



저작자표시-비영리-변경금지 2.0 대한민국

이용자는 아래의 조건을 따르는 경우에 한하여 자유롭게

- 이 저작물을 복제, 배포, 전송, 전시, 공연 및 방송할 수 있습니다.

다음과 같은 조건을 따라야 합니다:



저작자표시. 귀하는 원저작자를 표시하여야 합니다.



비영리. 귀하는 이 저작물을 영리 목적으로 이용할 수 없습니다.



변경금지. 귀하는 이 저작물을 개작, 변형 또는 가공할 수 없습니다.

- 귀하는, 이 저작물의 재이용이나 배포의 경우, 이 저작물에 적용된 이용허락조건을 명확하게 나타내어야 합니다.
- 저작권자로부터 별도의 허가를 받으면 이러한 조건들은 적용되지 않습니다.

저작권법에 따른 이용자의 권리는 위의 내용에 의하여 영향을 받지 않습니다.

이것은 [이용허락규약\(Legal Code\)](#)을 이해하기 쉽게 요약한 것입니다.

[Disclaimer](#)

Ph.D. DISSERTATION

# **Pruning Deep Convolutional Neural Networks for Fast Inference**

빠른 추론을 위한 깊은 길쌈형 인공신경망의 솜아내기

BY

ANWAR SAJID

FEBRUARY 2017

DEPARTMENT OF ELECTRICAL ENGINEERING AND  
COMPUTER SCIENCE  
COLLEGE OF ENGINEERING  
SEOUL NATIONAL UNIVERSITY

Pruning Deep Convolutional Neural  
Networks for Fast Inference

빠른 추론을 위한 깊은 길쌈형  
인공신경망의 솎아내기

지도교수 성 원 용

이 논문을 공학박사 학위논문으로 제출함

2017년 2월





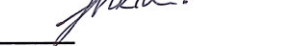
서울대학교 대학원

전기·컴퓨터 공학부

안 위 사 지 드

안위 사지드의 공학박사 학위논문을 인준함

2017년 2월

위 원 장:	최 기 영	
부위원장:	성 원 용	
위 원:	유 승 주	
위 원:	김 건 희	
위 원:	김 종 홍	

# Abstract

Deep learning algorithms have recently achieved human level classification performance on several diverse classification benchmarks including object and speech recognition. However these algorithms are computationally very expensive especially for resource limited portable machines. Several researches have proposed ideas to lower this cost and in this dissertation, we have addressed this problem. We have proposed pruning and fixed-point optimization techniques to reduce the computational complexity of deep neural networks. Pruning is a promising technique where a problem is first approximated with a large sized network followed by removing unimportant parameters.

The proposed work induces sparsity in a deep convolutional neural network (CNN) at three levels: feature map, kernels, and intra-kernel. Feature map pruning removes a large number of kernels and directly reduces the width of a layer and does not require any sparse representation. Thus the resulting network is thinner and runs faster than the predecessor unpruned network. However, feature map pruning removes all the incoming and outgoing kernels and thus affects a large number of parameters. We therefore may not achieve higher pruning ratios with feature map pruning. Kernel pruning eliminates  $k \times k$  kernels and is neither too fine nor too coarse. It can change the dense kernel connectivity pattern to a sparse one. Each convolution connection involves  $W \times H \times k \times k$  multiply and accumulate (MAC) operations where  $W$ ,  $H$  and  $k$  represents the feature map width, height and the kernel size, respectively. Further the

sparse representation for kernel pruning is also very simple. A single flag is enough to represent one convolution connection. Intra-kernel pruning removes scalar weights at the finest scale. The conventional pruning techniques induce irregular sparsity at the finest granularity by zeroing scalar weights. This sparsity can be induced in much higher rates but requires sparse representation in order to be translated into computational speedups in VLSI or parallel computer based implementations. Coarse pruning granularities demand very simple sparse representation but higher pruning ratios are comparatively difficult to achieve. On the contrary, fine grained pruning granularities can achieve higher pruning ratios but the sparse representation is more complicated. In this dissertation, we propose pruning techniques at the aforementioned three pruning granularities. We further show that various pruning granularities can be applied in combinations to compress the network size to the maximum limit.

The scalar weights inside a kernel is generally pruned in an irregular pattern. In this dissertation, we have proposed intra-kernel strided sparsity (IKSS). The IKSS prunes scalar weights at strided indices. We further impose a condition that all the outgoing kernels from a feature map must have the same stride and offset for IKSS. This has a direct impact on the sizes of matrices when convolutions are unrolled for matrix-matrix multiplications. The sparse representation for the constrained IKSS is only two numbers (*stride*, *offset*) per feature map. During pruning, it is important to locate the least adversarial pruning candidates. We have proposed three techniques for pruning candidate selection; particle filter, selecting the best of  $N$  random pruning masks, and activation sum voting for feature map pruning. The dissertation extensively discuss the best of  $N$  random masks technique and provide detailed analysis. We obtain more than 80% pruning ratios with various pruning granularities. Moreover, the pruned networks can be further compressed by quantizing the weights and

signals. This dissertation discusses our fixed-point optimization algorithm for deep convolutional neural network, where the network weights and signals are represented with 3-8 bits precision. We also discuss the layer-wise sensitivity analysis of deep convolutional neural networks. Thus we reduce the computational complexity of a CNN with pruning and fixed-point optimization.

