as in Figure 2.4(c), it is obvious that the learned function in Figure 2.4(b) was completely wrong.

When underfitting happens, the number of trainable parameters should be increased. When overfitting happens, there are two ways to address it: by decreasing the number of parameters in the function, or by collecting more data. Unfortunately, artificial vision is a complex problem and the number of parameters to represent the visual world is in millions. In fact, the current state-of-the-art image recognition networks are designed to have millions of parameters and are trained with millions of labeled images [35–37]. However, the task of labeling data is quite expensive, time-consuming, and requires tedious work. For example, several hundred hours were spent to create the ImageNet dataset [35] which includes a thousand categories. Thousands of hours would be needed to scale up the number of categories. To circumvent this problem, the research community recognizes that a large breakthrough lies in the use of unlabeled data which is freely available in abundant quantities.

2.3.2 Unsupervised Learning

Unsupervised learning is a type of machine learning algorithm that explores patterns from datasets consisting only of input data without labeled targets. Since the data is unlabeled, there is no error or reward signal to evaluate a potential solution. Therefore, there is an important debate in the research community about what is expected from an unsupervised learning algorithm and how to measure its success. Since an unsupervised algorithm does not know what kind of outputs are expected from it, how can it find the mapping function that produces outputs?

Currently, the common way to utilize unsupervised learning algorithms is to reduce the dimensions of inputs by exploring similarities of inputs and grouping these similarities. For example, in a classification task, after an unsupervised learning algorithm extracts important features, the inputs belong to the same classes are expected to be similar to each other in feature dimensions. It should be possible to look at the euclidian distance of the input features to decide which ones belong to the same classes. For this reason, a classifier is trained with inputs which are the features extracted from inputs by unsupervised learning algorithms. The training of the classifier is supervised and requires correct input-target pairs, however, because the unsupervised learning algorithms do the main job of moving the inputs in the feature space close to each other, few labeled data is required for the supervised training. Based on this usage, the success of an unsupervised learning algorithm is measured by the usefulness of the features that it extracts from inputs. Since these features are used in the classifier with labeled data, the accuracy of this training gives us an idea about the performances of the unsupervised learning algorithms.

Unsupervised learning is a very hot topic right now, because even though there have been great successes with supervised learning algorithms, the research community is aware that the scales that can be obtained with supervised learning algorithms are limited. Furthermore, supervised learning algorithms are not the way mammals learn to recognize objects. Mammals do not need thousands of examples to learn different classes. In most cases, even a single example is enough. Therefore, it is believed that we learn most of the hierarchy in

an unsupervised manner. Since human vision is vastly better at recognition than any of the current artificial vision algorithms, any insights from biology on how to proceed are likely to be very useful. There have been a number of unsupervised learning algorithms proposed. in this section, we will review two of the most important ones; Clustering learning and Auto-encoders.

Clustering

Given a set of data points, clustering algorithms group similar data points based on specific similarity metrics. The goal is to group the data points in such a way that points in the same cluster are more similar to each other than points in other clusters.

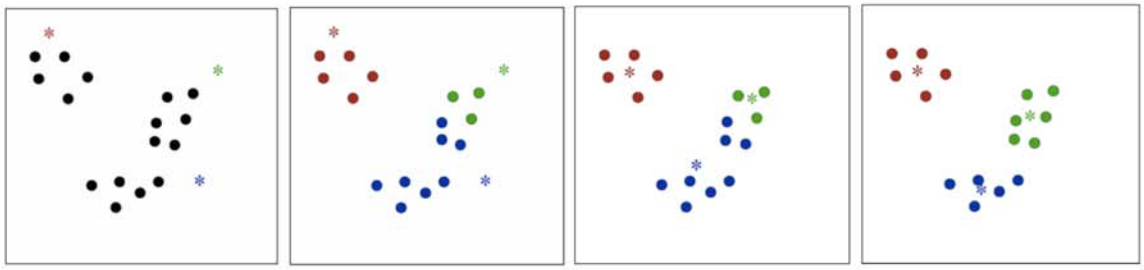


Fig. 2.5. Demonstration of k-means clustering algorithm. 1) User chooses the number of centroids, and the centroids are randomly initialized. 2) The points are assigned to the label of the centroid that is closest to them. 3) The points with the same labels represent clusters. Mean values of the points from the same clusters become the new values of the centroids. 4) 2 and 3 repeat until convergence (points stop changing labels).

One popular clustering algorithm is the k-means clustering algorithm which is demonstrated in Figure 2.5. The algorithm begins with some data points that are not yet labeled. It requires the user to select the number of clusters (number of centroids). This is a disadvantage of the algorithm because with high dimensional inputs where visualization is not possible, selecting the best number of centroids for the given problem would require multiple experiments. However, after the number of centroids are chosen, there are not any

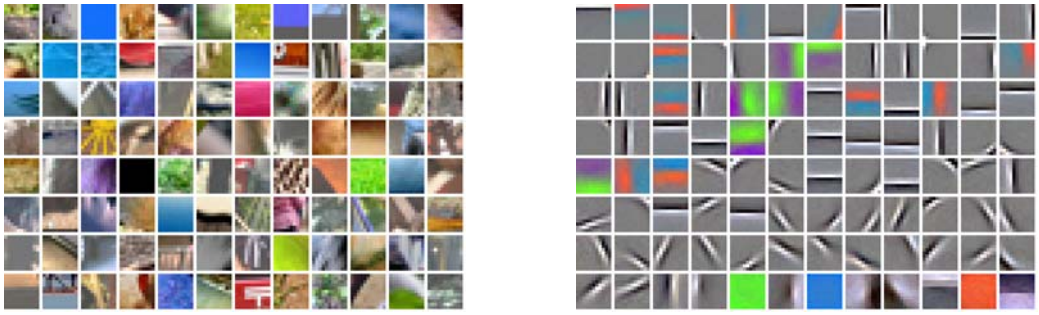
other parameters to be optimized for the k-means clustering algorithm. The k-means algorithm clusters the data points close to the centroids under the same label as the centroids. The algorithm then averages those points which provides us with the new centroids. This procedure is repeated until the centroids stop moving. Figure 2.5 is a good demonstration of the convergence. However, usually the input dimension is much higher than 2D, and we cannot visualize the clusters.

It turns out this simple algorithm works reasonably well for clustering input patches when the patches are pre-processed with ZCA-whitening. Input patches can be considered data points, but in much higher dimension than the examples from Figure 2.5. For example, if we want to train filters with size of 5×5, we will extract 5×5 patches from the images. Each 5×5 patch corresponds to a point in a 25 dimensional space. When these data points are clustered, the centroids can be used as filters.

A variant of k-means algorithm which is sometimes called the "gain-shape vector quantization" [38, 39] is used successfully to train filters from images. This algorithm minimizes:

$$\begin{aligned} \underset{D,z}{\text{minimize}} \quad & \sum_i ||Dz^{(i)} - x^{(i)}, ||_2^2 \\ \text{subject to} \quad & ||D^{(k)}||_2 = 1, \forall k \\ & ||z^{(i)}||_0 \leq 1, \forall i. \end{aligned} \tag{2.3}$$

where D is the filters that are trained and the vectors $z^{(i)} \in R^K$ are called code vectors, and $||z^{(i)}||_0 \leq 1$ means that each code vector may have at most a single non-zero element. That means a data point cannot be belong to two different clusters from Figure 2.5. This minimization can be optimized by alternating iteration over z and D, given by:



(a) Patches from images. (b) Filters.

Fig. 2.6. Randomly extracted patches of images and filters trained with k-means algorithm with the extracted patches. Filters are edge selective while the patches that are used to train the filters are not.

Repeat until convergence:

$$z_k^{(i)} := \begin{cases} {D^{(k)}}^T x^{(i)} & \text{if } k == \arg\max_l |{D^{(l)}}^T x^{(i)}| \\ 0 & \text{otherwise.} \end{cases} \tag{2.4}$$

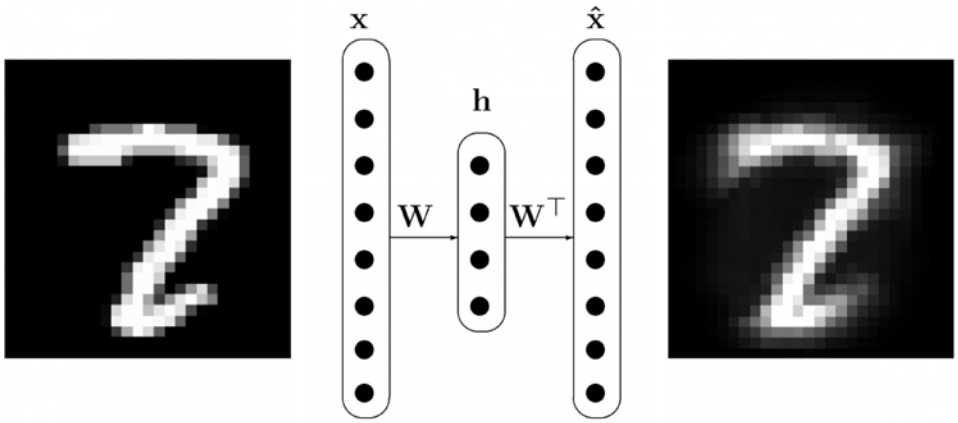
$$D := XZ^T$$

$$D^{(k)} := D^{(k)} / ||D^{(k)}||_2.$$

where $z^{(i)}$ is a code vector associated with the input $x^{(i)}$, and $D^{(k)}$ is the k'th column of the dictionary of filters. $x^{(i)}$ is usually extracted patches from input images that are the same size as the dictionary vectors, $D^{(k)}$.

Figure 2.6 is an illustration of $x^{(i)}$, extracted random patches from input images and $D^{(k)}$, the filters that are trained with k-means algorithms using the extracted patches. After the filters that can extract useful features from images are trained with the k-means clustering algorithm, they can be used in a ConvNet architecture to encode features. These features are used by a supervised learning algorithm to train a classifier with the labeled data.

Autoencoders



(a) Input (b) Neural encoding (c) Reconstruction

Fig. 2.7. Demonstration of training set-up of autoencoders. The network learns to compress images and decompresses them by minimizing the distance between input and output pairs. Figure taken from [40].

Autoencoders are basically neural networks that try to reconstruct input from latent variables [41]. They consist of two parts, encoders and decoders. Encoders map an input to a latent space, while decoders reconstruct the input from the latent space, as shown in Figure 2.7. Encoders have similar architecture as ConvNets. Most of the time, they are ConvNets without linear classifiers [42,43]. Decoders can be considered reverse ConvNets, where convolution is replaced with deconvolution and max-pooling is replaced with upsampling operations. Therefore, after an encoder decreases the size of the input, the decoder reflects it back to the original size of the input. This reconstruction task can be solved by minimizing the squared error between the input and output. The intention of this set-up is to compress information in a way that is possible to reconstruct it.

In such settings, it is important to take precautions to ensure that the network does not simply learn identity functions. This can be prevented by having a small number of latent variables which forces the encoder to compress the data by learning generic features.

However, we may need many variables to construct better networks, in which case gaussian noise is added to the input and the network is expected to reconstruct the original image without the noise.

For image classification tasks, the decoder is removed from the network and encoder is used to map inputs to a feature space. These features are used by a supervised learning algorithm to train a classifier with the labeled data.

2.3.3 Semi-supervised Learning

Semi-supervised learning algorithms use labeled data together with unlabeled data while training classifiers. Two different learning algorithms can be combined to learn the same parameters and one can use the labeled data while the other uses the unlabeled data. The cost function of both algorithms are minimized when used together. For example, a ConvNet can be combined with a decoder which is esentially an autoencoder. The encoder, ConvNet, can be trained with the labeled data in a supervised manner. Furthermore, it can also be trained together with the decoder to minimize the reconstruction error. There are two different cost functions in this setting and they can be minimized jointly [42, 44].

Another interesting approach is to learn a classifier from labeled data and use this classifier to classify the unlabeled data. Typically the unlabeled data for which the classifier is not certain about their categories are dismissed and the classifier is retrained with the remaining unlabeled data. This approach is called self-training and has been successfully deployed for several language processing tasks [45]. However, for images, the wrong predictions (errors) from the unlabeled data cause the accuracies to drop significantly [46]. Therefore, this approach has not been successful for image classification tasks.

2.4 Summary

In this chapter, we introduced Convolutional Neural Networks (ConvNets), a bio-inspired neural network extension, which advanced the state-of-the-art in many vision related tasks. We went through the ConvNet structure and explained the operational function of each

module and its role in the greater pipeline. ConvNets have trainable, differentiable parameters. These parameters make the ConvNets' end-to-end training possible with supervised training algorithms. In principle, one can also learn these parameters with unsupervised and semi-supervised learning algorithms that utilize unlabeled data which is freely available in abundant quantities. In this chapter, we introduced different kind of unsupervised and semi-supervised learning algorithms.

In this thesis, we will use ConvNets to extract feature hierarchies from images. In the next chapter, we will use clustering algorithm from unsupervised learning algorithms to learn the parameters of ConvNets.

3. CONVOLUTIONAL CLUSTERING FOR UNSUPERVISED LEARNING

The task of labeling data for training deep neural networks is daunting and tedious, requiring millions of labels to achieve the current state-of-the-art results. Such reliance on large amounts of labeled data can be relaxed by exploiting hierarchical features via unsupervised learning techniques. In this work, we propose to train a deep convolutional network based on an enhanced version of the k-means clustering algorithm, which reduces the number of correlated parameters in the form of similar filters, and thus increases test categorization accuracy. We call our algorithm *convolutional k-means clustering*. We further show that learning the connection between the layers of a deep convolutional neural network improves its ability to be trained on a smaller amount of labeled data. Our experiments show that the proposed algorithm outperforms other techniques that learn filters unsupervised. Specifically, we obtained a test accuracy of 74.1% on STL-10 and a test error of 0.5% on MNIST.

Furthermore, we test different initialization methods for ConvNets including unsupervised pre-training when there is an abundant amount of labeled data. Our results show that the initialization weights wash out with large datasets, and all the networks that are initialized with different methods converge to a similar training speed and overall performance.

3.1 Introduction

Deep neural networks require massive amounts of data to be trained. In large-scale datasets, supervised methods have been successfully trained over the past few years due to the advances in parallel computing [36,37]. Popular datasets such as ImageNet [47] contain more than a million labeled samples, and even larger datasets are already sought after by researchers in the field. Further pushing the boundaries, video datasets are becoming

increasingly important in the context of deep neural networks for event recognition tasks. In all such cases, labeling is necessary so that a supervised training algorithm can be used. However, the task of labeling data is quite expensive and time-consuming, requiring tedious work. For example, several hundreds of hours were spent to create ImageNet, and thousand of hours may be needed to annotate even the most simple video dataset [35]. To circumvent this problem, the research community recognizes that a large breakthrough lies in the use of unlabeled data, which is freely available in abundant quantities.

Over the last few decades, extensive research has been dedicated to learning feature hierarchies for deep learning in the context of image understanding. Examples include unsupervised, supervised, and semi-supervised learning. Such deep learning techniques use hierarchy of layers, which use "filters" to extract multiple input features and "connections" to combine extracted features together into inputs for the next layer. In earlier studies in the field, unsupervised pre-training was required for training deep networks by supervised learning methods. Recent advances in Convolutional Neural Networks (*ConvNets*) combined with abundant amounts of labeled data have shown great promises in object recognition tasks to remedy this issue [48].

On the other hand, unsupervised learning algorithms, such as k-means clustering, also increased the number of parameters in the network and achieved state-of-the-art results when labeled data are limited. Although unsupervised learning techniques using k-means algorithm were commonly used to train filters in several studies [49, 50], the network encoding structures present many similarities with ConvNets, such as the use of convolution and pooling in each layer.

The main differences between ConvNets and unsupervised learning techniques based on k-means applied to image recognition are the number of layers (depth) and the number of filters (width) at each layer, and the connections among layers. ConvNets improve accuracy by increasing network depth and width. Recent studies show that, significant performance of ConvNets was a result of the increased depth [51]. By contrast, unsupervised learning algorithms for deep networks were not able to scale to the same depth as conventional ConvNets. Therefore, recent unsupervised studies use large network width and two-to-

three layers with diminishing returns [49]. In this work, we demonstrate that learning the connections between the layers of deep neural networks plays a crucial role in improving the performance of unsupervised techniques.

While early work of ConvNets used to rely on a 'non-complete' connection scheme [52] to keep the number of connections within reasonable bounds, the trend has changed to fully-connected layers in order to exploit the benefits of parallel computing [37, 48]. Fully-connected layers perform a lot of potentially unnecessary operations because they connect every feature of the previous layer to every feature of the next.

In this study, a refined version of an unsupervised clustering algorithm that allows the filters to learn diverse features has been proposed. This is achieved by preventing the algorithm from learning redundant filters that are basically shifted version of each others as explained in detail in Section 3.3. Another major contribution of this work is that we learn sparse connection matrices between layers by forcing sparser group of features to map into the feature of next layer which has been explained in Section 3.4. We show that the convolutional k-means clustering algorithm can provide comparable mid-level feature hierarchies to the supervised networks with improved connection learning.

3.2 Related Work

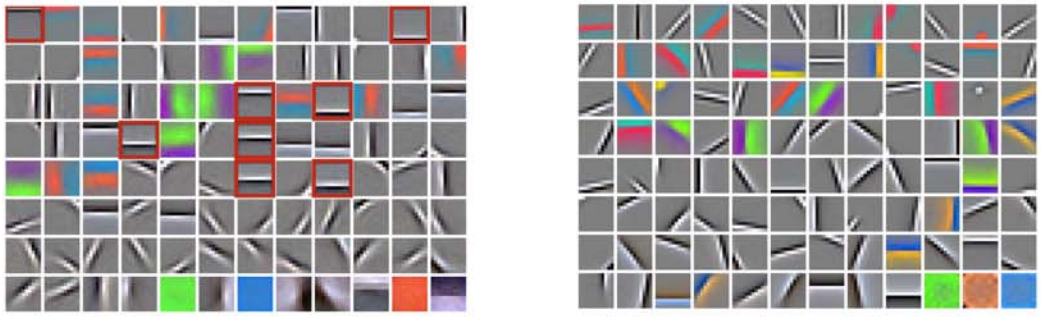
In recent years, there has been an increasing interest to learn ConvNets filters using unsupervised learning either in pre-training or when specifying the filter values. Earlier work suggested to use sparse coding and sparse modeling at patch level ignoring the fact that filters would be used in a convolutional manner [53, 54]. Such approaches result in duplicated filters that are simply shifted versions of each others. To address this problem, convolutional Restricted Boltzmann Machines trained with contrastive divergence [55] and convolutional sparse coding [56] methods were proposed.

Filters using k-means algorithm have gained significant attention in recent studies because of its simplicity and its competitive results when combined with the right pre-processing and encoding scheme [57, 58]. In these studies, filters trained with the k-means algorithm

are applied in a convolutional manner over the input maps to extract useful features. However, there has not been any attempt to reduce the redundancy between the filters learned with this algorithm, a problem that hampers efficiency and accuracy.

As in almost all statistical learning problems, curse of dimensionality is a known issue in deep neural networks. In particular, studies show that k-means performs poorly after the first layer [49]. The number of filters of the first layer have low dimensions, on the other hand, the subsequent layers increase the number of network parameters exponentially. As an example to the curse of dimensionality problem, if we have 32×32 RGB images, and we train 96 $3 \times 5 \times 5$ pixel filters in the first layer and convolve them with input images, we will get $96 \times 28 \times 28$ feature maps as output. If we want to train fully connected filters in the second layer (as in the first layer filters), we would need to train $96 \times 5 \times 5$ filters. The k-means algorithm fails to extract distinctive features and works poorly in such a high dimension. Therefore, for the mid level features, a smaller receptive field than fully connected layer should be preferred [49]. In the early work of ConvNets, [52] used parsimonious (not fully-connected) connection schemes to keep the number of connections within reasonable bounds and to force a break of symmetry in the network. Since different feature maps are fed with different input sets, the system is forced to extract different features. In techniques that use unsupervised algorithms, random connection [59], and grouping similar features [49] have been proposed; these results added additional layers and provided some improvement but not as significant as the ones obtained with supervised deep network.

In this work, we address the aforementioned problems by devising an optimized learning algorithm that avoids replication of similar filters. Since the filters will be used in convolutional operation, shifted versions of filters do not provide additional information to the feature hierarchy, and therefore should be avoided. We further propose to learn the connections between layers via supervised learning in the context of ConvNets. The connection setup uses 1D convolution across channels which is equivalent to the operation denoted as *mlpconv* layers in [60]. This layer has been used to enhance the abstraction



(a) kmeans (b) convolutional kmeans

Fig. 3.1. Filters trained on the STL-10 dataset with k-means and convolutional k-means. Filters are sorted by variance in descending order. While convolutional k-means learns unique features, the k-means algorithm introduces redundancy in filters. The duplicated features for horizontal edges are highlighted in red.

ability of the local model in [60], and to decrease the dimension of modules as well as to remove the computational bottlenecks in [36].

3.3 Learning Filters

3.3.1 Learning Filters with k-means

Our method for learning filters is based on the k-means algorithm. The classic k-means algorithm finds cluster centroids that minimize the distance between points in the Euclidean space. In this context, the points are randomly extracted image patches and the centroids are the filters that will be used to encode images. From this perspective, k-means algorithm learns a dictionary $D \in \mathbb{R}^{n \times k}$ from the data vector $w^{(i)} \in \mathbb{R}^n$ for $i = 1, 2, ..., m$. The algorithm finds the dictionary as follows:

$$
\begin{aligned}
s_j^{(i)} &:= \begin{cases} D^{(j)^T} w^{(i)} & \text{if } j = \underset{l}{\operatorname{argmax}} \left| D^{(l)^T} w^{(i)} \right|, \\ 0 & \text{otherwise,} \end{cases} \\
D &:= W S^T + D, \\
D^{(j)} &:= \frac{D^{(j)}}{||D^{(j)}||_2},
\end{aligned}
\tag{3.1}
$$

where $s^{(i)} \in \mathbb{R}^k$ is the code vector associated with the input $w^{(i)}$, and $D^{(j)}$ is the j'th column of the dictionary D. The matrices $W \in \mathbb{R}^{n \times m}$ and $S \in \mathbb{R}^{k \times m}$ have the columns $w^{(i)}$ and $s^{(i)}$, respectively. $w^{(i)}$'s are randomly extracted patches from input images that have the same dimension as the dictionary vectors, $D^{(j)}$.

Described learning scheme trains the centroid of each cluster at the patch level, however, in ConvNets, filters are applied to images in a convolutional manner. As observed in Figure 3.1(a), many of the centroids from the k-means training have almost the same orientation and they are shifted versions of each other in space. Therefore, after the convolution operation, they will produce redundant feature maps at neighboring locations. In the next section, we explain the proposed modifications of the k-means algorithm (*convolutional k-means*) that alleviates this problem.

3.3.2 Learning Convolutional Filters with k-means

In order to reduce the redundancy between filters at neighboring locations, we propose a new input patch extraction method. This method significantly reduces the redundancy in centroids produced by the k-means algorithm and keeps only the essential basis for them. The standard k-means algorithm extracts random patches from input images whose dimensions match those of the centroids. By contrast, the proposed method uses larger windows as inputs to decide which patch to extract for clustering.