In this dissertation, the proposed pruning techniques fit well to Graphics Processing Units. The IKSS can reduce the size of matrices and GPUs are quite good at multiplying matrices. Secondly, FFT based CNN implementations can benefit from kernel level pruning. For VLSI implementations, the fixed-point optimized techniques reduce the memory requirements and hence the networks can be hosted in the on-chip memory. Thus the proposed techniques can be exploited on a generic set of modern computing platforms.

**Keywords** : Convolutional Neural Network, Computational Complexity, Structured Pruning, Feature Map and Kernel Pruning, Fixed-Point Optimization

**Student Number** : 2012-31288

# Contents

<b>Abstract</b>	<b>i</b>
<b>Contents</b>	<b>iv</b>
<b>List of Figures</b>	<b>vii</b>
<b>List of Tables</b>	<b>x</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Convolutional Neural Networks (CNNs) . . . . .	2
1.2 Computational Complexity of Convolution Layers . . . . .	5
1.3 Publications Record . . . . .	8
1.4 Outline of the Dissertation . . . . .	9
<b>2 Background</b>	<b>10</b>
2.1 Introduction to Pruning . . . . .	10
2.2 Pruning Candidate Selection . . . . .	13
2.2.1 Evolutionary Particle Filter . . . . .	14
2.2.2 Activation Sum Voting . . . . .	17

2.2.3	Absolute Weight Sum Voting . . . . .	19
2.2.4	Best of $N$ Random Masks . . . . .	19
2.2.5	The Effect of Retraining on Pruning Masks . . . . .	25
2.3	Fixed-Point Optimization . . . . .	26
<b>3</b>	<b>Structured Pruning</b>	<b>28</b>
3.1	Introduction . . . . .	29
3.2	Feature Map and Intra-Kernel Pruning . . . . .	31
3.2.1	Intra Kernel Strided Sparsity . . . . .	31
3.3	Experimental Results . . . . .	37
3.3.1	Feature Map Pruning . . . . .	38
3.3.2	Intra-Kernel Pruning . . . . .	40
3.3.3	Pruning Granularities Applied in Combinations . . . . .	42
3.3.4	SVHN Dataset . . . . .	46
3.3.5	Execution Time Savings . . . . .	47
3.4	Comparison with the Previous Related Works . . . . .	49
3.5	Conclusions . . . . .	50
<b>4</b>	<b>Kernel Pruning</b>	<b>51</b>
4.1	Introduction . . . . .	51
4.2	Kernel and Feature Map Pruning . . . . .	52
4.3	Experimental Results . . . . .	57
4.3.1	CIFAR-10 . . . . .	57
4.3.2	CIFAR-100 . . . . .	62
4.3.3	SVHN . . . . .	64
4.4	Related Works . . . . .	65



4.5	Concluding Remarks . . . . .	66
<b>5</b>	<b>Quantizing the Pruned Networks</b>	<b>68</b>
5.1	Introduction . . . . .	68
5.2	Retraining Based Quantization . . . . .	70
5.2.1	L2 Error Minimization and Direct Quantization . . . . .	70
5.2.2	Layer Wise Sensitivity Analysis for Non-Uniform Quantization	72
5.2.3	Retraining with the Quantized Weights . . . . .	73
5.3	Separable Fixed-Point Kernels . . . . .	74
5.4	Quantizing the Pruned Networks . . . . .	77
5.4.1	Feature Map Pruned Networks . . . . .	79
5.4.2	Kernel Pruned Networks . . . . .	80
5.5	Concluding Remarks . . . . .	80
<b>6</b>	<b>Conclusion</b>	<b>82</b>
	<b>Bibliography</b>	<b>85</b>
	<b>Abstract in Korean</b>	<b>93</b>

# List of Figures

1.1	Sobel edge detector . . . . .	2
1.2	Convolutional Neural Network . . . . .	3
1.3	Average and max-pooling . . . . .	4
1.4	MAC operations in a single convolution . . . . .	6
2.1	Coarse and fine grained pruning granularities . . . . .	12
2.2	Example of state vector for particle filter . . . . .	15
2.3	Activation sum voting . . . . .	18
2.4	Absolute weight sum criterion . . . . .	20
2.5	MCR with the best of $N$ random masks for $CNN_{CIFAR-10}$ . . . . .	21
2.6	Distribution of $N$ random evaluations for $CNN_{CIFAR-10}$ . . . . .	21
2.7	MCR with the best of $N$ random masks for $CNN_{SVHN}$ . . . . .	22
2.8	Distribution of $N$ random evaluations for $CNN_{SVHN}$ . . . . .	22
2.9	Weight sum vs. best of $N$ random masks for $CNN_{MNIST}$ . . . . .	24
2.10	Weight sum vs. best of $N$ random masks for $CNN_{SVHN}$ . . . . .	24
2.11	In this plot, we prune a CNN network with various masks and compare their pre and post retraining performance. It can be observed that on the average, pre-retraining masks perform better after retraining.	26

3.1	Intra-kernel strided sparsity . . . . .	31
3.2	Convolution unrolling . . . . .	34
3.3	Convolution unrolling and IKSS . . . . .	35
3.4	Feature map pruning . . . . .	39
3.5	Intra-kernel pruning . . . . .	40
3.6	Feature map and IKSS, $CNN_{small}$ . . . . .	42
3.7	Feature map and IKSS, $CNN_{large}$ . . . . .	43
3.8	Constraint-less and constrained pruning, $CNN_{verylarge}$ . . . . .	44
3.9	Pruning SVHN network . . . . .	46
3.10	Profiling results . . . . .	48
4.1	Unconstrained kernel pruning . . . . .	53
4.2	Profiling kernel pruning . . . . .	55
4.3	GPU function scheduler call . . . . .	55
4.4	Best of $N$ random masks vs absolute weight sum . . . . .	56
4.5	Feature map and constraintless kernel pruning . . . . .	56
4.6	Feature map and kernel pruning, $CIFAR-10$ . . . . .	59
4.7	Combinations of feature map and kernel pruning . . . . .	59
4.8	Pruning $CNN_{CIFAR10.large}$ . . . . .	61
4.9	Pruning $CNN_{SVHN}$ . . . . .	61
4.10	$CIFAR-100$ $CNN$ . . . . .	63
4.11	Per class error on the SVHN dataset . . . . .	65
5.1	Direct quantization with L2 error minimization . . . . .	71
5.2	Layer-wise quantization sensitivity analysis . . . . .	72
5.3	Retraining with quantized weights . . . . .	73

5.4	A $3 \times 3$ separable kernel . . . . .	75
5.5	Network retraining with separable quantized kernels . . . . .	78

# List of Tables

2.1	Specifications of the three networks . . . . .	19
3.1	Specifications of the networks . . . . .	36
4.1	The CIFAR-10 networks . . . . .	57
4.2	Feature map and kernel level pruning (75%) in $CNN_{CIFAR10.small}$ . . . . .	60
5.1	Direct quantization of the MNIST network . . . . .	71
5.2	Retraining the quantized MNIST network . . . . .	77
5.3	Quantizing the $CNN_{SVHN}$ network . . . . .	79
5.4	Quantizing the feature map pruned $CNN_{SVHN}$ network . . . . .	79
5.5	Quantizing the kernel pruned $CNN_{SVHN}$ network . . . . .	80

# Chapter 1

## Introduction

Deep learning algorithms have long been known for their ability to approximate an empirical solution for a complex unknown function from the training examples. The deep learning research is enjoying a renaissance in recent years. This resurgence is mainly due to three reasons. First, the massively parallel Graphics Processing Units (GPUs) enable a research to experiment with very deep and diverse architectures. Moreover, GPUs also increase the productivity of researchers. Secondly, large amount of data is available for training deep neural networks. Thirdly, based on the first two reasons, high performance network architectures have been proposed in recent years. These factors have contributed to the human level performance of deep neural networks on some vision and speech recognition benchmarks. This development will significantly impact a large number of industries. To name a few, driverless cars, advanced driver assistance systems (ADAS), medical imaging, and intelligent software applications for smartphones are in the transition stage and rapidly changing. However, the computational complexity of these algorithms is still very high and

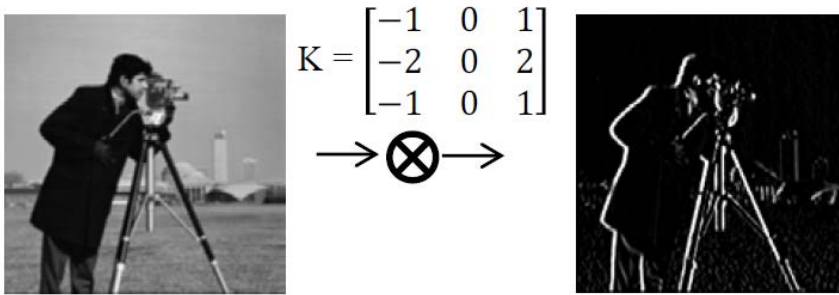


Figure 1.1: Sobel edge detector

the power consumption is also a serious concern. Several researches have therefore proposed ideas to lower the computational cost and this thesis also proposes solutions to this problem [1, 2, 3, 4, 5].

Network pruning is one promising technique that first learns a function with a sufficiently large sized network followed by removing less important connections [1, 2, 5]. This enables smaller networks to inherit knowledge from the large sized predecessor networks and exhibit a comparable level of performance. We propose structured pruning and fixed-point optimization techniques to reduce this complexity. In this dissertation, we will outline our contributions in pruning deep convolutional neural networks. In Section 1.1, we introduce the basic CNN architecture. The computational complexity of CNN is discussed in Section 1.2. We provide an outline for the rest of this dissertation in Section 1.4.