The windows are chosen to be two times bigger than the filter size and randomly selected from the input images. The centroids of the k-means algorithm convolve the entire window to compute a similarity metric at each location of the extracted area. The patch

which corresponds to the biggest activation from the window is meant to be the most similar feature to the centroid (given that ConvNets have translation invariance). Finally, the patch at that specific location (biggest activation) is extracted from the window and it is assigned to the corresponding centroid. The modified dictionary learning can be written as follows:

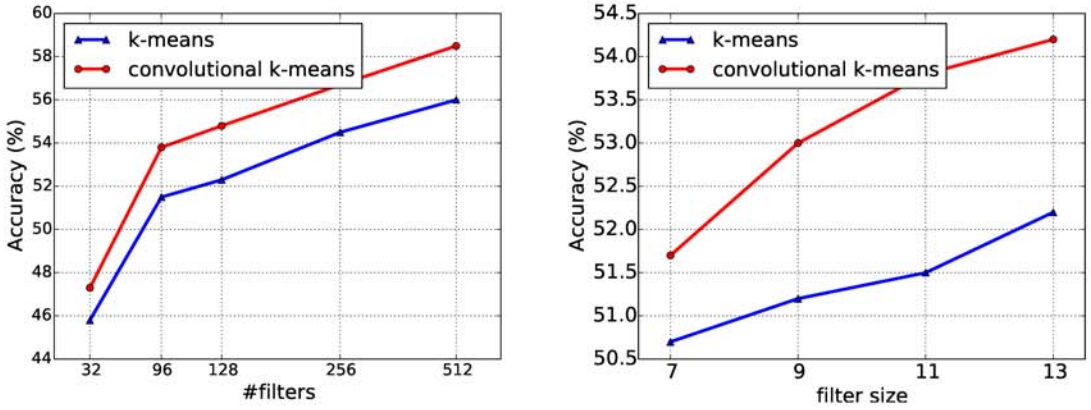
$$\begin{aligned} s_j^{(i)} &:= \begin{cases} D^{(j)^T} w_{(x,y)}^{(i)} & \text{if } (j,x,y) = \underset{(l,m,n)}{\operatorname{argmax}} \left| D^{(l)^T} w_{(m,n)}^{(i)} \right|, \\ 0 & \text{otherwise,} \end{cases} \\ D &:= W_{(x,y)} S^T + D, \\ D^{(j)} &:= \frac{D^{(j)}}{||D^{(j)}||_2}, \end{aligned} \tag{3.2}$$

where $D^{(j)}$ is the j'th column of the dictionary that corresponds to a $c \times s \times s$ 3D filter kernel and $w^{(i)}$ is the window with size $c \times 2s \times 2s$. x and y are the top-left location index of the input patch, and $w_{(x,y)}^{(i)}$ is the extracted patch from the location (x, y) with size $c \times s \times s$.

When these correlated filters are removed, there is more room for new filters to learn additional features. The filters that are trained with both k-means and convolutional k-means algorithms are presented in Figure 3.1. As can be observed from Figure 3.1(a), filters that are trained at the patch level with k-means algorithm have similar features but at different locations within a patch. As an example and also highlighted in red, there are many horizontal filters that are replicas of each other at different heights. By contrast, the filters that are trained with the convolutional k-means algorithm are significantly more diverse, as can be seen in Figure 3.1(b).

3.3.3 Experimental Results of Single Layer Network

We run experiments of a single layer network to analyze the effect of convolutional k-means. In our experiments, we use the STL-10 dataset that contains 96×96 RGB images in 10 categories [61]. This dataset has 500 images per class for training and 800 for testing. Additionally, it includes $100,000$ unlabeled images for unsupervised learning algorithms



(a) Accuracy versus number of filters

(b) Accuracy versus number of filters

Fig. 3.2. Comparisons of accuracy on the STL-10 dataset with filters that are trained by k-means and convolutional k-means. Tests use a single layer network and the sizes of filters are fixed to 11×11 for (a) while the number of filters is set to 96 for (b).

which are extracted from similar but broader distribution of images. For learning the filters with k-means clustering, only unlabeled data are used.For the training of filters with k-means and convolutional k-means, patches are randomly extracted from the raw images. Then, standard pre-processing, such as global contrast normalization and ZCA-whitening, is applied to the extracted patches.

For the encoding scheme, we only apply global contrast normalization to the input images. We fix the first layer filters of ConvNet with the trained dictionary by k-means and convolutional k-means. We reduce the dynamic range of the trained filters by dividing the filter values by a constant that is determined by cross-validation. In our experiments, we use the STL-10 dataset without any downsampling. However, the convolutional layer is applied with a stride of 4, which effectively reduces the dimension in the first layer. This layer is followed by a max-pooling operation, which reduces the dimension to $\mathrm{K} \times 2 \times 2$, where K is the number of filters that are used in convolutional layer. After pooling, rectification linear unit (ReLU) activation function is used; similar to recent works in ConvNets [48].

Note that to compare the effectiveness of filters that are trained with these two algorithms, we set a large pooling size which would decrease the dimension to K $\times$ 2 $\times$ 2, and train a single layer classifier. In the experiments, we use a learning rate of 0.1 with a momentum rate of 0.9.

In Figure 3.2, we compare the k-means algorithm with our convolutional k-means approach. In Figure 3.2(a), we fix the filter size to $3 \times 11 \times 11$ and change the number of filters. We observe that the increase in the number of filters provides us with higher performance for both algorithms, however, filters that are learned with convolutional k-means always outperform the ones with k-means algorithm. Note that to achieve a similar level of accuracy, such as 54%, the number of required filters for our approach is smaller than half of those for k-means. In Figure 3.2(b), we fix the number of filters to 96 and vary the size of the filters. Our approach outperforms k-means for all filter sizes.

3.4 Learning Connections

We also study a way to learn connections from one network layer to the next. Such connections are of extreme importance as creating groups of feature maps from which the following layer learns new features. While fully-connected layers make use of all the features of the previous layer into the next one, we use non-complete connection [52], which are more efficient in computation. These non-complete connections use multiple groups, each including a limited portion of the previous layer features. We use a sparse connection matrix that limits the local receptive field. Consequently, we can avoid the poor performance of the k-means algorithm when the input data are high dimensional [49].

Our method makes use of supervision with limited data while learning the connection weights between layers. The connections are described by a fully connected weight matrix that pools over the feature maps. Therefore, a single value in the weight matrix reflects how important that feature is for the corresponding group. To learn the relation between maps and organize them as groups (i.e., to define their weights), we add a convolutional layer with a predefined non-complete connection as illustrated in Figure 3.3. We attach a

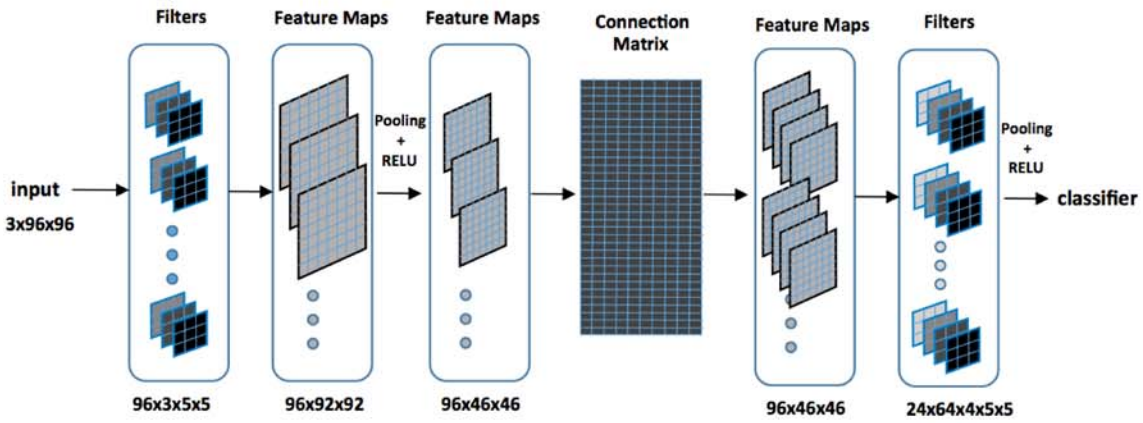


Fig. 3.3. Learning Connections setup. The setup network includes a connection matrix and a convolutional layer with a predefined non-complete connection scheme. First, the network with randomly initialized connection matrix is trained with supervised learning to learn the correct connection weights. Second, using the trained matrix, the next-layer filters are learned with convolutional k-means, as performed in the previous layer.

linear classifier after the convolutional layer and train the system using a backpropagation algorithm. The intuition of this setup is that since new filters are learned from groups of features and each weight matrix pools over features whose output is in a pre-determined group, the weight matrix is actually forced to learn the *proper* connections between the input feature maps and pre-determined groups by training. Therefore, even though we predefined the connections of the convolutional layer, the weight matrix provides the network with flexibility to define the connections in practice.

Note that the weight matrix that pools over feature maps can be considered as a 1D fully connected convolutional layer with enabled bias. This new approach allows us to limit the local receptive area for the k-means algorithm. It further allows us to create new complex and learnable interactions of cross channel information through the trained weights. After we learn the connections via supervised learning, we remove the learned filters from the network and only keep the connection matrix. This is because our goal is to learn the filters with unsupervised technique, using minimal labeled data, and to avoid overfitting. After these steps, our convolutional k-means algorithm is applied again on the pre-trained

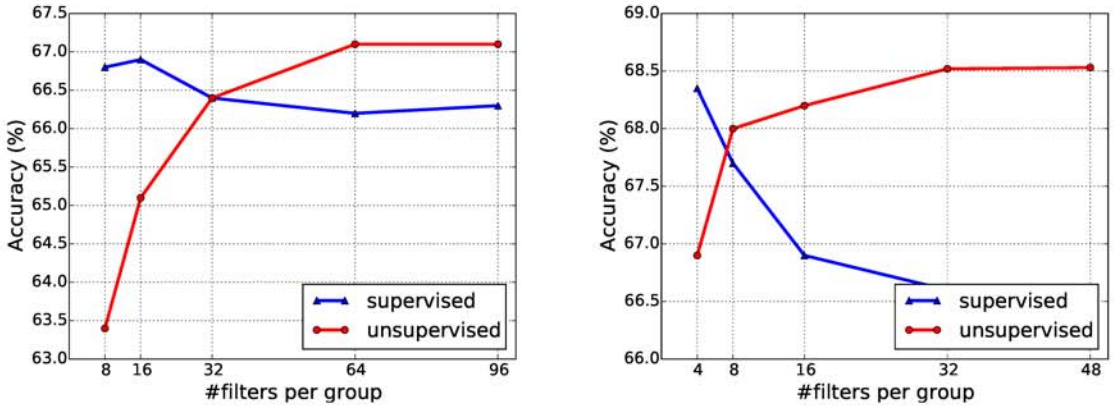
Table 3.1.
Classification accuracy on STL-10 testing set with 2 layer networks.

First Layer	Connection	Second Layer	Accuracy
Supervised	Supervised	Supervised	62.5%
Unsupervised	Random	Supervised	64.7%
Unsupervised	Random	Unsupervised	65.4%
Unsupervised	Supervised	Supervised	66.2%
Unsupervised	Supervised	Unsupervised	67.1%

connection matrix to learn the filters for the next layer, and the algorithm no longer suffers from the curse of dimensionality. Details and experimental results are presented in the next section.

3.4.1 Experimental Results of Multi-layer Networks

We conduct experiments combining (a) supervised and unsupervised learned filters and (b) supervised learned and random connections between layers. These experiments are designed to analyze the importance of learning connections. We set up a 2 layer network. The first layer has 96 filters of size 13×13. The convolutional layer is applied with a stride of 4 and followed by ReLU. Between the first layer and second layer feature extractors, we pre-define groups as 4 consecutive feature maps, which results in $96/4 = 24$ groups. From each group, we learn 64 filters. The size of second layer filter is chosen to be $4 \times 5 \times 5$, 4 comes from the choice of pre-defined non-complete connection scheme. After the convolution with the filters, we apply a pooling operation of 6×6 to decrease the dimensions. ReLU activation function follows the max-pooling operation. We use a linear classifier with 2 layers with a hidden neuron of 512 and interleaved with dropout [62].



(a) 2 layer network

(b) 3 layer network

Fig. 3.4. Comparisons of accuracy on the STL-10 dataset with filters that are trained by k-means and convolutional k-means. Performance comparisons of two and three layer networks with different learning methods on the STL-10 dataset. Supervised denotes that the corresponding layer trained via backpropogation and unsupervised denotes that filters are trained with the convolutional k-means algorithm after the connections are learned (this work).

For the *unsupervised learning* of filters, we apply convolutional k-means algorithm to the unlabeled data. *Random connection* refers to the case where the first layer filters are connected to the second layer with a pre-defined connection scheme. *Supervised connection* refers to the case where we train a supervised connection matrix of 96×96 before the pre-defined connection scheme. The second layer filters and the connection matrix are trained together, this forces the connection matrix to organize the feature maps such that the each group contains features that should be combined together. To learn the second layer filters in unsupervised manner with supervised connections, we fix the supervised trained connection matrix and train local filters for each group with convolutional k-means.

Table 3.1 shows the results of these experiments. First of all, training the whole network with supervised learning yields lower accuracy, as expected, due to overfitting to the limited number of labeled data. The unsupervised learning for the first layer provides a large performance increase over a fully supervised network. In our experiments, learning

the connections in a supervised manner boosts the performance for each case although the unsupervised learning still yields better performance. Furthermore, in Figure 3.4(a), we analyze the effect of the supervised and unsupervised learning of filters in the second layer. The unsupervised (k-means algorithm) and supervised (backpropogation algorithm) learning algorithms show different characteristics as we increase the number of filters in the second layer. K-means learning algorithm requires inclusion of increasing the number of filters to yield comparable results with the supervised backpropogation algorithm. Despite the fact that the supervised algorithm can more efficiently represent the data with fewer filters, it loses accuracy and overfits to the training set when the number of filters is increased. By contrast, the unsupervised algorithm (convolutional k-means) performs poorly with a low number of filters. This difference can be because the supervised algorithm is learning the discriminative features, whereas k-means learning algorithm learns all kind of common occurred features.

Finally, we extended the depth of the network to three to analyze whether the observed behavior continues with bigger networks. Using a configuration similar to the second layer, we add a third layer which includes a connection matrix that represents the connections and another convolution layer with non-complete connections. The connection matrix in this case decreases the dimension (size 1536×678) in a similar manner as [36]; this alleviates the computational bottlenecks. The other cascaded convolution layer groups each four feature maps and learns filters with dimensions $4 \times 3 \times 3$. This is followed by a ReLU activation function and max-pooling 4×4 to decrease the dimensions.

In Figure 3.4(b), we analyze the effect of supervised and unsupervised learning of filters in the third layer. The results present a similar behavior as in the second layer counterpart. The k-means algorithm requires to increase the number of filters to yield comparable results than the supervised backpropogation algorithm. By contrast, the performance of the unsupervised method can be increased further by concatenating the representations computed at different layers as an image feature vector for use in classification. Instead of just using the last layer output to feed the classifier, we concatenate intermediate layer outputs

to feed the classifier in our final results. The improvement is possible because our model does not overfit, as seen in other works [49, 58].

3.5 Final Classification Results

Finally, we compare our method against published state-of-the-art competing methods on the STL-10 and MNIST datasets. For this comparison, we mainly focus on algorithms that learn filters in an unsupervised manner. Multi-dictionary approach [49, 58] is the concatenation of the representations that are computed at different layers (i.e., output values) as an image feature vector. We use the same learning parameters, pre-processing and encoding scheme as were used in our other experiments (Section 3.3.3).

3.5.1 STL-10

For the final classification results, we use networks based on our two and three layer networks experiments. However, we increase the network size by replacing the stride in the first layer with a 2×2 max-pooling which increases the accuracy. We further increase the accuracy by the multi-dictionary approach. In detail, for the two layers with multi-dictionary network, we use a similar network from two layer experiment where we learned 64 filters from each 24 groups in the first layer output. We concatenate this network with a one layer network with 512 filters. The one layer network also includes ReLU activation and max-pooling to decrease the dimension of the output to $512 \times 4 \times 4$. For the three layers with multi-dictionary network, we use the network from three layer network experiment where we created 32 filters from 192 groups. We also concatenate this network with a one layer network with 512 filters. As in previous comparisons, the linear classifier uses 2 layers with a hidden layer of 512 and interleaved with dropout [62] with a rate of 0.5. As observed in Table 3.5.1, the two layer network with multi-dictionary achieves an accuracy of 71.4%. Note this value is significantly higher than all of the previously unsupervised learning algorithm work, while the network is an order of magnitude smaller (in number

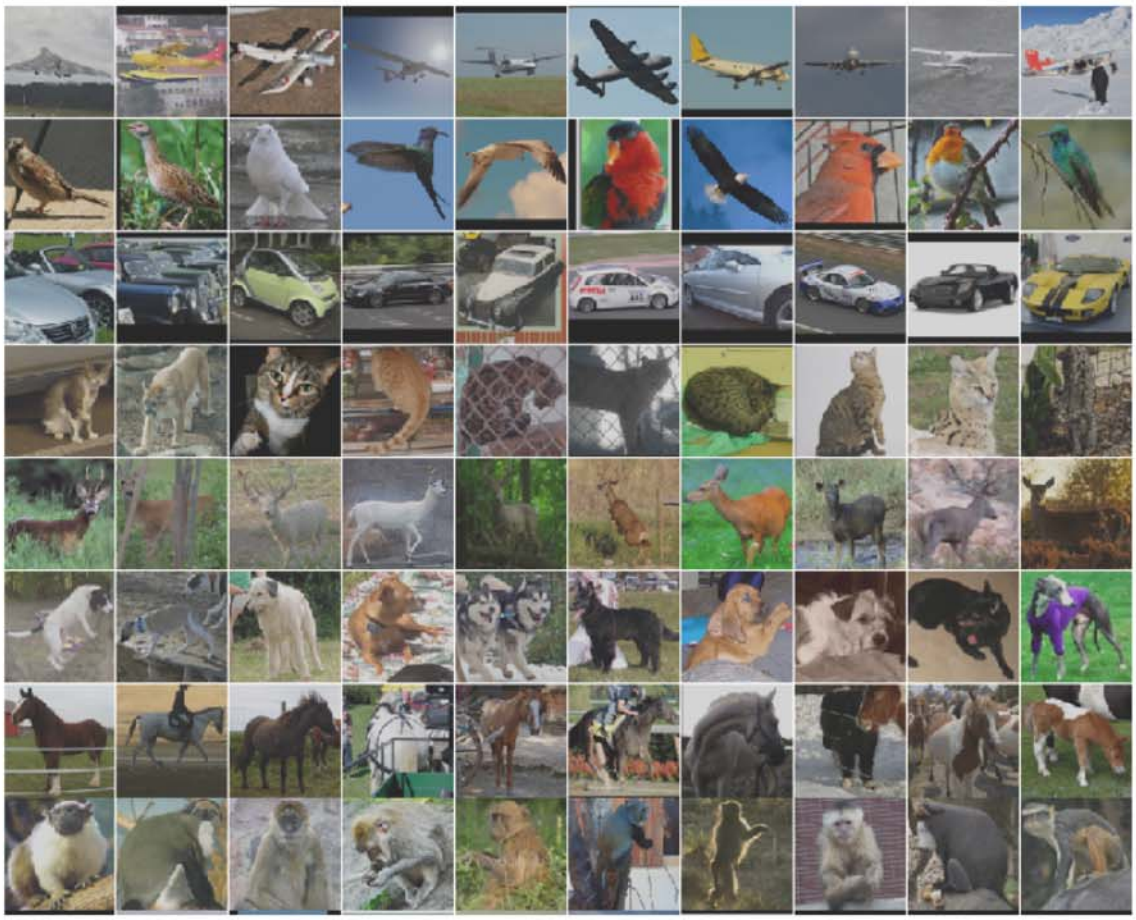


Fig. 3.5. Examples of images from the STL-10 dataset.

of parameters) than the networks used in [49, 58]. With an additional layer, our algorithm achieves an accuracy of 74.1%.

Table 3.2.
Classification accuracy on STL-10. Algorithms that learn the filters unsupervised.

Algorithm	**Test Accuracy**
[57] (1 layer)	59.0%
[49] (3 layers + multi dict.)	60.1%
[63] (3 layers)	63.7%
[50] (2 layers + multi dict.)	64.5%
[58] (3 layers + multi dict.)	67.9%
This work (2 layers + multi dict.)	71.4%
This work (3 layers + multi dict.)	74.1%

Table 3.3.
Classification accuracy on STL-10. Supervised and semi-supervised algorithms.

Algorithm	**Test Accuracy**
[64] (bayesian transfer learning)	70.2%
[65](unsupervised pre-training)	70.2%
[66] (triplet network)	70.7%
[67] (exemplar convnets)	72.8%
[42] (semi-supervised auto-encoder)	74.8%

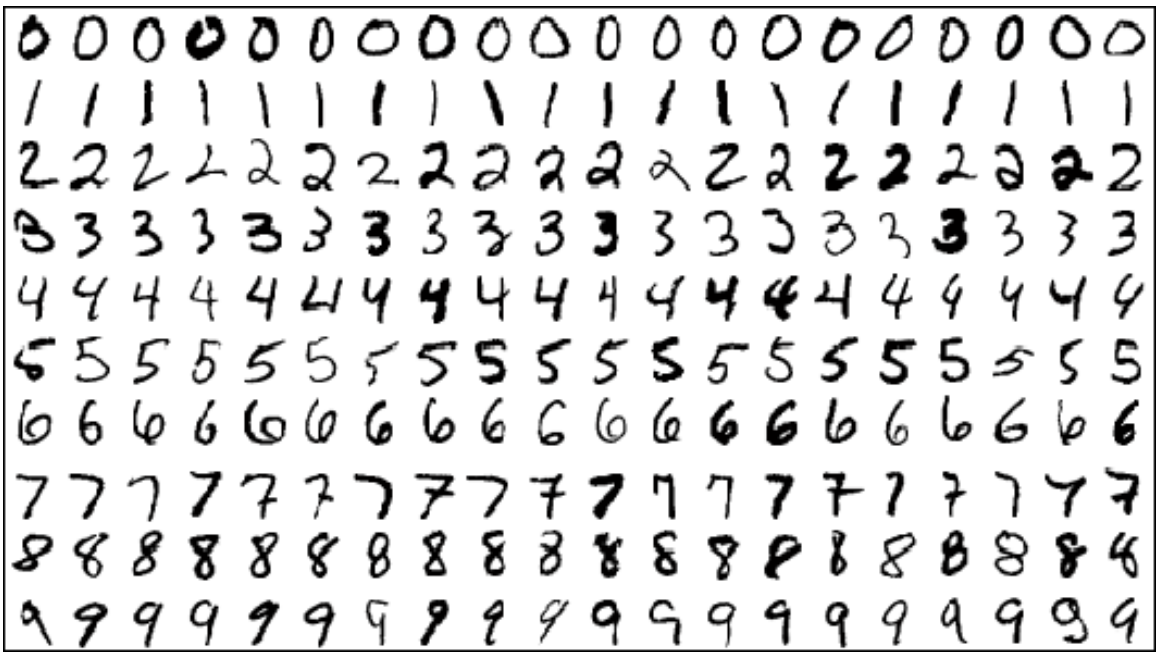


Fig. 3.6. Examples of images from the MNIST dataset.

3.5.2 MNIST

We run a series of experiments on MNIST dataset. Examples of MNIST dataset images are given in Figure 3.6. For testing, we use the standard $10,000$ test samples and use different sizes of labeled data for supervised trainings as presented in Table 3.5.2. The training data are randomly sampled from the entire dataset by making sure that each labels are uniformly distributed. For the unsupervised filter learning algorithm, we use the whole dataset, whereas for training the connections and the classifier, we only use the randomly extracted samples. We use the same two-layer network that was used on the STL-10 dataset, except this time we decrease the size of the hidden layer in the linear classifier to 256 and the concatenated one layer network has 96 filters. The experimental results for this dataset can be found in Table 3.5.2.

Table 3.4.
Classification test error on MNIST. Algorithms that learn the filters unsupervised.

Algorithm	600	1000	3000	All
[42] (auto-encoder)	8.4%	6.40%	4.76%	-
[68] (constractive auto-encoder)	6.3%	4.77%	3.22%	1.14%
This work (2 layers + multi dict.)	2.8%	2.5%	1.4%	0.5%

Table 3.5.
Classification test error on MNIST. Supervised and semi-supervised algorithms.

Algorithm	600	1000	3000	All
[52] (convnet)	7.68%	6.45%	3.35%	-
[69] (psuedo-label)	5.03%	3.46%	2.69%	-
[42] (semi-supervised auto-encoder)	3.31%	2.83%	2.10%	0.71%
[70] (generative models)	2.59%	2.40%	2.18%	0.96%
[44] (semi-supervised ladder)	-	1.0%	-	-

3.6 Summary

We have presented a novel framework that combines the strengths of an unsupervised clustering algorithm, k-means, and Convolutional Neural Networks when very few labeled data are available. Our framework modifies the k-means clustering algorithm so that, when used with ConvNets, it learns filters that are less redundant at neighboring locations. In addition, we proposed a supervised learning setup to learn the *proper* connections between layers. The idea of local connectivity applied to ConvNets mitigates the curse of dimensionality in filter learning and makes the algorithm scalable. Moreover, the proposed framework removes the necessity of data whitening on any of the layers including the input during the encoding phase (whitening is applied while learning the dictionary); which

makes the encoding stage very simple compared to the others [49, 63]. Our experiments show that the proposed algorithm performs better than the state-of-the-art among the techniques that learn deep neural network filters unsupervised.