## 1.1 Convolutional Neural Networks (CNNs)

CNN is primarily inspired from the Hubel and Wiesel model of mammal's brain [6, 7]. They performed experiments on monkeys and studied their vision system. They

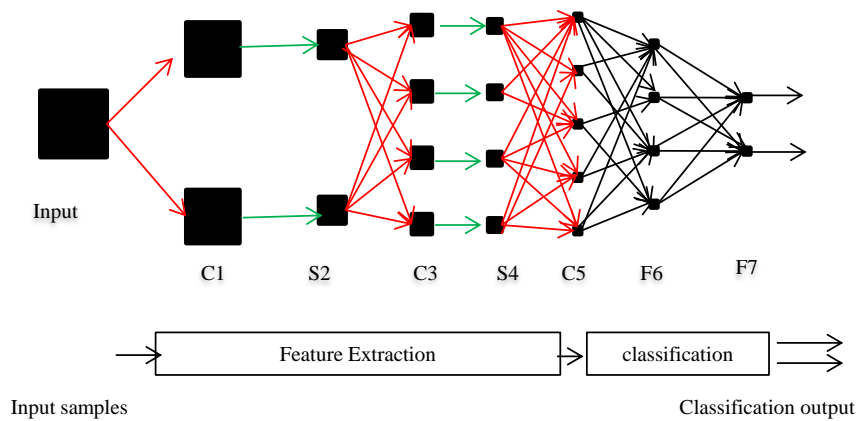


Figure 1.2: Convolutional Neural Network

found that the visual area contains two types of cells: simple cells responsible for feature extraction and complex cells spatially combining these features in a local region. A CNN can be conceptually divided into two parts. The frontal part, closer to the input layer, extracts features, while the rear layers (closer to the output layers) perform classification. The frontal part consists of convolution and pooling layers. The convolution and pooling layers act as feature extractors. The CNN kernels are the feature detectors which are not pre-designed like the Sobel edge detector shown in Fig. 1.1. CNN learns and employs a collection of such feature extractors for classification. Learning features with CNN is generally a better approach than searching for optimal hand designed features as CNN can learn problem specific features from the training data. A sample CNN is shown in Fig. 1.2. A feed forward deep neural network (FFDNN) ignores the spatial structure as features learnt at one place are not useful for detecting the same object at another spatial location. Compared to FFDNN, the learned CNN feature detectors have spatial invariance. The convolution layer consists of one or more feature maps. Weights in a convolution connection



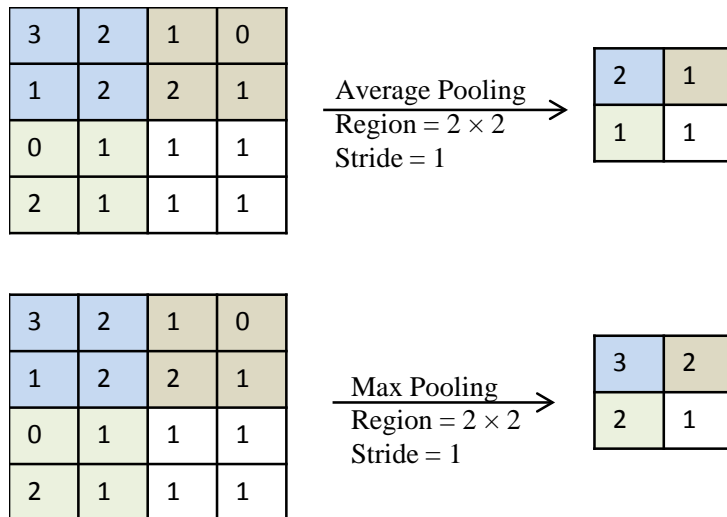


Figure 1.3: Average and max-pooling

are shared and receive inputs from a fixed neighborhood ( $3 \times 3$  etc.). The subsampling layer due to pooling from a  $k \times k$  region enables some degree of invariance to distortions. Each subsampling unit might take inputs from a  $k \times k$  region in the corresponding feature map and compute its average or max value [8]. An example of average and max-pooling is shown in Fig. 1.3. Generally, the pooling receptive fields are non-overlapping but overlapped pooling may perform better.

CNN have been widely used in pattern recognition problems. Yann Lecun et al. developed handwritten digit recognition CNN called LeNet-5 [7]. CNN have been widely used for face recognition [9], pedestrian detection [10] and general object recognition on ImageNet [11] etc. Due to their excellent performance, it is highly desirable to port the learning capabilities of CNN to resource limited portable devices. The next Section, discusses the computational complexity of a CNN.

## 1.2 Computational Complexity of Convolution Layers

In this Section, we discuss the computational complexity of a CNN. As earlier mentioned, a CNN has three types of layers which corresponds to three different connection types. The three computations are convolution, pooling and vector-matrix multiplications. The fully connected layers can be implemented as vector-matrix multiplication where activations from the source layer is arranged in the vector format and the weights constitute a matrix. The pooling layers have one to one connections and conventionally the average or max of a  $2 \times 2$  or  $3 \times 3$  region is computed. Therefore both the pooling and fully connected layers are not the hot-spot for optimization.

The convolution layers have a meshed connectivity pattern and thus are computationally most expensive. In this dissertation, we propose pruning and fixed-point optimization techniques to reduce this complexity. The convolution layer convolves the  $k \times k$  kernels with  $N$  feature maps of the source layer. If the destination layer has  $M$  feature maps, then  $N \times M$  convolutions are performed and  $N \times M \times H \times W \times k \times k$  multiply-accumulate (MAC) operations are needed, where  $W$  and  $H$  represents the feature map width and height of the destination layer, respectively. Thus, the convolution layers demand a large number of arithmetic operations in many cases. Further, the default memory access pattern of convolution layers is not computationally friendly [12]. It is therefore highly desirable to reduce the complexity of convolution layers. In the literature, there have been various works to reduce the computational complexity of convolution layers and we outline them in 3.4 and 4.4. The work of [13] and [14] unrolls convolutions into matrix-matrix multiplications and speeds up the computation by 3-4 times [13]. However, redundant data and kernels storage in-

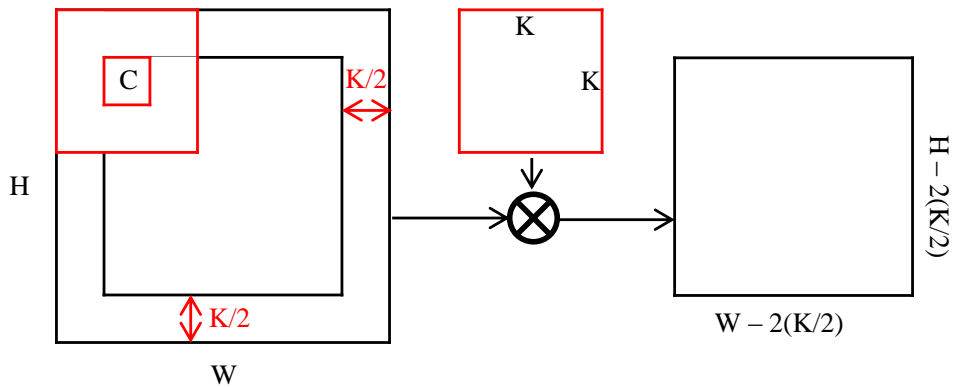


Figure 1.4: MAC operations in a single convolution

incurs its own cost of extra memory usage.

Network pruning is a promising technique to solve this problem. However, pruning usually results in irregular network connections that not only demand extra representation efforts but also do not fit well on parallel computation. We introduce structured sparsity at various scales for convolutional neural networks, which are feature map wise, kernel wise and intra-kernel strided sparsity. This structured sparsity is very advantageous for direct computational resource savings on embedded computers, parallel computing environments, and hardware based systems. It is therefore of prime importance to design high performance low complexity neural networks. This goal can be achieved by applying the pruning technique to high performance large sized networks, where the pruning reduces the computational cost. Further, these lightweight networks can be implemented using only on chip memory for energy savings as frequent DRAM accesses consume much energy. Pruning induces sparsity in a network and can be categorized as structured and unstructured. Unstructured pruning does not follow a specific geometry or constraint. In most cases, this technique needs

extra information to represent sparse locations. It leads to irregular sparsity which is difficult to exploit for efficient computation. On the other hand, structured sparsity places non-zero parameters at well-defined locations. This kind of constraint enables modern CPUs and graphics processing units (GPUs) to easily exploit computational savings. Network pruning has been studied by several researches [2, 3, 15, 16, 17, 18]. The works of Han et al. [2, 3] have shown that a much bigger portion of weights can be set to zero with minimum or no loss in performance. They train a network with an additional L1/L2 loss function on the weights and gradually prune it. If the weight of a connection is less than a threshold, the connection is dropped. The authors in [2] further extend this work by quantizing the finally pruned network [3]. However, both works have to explicitly locate non-zero weights with sparse representation. Conventionally sparse representation uses the compressed sparse row/compressed sparse column (CSR/CSC) format which represents  $m$  non-zero numbers with  $2m + n + 1$  numbers where  $n$  represents the number of rows or columns. The work of [3] shows that half of the AlexNet memory space is required for storing the indices of the non-zero parameters. This also doubles memory accesses as each weight fetch now becomes an *(index, weight)* pair. Our proposed work does not demand such extra representation. Our work proposes to reduce this complexity with structured pruning. We show that the proposed intra-kernel pruning is helpful in reducing the size of matrices in convolution unrolling [13]. Previous related works and discussions on this topic are provided in Chapter 3. This dissertation has the following important contributions.

- For pruning, it is important to select least adversarial pruning candidates. We propose three techniques in Chapter 2 in this regard. We particularly provide detailed discussions on the best of  $N$  random pruning masks.