3.7 Initialization on ImageNet

Celebrated deep neural networks do not have a long history. Before 2006, initialization techniques were insufficient for training deep multi layer neural networks. Recent advancements in optimization allowed scientists to successfully train deep networks [71]. Advancements have mostly included new initialization or more sophisticated training mechanisms. In this section, we will discuss different initialization techniques and the role of unsupervised learning algorithms that are used in pre-training. The main focus is to study the effects of several initialization techniques on the performance when large amount of labeled data are available. We experiment on these techniques on the ImageNet dataset which contains 1,500,000 images [47].

Deep ConvNets that include many layers and millions of parameters are currently trained with first-order optimization methods due to non-convex structure of the problem as well as the extremely large computational cost brought by second order updates. Batch optimization of deep ConvNets which only uses the first order information is ill-behaved due to the vanishing gradient problem [72]. The initial studies on this topic show that it is possible to train deep networks (3-4 layers) if the network is initialized with unsupervised pre-training [73, 74]. In these algorithms, each layer is trained in a greedy fashion using unsupervised learning algorithms. This procedure is followed by training the complete network with first-order optimization algorithm.

All the successful training methods introduced between 2006-2008 have the following property in common: They initialize their networks relying on unsupervised pre-training [41, 73–77]. Because of this, there has been an increasing interest in understanding how and why unsupervised pre-training helps deep learning. Erhan et al. [78] observe through extensive numerical studies that unsupervised pre-training acts as a regularizer that initializes the parameters in a "better" basin of attraction of the optimization procedure. More contemporary studies show that random initialization works fine as well, as long as they are well-designed [79]. Several recent results challenge the commonly held belief that first-order methods are insufficient to train deep networks with random initialization. Different

random initialization schemes have been proposed in that direction some of which we will cover in this section. Even though it has been suggested in several studies that unsupervised pre-training is helpful, it is not currently being used in any of the state-of-the-art models [15, 80].

3.7.1 Initialization Methods

It has been widely accepted that the starting values of the weights can have a significant impact on the training process. In particular, it is important to make sure that weights do not saturate in the non-linear layers. In the case of saturation, small gradients emerge and accordingly the learning tends to be slower [81]. We summarize several initialization techniques that are commonly used in practice:

1. **Heuristic Initialization [81]:** If the weights are very small/large as we go forward in the network, the resulting output values will get exponentially smaller/larger at each layer. This phenomenon causes vanishing or exploiting outputs due to numerical issues. Therefore, the magnitude of the weights should be kept in a reasonable range so that learning can proceed and no saturation occurs. To this extend, it is required that the distribution of the outputs at each layer has approximately unit standard deviation. This can be achieved by randomly drawn weights from a specific distribution (e.g. uniform) with mean zero and standard deviation given as:

$$\sigma_w = \frac{1}{\sqrt{n}} \tag{3.3}$$

   where n is the fan-in (the number of connections feeding into the node).

2. **Xavier Initialization [71]:** Extensive simulations show that heuristic initialization method that is discussed above causes the variance of the back-propogated gradients decrease as we go backwards in the network. From a forward-propagation point of view, the heuristic initialization method keeps information flowing with a constant variation throughout the layers. On the contrary, from a back-propagation point of

view, information flows with a smaller variation at each layer in this type of initialization technique. Xavier et al. [71] proposed a normalized initialization which approximately satisfies the objectives of maintaining activation variances and back-propagated gradients variance as one moves up or down the network. The proposed method uses the weights that are randomly drawn from a distribution (e.g. uniform) with mean zero and standard deviation as follows:

$$\sigma_w = \frac{\sqrt{6}}{\sqrt{n_i + n_{i+1}}} \tag{3.4}$$

where n_i is the fan-in (the number of connections feeding into the node) and n_{i+1} is the fan-out (the number of connections leaving out of the node).

3. **Normal Initialization [79]:** In normal initialization technique, convolutional weights are drawn from a standard Gaussian distribution, and the biases are set to zero. The justification of this approach is that the total amount of input to each unit should not depend on the size of the previous layer. Therefore, they do not easily saturate.

4. **K-means Initialization:** This method is a member of unsupervised pre-training in which weights are initialized by applying k-means algorithm at each layer.

3.7.2 Results

We perform our experiments on the ImageNet dataset [47]. This dataset is very challenging including 1000 categories most of which are fine-grained visual categories as shown in Figure 3.7. Another challenge is the size of the dataset. It includes 1,500,000 images for training and 50,000 images for testing where each image has dimensions $3 \times 256 \times 256$. Training networks with such high-dimensional data requires using GPUs with optimized libraries [82]. The average time of training is approximately 7 days using GPU (NVIDIA K40). The network architecture and the training parameters of OverFeat (the winner of the ImageNet 2013 localization challenge [47]) are used in our experiments [4]. The network has five convolutional layers and three linear layers. The number of parameters exceed 145 millions and the number of connections are about 2,810 millions.

Fig. 3.7. Two distinct classes from the 1000 classes of the ImageNet dataset. Left: Siberian husky, Right: Eskimo dog [36].

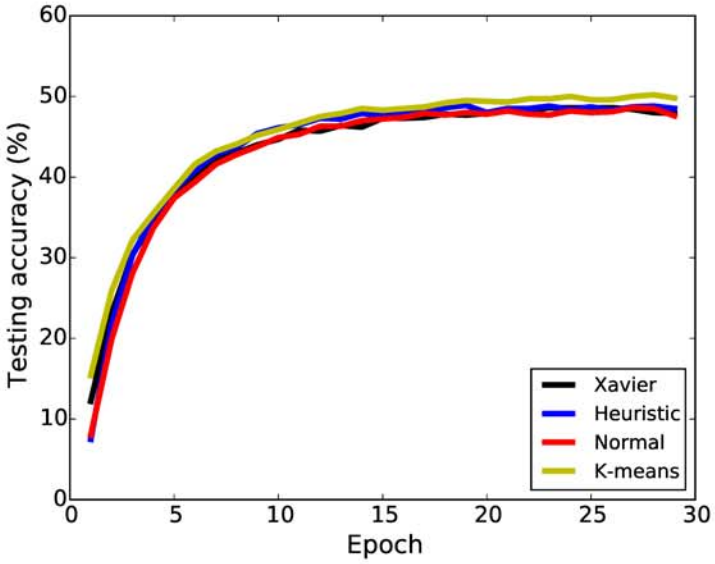
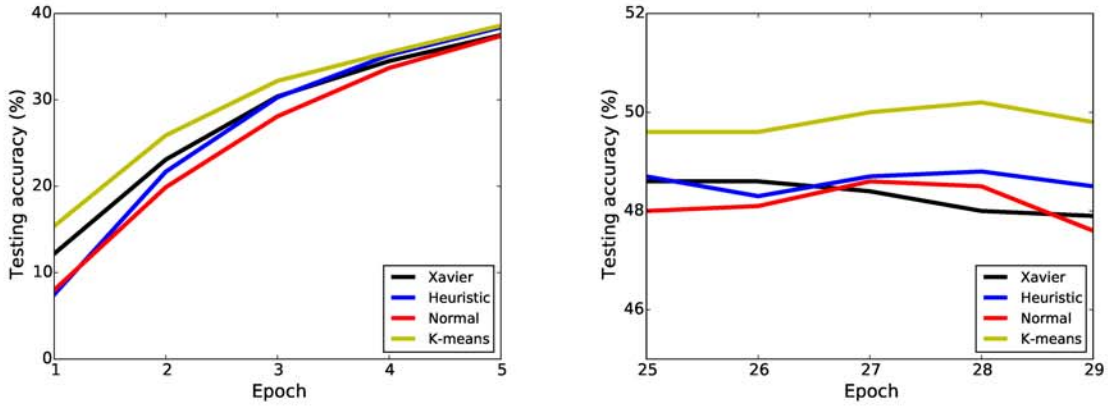


Fig. 3.8. Testing accuracy on the ImageNet dataset with different initialization techniques.



(a) Zoomed into the first 5 epochs.

(b) Zoomed into the last 5 epochs.

Fig. 3.9. Testing accuracy on the ImageNet dataset with different initialization techniques.

Figure 3.8 shows the learning curves of testing accuracy when the networks are initialized with different methods. The experiments show that when there are sufficiently large amount of labeled data, the importance of the initialization technique vanishes. Figure 3.9(a) zooms in to the first five epochs. Different initialization techniques result in different performance in the first few epochs. Initialization with k-means starts from a testing accuracy that is significantly higher than the others. On the other hand, Figure 3.9(b) (zoomed into the last five epochs) shows that differently initialized networks end up with similar performances. The network that is initialized with k-means unsupervised learning algorithm provides slightly better accuracy than the others.

3.8 Discussion

In this chapter, we showed that unsupervised learning can greatly help when there is a few number of labeled data. Later, in our experiments, we tested different initialization techniques and unsupervised pre-training methods when there is a large amount of labeled data. Our proposed framework is not used on the ImageNet dataset. Instead, we

tested algorithms that are already implemented in the well-known GPU libraries. These experiments were able to show us that the initialization technique has insignificant impact once there are sufficiently large amount of labeled data. The weights from different initialization techniques are washed out and after 3-4 epochs all the networks follow a similar learning trend independent of the used initialization technique. We conclude that unsupervised learning algorithms can provide a significant gain in performance when the number of labeled samples is small.

4. CONTEXT AUGMENTATION FOR LEARNING

Recent enhancements of deep convolutional neural networks (ConvNets) empowered by enormous amounts of labeled data have closed the gap with human performance for many object recognition tasks. These impressive results have generated interest in understanding and visualization of ConvNets. In this chapter, we study the effect of background in the task of image classification. Our results show that changing the backgrounds of the training datasets can have drastic effects on testing accuracies. Furthermore, we enhance existing augmentation techniques with the foreground segmented objects. The findings of this work are important in increasing the accuracies when only a small dataset is available, in creating datasets, and creating synthetic images.

4.1 Introduction

In recent years, ConvNets empowered by abundant amounts of labeled data have increased accuracies in many tasks, including object classification, detection, and face recognition [15,83,84]. However, when the training dataset is small, the ConvNets are not able to learn the necessary features, resulting in low accuracies in these tasks. To address this issue, several approaches such as transfer learning, unsupervised, and semi-supervised learning have been proposed. Nevertheless, they still require additional datasets or underperform their supervised counterparts.

In this work, we investigate the effect of the context (i.e., background) in training of ConvNets. 500 images from a common dataset, the STL-10 dataset [61], have been parsed as the background and the foreground objects (i.e., the object to be classified). These segmented images allow us to improve the accuracy on test dataset and also enable us to understand the importance of the context. Additionally, we present new augmentation tech-

niques based on the segmentation of the training dataset into background and foreground objects.

Our approach can be beneficial in several scenarios: i) Understanding ConvNets: The experiments with different backgrounds combined with foreground objects give us information about how ConvNets use background information. ii) To augment the available datasets to improve accuracies: Especially for categories, it is hard to collect data. In these cases, augmenting the data can lead to significant improvements in accuracy. iii) When a new dataset is collected. iv) For synthetic image creation; To bypass the effort of labeling and finding images, the images can be created synthetically. During this process, it is important to create backgrounds for the objects that are of interest. In this chapter, we analyze how the context directly impacts the accuracy of the ConvNets. We present several results showing how the user can identify such scenarios and improve dataset creation.

We analyze how the background plays a very important role in the ConvNets accuracy when the dataset is of limited size. We also present new augmentation techniques and enhance other commonly utilized augmentation techniques to improve the accuracy. Note that for our analysis, we use a very small amount of labeled data, i.e., 500 images, in order to explore how much we can advance the performance of the network when minimal data are available. The ability to train a ConvNet using a small amount of data has an advantage because data can be very expensive and for some categories large amounts of data may be difficult to find.

The main contributions of this work include:

- An analysis of the context (i.e., the background) on the accuracy of a ConvNet. Using our segmented images, we analyze the importance of the background and conclude that the training dataset should contain a diverse set of backgrounds, in order to facilitate stronger accuracy in the network.

- A novel augmentation technique based on the segmentation of images. When the dataset is small, the dataset can be augmented by pre-processing each image and

increasing the number of input images. We enhance the aforementioned techniques with the segmented information.

The results of this study are organized as follows: Section 4.2 discusses the related work. Section 4.3 explains the segmentation procedure. Section 4.4 and section 4.5 show our experimental setup for different backgrounds and training details respectively. Section 4.6 presents the results. Section 4.7 reviews augmentation techniques (2D affine transformations) and proposes segmented versions of these augmentations. Section 4.8 concludes.

4.2 Related Work

In this section, we describe alternative techniques to address the problem of learning from few data with ConvNets such as transfer learning, and techniques related to our work such as data augmentation, synthetic dataset creation, and understanding ConvNets.

Learning with few labeled data: The features that are learned from big labeled datasets also perform learned tasks well on other datasets which makes transfer learning a popular method while dealing with a small dataset [85, 86]. Transfer learning occurs when ConvNet's knowledge of an existing task is transferred to a new task in order to improve the networks ability to learn the new task. Transfer learning is applied by training a network with big labeled data and fine-tuning the classifier for the other small dataset. Despite the success of this approach, there is still a need for big labeled datasets to train the initial network. Furthermore, the transferred features perform poorly when the datasets, tasks, are less similar [87]. To decrease the need for big labeled datasets, extensive research has also been dedicated to unsupervised and semi-supervised learning algorithms which also utilize unlabeled data (e.g. [67]. However, these approaches underperform their supervised counterparts when the amount of labeled data is large, which shows that they do not efficiently represent the necessary features.

Synthetic Dataset: One approach to bypass the need for creating a labeled dataset manually is to create the images synthetically. Jaderberg et al. [88] successfully use a synthetic dataset for Optical Character Recognition (OCR) systems. In order to generate

the synthetic images, they randomly select fonts and render them with different colors over a background. This process, known as background blending, is especially important for scene text recognition in synthetic OCR systems. This process is an improvement to traditional OCR techniques which fail at detecting text in scene images due to the fact that they are tuned to work on black-and-white text and line-based printed documents. Thus, to create a synthetic dataset for generic object recognition, a more elaborate scheme is necessary for the background than was utilized in traditional OCR systems.

Recently, Peng et al. [89] use 3D CAD models to create a synthetic dataset for object detection. While they perform experiments with different backgrounds on their detector, their experiments are built on a pretained ConvNet [15] on the Imagenet dataset [47]. Therefore, they do not show the importance of having a variety of backgrounds while training an object classifier.

Data Augmentations: Data augmentation is the technique which improves the accuracy of ConvNets by increasing the size of the training dataset. Most augmentation techniques perform a 2D local transformation on the image (e.g., rotation and translation) to generate multiple variations of one input image [80]. This technique is commonly used to learn invariant features [15,80,90]. To facilitate augmentation, Dosovitskiy et al. [67] use a single image (i.e., a seed) per class to create an augmented dataset. Each image belongs to a different category and is accompanied by hundreds of transformed versions of itself (e.g. color, contrast). The network learns discriminative features that are invariant to some typical transformations. After the network is trained, the classifier is retrained by the labeled samples of the recognition task.

Understanding ConvNets: With the increasing success of ConvNets, there has been a growing interest in understanding various dimensions of its behavior such as the training process and the features that are learned. Several visualization techniques have been proposed to analyze and investigate what kind of features are learned in different layers of ConvNets [26,51,91]. There has also been further effort to measure the encoded invariance of a ConvNet: invariance to 2D transformations [92] and low-level cues [89].

Fig. 4.1. The foreground segmented images in the first row and the original image in second. All images belong to the cat category.

Additionally, there have been studies to analyze which inputs can fool the ConvNets. For example, the study conducted by Szegedy et al. [93] shows that small perturbations in the input images can make the network alter its prediction about the image from a correct label to a wrong one. Furthermore, Nguyen et al. [94] show examples of images that are unrecognizable to humans, but are categorized by state-of-the-art deep ConvNets as familiar objects with certainties greater than 99.6%. These findings raise questions about the generality, or the ability of ConvNets to be tolerant to small changes in images.

4.3 Segmentation-based Augmentation

In this section, we describe the importance of image context, how images are segmented, and how we use the segmented images to create a novel augmentation technique.

4.3.1 Image Context

The context of an image (i.e., the background) has significant importance for us while recognizing objects, especially when images are small. Figure 4.1 shows the foreground

Fig. 4.2. Examples of segmented images.

in the first row (i.e., the main object to classified) and the original image in second. According to our observations, it is hard to recognize some objects using just the foreground information because the context of the image helps to classify them. The key inspiration of adding the dimension of foreground and background in object recognition comes from how we perceive the world. Most animals have binocular vision, which enables them to perceive depth, and to distinguish objects by differentiating the object in respect its surrounding context. In the following section, we analyze how the extra information added by segmentation can be used to augment the dataset and improve the accuracy.

4.3.2 Segmentation

To be able to analyze the importance of the context, we need to discriminate between foreground and background. We explore several options to segment the foreground objects: i) datasets with depth data, ii) automatic segmentation, iii) manual segmentation. We discarded datasets that contain depth information because they are very limited in size, and it is hard to compare with other well-known datasets. We expect that these datasets will become more abundant with the development of depth cameras in mobile environments ([95,96]). We explored the automatic segmentation approach using the methods proposed

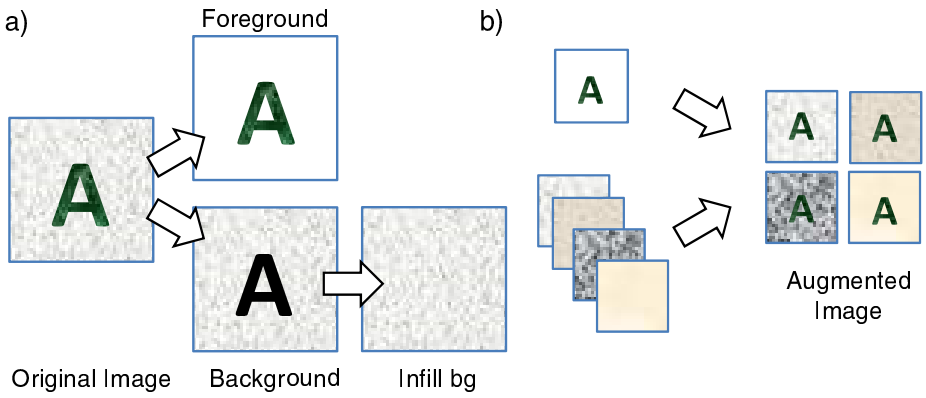


Fig. 4.3. Segmented-based Augmentation Overview. a) The image is split into foreground and background. The background is filled in. b) Each foreground can now be combined with the infilled backgrounds which creates new examples for the training.

by Long et al. [97] and Comaniciu et al. [98], as well as by using a graph-cut-based segmentation [99]. However despite their strong performance on other datasets, these algorithms perform poorly on the STL-10 dataset. The reason for that can be that the images are small 96×96 and are also noisy. We decided to pursue a manual segmentation. This has two main advantages: One, the segmentation does not contain noise that would propagate to the augmentation, and two, it allows us to use a well-known dataset.

To speed up the process of creation and save additional information like the click positions over time as well as the final mask, we developed an interactive segmentation application. We realized that we needed to present the same image in several sizes to the users for the users to segment the objects properly. Several examples of the segmentations can be seen in Figure 4.2.

4.3.3 Segmentation Augmentation

Once the images are segmented, we can alter their background to analyze their importance and create our segmentation-based augmentation (Figure 4.3). Note that since the foreground of each image does not match the others, we need an algorithm to fill in the

Fig. 4.4. Images of extracted backgrounds. Gaps are automatically filled using *patch-match* approach [102].

gaps that have occurred because of the removal of segmented objects from the images. Recently there have been several approaches to completing the scene of a photograph. Hays et al. [100] patch up holes in images by exploiting a large database of pictures with similar statistics. Other studies focus on the minimization of artifacts of pattern-based inpainting [101].

For our task, we use the method proposed by Barnes et al. [102]. This algorithm quickly finds correspondences between small square regions of an image. Using corresponding patches extracted from its surroundings, the algorithm infills gaps in images. Some examples of infilled background images can be seen in Figure 4.4.

4.4 Set-up with Backgrounds

In this section, we present different combinations of backgrounds and foreground objects we use in our experiments. They are as follows:

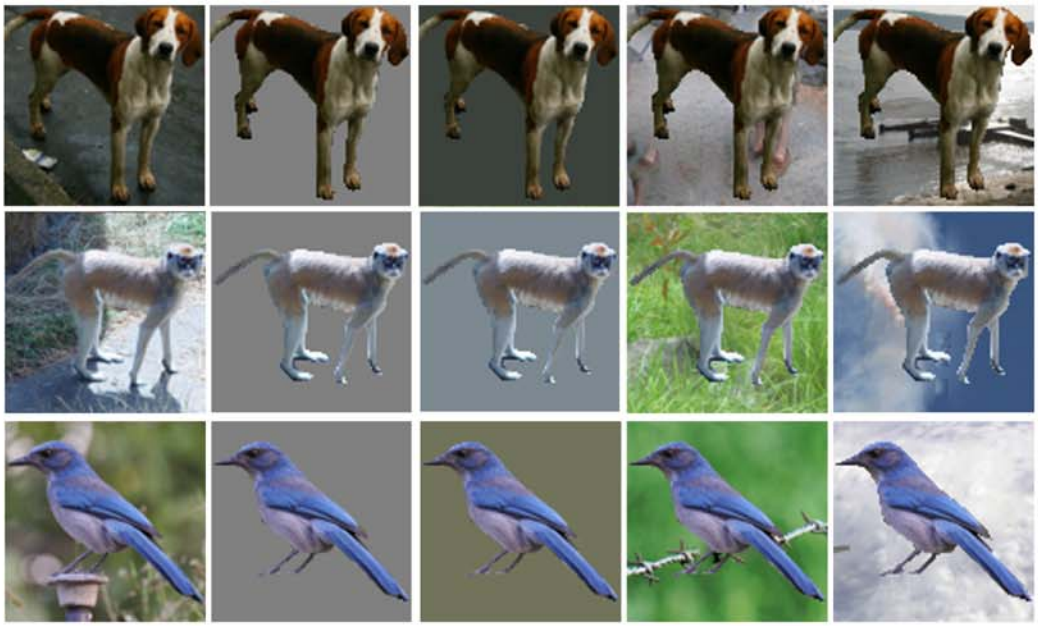


Fig. 4.5. Different set-up of context during training. First column has original images. Second column images are foreground objects with uniform gray color as background. Third column has mean value of the extracted background replaced with the background. Forth column has examples of foreground objects combined with different backgrounds from the same category as themselves. Fifth column images are foreground objects that are combined with backgrounds from other categories.

1. **Only bg** (only background): After the foreground objects are segmented and removed from the background, the gaps are filed with *patch-match* approach [102]. Examples of the backgrounds that are created with this process are shown in Figure 4.4. Training with these images is interesting because even though they do not include the objects in which we are interested, they have the information of the environment in which the object is normally found.

2. **Gray bg with fg** (gray background with the segmented foreground objects): These images have the segmented foreground objects while their backgrounds are replaced with a gray value. This process takes all the information about the background and

leaves only the foreground objects. Examples can be seen in Figure 4.5 (second column).

3. **Mean value of bg with fg** (mean value of the removed background with the segmented foreground): These images instead of a uniform gray color as background have the mean value colors of the extracted backgrounds. Therefore, if the image was a plane in the air, the mean value is most likely to be blue, and then the background becomes a uniform blue. Examples of these images can be seen in Figure 4.5 (third column).

4. **Bg same category with fg** (backgrounds from the same category combined with the foreground objects from the same category): In this set-up, segmented foreground objects from each category are combined with the background images from the same category as well. Examples can be seen in Figure 4.5 (fourth column), in the first row a segmented dog from the first image is combined with a background image from image that belongs to a dog category. These images preserve the 'correct label' for the backgrounds and the foreground objects. For segmented 500 images, 50 images per category, this process creates $50 \times 50 \times 10 = 25000$.

5. **Bg all categories with fg** (backgrounds from all the categories combined with each foreground object): These images are basically all possible combinations of the segmented foreground objects and the background images. For segmented 500 images, this process creates $500 \times 500 = 250000$. Two of the examples can be seen in the last column of Figure 4.5, the first one is an image of a dog with a background from an image of a plane. The second one is a monkey with the background again that belongs to a plane. These images look less realistic than the previous examples to human observers because we know that a monkey can not hang in the air, and there will not be a similar image in the test dataset. Combining the images as such provides many examples for each category. Therefore, it may decrease the problem of the network overfitting to the training dataset. On the other hand, it removes the background

information in a way that the network cannot take advantage of the background while categorizing an object because the same background appears for each category.

4.5 Experimental Setup

We use the STL-10 dataset for our experiments [61]. This dataset contains 5000 training and 8000 testing images from 10 categories. In our experiments, we use 500 of these training images, 50 from each category. The images are RGB colors and 96×96. We only pre-process the images by global contrast normalization.

In the experiments, we use a 4 layer ConvNet configured as (96) 7c- 3p - (256) 5c - 2p - (512) 3c - 2p - (10) c1 where (96) 7c denotes convolution with 96 filters each is 7×7. 3p denotes 3×3 pooling with a stride of 3. ReLU non-linearity operation follows each convolution layer. As a classifier, global average pooling is used [60] followed by a softmax layer.

The networks are trained with a learning rate of 0.1 and a momentum of 0.9. The training used stochastic gradient descent algorithm with a batch size of 10. For each experiment, we repeat the training 10 times with different random seeds and provide mean value and standard deviations of the test accuracies. For the experiments that each foreground objects may appear with many different backgrounds, we make sure that in the same epoch (500 unique images) each foreground object and background appear only once with random combinations.

4.6 Results on the Background

The 500 training images without any augmentation results in the accuracy of $47.42 \pm 1.14\%$. With the 500 segmented foreground images and their filled background counterparts, we perform many experiments as shown in Table 4.2.

Only bg: In our first experiment, we use only the background for training. Examples are shown in Figure 4.4. We train the network with these images by conserving the labels of the original images. Even though we have removed the foreground objects (i.e., the

Table 4.1.
Experiments with different combination of backgrounds (bg) with the foreground objects (fg).

	#images	Accuracy
Original images	500	$47.4 \pm 1.1\%$
Only bg	500	$31.3 \pm 0.8\%$
Gray bg with fg	500	$28.7 \pm 1.0\%$
Mean value of bg with fg	500	$36.7 \pm 0.9\%$
Bg same category with fg	25000	$54.4 \pm 0.9\%$
Bg all categories with fg	250000	$48.5 \pm 1.0\%$

objects of the categories) from the examples, we are experimenting to see if the network can still learn something from the background that would result in a test accuracy that is better than chance (10%). Ships are usually in the water, planes are in the sky, and birds are on the branches of trees. Therefore, it would not be surprising if the network achieves test accuracy better than chance. In fact, the network correctly classifies 31.3% of the images from the test dataset which is quite surprising.

Gray bg with fg: In the second experiment, we use the foreground objects with a gray background. This removes a lot of information from the training dataset. Therefore, the network performs poorly on the test dataset. In fact, it gives worse performance than the training with only background images (28.7% as opposed to 31.3%).

Mean value of bg with fg: In the third experiment, the gray value is replaced with the mean color of the background as can be seen in Figure 4.5-third row. It provides more information and variety in the training dataset. The mean value is also partially adding a background information.

Bg same category with fg: In our fourth experiment, we increase the dataset by combining the foreground objects with the backgrounds from the same categories. Examples can be seen in Figure 4.5 (fifth column). This combination preserves the correct label for

both the foreground images and the background images while creating 25000 examples. Another surprising result of this experiment is that the accuracies increase 15% compared to the training with original images.

Bg all categories with fg: In the fifth experiment, we combine each foreground object with each background object. Segmentation of 500 images gives us 500 unique foreground and background images. Combining them creates 250000 different images. The result is slightly better than the training with original images (48.5% as opposed to 47.4%). This experiment is also in a way removing the background information from the dataset. Because the same background appears for each category, the network cannot take advantage of the statistical information of the background while classifying an object during training. From this perspective, it is similar to the experiment with gray backgrounds. On the other hand, it also creates many examples of the foreground objects in different backgrounds and reduces the overfitting problem.

These experiments show us the importance of the context while training ConvNets. As they learn the foreground images, they also learn the backgrounds. Interestingly, as evidenced in our first experiment, sometimes background information is all that is needed to categorize some images. Another interesting finding is the significant increase in accuracy when the data are augmented with the combinations of foreground and background images from each category.

Finally, our last experiment shows the delicate balance between having more examples which can potentially increase accuracy and removing background information in order to create these examples which decreases accuracy. The result is better than the training with original images, but is much worse than the training with the combinations of the same label of foreground and background images. On the other hand, even though the results are worse, we know that the network uses the foreground objects to recognize the categories. This recognition can be useful for many applications and is still better than just using the original images, because despite the fact that the original images use background information, they still result in lower accuracy.

4.7 Augmentation Techniques

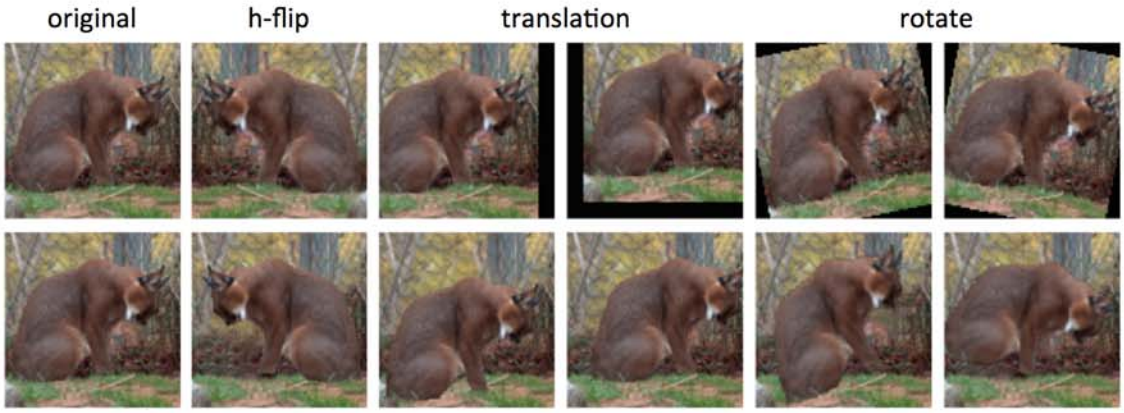


Fig. 4.6. First row are images that standard data augmentations applied to the whole image. Second row are the images where the backgrounds stay the same and the augmentations are only applied to the foreground objects. They are referred to as segmented versions of the corresponding data augmentations.

We are interested in understanding if the segmented objects and background can be useful for data augmentation that would result in increased accuracy. In the previous section, we saw that combining foreground and background images can provide an increase in accuracy. In this section, we experiment with 2D transformations of the segmented objects.

First, we review standard data augmentation approaches that are used to train ConvNets. We are studying how each augmentation technique improves the test accuracies when the amount of labeled data is very few. In addition, we propose new augmentation techniques that are based on segmented object images and background images. Examples of the augmentation techniques we used in our experiments can be seen in Figure 4.6 where the first row contains examples of standard augmentation methods and the second row shows examples of segmented version of these augmentations. They are as follows:

- Horizontal flipping (hflip): Images are flipped horizontally, which gives us the mirror images of the originals. Segmented horizontal flip is when only the the foreground objects are flipped while the background stays the same (Figure 4.6 - second column).

- Translation: Images are shifted of random values which are uniformly selected between $(-20, 20)$ pixels in vertical and horizontal directions. Segmented translation of the images corresponds to the same operation but only applied to the segmented foreground object with the values of $\{-20, -15, -10, -5, 5, 10, 15, 20\}$ (Figure 4.6 - third and forth column)).

- Rotation: Images are rotated by a random angle between $(-10, 10)$. Segmented rotation of the images, similar to the previous examples, rotation is only applied to the foreground objects with angles of $\{-10, -5, 5, 10\}$ (Figure 4.6 - fifth and sixth column)).

We also experimented with several other data augmentation methods in addition to the above (e.g. changing the color - adding a value to the hue component in HSV representation, and changing the contrast of the whole images, changing the color of the foreground objects and the background independently and adding Gaussian Noise), but we did not observe any accuracy gain with these augmentations.

In our first experiment in this section, we test standard data augmentations on the original dataset. We run experiments in 2 scenarios: 500 training images and 5000 training images (all labeled examples from the STL-10 dataset). The network achieves an accuracy of 47.4% with 500 training images, and it achieves an accuracy of 69.8% with 5000 training images, without any augmentation techniques applied.

Figure 4.7 displays the influence of each data augmentation on the test accuracy. The segmented augmentation versions increase accuracy less than their original counterparts. In the segmented augmentations, the background stays the same. In standard augmentation the background is also augmented which further increases variety in the training dataset. On the other hand, the segmented and the original augmentation techniques create different images, and they can be used together to further boost the performance.

In our final experiments, we combine all the techniques that result in improvements when training the ConvNet with 500 images. Table 4.2 presents the results. Each row includes data augmentations from the previous rows. Note that when combined with other

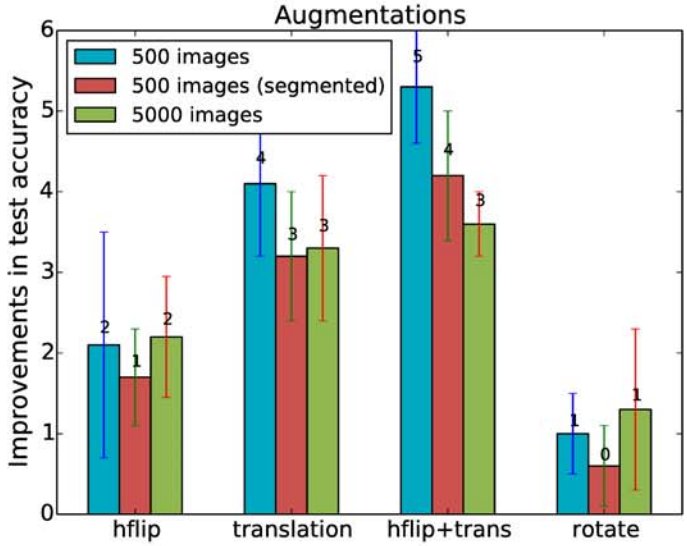


Fig. 4.7. Influence of augmentations on the STL-10 dataset when 500 images and 5000 images are used. Baseline accuracy value for the training with 500 images is 47.42% and with 5000 images, it is 69.8%. For 500 images, the segmented versions of the augmentations results (label 500 images segmented) are given as well as the standard augmentation techniques' (label 500 images). For 5000 images, only the standard augmentations are applied (label 500 images).

augmentation techniques, rotation did not improve results. Therefore, rotation has been removed from the table. With only 500 images enhanced with the augmentation techniques, the network achieves an accuracy of $59.5 \pm 0.8\%$. The cumulative augmentation and segmentation algorithms applied, resulted in a 25.5% improvement in image recognition from the initial dataset.

4.8 Summary

We provided an analysis of the impact of background while training ConvNets. We also proposed different techniques to improve the accuracies when only a small dataset is available to us.

Table 4.2.
Accuracies with 500 unique examples. Each row includes the augmentation from the previous rows.

	Accuracy
Original images	$47.4 \pm 1.1\%$
Bg same category with fg	$54.4 \pm 0.9\%$
Horizontal flip	$55.4 \pm 0.7\%$
Translate	$58.9 \pm 0.4\%$
Segmented hflip	$59.5 \pm 0.8\%$

Recently, deep ConvNets have closed the gap with human performance in many tasks and contain promising advances in the field of visual understanding. The problem is their need for huge labeled datasets. The question which arises is whether the ConvNets fully utilize the data provided to them. In this work, we observed that creating datasets which utilize foreground and background objects from the same category increased the performance of the network.

We also observed that in certain instances background may give enough information for the network to categorize some of the objects while in other instances the network is unable to do so with the given background information. Furthermore, we proposed new augmentation techniques that provide accuracy increase in ConvNets. The findings of this work are important in increasing the accuracies when only a small dataset is available, in creating datasets, and creating synthetic images. The ability to train a ConvNet using a small amount of data has an advantage because data can be very expensive and for some categories large amounts of data may be difficult to find.

5. EMBEDDED STREAMING CONVNETS ACCELERATOR

Deep Convolutional Neural Networks (ConvNets) have become a very powerful tool in visual perception. ConvNets have applications in autonomous robots, security systems, mobile phones, and automobiles where high throughput of the feed-forward evaluation phase and power efficiency are important. Because of this increased usage, many FPGA-based accelerators have been proposed. In this chapter, we present an optimized streaming method for ConvNets' hardware accelerator on an embedded platform. The streaming method acts as a compiler, transforming a high-level representation of ConvNets into operation codes to execute applications in a hardware accelerator. The proposed method utilizes maximum computational resources available based on a novel scheduled routing topology that combines data reuse and data concatenation. It is tested with a hardware accelerator implemented on the Xilinx Kintex-7 XC7K325T FPGA. The system fully explores weight level and node level parallelizations of ConvNets and achieves a peak performance of 247 G-ops while consuming less than 4 watts of power. Our results indicate high performance efficiency, outperforming all other presented platforms while running these applications.

Furthermore, this chapter includes an extensive review of related work. There are basically two approaches to accelerate ConvNets: 1) Decreasing the number of calculations that are needed. 2) Exploring different level of parallelisms of ConvNets in different hardware resources such as CPUs, GPUs and FPGAs. In the related work, we cover all these approaches.

5.1 Introduction

Recent advances in optimizing deep networks such as dropout [103], sub-sampling [104, 105] and efficient GPU implementations for training have enabled researchers to increase the scale of ConvNets with more layers and parameters that were not previously

practical to train. This development has resulted in increased accuracy of many tasks, including object classification and detection [15]. As a result, ConvNets have become particularly useful for applications in autonomous robots, security systems, mobile phones, and automobiles, which require real-time execution and high accuracy. ConvNets are, however, computationally very expensive due to the fact that over ninety percent of the time required for convolution operations with images is spent on processing by the filters [106].

One approach to accelerate ConvNets is to reduce their size, thus removing unnecessary parameters which results in a significant reduction in computational calculations. Recently, many studies show that ConvNets do no use all their capacities [107]. Whereas a small network does not learn as good features as a bigger network, it is possible to decrease the pre-trained big network's size to a smaller one without loss in the accuracy. This fact shows that big ConvNets are overly parameterized, however, this redundancy seems necessary to solve a highly convex optimization [108].

Another line of research to accelerate ConvNets is to exploit different level of parallelisms. Parallelism in ConvNets can be explored at many levels, including weight parallelism (parallel sum of products computation in convolution) and node parallelism (computation over multiple convolutional planes), and specific hardware architectures take advantage of a specific subset of them. Extensive research has been conducted to find the best routing schemes of ConvNets on CPUs, GPUs, and FGPAs (custom hardware).

GPUs are becoming a common alternative to custom hardware in vision applications because they are inexpensive and easily programmable [82, 109]. In particular, while training ConvNets, GPUs provide high-speed processing of hundreds of images at the same time. However, custom hardware can provide better performance during the feed-forward prediction phase with less power consumption, which is necessary for embedded systems. By developing a custom architecture and control method adapted to ConvNets, the product of power consumption by performance can be improved. Because of these advantages, extensive research has been devoted to custom architectures for convolutional networks or related algorithms [110–116].

This study presents a highly optimized control method for the scalable hardware architecture for ConvNets. The control method acts as a compiler and transforms high-level representations of ConvNets into operation codes for the purpose of executing applications within a hardware accelerator. The control method takes advantage of the characteristics of ConvNets and explores the computational power of the custom hardware.

The results of this study are organized as follows: Section 5.2 explains related work; Section 5.2.1 covers the methods that successfully remove unnecessary parameters of ConvNets. Section 5.2.2 describes different implementations of ConvNets on CPUs and GPUs and custom hardware. Section 5.3 describes the architecture of the custom hardware. The strengths and limitations of the custom hardware are presented in this section. Section 5.4 describes our control method that is optimized for ConvNets and custom hardware. Section 5.5 gives experimental results on the performance of the system. The final section is a comprehensive summary.

5.2 Related Work

With the recent success of Deep ConvNets in object recognition, there has been extensive interest in accelerating forward and backward data processing of ConvNets. There are basically two approaches to accelerate ConvNets. First approach, covered in Section 5.2.1, is decreasing the network size. Second approach, studied in Section 5.2.2, is exploring different level of parallelisms in ConvNets on different hardware resources such as CPUs, GPUs and FPGAs.

5.2.1 Acceleration through reducing ConvNet size

A recent study shows that ConvNets do not utilize all their capacities because the study is able to predict a big chunk of ConvNet's parameters given only a subset of them [117]. This means that parameters of ConvNets are correlated. In other words, ConvNets store similar information wasting computing resources. Inspiring from this work, researchers propose different ways to remove the redundant parameters in ConvNets. One way is to re-

move correlated parameters of pre-trained networks and retrain them (fine-tune) to recover the original performance. The other way is to train a network from scratch in a way that is designed to penalize redundant parameters to emerge. In this section, we will cover these two approaches.

Reducing the size of a pre-trained network

To reduce the redundancy in ConvNet filters, some studies aim at finding appropriate low-rank approximations of each convolutional layer filter [118, 119]. They then fine-tune the networks until the original prediction performances are restored. The low-rank approximations include several elementary tensor decompositions based on singular value decompositions and filter clustering methods that explore the similarities between the learned tensors. These studies obtain a speedup of $2\times$ by keeping the accuracy within 1% of the original models.

Another approach to reduce the size of a network is network pruning. Network pruning has been used to reduce network complexities in many studies such as *optimal brain damage* [120] and *optimal brain surgeon* [121]. However, these approaches are computationally expensive because they prune networks based on the Hessian loss functions which require second order derivatives. On the other hand, recent studies show that it is possible to remove parameters without any approximations or additional computations and any losses in accuracies [122, 123]. These studies take advantage of the fact that parameters in ConvNets are very sparse, many of them having values close to zero. They prune networks using a three-step method. First, they train a network to learn which of its connections are important. Next, they prune the unimportant, low-weight, connections. Finally, they retrain the network to fine tune the weights of the remaining connections. With this recipe, these studies reduce the number of parameters of popular networks by a factor of $10\times$. Furthermore, removing these connections reduces the convolution operation to sparse matrix multiplication which provides a speed-up gain by $5\times$.

Learning a small network from scratch

There has been an extensive effort to learn small networks from scratch that have similar performance with bigger networks. Jonghoon et al. [124] apply structural constraints to conventional ConvNets in order to learn low rank, 1D separated filters. The constrain sets 1D filters that are cascaded in across channel, vertical, and horizontal dimensions. Therefore, a network learns separated filters instead of a standard 3D filters. This structure while achieving a similar performance with vanilla ConvNets, uses $10\times$ fewer parameters and provides 2 to $3\times$ speedup.

Another interesting approach to accelerate ConvNets is to teach a small network to mimic a bigger one [108]. In almost any machine learning algorithms, a simple way to improve the performance is to train many different models on the same data and average the predictions of these models [125]. Caruna et al. [126] show that the knowledge of an ensemble can be compressed into a single model. Inspired by this work, Hinton et al. [108] use transfer learning to compress the knowledge of a big network into a smaller one. They achieve many interesting results, such as small network learns so much faster and converges to a better accuracy when it learns from a bigger network instead of correct labels of a dataset. However, this decrease in the network size accompanies with a decrease in the performance as can be seen in Figure 5.1.

Figure 5.1 shows an experiment we conducted by using a popular dataset called Cifar-10 [127] with a popular network structure called *network in network* [60]. Big network refers to the reference network [60], and the small network is the network that is half size of the big network, e.g. whereas big network has 192 filters in the first layer, the small network has 96 filters. Hard targets refers to real labels from the dataset, whereas soft targets refer to the outputs of the big network. Therefore, when soft targets are used, an image is processed by the big network and its output is used as the correct label while training the small network. The intuition behind this set-up is to remove the hard constraint that only one of the categories should be on with 100% confidence while training a network. For example, if an image of 7 looks like 2, forcing the network to give the output as 7 with

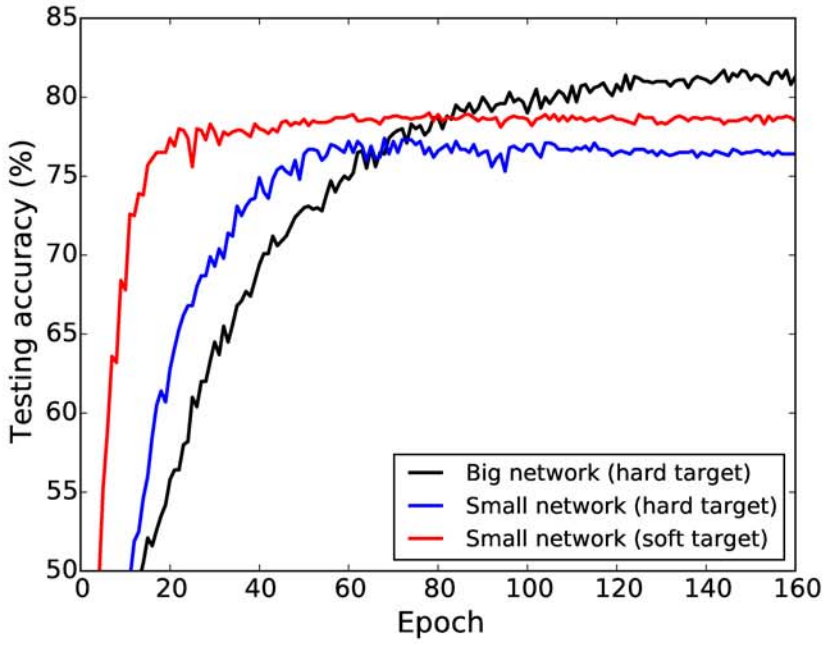


Fig. 5.1. Testing accuracy on Cifar-10 dataset.

100% confidence is a very hard constraint. It would be better if the network thinks that this image also looks like a 2, and gives some small possibility for it to be classified as 2. On the other hand, we only have the information of the correct labels in our hand. That is the reason, Hinton et al. [108] propose to train a big model first since big networks have a capacity to learn under hard constraints. As can be seen from Figure 5.1, the small network that learns from soft targets converge to a better accuracy than the one that learns from hard targets. Such a reduction in the network gives $2\times$ speed-up with a loss in the original accuracy of the big network.

5.2.2 Acceleration through parallelizations

Progress in computer technology has played a crucial role in the recent success of machine learning applications. In the late 90s, the first successful applications of continuous speech recognition and handwriting recognition were deployed as the scientists began train-

ing bigger models due to faster CPUs. While these networks have around 60.000 trainable parameters in the late 1990s [52], today the networks with millions of parameters are utilized to produce state-of-the-art results [109]. The rapid increase in the network sizes has occurred in the last five years mainly because of the efficient implementations of ConNets on parallel processors such as GPUs. The speed of optimized implementations allows researchers to explore significantly higher capacity networks. Whereas the training of these networks takes about a week on multiple GPUs [109], the feedforward real-time image processing becomes a need for applications. This section discusses several efficient implementations of ConvNets on CPUs and GPUs.

The implementations of ConvNets are mostly dedicated to efficient convolution operations due to the fact that convolution operations with images spend over ninety percent of their time being processed by the filters. Convolution operation takes two inputs: an image ($D \in \mathbf{R}^{N \times C \times H \times W}$) and a filter ($F \in \mathbf{R}^{K \times C \times R \times S}$). The image ranges over N images in a mini-batch, C input feature maps (e.g. $C = 3$ in RGB images), H height of the image, W width of the image. The filters range over K number of filters, C input feature maps, R height of each filter, S width of each filter. The output is also a four-dimensional tensor $O \in \mathbf{R}^{N \times K \times P \times Q}$, where N and K defined as previously, and $P = H - R + 1$, $Q = W - S + 1$. The operation is as follow:

$$O(n, k, p, q) = \sum_{c=1}^{C} \sum_{r'=1}^{R} \sum_{s'=1}^{S} D(n, c, r - r', s - s') F(k, c, r', s') \tag{5.1}$$

As can be seen from Equation 5.1, convolution operation includes seven nested for loops. The calculations across the dimensions N and K are independent loops. The calculations across C have independent loops that need to be summed in the end.