- We introduce structured pruning at various granularities for maximum pruning benefits. The pruned networks are easily accelerated with very simple sparse representation. Feature map pruning reduces the width of a convolution layer and directly produces a low complexity network. Detailed explanations and experimental analysis is provided in Chapter 3.
- We propose constrained intra-kernel stride sparsity (IKSS) where convolutions are efficiently implemented as matrix-matrix multiplications.
- We propose constraintless kernel level pruning where the sparse representation is very simple. Moreover, the technique is applied in combination with feature map pruning to further improve sparsity ratios.
- The weights and signals of deep neural networks are generally saved in 32-bit floating point precision. We propose fixed-point optimization algorithms for representing weights and signals with 3-8 bits while keeping the same level of performance as the floating point network. The algorithms and details are outlined in Chapter 5.

### 1.3 Publications Record

- Sajid Anwar, Kyuyeon Hwang, and Wonyong Sung. Structured pruning of deep convolutional neural networks. arXiv:1512.08571, 2015. Accepted for publication in the ACM Journal of Emerging Technologies in Computing Systems. URL: DOI: <http://dx.doi.org/10.1145/3005348>
- Coarse Pruning of Convolutional Neural Networks with Random Masks.” [Under review in ICLR 2017]

- Anwar Sajid, Kyuyeon Hwang, and Wonyong Sung. "Fixed-point optimization of deep convolutional neural networks for object recognition." In Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on, pp. 1131-1135. IEEE, 2015.
- Anwar Sajid, Kyuyeon Hwang, and Wonyong Sung. "Learning separable fixed point kernels with deep CNN." In Acoustics, Speech and Signal Processing (ICASSP), 2016 IEEE International Conference on, IEEE, 2016.

## 1.4 Outline of the Dissertation

This dissertation is organized as follows. Chapter 2 describes the background of pruning, pruning granularities, and pruning candidate selection. In Chapter 3, we introduce feature map and intra-kernel pruning. We particularly discuss the proposed intra-kernel strided sparsity and show its relationship with convolution unrolling. We provide extensive experimental results in Section 3.3. Chapter 4 proposes that very high pruning ratios can be achieved when we combine kernel and feature map pruning. The fixed-point optimization algorithms are discussed in Chapter 5. We discuss direct quantization and layer-wise quantization sensitivity analysis. We provide a re-training based algorithm for fixed-point optimization. Finally, Chapter 6 concludes this dissertation.

The material in this dissertation was presented in [19, 5, 20, 21].

# Chapter 2

## Background

This chapter briefly reviews pruning and fixed point optimization. We introduce pruning in Section 2.1, where we discuss the pruning granularities, pruning ratios, and sparse representation. We outline three methods for pruning candidate selection in Section 2.2 along with necessary experimental evaluations. We briefly introduce fixed-point optimization in Section 2.3.

### 2.1 Introduction to Pruning

It is of prime importance to design high performance low complexity neural networks. This goal can be achieved by applying the pruning technique to high performance large sized networks, where the pruning reduces the computational cost. Further, these lightweight networks can be implemented using only on chip memory for energy savings as frequent DRAM accesses consume much energy. Sparsity in a deep convolutional neural network (CNN) can be induced at various levels. Fig-

Figure 2.1 shows four pruning granularities. At the coarsest level, a full hidden layer can be pruned. This is shown with a red colored rectangle in Fig. 2.1(a). Layer wise pruning affects the depth of the network and a deep network can be converted into a shallow network. Increasing the depth improves the network performance and layer-wise pruning therefore demand intelligent techniques to mitigate the performance degradation. The next pruning granularity is removing feature maps [9, 5].

Feature map pruning removes a large number of kernels and is therefore destructive in nature. We therefore may not achieve higher pruning ratios with this granularity. For the depicted architecture in Fig. 2.1 (b), pruning a single feature map removes four kernels. Feature map pruning affects the layer width and we directly obtain a thinner network and no sparse representation is needed. Kernel pruning is the next pruning granularity and it prunes  $k \times k$  kernels. It is neither too fine nor too coarse and is shown in Fig. 2.1(c). Kernel pruning is therefore a balanced choice and it can change the dense kernel connectivity pattern to a sparse one. Each convolution connection involves  $W \times H \times k \times k$  multiply and accumulate (MAC) operations where  $W$ ,  $H$  and  $k$  represents the feature map width, height and the kernel size, respectively. Further the sparse representation for kernel pruning is also very simple. A single flag is enough to represent one convolution connection. The conventional pruning techniques induce sparsity at the finest granularity by zeroing scalar weights. This sparsity can be induced in much higher rates but high pruning ratios do not directly translate into computational speedups in VLSI or parallel computer based implementations [2]. Figure 2.1(d) shows this with red colored zeroes in the kernel. Pruning induces sparsity in a network and can be categorized as structured and unstructured. Unstructured pruning does not follow a specific geometry or constraint. In most cases, this technique needs extra information to represent sparse locations. It leads to irreg-



Inducible pruning ratios Inside allowable budget (increasing from left to right), (e.g. budget = ↓Accuracy)

Pruning granularity (course (left) to fine grained (right))