Parallelism can be explored at many levels, including weight parallelism (parallel sum of the products of computation in convolution) and node parallelism (computation over multiple convolutional planes) [128]. Because the training phase involves the processing of many input images before a weight update, parallelization across images is also explored in several popular libraries. Note that in real-time applications this acceleration is not possible

since the input images arrive in regular time intervals and an output is expected immediately after each image is sent.

Parallelization on CPUs

Regarding CPUs, there have been highly optimized implementations of ConvNets [129, 130] that explore cross image parallelization with multi-threading and implementing convolutions as highly optimized matrix-matrix multiplications (using BLAS).

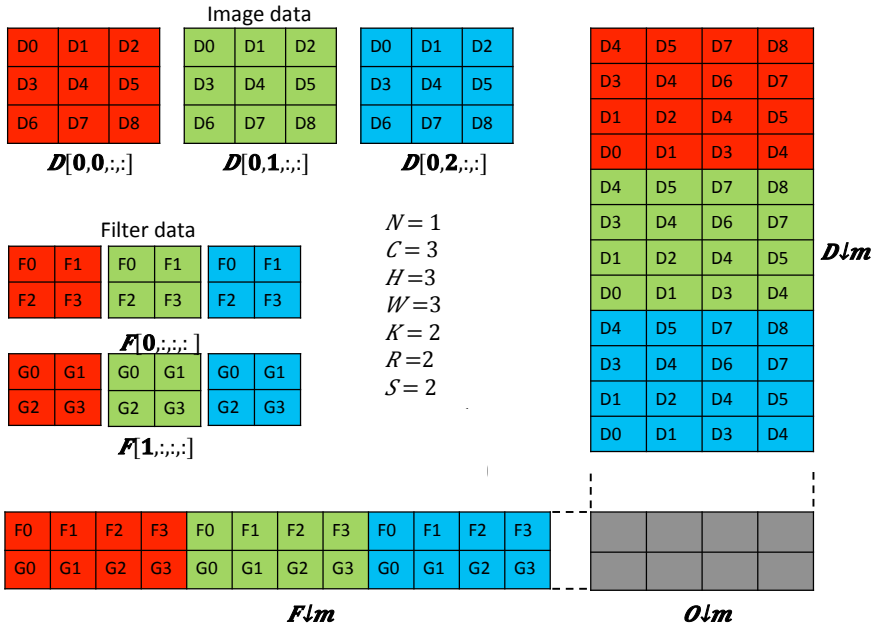


Fig. 5.2. Figure taken from [82]. Illustration of how a simple convolution can be lowered to a matrix multiplication. N: Number of images in mini-batch, C: Number of input feature maps, H: Height of input image, W: Width of input image, K: Number of output feature maps, R: Height of filter kernel, S: Width of filter kernel. The colors in this illustration represent the input feature maps, and elements of D and F are uniquely labeled in the illustration so as to show how each participates in forming D_m and F_m. The filter matrix Fm has dimensions K $\times$ CRS $= 2 \times 12$, while the data matrix Dm has dimensions CRS $\times$ NPQ $= 12 \times 4$. Note that each element of D is duplicated up to RS = 4 times in D_m. The output matrix O_m has dimensions K $\times$ NPQ $= 2 \times 4$.

A very successful method is to lower the convolution operation to a matrix multiplication. The conversion of convolution operation to a matrix multiplication is illustrated in Figure 5.2. Even though this conversion requires multiple memory copies of the same elements, it provides speed-up by taking advantage of highly optimized matrix-matrix multiplications libraries (BLAS [131]). Matrix multiplication is fast because it has a high ratio of floating-point operations per byte of data transferred [82]. The ratio increases as the matrixes get larger, meaning that the operation is less efficient if the convolution operation is performed on a small input with a small filter. However, in ConvNets, inputs and filters because of their three-dimensional nature produce big matrixes which make the lowering convolution into a matrix multiplication approach appealing.

Other approaches are proposed, such as the unrolling of convolution operations (multiple nested for loops) for cache-friendly memory access [132], and the linear quantization of network weights and inputs [133]. Despite the benefits of such implementations, ConvNets on general purpose processors are still too demanding, power-consuming, and slow to be used for real-time applications in an embedded system environment.

Parallelization on GPUs

One of the main reason for the success of ConvNets is the fast implementations in GPUs which have enabled researchers to increase the scale of ConvNets with more layers and parameters that were not previously practical to train. Currently, all the models which provide state-of-the-art performance on big datasets are trained on GPUs [36, 37, 48, 51].

While GPUs are gaining popularity in the training phase, they are not commonly present in embedded systems. The challenges in running ConvNets on GPUs are mostly centered on memory and communication. Different communication methods for GPUs have been explored for the purpose of overcoming the bandwidth limitation between the host system and GPU memory [82, 134–136]. The small local memory spaces in the GPU limit the ability to properly store the images, as do the filter coefficients of large networks [137]. Other optimizations take advantage of processing multiple inputs for training in a mini

batch mode [109]. However, this is not practical for real-time applications where there is a continuous feed of input, and outputs are expected immediately.

The most successful approach to speed up convolution operation on GPUs, similar to on CPUs, is to lower it to a matrix multiplication operation (Figure 5.2). In popular frameworks [138, 139] for training ConvNets, convolution operation has been implemented as a matrix multiplication operation to be run on GPUs. The disadvantage of this approach on GPUs is that forming input matrix, D_m, involves duplicating the input data up to RS times. Sharan et. al. [82] propose a GPU implementation of convolution where the memory copy of the duplicated inputs are hidden, allowing the matrix multiplication computation to be limited only by the time it takes to perform the arithmetic. This is achieved by computing a sub-matrix of results while fetching the next tiles of input and filter from off-chip memory into on-chip caches and other memories. These implementations all benefit from the large batch size. Batch size (N), the number of the images for training one batch, is usually set to $128 - 256$ to utilize all the resources in GPUs. On the other hand, in applications, there is a continuous feed of input, and outputs are expected immediately (in such cases $N = 1$). These implementations lose efficiency in such scenario.

Parallelization on FPGAs

Despite the fact that GPUs are becoming a common alternative to custom hardware, custom hardware can provide better performance with significantly less power consumption, which is necessary for mobile, embedded platforms. By developing a custom architecture and control method fully adapted to ConvNets, the product of power consumption by performance can be improved enormously. Because of this, there has been extensive interest in utilizing custom hardware to accelerate convolutional neural networks [140–145]. Custom hardware can uncover massive parallelism compared to CPUs. It also consumes low power, unlike GPUs. This advantage comes from a custom implementation where a large number of logic units, specialized to operations required for ConvNets, can operate in parallel.

Node parallelism provides a significant performance benefit for ConvNets [128, 140, 143] based on the fact that each convolutional plane is independent of others in the same layer. The degree of parallelism can be increased by placing as many processing units as silicon can hold. However, the actual performance is often limited by its memory bandwidth during ConvNet computation. Since the number of connections to produce a single output plane is much higher than the number of processing units, node parallelism necessarily generates intermediate results that need to be stored in memory. Such intermediates require frequent memory access between a host processor and memory. This causes significant overhead time when used for large-scale neural networks.

Efficient memory and operation scheduling, as well as unit parallelism, are crucial factors in ConvNet acceleration. Recently, an FPGA-based accelerator for ConvNets was proposed, containing an analytical design scheme using roofline model [146]. This method uses floating point operations in comparison to our study, which uses fixed point operations. Overall, our method achieves better peak performance in all of the experiments we tested. Furthermore, our method is capable of running the whole network except the linear layer, as opposed to only convolutional layers.

In this study, we propose a routing scheme that combines data reuse and concatenation to reduce the memory access limitation of custom hardware. This routing scheme enables maximum node level parallelization of ConvNets. Whereas the hardware fully explores the weight parallelism, the control method enables the full node parallelism with available resources in the hardware accelerator. Our control method acts as a compiler by transforming a high-level representation of ConvNets into operation codes to execute applications in a hardware accelerator. We discuss different optimizations and improvements in our control method while running ConvNets. This study is in line with previous work [7, 147, 148] but includes many improvements and an in-depth explanation of our methods. The initial study [147] was based on a small FPGA; therefore, the limitation was on computational resources rather than the memory access bandwidth. In other words, there was no need for an optimized routing scheme. The study presented herein [148] focuses on the convolution engine rather than the control method that runs the applications. While the recent study [7]

investigates memory optimizations, this method is not generic and does not utilize the resources after the first iteration because of the intermediate values occupying the bandwidth. In our work, we present a full system that combines data reuse and concatenation, and we present object classification and detection applications that run on our system.

5.3 Embedded Streaming ConvNets Hardware Accelerator

In this section, we present the architecture of the hardware accelerator for running ConvNets in the feed-forward prediction phase. The capabilities and limitations of this hardware were the most important considerations when we implemented our control method. The hardware system is implemented on the Xilinx Kintex-7 XC7K325T FPGA.

5.3.1 Architecture

The architecture of the hardware accelerator is based on [147] where the hardware is divided into two main areas: the memory router and collections that include the operators for processing images. Hardware operators are bundled together into a unit called a *collection*. All the operators and routers can be configured at run time and each module operates independently. In other words, configuring one module does not require stalling any others. All operators use the Q8.8 number format - which has been tested to provide virtually identical results to 32-bit floating number format - for the feed-forward phase computation, even though the ConvNet has been trained in 32-bit floating number format. These two main parts, memory router and collections, are explained in the subsections below.

Memory Router

The memory router interfaces with eight high-throughput I/O ports and four collections. It acts as a gateway to the accelerator and directly communicates with AXI bus. It is implemented as a crossbar switch where all incoming and outgoing streams pass through this module. Having a larger crossbar switch could ease the data traffic in the memory

router. However, it would also cause inefficient use of logic area due to the dense grid pipeline. In order to meet these area constraints, only essential streams are routed to the memory router while intermediate streams are delivered to the neighboring collections. The memory router can route incoming data streams to one or more outputs based on the configured routing topology. Moreover, one part of the router can be dynamically reconfigured without halting the functionality of the rest.

Collections

Hardware operators are bundled together into a single module called a *collection*. Each collection contains a convolution, a max-pooling, a non-linearity module, and a stream-adder. The presented system can hold four collections. Each convolution operation in a collection can be configured to do 1 - 12×12 convolution or smaller, 4 - 6×6 convolution or smaller, 16 - 3×3 convolution or smaller, and so on.

Inputs and outputs appear as data streams for all modules. The output from the convolution operator can be streamed into the max-pooling or non-linearity operator within the same collection. Producing output maps for 3D convolutions requires the accumulation of element-wise sums over multiple 2D convolved maps. To achieve this, the output of the convolution from one collection can be streamed into the second collection where it can be summed together with the output of the convolution from the second collection. This process can be repeated among all collections.

5.3.2 High-throughput Data Ports

The FPGA has eight high-performance ports to DDR3 memory. These high-performance ports tap into DDR3 memory using the AXI buses. Each AXI DMA is bidirectional and can transfer 128-bit data words per clock cycle per direction.

ConvNets process hundreds of images with different filters and produce hundreds of intermediate results. Therefore, the bandwidth of the system can be a significant limitation in data transfer. Since the presented system has four collections, each capable of process-

ing images with multiple filters, enough bandwidth could stream different images to each convolution operator in each collection and sum all of them together in one transaction. However, having only eight AXI ports to stream data in and out of the hardware accelerator proves to be a prominent handicap. It is not possible to send unique streams of data from memory to each convolution engine. Our method presents a novel routing scheme to achieve the full utilization, as described in subsection 5.4.2.

5.3.3 Computational Resources

The operations used by ConvNets are implemented in the custom hardware as follows:

Convolver

The convolution operation is the basis of ConvNets. Convolution with trained filters is used to extract useful features from the input images or from the output of the previous layers, in which case convolution extracts more complex features.

Weights in ConvNets can be described as 4-dimensional filters: $W \in \mathbf{R}^{C\times X\times Y\times F}$, where C is the number of input channels, X and Y are the spatial dimensions of the filter, and F is the number of filters or the number of output channels. The output maps of the convolution operation are called feature maps. The number of feature maps is equal to the number of filters. For each feature map, inputs are convolved with a filter $W \in \mathbf{R}^{C\times X\times Y}$ and are described as:

$$\begin{aligned} F_f(x,y) &= I * W_f \\ &= \sum_{c=1}^{C}\sum_{x'=1}^{X}\sum_{y'=1}^{Y} I(c, x-x', y-y')W_f(c,x',y') \end{aligned} \tag{5.2}$$

assuming a stride of one where f is an index of the feature maps, $I \in \mathbf{R}^{C\times N\times M\times F}$ is the input map, and N and M are the spatial dimensions of the input.

$$F_f(c,x,y) = \sum_{x'=1}^{X} \sum_{y'=1}^{Y} I(c, x-x', y-y') W_f(c, x', y') \tag{5.3}$$

The kernel is pre-loaded from memory and cached for the duration of the convolution. As an image is streamed in, one input pixel results in one output pixel. This does not include an initial set up delay that occurs due to the pipelined nature of the hardware.

Max-pooler

In the max-pooling operator, the images are divided into a set of squares $p \times p$, and for each square, the maximum value is outputted. This operation in practice is a non-linear down-sampling by a factor of p. It gives strength to ConvNets by providing a form of translation invariance. Furthermore, it reduces computations for the upper layers.

Non-linearity operator

In ConvNets, a non-linearity operation usually follows either convolution or max-pooling operations. Non-linearity operations increase the capabilities of ConvNets to separate high-dimensional inputs. Without a non-linearity, a composition of linear functions is itself a linear function, which has limited power to encode information.

The Rectified Linear Unit (ReLU), $f(x) = \max(0; x)$, is one of the most widely used non-linearity operations [15]. The ReLU operation in the custom hardware produces one output per clock cycle.

Stream Adder

This module computes an element-wise addition of two incoming streams and produces a single stream as output. It can process element-wise arithmetics, and its main role in the context of ConvNets is to complete the 3D convolutions by summing up multiple 2D convolution maps from the convolution engines. This operator takes one stream from the neighbor collection and performs operations with the other stream that comes to the collec-

tion that contains it. The arithmetic occupies a single DSP slice but effectively processes the accumulation of all elements in the plane at one clock cycle when used along with other pipelined processing units.

5.4 Control Method

This hardware accelerator requires special software to interpret high-level ConvNet abstraction into a sequence of control instructions for configuring the connections in the accelerator. This software, the control method, acts as a compiler and transforms high-level representations of ConvNets into operation codes to execute applications within a hardware accelerator. The method takes sequential descriptions of networks from the Torch environment [139] and parses them to exploit different levels of parallelisms [128] and optimizations.

There are three main features of the control method that significantly improve performance:

- *Across modules optimizations:* Max-pooling and non-linearity operations generally follow a convolution operation. These three operations can be cascaded. As the input streams in, the output of the layer can be produced in a pipelined manner. This concatenation part takes place in the *network parser* and will be described in section 5.4.1.

- *Across images, within a module:* Operations that are independent of each other can be parallelized. However, each independent operation requires its own input and output streams, and the bandwidth of systems becomes a major bottleneck that prevents efficient energy flow. We propose a novel routing which combines data reuse and data concatenation for maximum parallelization. This part is described in section 5.4.2.

- *Smart configuration:* For videos or a collection of images, the same network is run multiple times on different images. The sequence of codes for configuration is cre-

ated only in the beginning and is serialized for memory-optimized access. Caching these sequences of codes in hardware further boosts the performance. This process takes place in the *configurator* and will be described in section 5.4.3.

The hierarchy of our control method, from top down, includes: network parser, resource allocator, and finally hardware configurator. Each part of the control method is explained below.

5.4.1 Network Parser

The network parser explores across-module optimizations while creating a list of operations needed to execute a given ConvNet. The network parser scans the sequential description of ConvNets (Torch network definition [139]) and combines the operations that can be calculated in one collection in the custom hardware. Across-module optimizations are obtained by cascading operations such as convolution, max-pooling, and non-linearity. These operations in ConvNets generally follow each other; hence, these three operations can be cascaded (i.e., the stream of the output of convolution is fed to the max-pooling operation and the output of that is fed to the non-linearity operation). In this process, input and output streams do not need to be sent to the memory router between each operation. In this step, the network parser does not use any low-level information, and is therefore independent of available resources in the custom hardware. After the list is created, it is analyzed in the *resource allocator* to run as many of the operations in parallel as possible based on the resources.

5.4.2 Resource Allocator

The resource allocator manages the hardware resources of the coprocessor and assigns queued operations from the network parser into the collections (computing units). The routing topology is drawn by the allocator in order to maximize the utilization of computing power. The allocator combines data reuse and data concatenation to alleviate the bandwidth

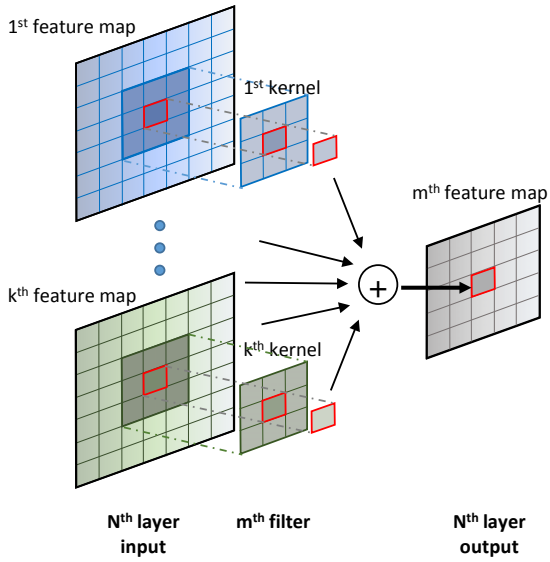


Fig. 5.3. Demonstration of calculating an output feature at each layer. Each input is convolved with a different kernel and summed together.

bottleneck. This step is essential to exploiting the parallelism of ConvNets, i.e., parallelism over multiple convolutional planes.

Overview of Convolution Operation in ConvNets

ConvNets contain hundreds of filters at each layer to extract features by convolving inputs. Convolution with each filter includes multiple 2D convolutions and an accumulation step:

$$F_f(x, y) = \sum_c F_f(c, x, y) \tag{5.4}$$

where c is the number of input feature maps and $F_f(c, x, y)$ is the output of the 2D convolution as given in Equation 5.3. This operation is depicted in Figure 5.3, where each line refers to a 2D convolution operation and the arrows going to the same output are summed together.

The breakdown of this operation in our system is as follows:

1. Each convolution engine from each collection processes a 2D input feature map by convolving it with a 2D filter.

2. The results of the convolution of the first collection are streamed into the neighbor collection's stream adder for summation.

3. The stream adder performs an element-wise summation of the upcoming stream from the neighbor collection and the stream coming from the convolution operator in its own collection.

4. The results of the stream adder go to the neighbor collection's stream adder for further accumulation of 2D convolutions.

There are generally not enough resources to calculate all the 2D convolutions needed to produce one output map. Hence, the outputs of each DMA transaction (partial summation over c in Equation 5.4) are saved as intermediate values. In the next transaction, the intermediate values are streamed into the custom hardware for further accumulation with the outputs from the other 2D convolution operations.

Node Parallelism

The node parallelism, in this context, refers to a parallelization across images within a module. In other words, each 2D convolution across images is an independent operation and can be parallelized. However, each instance requires its own input and output streams. Ideally, the parallelism is equal to the number of collections that can fit into an FPGA. Nevertheless, due to the limited bandwidth of the custom hardware, streaming input and output for each operation requires an optimized routing scheme for the available resources. We combine data reuse and data concatenation to achieve maximum node parallelism.

To justify the need for an optimized routing scheme, we first calculate the utilization of the naive approach for a system with four ports (I/O bus) and eight collections as shown in Figure 5.4(a). Let's say we use one convolution engine from each collection. Though calculating one output map generally requires hundreds of 2D convolutions, the system can

calculate the intermediate values for two output maps together. While a related routing scheme was proposed in [7], this method only provided full utilization where streaming of the intermediate values was not required. When intermediates are created and need further processing, two ports would need to be used to stream them in and out, giving only 50% utilization in general.

This work achieves optimum utilization by taking advantage of both the data concatenation and this routing scheme at the same time.

Concatenation of Data

The 32-bit AXI bus carries data words represented in Q8.8 format (8 bits for integers and 8 bits for fractions) via DMA transfer. Since each word uses only 16 bits, the 16 most significant bits remain vacant, lowering effective data bandwidth by half (Figure 5.4(a)). Therefore, two consecutive data words can be concatenated in a single bus during the transfer, which can increase the effective bandwidth. However, if data are not produced together, a waiting time in the processing pipelines occurs due to the delay between transferring and processing.

If two consecutive data words are produced together, and resultantly consumed together, they can be concatenated, increasing the bandwidth without causing any delay. This routing scheme encourages concatenation because two intermediate values or two output values are produced concurrently. Those values will also be used together later on. With the approach from Figure 5.4(a), concatenation of outputs would not be possible since each layer requires hundreds of convolution operations. It would therefore not be possible to produce two outputs in one cycle. However, with the approach from Figure 5.4(b)), outputs and intermediates are created together, so they can be concatenated and brought together in the next cycle in order to calculate the next layers' outputs. Figure 5.4 is depicted for four ports for simplicity. In our case, we have eight ports and multiple convolution engines, e.g. 64 - 3×3 convolution engines. This routing scheme is flexible for any number of ports and

computational power, and provides benefits when the memory access is the limitation. It effectively increases the bandwidth eightfold.

Note that two data words are encapsulated into a 32-bit stream (each word is 16-bit). As soon as the stream reaches the memory router, two data words are outputted and fed to different convolution engines.

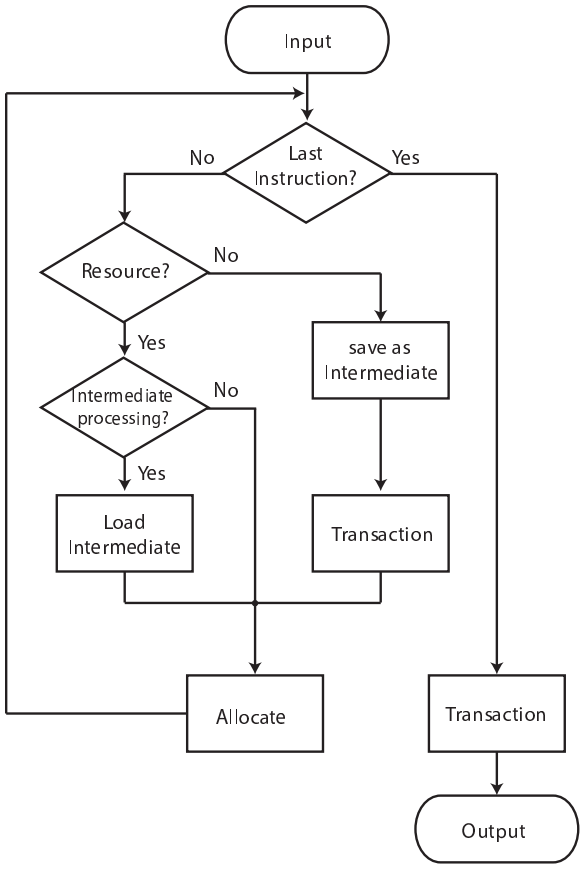


Fig. 5.5. The flowchart of the control method. Each instruction is scheduled as long as resources are available. Otherwise the instructions that are scheduled are processed. If there are not enough resources to complete the calculations for one output, the partial output from the calculations is saved as an intermediate value which comes in the later transaction for further processing.

Flow of Resource Allocator

The flowchart of the system is shown in Figure 5.5. The table containing rows of instructions from the network parser is scheduled row by row (as long as resources permit). As described in subsection 5.4.2, when resources are allocated, each half of the collections is allocated for each set of filters. As long as ports and collections are available, resources continue to be allocated, and once one kind of resource is completely allocated, the transaction is performed. If the calculations have not been completed for an output feature map, the outputs of the transaction are saved as intermediate values. In the next cycle, the system checks if there are intermediate values that need to be processed further. If so, it allocates resources for them as well. This routing scheme is very flexible and can process ConvNets with any number of filters and layers.

5.4.3 Hardware Configurator

The hardware configurator is the lowest level of our control method to execute a ConvNet application. It provides an environment for communication between the resource allocator and the custom hardware through AXI memory-mapped and high-throughput drivers. For a given routing topology (from the resource allocator), this module produces corresponding operation codes as output. It uses a static routing method so that it only has to be determined once (since codes are generated one time at the start, and the hardware loops over the operation sequence during actual execution).

The hardware configurator produces and serializes operation codes for coalesced memory access. The sequence of codes resides in the contiguous memory, which facilitates DMA transfer and minimizes lead time without scatter-gather addressing. The length of operation codes for a large-scale network is often greater than 30 Mbytes for each frame. The high number of operations must then be directed to the accelerator, and this consumes a large portion of transfer time within the overall execution time. An additional performance boost can be created by caching the operation codes in the on-chip memory within the custom logic, but this comes at the cost of an area. In this case, the hardware accelerator

only requires a single code to trigger a certain routing topology, providing us a speed close to the theoretical maximum.

5.5 Experimental Results

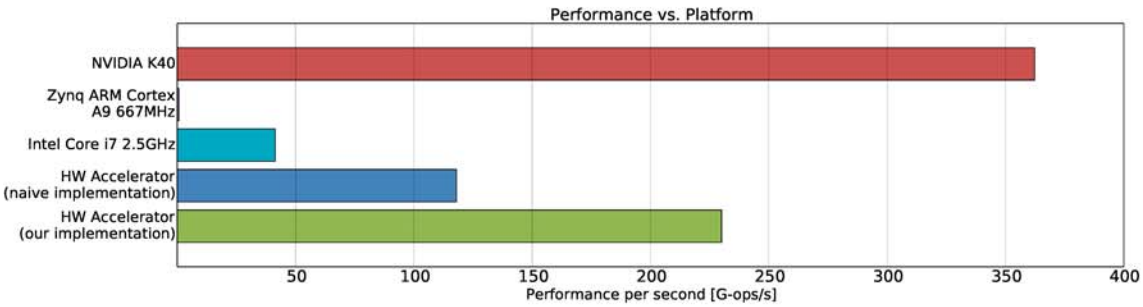


Fig. 5.6. Performance per second test over one layer network on different platforms. The input is an 500×500 RGB image. The network has 6×6 128 filters, 4×4 max-pooling and non-linearity operation. We also compare the hardware accelerator that is configured with the naive implementation described above and our implementation.

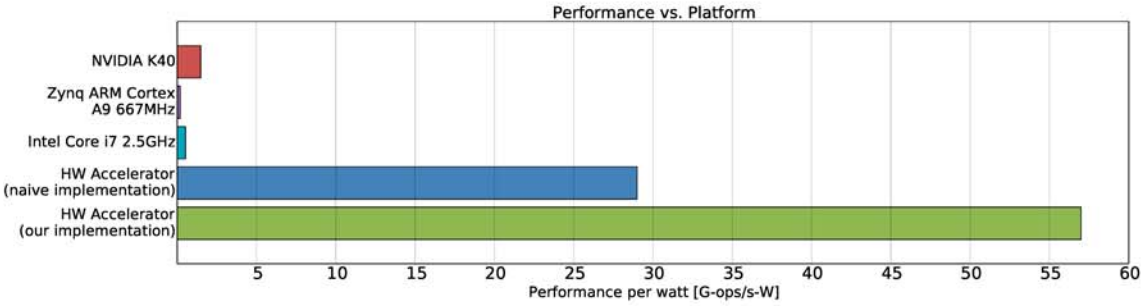


Fig. 5.7. Performance per watt test over one layer network on different platforms. The input is an 500×500 RGB image. The network has 6×6 128 filters, 4×4 max-pooling and a ReLU non-linearity operation. We also compare the hardware accelerator that is configured with the naive implementation described above and our implementation.

We compare the performance of our custom hardware with other platforms. We present our custom hardware with two different configurations: our novel routing scheme (input reuse and concatenation) and the naive routing scheme. Other platforms include Intel Core-i7 2.5GHz CPU, NVIDIA K40, and ARM Cortex A9 processor. In our first experiment, we run a single layer network. This experiment uses an input image of $3 \times 500 \times 500$, where 3 is the number of channels (e.g. RGB images). The single layer network consists of 3×128 convolution operations with 6×6 filters, a max-pooling operation with 4×4 window size and a non-linearity operation. This experiment is designed to showcase the peak performances of platforms with large image sizes and filters.

As can be seen in Figure 5.6, the performance per second of NVIDIA K40 is five times greater than the system presented here. However, desktop GPUs have a TDP consumption around $225 - 300$ watts (245 for the NVIDIA K40), whereas our platform consumes a maximum of 4 watts. Figure 5.7 presents the comparison performance per watt. In this case, our implementation is 10 times better than the GPU alternative in performance per watt. In the comparison of the novel vs. the naive routing scheme, our system reports 164 G-ops/s, while the naive approach achieves just 80 G-ops/s. The hardware accelerator with the optimized control method is 96 times faster than the baseline embedded processor, a dual-core ARM Cortex A9. The peak performance of the presented system is 247 G-ops/s when the filters are larger (12×12). In the following chapter, more experimentations with a multi-layer network are utilized for the presented applications.

5.6 Summary

In this section, we presented a novel control method for a hardware-accelerated real-time implementation of deep convolutional neural networks (ConvNet) on an embedded platform. This control method achieves full utilization of the resources of the hardware by exploiting the characteristics, architecture, and configuration of ConvNets. Our method combines data concatenation and data reuse for the maximum speed-up. Our results show that our system is extremely performance efficient.

The proposed routing scheme is implemented using custom hardware with eight ports and four collections; each has multiple convolution operations, but this routing scheme can be uncovered in any other configuration. While specialization emerges for energy efficiency, flexibility of the system is also important. The system presented here can run different architectures of ConvNets, with any numbers of filters and layers, using the implemented control method. Furthermore, this system can be used for generic image processing applications which use convolution-like data flow. For example, convolution operations can be replaced with sum-of-absolute-differences (SAD) or sum-of-square-differences (SSD) operations, widely used in tracking and motion estimation algorithms. Additional examples demonstrating the usability of this system include SIFT, median-filtering, and video processing, among others.

In this chapter, we also discussed different ways to accelerate ConvNets. Some of these ways include removing correlated parameters and implementing highly efficient convolution operations. The amount of acceleration can be further increased by taking advantage of both ConvNet reduction techniques and highly optimized ConvNet implementations, although most of the time reduction in the network size causes a loss in the network accuracy.

6. APPLICATIONS OF THE CONVNETS ACCELERATOR

In this chapter, we test our hardware system presented in the last chapter with applications on object classification and object detection in real-world scenarios. The main components of these applications are ConvNets with multi-layer architecture. As we mentioned in the last chapter, our hardware system can be used for generic image processing applications which use convolution-like data flow. For example, convolution operations can be replaced with sum-of-absolute-differences (SAD) or sum-of-square-differences (SSD) operations, widely used in tracking and motion estimation algorithms. In this chapter, we replace the convolution operation with similarity-matching-ratio (SMR) and showcase the flexibility of our proposed system. SMR is an algorithm we propose which achieves state-of-the-art results on challenging video sequences. We introduce the SMR algorithm, run it on our hardware system, and achieve real-time tracking in videos.

6.1 Object Classification and Detection with ConvNets

In this section, we present two different applications that we run on our system: object classification and object detection. First, the details of the architecture and training are presented, and then the applications are explained, providing screen shots for each application. Videos of the applications are available online[1].

6.1.1 Network Architecture and Training

The network set-up has five layers of convolution, max-pooling, ReLU modules, and three fully connected linear layers. The parameters of the network architecture are given in Table 6.1. In the first layer, inputs are padded with zeroes to obtain the information in

[1] `http://web.ics.purdue.edu/~adundar/`

Table 6.1.
The specifications of the network used for the applications. Input size of each layer is calculated based on the zero-padding size, parameters of convolution and pooling operations.

Layer	1	2	3	4	5	6	7	8
Stage	conv+max	conv+max	conv+max	conv+max	conv	full	full	full
#channels	48	64	64	64	32	128	128	20
Filter size	11×11	5×5	3×3	3×3	3×3	-	-	-
Pooling size	4×4	2×2	2×2	2×2	-	-	-	-
Pooling stride	4×4	2×2	2×2	2×2	-	-	-	-
Zero-Padding size	5×5	-	-	1×1	1×1	-	-	-
Spatial input size	231×231	58×58	27×27	12×12	6×6	6×6	1×1	1×1

the borders of the images. Padding is applied to the fourth and fifth layers as well. In the first layer, max-pooling is applied by a stride of four. The input size of the network is $3 \times 231 \times 231$ and the spatial size of the input of each layer decreases as we move to the end of the network, as given in Table 6.1. The output of the network is fed to a soft-max operation which normalizes the summation of values to one so they represent a probability distribution. This network is trained with twenty categories, which is the size of the network output. The network is trained in the Torch environment [139] on NVIDIA K40.

DropOut [103] with a rate of 0.5 is employed on the fully connected layers (sixth and seventh) in the classifier. Dropout sets the output of each hidden layer to zero with a probability of 0.5. In this process, only half of the network is trained in each forward-backward pass. As a result, complex co-adaptations of neurons decrease, as does over-fitting. In the test period, outputs of neurons are multiplied by 0.5, so the whole network contributes to the prediction.

We follow the input preparation method as [15]. It starts by cropping images that are 25×25 larger than the network's input size, 263×263. Each image is down-sampled so that the smallest side of the image is 263. Images are cropped from the center to obtain 263×263 size images. To avoid over-fitting and increase the training dataset, 231×231 random patches are extracted from the images at each training step; some patches are randomly flipped horizontally. Finally, as pre-processing, patches are normalized by subtracting the mean and dividing by the standard deviation of the dataset. The same network is used for both applications.

6.1.2 Object Classification

The first task is using the network to classify objects (i.e., assign trained classes to an image). Figure 6.1 shows a screenshot from a video we recorded. Because an image can have multiple objects, we display the top 2 probabilities. Images from the camera were down-sampled to the size of 231×231 and fed to the network. The output of the network

is the probability distribution of each class in the image. As demonstrated in the figure, the system is able to recognize two presented objects correctly, despite the fact that they are not centered.

Note that when just one object is present, the second highest probability can be low or irrelevant, interpreted as noise.

In Table 6.2, a comparison of the performance per watt of our system for this application is presented. There is a drop in the performance in comparison to Figure 5.7 because of the small filter sizes that decrease the weight level parallelization. However, our system still outperforms all other presented platforms in performance per watt. In Table 6.3, the performance numbers for each layer in the classification application are given. The number of operations needed for each layer and the time it takes to process each layer in our system are provided in this table. Differing numbers of filters, filter sizes, or input sizes to each layer make a big difference in the number of computations and the overall performance. Note that only the convolutional layers are processed in each platform because a linear layer is not implemented in our system.

6.1.3 Object Detection

Object detection is a more complex task than simple classification since a bounding box for the predicted object must be returned. That is, the network should predict both the correct category and the location of the object. Also there might be multiple objects in a scene, and these objects might not belong to any trained category.

To return the bounding box, we follow the *sliding-window* approach without any pre-processing (in a similar manner as [4, 149, 150]). In ConvNets, this approach is efficient since they share computations that are common to the overlapping regions. The output of the network is a three-dimensional feature map, where one dimension has the information of the probability distributions and the other two dimensions have the information of the spatial coordinates.

Table 6.2.
Performance per second and performance per second-watt numbers over classification and detection applications on different platforms.

	Object Classification		Object Detection	
Performance	Per Second G-ops/s	Per Second - Watt G-ops/s-W	Per Second G-ops/s	Per Second - Watt G-ops/s-W
NVIDIA K40	573.3	2.34	828.7	3.38
Zynq ARM Cortex A9 667MHz	1.7	0.43	1.7	0.43
Intel Core i7 2.5GHz	88.6	1.1	78	1.0
HW Accelerator (Naive implementation)	58.2	14.6	81.2	20.3
HW Accelerator (Our implementation)	115.2	28.3	179	43.5

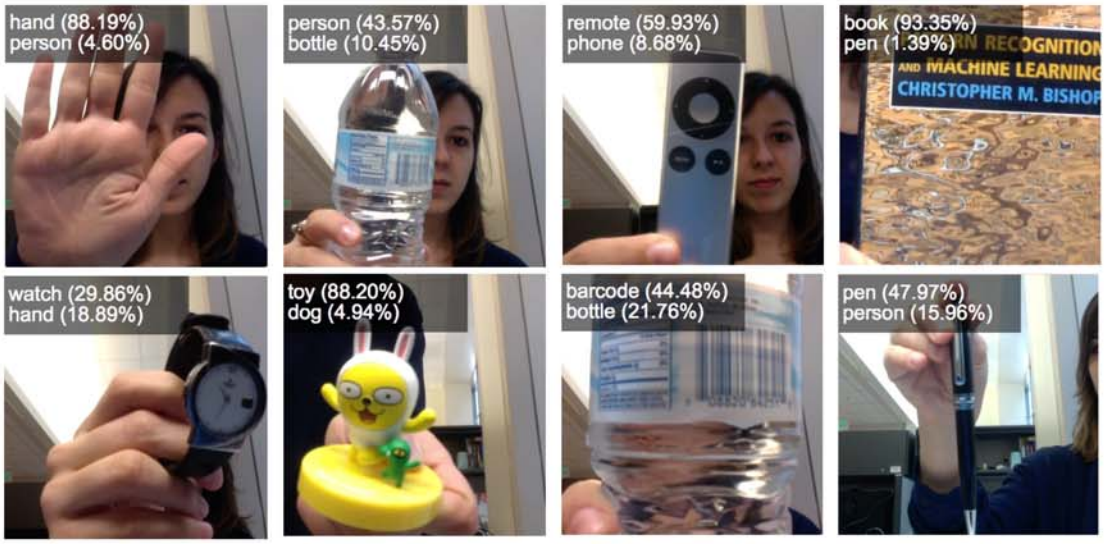


Fig. 6.1. Object classification: ConvNet takes each frame and produces a probability distribution among the classes the network is trained with. The top 2 classes with the highest probabilities are displayed on the left corner of images with the probabilities. The network is able to recognize two objects at the same time. When there is only one object in the image and the second probability is low, it can be considered noise.

To remove closely detected duplicates among the detections that are spatially very close, the one with the highest probability is chosen while the rest are discarded. Because the network is not trained with all the visible categories in each scene, a fixed low threshold of probability is set for a prediction to be valid. This is necessary because the network consistently attempts to pick a category for each patch despite the possibility that there is not a recognizable object.

In Figure 6.2, the detection application's screenshots of a movie trailer are presented. The video contains 360×720 RGB images. Frames of the video are fed to the network after subtracting the mean and dividing by the standard deviation of the training dataset. This detection process could be further improved by implementing more categories (so that the system is able to detect more objects), using the multi-scale approach, or combining the prediction of ConvNets with pre or post-processing.

Table 6.3.
Performance per second and performance per second-watt numbers for each layer for classification application with our implementation.

HW Accelerator (Our implementation)			
Network given in Table 1		Performance	
	G-ops	ms	G-ops/s
layer 1	1.86470	15.28	122.0
layer 2	0.44827	4.27	105.0
layer 3	0.04616	0.67	69.0
layer 4	0.00739	0.23	32.1
layer 5	0.00059	0.10	5.9
Overall	2.36711	20.55	115.2

In Table 6.2, comparisons of the performance per second and per watt of our system are presented for this application. The performance numbers for our system improve compared to the classification application because of the larger image sizes and the streaming architecture of our system.

6.2 Tracking with SMR

In this section, we replace the convolution operation with a template matching algorithm for visual tracking and showcase the flexibility of our proposed system. We first present a novel approach to visual tracking: Similarity Matching Ratio (SMR) which is also our work. The traditional approach of tracking is minimizing some measures of the difference between the template and a patch from the frame. This approach is vulnerable to outliers and drastic appearance changes and an extensive study is focusing on making the approach more tolerant to them. However, this often results in longer, corrective algorithms which do not solve the original problem. Our work proposes a novel approach to the defini-

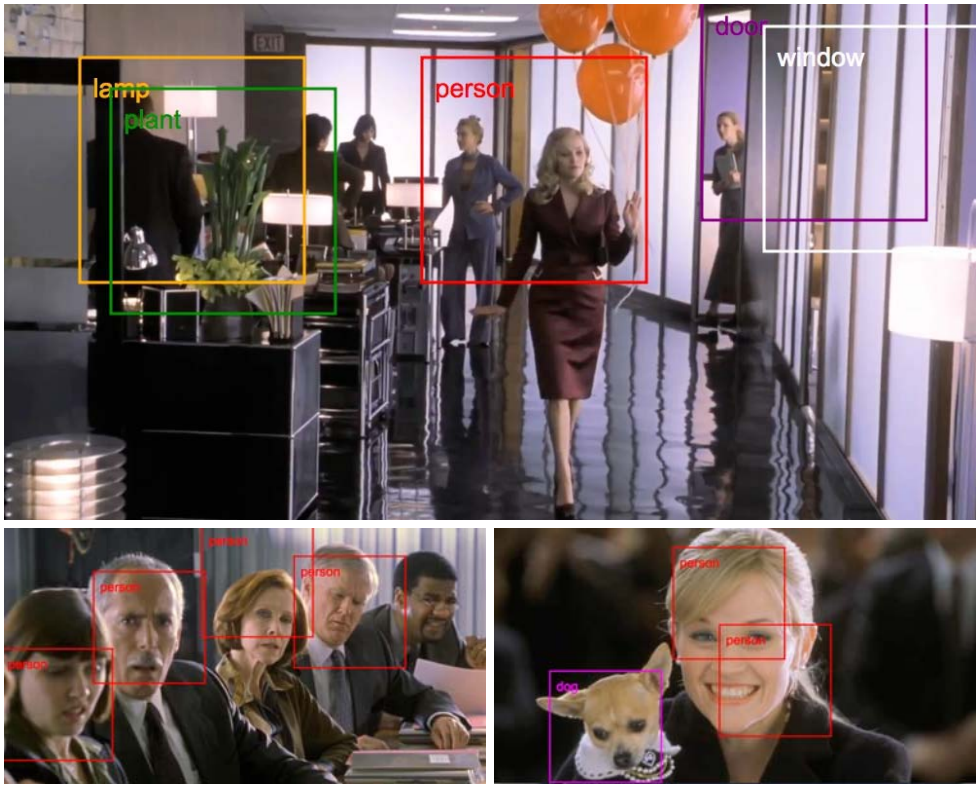


Fig. 6.2. Object detection: ConvNet scans the frame in a sliding window fashion. A simple post-processing is applied to show ConvNets capabilities. Detections above a threshold are displayed after pruning the detections in close regions. Detections can be improved with more sophisticated post-processing or by using multiple scales.

tion of the tracking problems, SMR, which turns the differences into probability measures. Only pixel differences below a threshold count towards deciding the match, the rest are ignored. This approach makes the SMR tracker robust to outliers and points that dramatically change appearance. The SMR tracker is tested on challenging video sequences and achieves state-of-the-art performance.

6.2.1 Introduction

Visual tracking of objects in a scene is a very important component of a unified robotic vision system. Robots need to track objects in order to interact. As such as they move closer, robots and other autonomous vehicles will have to avoid other moving objects, humans, animals, as they operate in our everyday environment.

The human visual system object tracking performance is currently unsurpassed by engineered systems, thus our research tries to take inspiration and reverse-engineer the known principles of cortical processing during visual tracking. Visual tracking is a complex task, with neuroscience studies of cortical processing painting an incomplete picture, and thus is only partially able to guide the design of a synthetic solution. Nevertheless a few key features arise from studying the human visual system and its tracking abilities: (1) the human visual system is not limited to three-dimensional conventional objects in space, rather is able to track a set of visual features [151]. Thus *object* in this section refers to a distinct group of features in the two-dimensional space. (2) It is not necessary for humans to have knowledge of the object class before visual tracking, and (3) humans can track an object after a very brief presentation. Even though the human visual system does not operate with frames it is common to desire synthetic systems to be able to track from a single frame.

Visual tracking in artificial systems has been studied for decades, with laudable results [158]. In this study, we focus on bio-inspired visual tracking systems that can be part of a unified neurally-inspired vision system. Ideally, a unified visual model would be able to parse and detect an object every frame, but right now there is no bio-inspired model that can do this in real-time [159–161]. Deep neural networks come close to this performance when trained to look for a single object on a large collection of images [162].

A bio-inspired synthetic visual tracker is generally thought of having two outputs of the same unified stream: one is a deep neural network classifier that is capable of categorizing object, another is a shallower classifier that can group features into objectness. The first deep system is used to be able to continue tracking an object as it disappears and reappears in the scene, while the second system provides rapid grouping of local features, by tracking

Table 6.4.
Properties of the video dataset used in this work [152].

	Video Sequence					
	1. David	2. Jumping	3. Ped1	4. Ped2	5. Ped3	6. Car
Camera Movement	yes	yes	yes	yes	yes	yes
Partial Occlusion	yes	no	no	yes	yes	yes
Full Occlusion	no	no	no	yes	yes	yes
Pose Change	yes	no	no	no	no	no
Illumination Change	yes	no	no	no	no	no
Scale change	yes	no	no	no	no	no
Similar Objects	no	no	no	yes	yes	yes

Table 6.5.
Number of correctly tracked frames from the state-of-art trackers and the SMR tracker. Table is taken and modified from [153].

	Video Sequence					
	1. David	2. Jumping	3. Ped1	4. Ped2	5. Ped3	6. Car
Number of Frames	761	313	140	338	184	945
[154]	17	75	11	33	50	163
[155]	n/a	313	6	8	5	n/a
[156]	94	44	22	118	53	10
[157]	135	313	101	37	49	45
[153]	761	170	140	97	52	510
SMR (this work)	761	313	140	236	66	510

local maxima in the retinal space. Such distinction might be necessary as a deep system will need 100-200 ms to process one visual scene [163], while tracking without predicting object movement, as the one required for the oculo-motor control of smooth-pursuit [164], requires faster processing of the visual stream.

Inspired by recent findings on shallow feature extractors of the visual cortex [165], we postulate that simple tracking processes are based on a shallow neural network that can quickly identify similarities between object features repeated in time. We propose an algorithm that can track and extract motion of an object based on the similarity between local features observed in subsequent frames. The local features are initially defined as a bounding box that defines the object to track.

Our work uses a modified template matching algorithm but offers an advantage over traditional template matching algorithms. Traditional template matching algorithms define the tracking problem as follows: We are given two images, $F(x, y)$ and $G(x, y)$, which represent the pixel values at each location (x, y). $G(x, y)$ is the template, representing the object that wanted to track, that may come from the user selection or an automatic detection algorithm, and $F(x, y)$ is the new image that comes from a camera. The goal is to find the new location of the object (h_1, h_2) by minimizing some measures of the difference between $F(x + h_1, y + h_2)$ and $G(x, y)$ in different configurations.

In our work we change this definition of tracking and propose a novel approach, Similarity Match Ratio (SMR). This approach is more robust to appearance change, disappearance and outliers because instead of trying to minimize some measures of difference between $F(x + h_1, y + h_2)$ and $G(x, y)$ as a whole, we want to find (h_1, h_2) that gives the best match ratio between $F(x + h_1, y + h_2)$ and $G(x, y)$. To do this, we are turning pixel differences between $F(x + h_1, y + h_2)$ and $G(x, y)$ into probability values and accumulating them for every pixel that has a good match. If there is no good match between some pixels, these pixels provide zero probabilities because we are not interested in how badly the two pixels match. The method is tested on challenging benchmark video sequences which include camera movement, partial/full occlusion, illuminance change, scale change and similar objects. State-of-the-art performance is achieved from these video sequences.