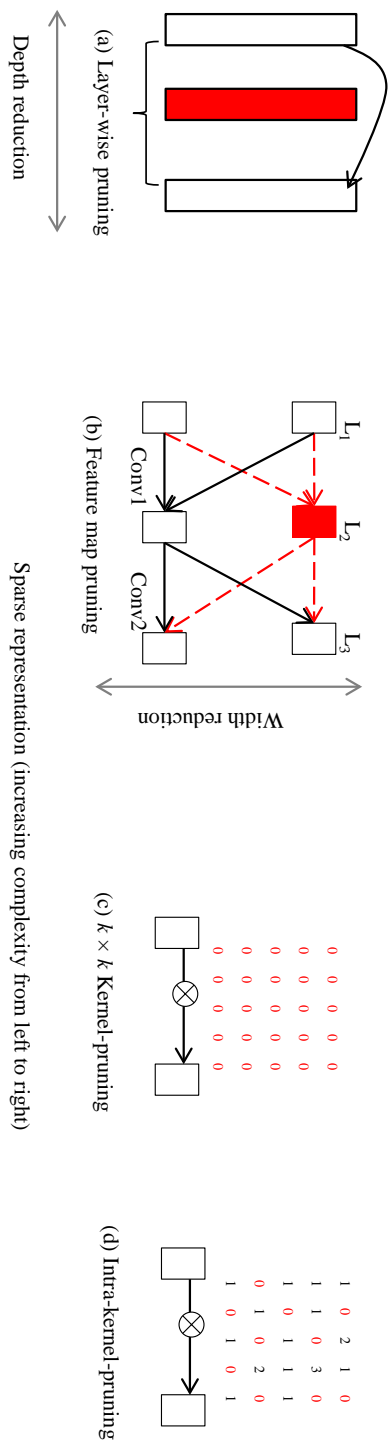


Figure 2.1: Coarse and fine grained pruning granularities

ular sparsity which is difficult to exploit for efficient computation. On the other hand, structured sparsity places non-zero parameters at well-defined locations. This kind of constraint enables modern CPUs and graphics processing units (GPUs) to easily exploit computational savings. Further Fig. 2.1 summarizes the relationship between three related factors: the pruning granularities, the pruning ratios and the sparse representations. Coarse pruning granularities demand very simple sparse representation but higher pruning ratios are comparatively difficult to achieve. Similarly fine grained pruning granularities can achieve higher pruning ratios but the sparse representation is more complicated. The proposed work prunes then network at three granularities: feature map, kernel and intra-kernel. In Chapter 3, we demonstrate feature and intra-kernel strided pruning, while in Chapter 4, we propose pruning full  $k \times k$  kernels. We achieve best pruning results when we prune a network with combined granularities.

## 2.2 Pruning Candidate Selection

The learning capability of a network is determined by its architecture and the number of effective learnable parameters. Pruning reduces this number and inevitably degrades the classification performance. The pruning candidate selection is therefore of prime importance. It is important to select good pruning masks as some of the masks may be less adversarial than others. Further the pruned network is retrained to compensate for the pruning losses [1]. For a specific pruning ratio, we search for the best pruning masks which afflicts the least adversary on the pruned network. Indeed retraining can partially or fully recover the pruning losses, but the lesser the losses, the more plausible is the recovery [22]. Further small performance degradation also means that the successor student network has lost little or no knowledge of the prede-

cessor teacher network. If there are  $M$  potential pruning candidates, the total number of pruning masks is  $(2^M)$  and an exhaustive search is therefore infeasible even for a small sized network. We therefore propose a simple strategy for selecting pruning candidates. We outline here particle filter, activation sum voting, and best of  $N$  random masks. All the schemes are elaborated and discussed in this Chapter.

### 2.2.1 Evolutionary Particle Filter

The pruning process needs to select less important connection combinations. These connections, when pruned, have least adversary on the network performance, which can be compensated with retraining. Note that considering all the pruning patterns is too complex in most cases. In this work, we propose to locate pruning candidate connections with the sequential Monte Carlo (SMC) method also known as particle filters [23]. Particle (PF) filtering finds its applications in several fields [24, 25, 26, 27]. With a set of weighted particles, the PF represents the filtering distribution [28]. Particle filter is usually applied to the system model shown in 2.1 and 2.2.

$$\mathbf{x}_k = \mathbf{f}(\mathbf{x}_{k-1}) + \mu_k \quad (2.1)$$

$$\mathbf{z}_k = \mathbf{h}(\mathbf{x}_k) + \mathbf{v}_k, \quad (2.2)$$

where  $k$  shows the time step,  $\mathbf{z}$  represents the observation vector,  $\mathbf{v}$  denotes the observation noise and  $\mu$  is the process noise. The state vector is represented by  $\mathbf{x}$ . Figure 2.2 (a) shows the dotted matrices as the pruning masks for weights between two layers. The  $\mathbf{x}$  state vector represents the inverse of the pruning mask. Non-existing entries in the state vector are pruned. In Figure 2.2 (b), one example of the state vector