6.2.2 Previous Work

Most popular trackers that are based on the traditional definition of the tracking problem (e.g. Sum-of-Squared-Distances (SSD), Sum-of-Absolute-Differences (SAD), Lucas-Kanade tracker) try to find distance vector (h_1, h_2) that minimizes the difference between $F(x + h_1, y + h_2)$ and $G(x, y)$ either on the grayscale or color image. However, the template $G(x, y)$ may be including outliers or some parts that dramatically change or disappear, which cause tracking failure. The common approach to overcome these tracking failures is that trackers should not treat all pixels in a uniform manner but eliminate outliers from the computation.

Some studies [166, 167] propose using a weighted histogram as a measure to minimize for tracking an object. By assuming that pixels close to the center are the most reliable, these methods weigh them higher, since occlusions and interferences tend to occur close to boundaries. However, a dramatical change in the appearance can occur even in the center, which cannot be handled by this method.

There are studies that aim to detect outliers and suppress them from the computation. [168] uses the common approach that outliers produce large image differences that can be detected by the estimation process [169]. Residuals are calculated iteratively and if the variations of the residual are bigger than a user defined threshold they are considered outliers and suppressed. [170] uses the spatial coherence property of the outliers which means that outliers tend to form a spatially coherent group rather than being randomly distributed across the template. In that work the template is divided into blocks and constant weights are assigned for each block. If the image differences of the blocks between the frames are large, it means these blocks include a significant amount of outliers. The method excludes the blocks that contain outliers from the computation of minimization. These methods are more robust to outliers. However, they are computationally expensive.

[153] proposes forward backward error which is based on the fact that correct tracking should be independent of the direction of time-flow. Firstly, points are tracked in the forward direction. Then, backward tracking is applied to validate the trajectories. This

method enables trackers to avoid tracking points that disappear from the camera view or change appearance drastically. Before our work, Kalal's tracker was the state-of-the-art.

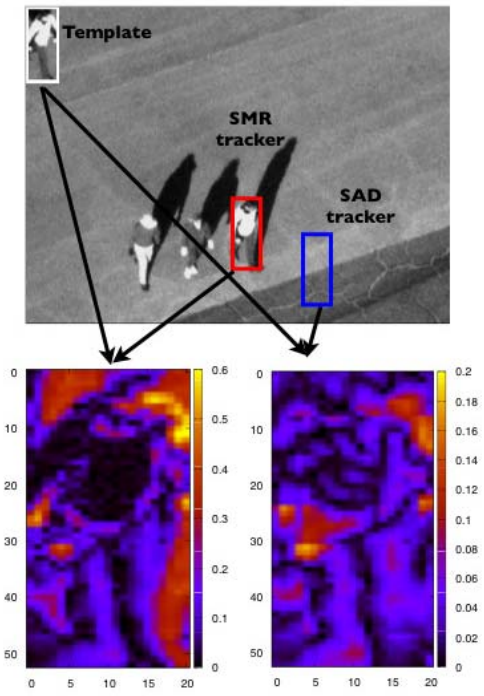
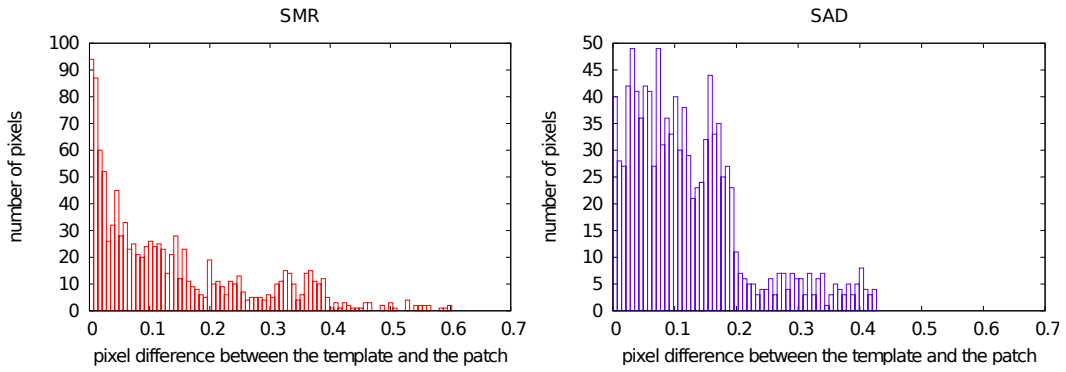


Fig. 6.3. (Top) The red box is the SMR tracker's output, the blue box is the SAD tracker's output. The ground-truth from the first frame is used as a template which is shown on the left top corner of the frame. (Bottom) The absolute differences for each pixel between the template and result from the SMR tracker are mapped on the left and from the SAD tracker on the right. Dark values (close to zero) report a better match. Note that even though there are higher differences, the SMR tracker is able to find the correct patch.

6.2.3 Similarity Matching Ratio (SMR) Tracker

The SMR tracker uses a modified template-matching algorithm. In this algorithm, we look for similarity between a template $G(x, y)$ and patches of a new video frame $F(x + h_1, y+h_2)$. The SMR computes the difference between the template and the patches at each pixel. Templates are moved convolutionally on the new video frame, and stepped by one



(a) (b)

Fig. 6.4. Histogram of the pixel differences that were mapped in Figure 6.3. (a) Map between the template and result from the SMR tracker and (b) result from the SAD tracker. The SAD tracker minimizes the number pixels with large differences, whereas the SMR tracker maximizes the number of pixels that have small differences.

pixel. If this difference is lower than a threshold, it is summed to the output after negative exponential distance conversion. This thresholding eliminates outlying pixels, in such a way that they do not appear in the final output. The SMR algorithm is as follows:

1. The search area, (h_1, h_2), is limited to the neighborhood of the target's previous position.

2. For each pixel in the template $G(x, y)$, the method is checking if the condition $|F(x+h_1, y+h_2) - G(x, y)| \leq \alpha$ is satisfied, where α is a dynamic threshold defined in 6.

3. If satisfied, we are interested in how close the match is, so the pixel difference is converted into a probability value p by $p = \exp(-|F(x+h_1, y+h_2) - G(x, y)|)$. If not these pixels are ignored.

4. The probability values are summed up for each patch. The algorithm finds the (h_1, h_2) that gives the highest similarity matching ratio, $\arg\max_{h_1,h_2} \sum p$.

5. $G(x,y)_{t+1} = F(x+h_1, y+h_2)_t$ The patch is extracted in every detection and assigned as new template.

6. Dynamic threshold $\alpha = \max(G(x,y)_t - G(x,y)_{t+1}) \cdot k$ where $k = 0.25$ is a constant determined experimentally.

The biggest advantage of the SMR is that pixel differences above α are not contributing to the matching similarity output. These pixels may be outliers or points that dramatically change appearance, and thus should not affect the matching similarity. Outlying pixels usually only increase the error and cause failure, so we chose to ignore them in this method. This way, only reliably matching pixels contribute to the output of each matching step.

6.2.4 Accuracy Results

This approach is tested on a challenging benchmark: the TLD dataset [152]. From this dataset six videos with different properties were selected as displayed in Table 6.2.1. Each video contains only one target. The metric used is the number of correctly tracked frames. For this test, color videos are converted to grayscale. State-of-the-art performance is achieved and results are presented in Table 6.5.

To illustrate how the qualitatively different way of defining the tracking problem of the SMR tracker provides better results than the traditional approach, we will compare the SMR tracker with the SAD tracker in the present section.

Figure 6.3 shows the detections from the SAD tracker and the SMR tracker where they have used the same template. Points that dramatically changed appearance cause the SAD tracker to fail whereas the SMR tracker correctly detects the object. For illustration purposes, the differences for each pixel between the template and the patches the SAD tracker and the SMR tracker detected are mapped in Figure 6.3. The patch the SMR tracker detected has a bigger sum of absolute differences. However, that is because of the region that dramatically changed appearance. That patch has many close matches with the template as can be seen in Figure 6.4. As such, the SMR tracker is able to detect it. Again, with the

same principle the SMR tracker is able to track the object when it is going out of the scene as shown in Figure 6.5.

Fig. 6.5. The red boxes are the SMR tracker's outputs. The video frame is extended and padded by zeroes. The SMR tracker is able to track when the target is going out of the frame. The template update is ceased in these situations which prevents the drifting from the object.

Fig. 6.6. (Top) The red boxes are the SMR tracker's outputs. (Bottom) The blue boxes are the SAD tracker's outputs. Outlying pixels cause the SAD tracker to drift, whereas the SMR tracker is not affected by them.

The SMR tracker is more robust to outliers than the traditional approach. As can be seen in Figure 6.6, outliers cause the SAD tracker to drift away from the object, whereas the SMR tracker (Figure 6.6) finds the target. Ideally, the bounding box should be entirely filled with the target. However, during long-term tracking, the object may move back and forth and rotate which causes some background pixels to be included in the next template. A tracker does not know which pixels belong to the object and which ones belong to the background. On the other hand, the SMR tracker has a higher probability of rejecting background pixels, as they tend to change more.

The SAD tracker from the 2nd frame to 3rd in Figure 6.6 (bottom) drifts away from the object, because the pixels from the background have become included in the bounding box and they propagate to the template. When the face moves right, the SAD tracker does not move and drifts away from the object because the background, which has high contrast, gives big differences if the bounding box shifts to a new position. Therefore, the traditional approach gives priority to preventing big differences when it is making a decision, even if these pixels are not the majority of the template. On the other hand, the SMR tracker is focusing on the number of pixels that have small differences with the template, which is a human face in this case (Figure 6.6 top).

Failure Mode

Even though the SMR tracker updates the template at every frame in this presented work, drifts caused by the accumulation of small errors during each detection are not observed by applying this method on the benchmark dataset. However, when an object becomes occluded very slowly, updating the template at every frame causes the template to include foreground pixels that do not belong to the object. An example can be seen in Figure 6.7. A better template update mechanism will prevent this kind of failure. This will most probably require the use of a classifier, which is out of the scope of the work in this chapter. Another limitation of this method is the inability of updating the template size. This may become a problem when the object goes further away from the camera. In

that case, the object will get smaller and may become a minority of the pixels within the bounding box which would cause the failure of the tracker.

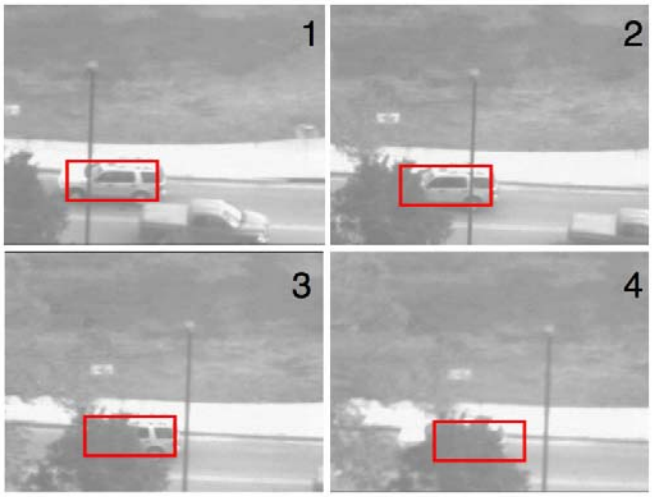


Fig. 6.7. Red boxes are the SMR tracker's results. The every-frame template update causes the outlying pixels to propagate to the templates. When outlying pixels dominate the template, the SMR tracker fails.

6.2.5 SMR Accelerator

The SMR hardware accelerator is implemented on the Avnet Zedboard which has the Zynq 7000 All Programmable SoC consisting of a Dual core Cortex-A9 and a Programmable Logic.

We follow the same architecture as presented in the last chapter. Such that we group all patch operations into one single accelerator which provides us with following:

1. Exploits parallelism between patches and within patches.

2. Chains multiple operations in one clock cycle. These operations include sum of absolute differences, comparison with threshold, exponential operations and summation of the exponentials.

Fig. 6.8. Tracking results obtained on the hardware accelerator. The application takes video sequences from the TLD dataset, Pedestrian 1 [152].

3. Makes the communication efficient.

The architecture is a fully parallel pipelined that outputs a SMR sum of probabilities every clock cycle. The partial sum of the previous operation and the input are held in the pipeline registers. This algorithm has an exponential operation which is implemented as a look up table in the custom hardware. The final row wise addition is calculated in a single cycle with a logarithmic tree structure adder.

The input frames are 320×240 RGB images. From the first frame, a template is extracted by clicking on the target. The coordinates of the template and the image are sent to the hardware accelerator. The template size in the example from Figure 6.8 is $3 \times 15 \times 15$. The template matching operation is performed in the 100×100 spatial area centered around the object's previous location. Figure 6.8 displays the tracking results from the Zed board where the tracker successfully finds the target in the images.

By exploiting the node and weight level parallelisms of the algorithm, the presented hardware accelerator for SMR provides 60 times speed up compared to the baseline embedded processor, a dual-core ARM Cortex A9.

6.3 Summary

We presented our system with two different applications of ConvNets: object classification and object detection. Our experiments showed that it is possible to run successful applications of ConvNets in real-time with significant power efficiency. The applications using ConvNets are extensive, including tracking [171, 172], action recognition [173], face recognition [83], pixel-level scene labeling [174] and stereo matching [175]. These applications can be run on our system without any modification.

In this chapter, we also proposed a novel approach of tracking: the Similarity Matching Ratio (SMR). The SMR tracker is more robust to outliers than the traditional approaches because it is not collecting differences between the template and the frame for each pixel. Instead, it is collecting probabilities from the pixels that have small differences from the template. The SMR tracker tries to find a region which maximizes the good match instead of minimizing the differences for the whole template. The SMR tracker is tested on challenging video sequences and achieves state-of-the-art performance. These results show that SMR is a superior approach.

We implemented the SMR algorithm on the proposed system by slightly modifying the custom hardware architecture and achieved significant speed-up. We showed that the hardware system that is presented in the last chapter can be used for generic image processing applications which use convolution-like data flow. SMR is one type that uses convolution-like data flow, simpler examples include sum-of-absolute-differences (SAD) or sum-of-square-differences (SSD) operations, widely used in tracking and motion estimation algorithms.

7. CONCLUSION

This thesis has extensively studied Convolutional Neural Networks (ConvNets). ConvNets with millions of parameters trained on large-scale datasets close the gap with human performance in many tasks. In this scenario, there were two big issues to be resolved: 1) to obtain the state-of-the-art performance with minimal number of labeled data, 2) to be able to use ConvNets in real-time applications on mobile phones, security cameras or in robotics despite the fact that they have millions of parameters and connections.

In order to achieve high performance using a few amount of labeled data, there are two approaches: 1) exploiting unlabeled data for learning feature hierarchies, 2) increasing the size of a dataset by augmentation techniques. We have studied both of these approaches. In order to discover patterns from unlabeled data, we used clustering learning algorithm. We trained a deep ConvNet based on an enhanced version of a clustering algorithm. The learned patterns from our proposed algorithm provided us with the filters in ConvNets. We also learned the connections between the layers of a deep ConvNet which improves its ability to be trained on a smaller amount of labeled data using an unsupervised learning technique. Our framework obtained state-of-the-art performance results in different tasks. We also extensively studied the augmentation techniques which result in higher performer networks. In order to increase the dataset size, we proposed new augmentation techniques that are based on segmenting the foreground objects from background. The proposed augmentation methods provided increased accuracy of networks.

For the real-time applications of ConvNets, we studied different ways to reduce the number of parameters and also explored specific kind of parallelizations on specific hardware architectures. We presented an optimized streaming method for ConvNets' hardware accelerator on an embedded platform. We built different applications of ConvNets. Our proposed system outperformed all other existing platforms while running these applications.

We summarize our contributions below:

1. We showed that the accuracy obtained using unsupervised learning algorithms can be greatly improved by reducing the correlation among parameters.

2. By limiting the connections between layers of deep networks, we were able to scale-up the depth of networks via an unsupervised learning algorithm.

3. We showed that background in the training datasets can have drastic effects on the testing accuracy. We improved the performance of a network by using segmented objects in augmenting datasets.

4. We reviewed different methods of accelerating ConvNets on CPUs, GPUs, and custom hardware. We presented a custom hardware accelerator for ConvNets and showed that it can provide significant energy efficiency compared to other existing platforms.

5. We built different applications of ConvNets and showcased that they can be used for practical applications on embedded environments. We also proposed a new template matching algorithm for visual tracking which obtained state-of-the-art performance on several well-known datasets. We ran this tracking algorithm on our proposed system and demonstrated its applicability and flexibility.

Overall, this work shows that ConvNets do not need enormous amount of labeled data and similar performance results can be obtained with much fewer labeled data by utilizing unlabeled data and aggressive augmentation techniques. Furthermore, applications of ConvNets are ready to be used in embedded platforms which exploit parallelizations at many levels.

REFERENCES

REFERENCES

[1] A. L. Samuel, "Some studies in machine learning using the game of checkers," *IBM Journal of research and development*, vol. 3, no. 3, pp. 210–229, 1959.

[2] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The bulletin of mathematical biophysics*, vol. 5, no. 4, pp. 115–133, 1943.

[3] Web, *sebastianraschka.com/Articles/2015_singlelayer_neurons.html*, 2015.

[4] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun, "Overfeat: Integrated recognition, localization and detection using convolutional networks," *arXiv preprint arXiv:1312.6229*, 2013.

[5] G. Leuba and R. Kraftsik, "Changes in volume, surface estimate, three-dimensional shape and total number of neurons of the human primary visual cortex from midgestation until old age," *Anatomy and embryology*, vol. 190, no. 4, pp. 351–366, 1994.

[6] A. Dundar, J. Jin, and E. Culurciello, "Convolutional clustering for unsupervised learning," *Internatinal conference on learning representations*, 2016.

[7] A. Dundar, J. Jin, V. Gokhale, B. Martini, and E. Culurciello, "Memory access optimized routing scheme for deep networks on a mobile coprocessor," in *IEEE High Performance Extreme Computing Conference (HPEC)*, 2014.

[8] A. Dundar, J. Jin, B. Martini, and E. Culurciello, "Embedded streaming deep neural networks accelerator with applications," *IEEE Transactions on Neural Networks and Learning Systems*, vol. PP, no. 99, pp. 1–12, 2016.

[9] A. Dundar, J. Jin, V. Gokhale, B. Krishnamurthy, A. Canziani, B. Martini, and E. Culurciello, "Accelerating deep neural networks on mobile processor with embedded programmable logic," in *Neural information processing systems conference (NIPS)*, 2013.

[10] A. Dundar, J. Jin, and E. Culurciello, "Visual tracking with similarity matching ratio," *In Proceedings of the International Conference on Computer Vision Theory and Applications (VISAPP)*, 2013.

[11] D. H. Hubel and T. N. Wiesel, "Receptive fields and functional architecture in two nonstriate visual areas (18 and 19) of the cat," *Journal of neurophysiology*, vol. 28, no. 2, pp. 229–289, 1965.

[12] L. Wiskott, "How does our visual system achieve shift and size invariance," *Chapter 16 in 23 Problems in Systems Neuroscience*, 2005.

[13] M. Riesenhuber and T. Poggio, "Models of object recognition," *Nature neuroscience*, vol. 3, pp. 1199–1204, 2000.

[14] T. Serre and M. Riesenhuber, "Realistic modeling of simple and complex cell tuning in the hmax model, and implications for invariant object recognition in cortex," DTIC Document, Tech. Rep., 2004.

[15] A. Krizhevsky, I. Sutskever, and G. Hinton, "ImageNet classification with deep convolutional neural networks," *Advances in Neural Information Processing Systems*, vol. 25, 2012.

[16] S. Lazebnik, C. Schmid, and J. Ponce, "Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories," in *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, vol. 2, 2006, pp. 2169–2178.

[17] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool, "Speeded-up robust features (surf)," *Comput. Vis. Image Underst.*, vol. 110, no. 3, pp. 346–359, Jun. 2008.

[18] T. Serre, L. Wolf, S. Bileschi, M. Riesenhuber, and T. Poggio, "Robust object recognition with cortex-like mechanisms," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 29, no. 3, pp. 411–426, 2007.

[19] T. Serre, L. Wolf, and T. Poggio, "Object recognition with features inspired by visual cortex," in *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, vol. 2. IEEE, 2005, pp. 994–1000.

[20] J. Mutch and D. G. Lowe, "Multiclass object recognition with sparse, localized features," in *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, vol. 1. IEEE, 2006, pp. 11–18.

[21] M. Oquab, L. Bottou, I. Laptev, and J. Sivic, "Is object localization for free?-weakly-supervised learning with convolutional neural networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 685–694.

[22] S. Hong, H. Noh, and B. Han, "Decoupled deep neural network for semi-supervised semantic segmentation," in *Advances in Neural Information Processing Systems*, 2015, pp. 1495–1503.

[23] K. Simonyan and A. Zisserman, "Two-stream convolutional networks for action recognition in videos," in *Advances in Neural Information Processing Systems*, 2014, pp. 568–576.

[24] S. Marčelja, "Mathematical description of the responses of simple cortical cells*," *JOSA*, vol. 70, no. 11, pp. 1297–1300, 1980.

[25] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional neural networks," *arXiv preprint arXiv:1311.2901*, 2013.

[26] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller, "Striving for simplicity: The all convolutional net," *arXiv preprint arXiv:1412.6806*, 2014.

[27] J. Yosinski, J. Clune, A. Nguyen, T. Fuchs, and H. Lipson, "Understanding neural networks through deep visualization," *arXiv preprint arXiv:1506.06579*, 2015.

[28] L. Bottou, "Online learning and stochastic approximations," *Online learning in neural networks*, vol. 17, no. 9, p. 25, 1998.

[29] ——, "Large-scale machine learning with stochastic gradient descent," *Proceedings of COMPSTAT'2010*, pp. 177–186, 2010.

[30] L. B. Y. Le Cun and L. Bottou, "Large scale online learning," *Advances in neural information processing systems*, vol. 16, p. 217, 2004.

[31] D. C. Cireşan, U. Meier, J. Masci, L. M. Gambardella, and J. Schmidhuber, "High-performance neural networks for visual object classification," *arXiv preprint arXiv:1102.0183*, 2011.

[32] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural computation*, vol. 1, no. 4, pp. 541–551, 1989.

[33] Y. LeCun, F. J. Huang, and L. Bottou, "Learning methods for generic object recognition with invariance to pose and lighting," in *Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on*, vol. 2. IEEE, 2004, pp. II–97.

[34] K. Jarrett, K. Kavukcuoglu, M. Ranzato, and Y. LeCun, "What is the best multi-stage architecture for object recognition?" in *Computer Vision, 2009 IEEE 12th International Conference on*. IEEE, 2009, pp. 2146–2153.

[35] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, pp. 1–42, April 2015.

[36] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," *arXiv preprint arXiv:1409.4842*, 2014.

[37] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[38] I. S. Dhillon and D. S. Modha, "Concept decompositions for large sparse text data using clustering," *Machine learning*, vol. 42, no. 1-2, pp. 143–175, 2001.

[39] C. Zetzsche, G. Krieger, and B. Wegmann, "The atoms of vision: Cartesian or polar?" *JOSA A*, vol. 16, no. 7, pp. 1554–1565, 1999.

[40] A. Lemme, R. F. Reinhart, and J. J. Steil, "Efficient online learning of a non-negative sparse autoencoder." in *ESANN*. Citeseer, 2010.

[41] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, no. 5786, pp. 504–507, 2006.

[42] J. Zhao, M. Mathieu, R. Goroshin, and Y. Lecun, "Stacked what-where auto-encoders," *arXiv preprint arXiv:1506.02351*, 2015.

[43] A. Rasmus, T. Raiko, and H. Valpola, "Denoising autoencoder with modulated lateral connections learns invariant representations of natural images," *arXiv preprint arXiv:1412.7210*, 2014.

[44] A. Rasmus, H. Valpola, M. Honkala, M. Berglund, and T. Raiko, "Semi-supervised learning with ladder network," *arXiv preprint arXiv:1507.02672*, 2015.

[45] R. Mihalcea, "Co-training and self-training for word sense disambiguation," in *Proceedings of the Conference on Computational Natural Language Learning (CoNLL-2004)*, 2004.

[46] S. Sukhbaatar and R. Fergus, "Learning from noisy labels with deep neural networks," *arXiv preprint arXiv:1406.2080*, 2014.

[47] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *CVPR*, 2009, pp. 248–255.

[48] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[49] A. Coates and A. Y. Ng, "Selecting receptive fields in deep networks," in *Advances in Neural Information Processing Systems*, 2011, pp. 2528–2536.

[50] L. Bo, X. Ren, and D. Fox, "Unsupervised feature learning for rgb-d based object recognition," in *Experimental Robotics*. Springer, 2013, pp. 387–402.

[51] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *Computer Vision–ECCV 2014*. Springer, 2014, pp. 818–833.

[52] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[53] C. Poultney, S. Chopra, Y. L. Cun *et al.*, "Efficient learning of sparse representations with an energy-based model," in *Advances in neural information processing systems*, 2006, pp. 1137–1144.

[54] M. D. Zeiler, D. Krishnan, G. W. Taylor, and R. Fergus, "Deconvolutional networks," in *CVPR*, 2010, pp. 2528–2535.

[55] H. Lee, R. Grosse, R. Ranganath, and A. Y. Ng, "Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations," in *ICML*. ACM, 2009, pp. 609–616.

[56] K. Kavukcuoglu, P. Sermanet, Y.-L. Boureau, K. Gregor, M. Mathieu, and Y. L. Cun, "Learning convolutional feature hierarchies for visual recognition," in *Advances in neural information processing systems*, 2010, pp. 1090–1098.

[57] A. Coates and A. Y. Ng, "The importance of encoding versus training with sparse coding and vector quantization," in *ICML*, 2011, pp. 921–928.

[58] T.-H. Lin and H. Kung, "Stable and efficient representation learning with nonnegativity constraints," in *ICML*, 2014, pp. 1323–1331.

[59] E. Culurciello, J. Jin, A. Dundar, and J. Bates, "An analysis of the connections between layers of deep neural networks," *arXiv preprint arXiv:1306.0152*, 2013.

[60] M. Lin, Q. Chen, and S. Yan, "Network in network," *arXiv preprint arXiv:1312.4400*, 2013.

[61] A. Coates, A. Y. Ng, and H. Lee, "An analysis of single-layer networks in unsupervised feature learning," in *International conference on artificial intelligence and statistics*, 2011, pp. 215–223.

[62] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving neural networks by preventing coadaptation of feature detectors," *arXiv preprint arXiv:1207.0580*, 2012.

[63] K. Y. Hui, "Direct modeling of complex invariances for visual object features," in *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, 2013, pp. 352–360.

[64] K. Swersky, J. Snoek, and R. P. Adams, "Multi-task bayesian optimization," in *Advances in Neural Information Processing Systems*, 2013, pp. 2004–2012.

[65] T. L. Paine, P. Khorrami, W. Han, and T. S. Huang, "An analysis of unsupervised pre-training in light of recent advances," *arXiv preprint arXiv:1412.6597*, 2014.

[66] E. Hoffer and N. Ailon, "Deep metric learning using triplet network," *arXiv preprint arXiv:1412.6622*, 2014.

[67] A. Dosovitskiy, J. T. Springenberg, M. Riedmiller, and T. Brox, "Discriminative unsupervised feature learning with convolutional neural networks," in *Advances in Neural Information Processing Systems*, 2014, pp. 766–774.

[68] S. Rifai, P. Vincent, X. Muller, X. Glorot, and Y. Bengio, "Contractive auto-encoders: Explicit invariance during feature extraction," in *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, 2011, pp. 833–840.

[69] D.-H. Lee, "Pseudo-label: The simple and efficient semi-supervised learning method for deep neural networks," in *Workshop on Challenges in Representation Learning, ICML*, vol. 3, 2013.

[70] D. P. Kingma, S. Mohamed, D. J. Rezende, and M. Welling, "Semi-supervised learning with deep generative models," in *Advances in Neural Information Processing Systems*, 2014, pp. 3581–3589.

[71] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *International conference on artificial intelligence and statistics*, 2010, pp. 249–256.

[72] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *Neural Networks, IEEE Transactions on*, vol. 5, no. 2, pp. 157–166, 1994.

[73] G. E. Hinton, S. Osindero, and Y.-W. Teh, "A fast learning algorithm for deep belief nets," *Neural computation*, vol. 18, no. 7, pp. 1527–1554, 2006.

[74] Y. Bengio, P. Lamblin, D. Popovici, H. Larochelle *et al.*, "Greedy layer-wise training of deep networks," *Advances in neural information processing systems*, vol. 19, p. 153, 2007.

[75] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol, "Extracting and composing robust features with denoising autoencoders," in *In Proceedings of the Twenty-fifth International Conference on Machine Learning (ICML 2008)*, 2008.

[76] J. Weston, F. Ratle, H. Mobahi, and R. Collobert, "Deep learning via semi-supervised embedding," in *Neural Networks: Tricks of the Trade*. Springer, 2012, pp. 639–655.

[77] H. Lee, C. Ekanadham, and A. Y. Ng, "Sparse deep belief net model for visual area v2," in *Advances in neural information processing systems*, 2008, pp. 873–880.

[78] D. Erhan, Y. Bengio, A. Courville, P.-A. Manzagol, P. Vincent, and S. Bengio, "Why does unsupervised pre-training help deep learning?" *The Journal of Machine Learning Research*, vol. 11, pp. 625–660, 2010.

[79] I. Sutskever, J. Martens, G. Dahl, and G. Hinton, "On the importance of initialization and momentum in deep learning," in *Proceedings of the 30th international conference on machine learning (ICML-13)*, 2013, pp. 1139–1147.

[80] D. Ciresan, U. Meier, and J. Schmidhuber, "Multi-column deep neural networks for image classification," in *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*. IEEE, 2012, pp. 3642–3649.

[81] Y. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller, "Efficient backprop," in *Neural Networks: Tricks of the Trade*. Springer, 1998, pp. 9–50.

[82] S. Chetlur, C. Woolley, P. Vandermersch, J. Cohen, J. Tran, B. Catanzaro, and E. Shelhamer, "cudnn: Efficient primitives for deep learning," vol. abs/1410.0759, 2014.

[83] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf, "Deepface: Closing the gap to human-level performance in face verification," in *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*. IEEE, 2014, pp. 1701–1708.

[84] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *CoRR*, vol. abs/1409.1556, 2014.

[85] A. S. Razavian, H. Azizpour, J. Sullivan, and S. Carlsson, "Cnn features off-the-shelf: an astounding baseline for recognition," in *Computer Vision and Pattern Recognition Workshops (CVPRW), 2014 IEEE Conference on*. IEEE, 2014, pp. 512–519.

[86] J. Donahue, Y. Jia, O. Vinyals, J. Hoffman, N. Zhang, E. Tzeng, and T. Darrell, "Decaf: A deep convolutional activation feature for generic visual recognition," *arXiv preprint arXiv:1310.1531*, 2013.

[87] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, "How transferable are features in deep neural networks?" in *Advances in Neural Information Processing Systems*, 2014, pp. 3320–3328.

[88] M. Jaderberg, K. Simonyan, A. Vedaldi, and A. Zisserman, "Synthetic data and artificial neural networks for natural scene text recognition," *arXiv preprint arXiv:1406.2227*, 2014.

[89] X. Peng, B. Sun, K. Ali, and K. Saenko, "Exploring invariances in deep convolutional neural networks using synthetic images," *arXiv preprint arXiv:1412.7122*, 2014.

[90] R. Wu, S. Yan, Y. Shan, Q. Dang, and G. Sun, "Deep image: Scaling up image recognition," *arXiv preprint arXiv:1501.02876*, 2015.

[91] A. Mahendran and A. Vedaldi, "Understanding deep image representations by inverting them," *arXiv preprint arXiv:1412.0035*, 2014.

[92] K. Lenc and A. Vedaldi, "Understanding image representations by measuring their equivariance and equivalence," *arXiv preprint arXiv:1411.5908*, 2014.

[93] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," *arXiv preprint arXiv:1312.6199*, 2013.

[94] A. Nguyen, J. Yosinski, and J. Clune, "Deep neural networks are easily fooled: High confidence predictions for unrecognizable images," *arXiv preprint arXiv:1412.1897*, 2014.

[95] R. Ng, M. Levoy, M. Brédif, G. Duval, M. Horowitz, and P. Hanrahan, "Light field photography with a hand-held plenoptic camera."

[96] M. Martinello, A. Wajs, S. Quan, H. Lee, C. Lim, T. Woo, W. Lee, S.-S. Kim, and D. Lee, "Dual aperture photography: Image and depth from a mobile camera."

[97] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 3431–3440.

[98] D. Comaniciu and P. Meer, "Mean shift: A robust approach toward feature space analysis," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 24, no. 5, pp. 603–619, 2002.

[99] C. Rother, V. Kolmogorov, and A. Blake, "Grabcut: Interactive foreground extraction using iterated graph cuts," *ACM Transactions on Graphics (TOG)*, vol. 23, no. 3, pp. 309–314, 2004.

[100] J. Hays and A. A. Efros, "Scene completion using millions of photographs," *ACM Transactions on Graphics (SIGGRAPH 2007)*, vol. 26, no. 3, 2007.

[101] M. Daisy, D. Tschumperlé, and O. Lézoray, "A fast spatial patch blending algorithm for artefact reduction in pattern-based image inpainting," in *SIGGRAPH Asia 2013 Technical Briefs*, ser. SA '13. New York, NY, USA: ACM, 2013, pp. 8:1–8:4.

[102] C. Barnes, E. Shechtman, A. Finkelstein, and D. Goldman, "Patchmatch: A randomized correspondence algorithm for structural image editing," *ACM Transactions on Graphics-TOG*, vol. 28, no. 3, p. 24, 2009.

[103] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," *CoRR*, vol. abs/1207.0580, 2012.

[104] M. A. Erdogdu and A. Montanari, "Convergence rates of sub-sampled newton methods," in *Advances in Neural Information Processing Systems*, 2015, pp. 3034–3042.

[105] M. A. Erdogdu, "Newton-stein method: A second order method for glms via stein's lemma," in *Advances in Neural Information Processing Systems 28*, 2015, pp. 1216–1224.

[106] C. Farabet, B. Martini, P. Akselrod, S. Talay, Y. LeCun, and E. Culurciello, "Hardware accelerated convolutional neural networks for synthetic vision systems," in *Circuits and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium on*, 2010, pp. 257–260.

[107] Y. N. Dauphin and Y. Bengio, "Big neural networks waste capacity," *arXiv preprint arXiv:1301.3583*, 2013.

[108] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *arXiv preprint arXiv:1503.02531*, 2015.

[109] A. Krizhevsky, "One weird trick for parallelizing convolutional neural networks," *arXiv preprint arXiv:1404.5997*, 2014.

[110] C. Farabet, C. Poulet, and Y. LeCun, "An fpga-based stream processor for embedded real-time vision with convolutional networks," in *Computer Vision Workshops (ICCV Workshops), 2009 IEEE 12th International Conference on*. IEEE, 2009, pp. 878–885.

[111] M. Sankaradas, V. Jakkula, S. Cadambi, S. Chakradhar, I. Durdanovic, E. Cosatto, and H. Graf, "A massively parallel coprocessor for convolutional neural networks," in *Application-specific Systems, Architectures and Processors, 2009. ASAP 2009. 20th IEEE International Conference on*, 2009, pp. 53–60.

[112] S. Cadambi, A. Majumdar, M. Becchi, S. Chakradhar, and H. P. Graf, "A programmable parallel accelerator for learning and classification," in *Proceedings of the 19th international conference on Parallel architectures and compilation techniques*, ser. PACT '10. New York, NY, USA: ACM, 2010, pp. 273–284.

[113] H. P. Graf, S. Cadambi, I. Durdanovic, V. Jakkula, M. Sankaradass, E. Cosatto, and S. Chakradhar, "A massively parallel digital learning processor," in *Advances in Neural Information Processing Systems 21*, D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, Eds., 2009, pp. 529–536.

[114] W. Qadeer, R. Hameed, O. Shacham, P. Venkatesan, C. Kozyrakis, and M. A. Horowitz, "Convolution engine: balancing efficiency & flexibility in specialized computing," in *ACM SIGARCH Computer Architecture News*, vol. 41, no. 3. ACM, 2013, pp. 24–35.

[115] C. Yan, Y. Zhang, J. Xu, F. Dai, J. Zhang, Q. Dai, and F. Wu, "Efficient parallel framework for hevc motion estimation on many-core processors," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 24, no. 12, pp. 2077–2089, 2014.

[116] C. Yan, Y. Zhang, J. Xu, F. Dai, L. Li, Q. Dai, and F. Wu, "A highly parallel framework for hevc coding unit partitioning tree decision on many-core processors," *Signal Processing Letters, IEEE*, vol. 21, no. 5, pp. 573–576, 2014.

[117] M. Denil, B. Shakibi, L. Dinh, N. de Freitas *et al.*, "Predicting parameters in deep learning," in *Advances in Neural Information Processing Systems*, 2013, pp. 2148–2156.

[118] E. L. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus, "Exploiting linear structure within convolutional networks for efficient evaluation," in *Advances in Neural Information Processing Systems*, 2014, pp. 1269–1277.

[119] M. Jaderberg, A. Vedaldi, and A. Zisserman, "Speeding up convolutional neural networks with low rank expansions," *arXiv preprint arXiv:1405.3866*, 2014.

[120] Y. LeCun, J. S. Denker, S. A. Solla, R. E. Howard, and L. D. Jackel, "Optimal brain damage." in *NIPs*, vol. 89, 1989.

[121] B. Hassibi and D. G. Stork, *Second order derivatives for network pruning: Optimal brain surgeon*. Morgan Kaufmann, 1993.

[122] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

[123] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *Advances in Neural Information Processing Systems*, 2015, pp. 1135–1143.

[124] J. Jin, A. Dundar, and E. Culurciello, "Flattened convolutional neural networks for feedforward acceleration," *arXiv preprint arXiv:1412.5474*, 2014.

[125] T. G. Dietterich, "Ensemble methods in machine learning," in *Multiple classifier systems*. Springer, 2000, pp. 1–15.

[126] C. Bucilu?, R. Caruana, and A. Niculescu-Mizil, "Model compression," in *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2006, pp. 535–541.

[127] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," 2009.

[128] A. R. Omondi and J. C. Rajapakse, *FPGA implementations of neural networks*. Springer, 2006.

[129] P. Sermanet, K. Kavukcuoglu, and Y. LeCun, "Eblearn: Open-source energy-based learning in c++," in *Tools with Artificial Intelligence, 2009. ICTAI'09. 21st International Conference on*. IEEE, 2009, pp. 693–697.

[130] R. Collobert, K. Kavukcuoglu, and C. Farabet, "Implementing neural networks efficiently," in *Neural Networks: Tricks of the Trade*. Springer, 2012, pp. 537–557.

[131] "Netlib blas," *http://www.netlib.org/blas/*, 2014.

[132] K. Chellapilla, S. Puri, P. Simard *et al.*, "High performance convolutional neural networks for document processing," in *Tenth International Workshop on Frontiers in Handwriting Recognition*, 2006.

[133] V. Vanhoucke, A. Senior, and M. Z. Mao, "Improving the speed of neural networks on cpus," in *Proc. Deep Learning and Unsupervised Feature Learning NIPS Workshop*, 2011.

[134] D. C. Cireşan, U. Meier, J. Masci, L. M. Gambardella, and J. Schmidhuber, "Flexible, high performance convolutional neural networks for image classification," in *Proceedings of the Twenty-Second international joint conference on Artificial Intelligence-Volume Volume Two*. AAAI Press, 2011, pp. 1237–1242.

[135] F. Nasse, C. Thurau, and G. A. Fink, "Face detection using gpu-based convolutional neural networks," in *Computer Analysis of Images and Patterns*. Springer, 2009, pp. 83–90.

[136] D. Scherer, H. Schulz, and S. Behnke, "Accelerating large-scale convolutional neural networks with parallel graphics multiprocessors," in *Artificial Neural Networks–ICANN 2010*. Springer, 2010, pp. 82–91.

[137] A. Coates, B. Huval, T. Wang, D. Wu, B. Catanzaro, and N. Andrew, "Deep learning with cots hpc systems," in *Proceedings of The 30th International Conference on Machine Learning*, 2013, pp. 1337–1345.

[138] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," *arXiv preprint arXiv:1408.5093*, 2014.

[139] R. Collobert, C. Farabet, and K. Kavukcuoglu, "Torch7: A matlab-like environment for machine learning," in *BigLearn, NIPS Workshop*, no. EPFL-CONF-192376, 2011.

[140] C. Farabet, B. Martini, B. Corda, P. Akselrod, E. Culurciello, and Y. LeCun, "Neuflow: A runtime reconfigurable dataflow processor for vision," in *Computer Vision and Pattern Recognition Workshops (CVPRW), 2011 IEEE Computer Society Conference on*, 2011, pp. 109–116.

[141] U. J. Kapasi, S. Rixner, W. J. Dally, B. Khailany, J. H. Ahn, P. Mattson, and J. D. Owens, "Programmable stream processors," *IEEE Computer*, pp. 54–62, Aug. 2003.

[142] J. Cloutier, E. Cosatto, S. Pigeon, F.-R. Boyer, and P. Simard, "Vip: an fpga-based processor for image processing and neural networks," in *Microelectronics for Neural Networks, 1996., Proceedings of Fifth International Conference on*, 1996, pp. 330–336.

[143] M. Peemen, A. A. Setio, B. Mesman, and H. Corporaal, "Memory-centric accelerator design for convolutional neural networks," in *Computer Design (ICCD), 2013 IEEE 31st International Conference on*. IEEE, 2013, pp. 13–19.

[144] P. Merolla, J. Arthur, R. Alvarez-Icaza, A. Cassidy, J. Sawada, F. Akopyan, B. Jackson, N. Imam, C. Guo, Y. Nakamura, B. Brezzo, I. Vo, S. Esser, R. Appuswamy, B. Taba, A. Amir, M. Flickner, W. Risk, R. Manohar, and D. Modha, "A million spiking-neuron integrated circuit with a scalable communication network and interface," *Science*, pp. 668–673, August 2014.

[145] K. Ovtcharov, O. Ruwase, J.-Y. Kim, J. Fowers, K. Strauss, and E. S. Chung, "Accelerating deep convolutional neural networks using specialized hardware," *Microsoft Research Whitepaper*, vol. 2, 2015.

[146] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, "Optimizing fpga-based accelerator design for deep convolutional neural networks," in *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM, 2015, pp. 161–170.

[147] J. Jin, V. Gokhale, A. Dundar, B. Krishnamurthy, B. Martini, and E. Culurciello, "An efficient implementation of deep convolutional neural networks on a mobile coprocessor," in *Circuits and Systems (MWSCAS), Proceedings of 2014 IEEE 57th International Midwest Symposium on*, 2014.

[148] V. Gokhale, J. Jin, A. Dundar, B. Martini, and E. Culurciello, "A 240 g-ops/s mobile coprocessor for deep neural networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2014, pp. 682–687.

[149] M. Delakis and C. Garcia, "text detection with convolutional neural networks." in *VISAPP (2)*, 2008, pp. 290–294.

[150] C. Garcia and M. Delakis, "A neural architecture for fast and robust face detection," in *Pattern Recognition, 2002. Proceedings. 16th International Conference on*, vol. 2. IEEE, 2002, pp. 44–47.

[151] E. Blaser, Z. Pylyshyn, A. Holcombe *et al.*, "Tracking an object through feature space," *Nature*, vol. 408, no. 6809, pp. 196–198, 2000.

[152] Z. Kalal, J. Matas, and K. Mikolajczyk, "P-N Learning: Bootstrapping Binary Classifiers by Structural Constraints," *Conference on Computer Vision and Pattern Recognition*, 2010.

[153] Z. Kalal, K. Mikolajczyk, and J. Matas, "Forward-backward error: Automatic detection of tracking failures," in *Pattern Recognition (ICPR), 2010 20th International Conference on*. IEEE, 2010, pp. 2756–2759.

[154] J. Lim, D. Ross, R. Lin, and M. Yang, "Incremental learning for visual tracking," *Advances in neural information processing systems*, vol. 17, pp. 793–800, 2004.

[155] R. Collins, Y. Liu, and M. Leordeanu, "Online selection of discriminative tracking features," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 27, no. 10, pp. 1631–1643, 2005.

[156] S. Avidan, "Ensemble tracking," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 29, no. 2, pp. 261–271, 2007.

[157] B. Babenko, M. Yang, and S. Belongie, "Visual tracking with online multiple instance learning," in *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*. IEEE, 2009, pp. 983–990.

[158] A. Yilmaz, O. Javed, and M. Shah, "Object tracking: A survey," *Acm Computing Surveys (CSUR)*, vol. 38, no. 4, p. 13, 2006.

[159] J. DiCarlo, D. Zoccolan, and N. Rust, "How does the brain solve visual object recognition?" *Neuron*, vol. 73, no. 3, pp. 415–434, 2012.

[160] Y. LeCun, F. Huang, and L. Bottou, "Learning methods for generic object recognition with invariance to pose and lighting," in *Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on*, vol. 2. IEEE, 2004, pp. II–97.

[161] T. Serre, L. Wolf, S. Bileschi, M. Riesenhuber, and T. Poggio, "Robust object recognition with cortex-like mechanisms," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 29, pp. 411–426, 2007.

[162] P. Sermanet, K. Kavukcuoglu, and Y. LeCun, "Traffic signs and pedestrians vision with multi-scale convolutional networks," *Snowbird Machine Learning Workshop*, 2011.

[163] S. Thorpe, D. Fize, C. Marlot *et al.*, "Speed of processing in the human visual system," *nature*, vol. 381, no. 6582, pp. 520–522, 1996.

[164] J. Wilmer and K. Nakayama, "Two distinct visual motion mechanisms for smooth pursuit: Evidence from individual differences," *Neuron*, vol. 54, no. 6, pp. 987–1000, 2007.

[165] B. Vintch, J. A. Movshon, and E. P. Simoncelli, "Characterizing receptive field structure of macaque v2 neurons in terms of their v1 afferents," *Annual meeting in Neuroscience*, November 2010.

[166] D. Comaniciu, V. Ramesh, and P. Meer, "Kernel-based object tracking," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 25, no. 5, pp. 564–577, 2003.

[167] J. Shi and C. Tomasi, "Good features to track," in *Computer Vision and Pattern Recognition, 1994. Proceedings CVPR'94., 1994 IEEE Computer Society Conference on*. IEEE, 1994, pp. 593–600.

[168] G. Hager and P. Belhumeur, "Efficient region tracking with parametric models of geometry and illumination," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 20, no. 10, pp. 1025–1039, 1998.

[169] M. Black and A. Jepson, "Eigentracking: Robust matching and tracking of articulated objects using a view-based representation," *International Journal of Computer Vision*, vol. 26, no. 1, pp. 63–84, 1998.

[170] T. Ishikawa, I. Matthews, and S. Baker, *Efficient image alignment with outlier rejection*. Citeseer, 2002.

[171] J. Fan, W. Xu, Y. Wu, and Y. Gong, "Human tracking using convolutional neural networks," *Neural Networks, IEEE Transactions on*, vol. 21, no. 10, pp. 1610–1623, 2010.

[172] J. Jin, A. Dundar, J. Bates, C. Farabet, and E. Culurciello, "Tracking with deep neural networks," in *Information Sciences and Systems (CISS), 2013 47th Annual Conference on*, March 2013, pp. 1–5.

[173] S. Ji, W. Xu, M. Yang, and K. Yu, "3d convolutional neural networks for human action recognition," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 35, no. 1, pp. 221–231, 2013.

[174] C. Farabet, C. Couprie, L. Najman, and Y. LeCun, "Learning hierarchical features for scene labeling," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 35, no. 8, pp. 1915–1929, 2013.

[175] J. Žbontar and Y. LeCun, "Computing the stereo matching cost with a convolutional neural network," *arXiv preprint arXiv:1409.4326*, 2014.

VITA

VITA

Aysegul Dundar received a Bs.C. degree with honors in Electrical and Electronic Engineering from Bogazici University, Istanbul, Turkey in 2011. She is pursuing a Ph.D. at the Weldon School of Biomedical Engineering at Purdue University under the direction of Eugenio Culurciello.

Her research is focused on synthetic models of the human vision system, specifically deep convolutional neural networks in hardware. Her work on nn-X embedded vision systems has been featured on the MIT Technology Review and involved in the start-up company Teradeep.