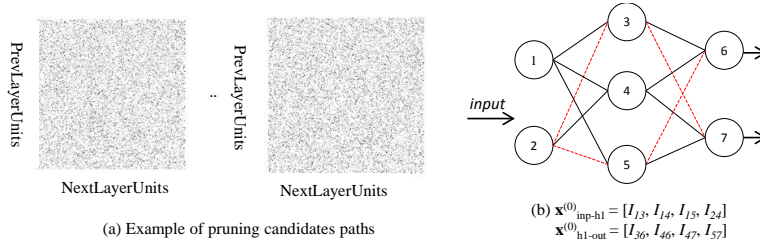


Figure 2.2: Example of state vector for particle filter

is provided. The circles represent the neurons while  $I_{ij}$  shows the index of the weight going from neuron  $i$  to  $j$ . The red dotted connections are pruned while the black solid lines survive through the pruning process. The observation function is represented by  $h()$  whereas  $f()$  represents the transition function. When pruning  $n$  connections, the possible combinations are on the order of  $\mathcal{O}(2^n)$ , which means that exhaustive search is not feasible in most cases. With  $N$  particles, we simulate several possible connection combinations. The state vector contains a set of possible connections through which the input are forward propagated. Thus the likelihood computation is simulated with pruning masks. The trained network is used as the observation function which is noisy as the classification error rate is greater than 0%. Thus each particle simulates a set of connections as the possible candidate for the most likely path through the network. The likelihood to each particle is assigned based on the misclassification rate (MCR) on the validation set. This way importance weight is computed by  $1 - MCR$  and shows the likelihood of keeping. Thus connections with high importance survive while the rest are pruned. The MCR likelihood criterion compares the network assigned label with the true one which, through iterative pruning, guides the network learn the true labels. Once all particles are assigned probabilities, then we construct the cumulative distribution function (CDF) and resample with the se-

quential importance resampling (SIR) [24]. The transition function  $f()$  is simulated by perturbing the pruning mask.

Due to the finite number of samples, PF suffers from degeneracy and impoverishment problems when the highly likely particles replace the less likely ones [29]. Kwok *et al.* suggested the evolutionary particle filter (EPF) and proposed a hybrid approach where genetic algorithm (GA) is combined with PF to solve these problems. The authors argue that particles are similar to chromosomes and survival of the fittest has equivalence to the resampling algorithm [30]. With the hybrid approach, the aim is to increase the fitness of the whole population. An important merit of the GA augmented approach is that it not only maintains the highly likely genes in its chromosome but also re-defines particles in the less likely region [30]. EPF reduces computational cost as it requires fewer particles than conventional particle filters [30]. This property suits us as it will reduce the cost of finding pruning candidates. We also believe that SMC techniques have more potential usages in exploring the network parameters space. One usage can be finding a class specific routes in a deep neural network.

Considering the case of feature map pruning, we explain here the EPF approach candidate selection with  $N$  particles. We represent each particle with  $P[i]$  and the state vector as  $\mathbf{x}^{(i)}$ . Suppose that we are pruning a layer  $L$  with  $F$  non-pruned feature maps. Let's further suppose that the current pruning rate is denoted with  $pr$ . We compute the dimension  $d$  of the state vector with  $d = F \times (1 - pr)$ . In order to make the likelihood computation connections count invariant, the state vectors of all the particles have the same dimension  $d$  for a given value of  $pr$ . Let's consider a simple example where  $F$  is 10, the pruning rate is 25% and  $N$  is 3. The  $d$  in this case will be  $\lfloor 10 \times (1 - 0.25) \rfloor = 7$ . The state vectors  $\mathbf{x}^{(i)}$  for each particle is randomly assigned and may be as follows:

$\mathbf{x}^{(1)} = [10\ 2\ 9\ 7\ 1\ 3\ 6]$ ,  $\mathbf{x}^{(2)} = [10\ 3\ 2\ 7\ 4\ 5\ 9]$ ,  $\mathbf{x}^{(3)} = [2\ 5\ 1\ 8\ 6\ 7\ 3]$ . Each particle is only allowed to compute the likelihood with the set of feature maps listed in its state vector. Other feature maps not included in the set are zeroed. For example, for the particle  $P[3]$ , the pruning mask is  $[1\ 1\ 1\ 0\ 1\ 1\ 1\ 1\ 0\ 0]$  where the feature maps  $[4\ 9\ 10]$  are zeroed. This way likelihood is computed for each particle. These  $N$  particles then undergo the GA based SIR as explained in Section 2.3. This way EPF helps in reducing the adversarial effect of pruning.

There can be a concern regarding the computational cost of this approach. There are two reasons that we outline here which mitigates this concern. Likelihood computation only uses the feed forward computation with the validation set. The validation set is 10-20% of the training set and is thus not very large. Secondly the hybrid approach (genetic algorithm + particle filter) in particle filtering enables us to select the pruning candidate with few particles. In all our experiments we used not more than 100 particles. The next subsection introduces a simple pruning candidate selection scheme and compares it with the EPF based method.

### 2.2.2 Activation Sum Voting

The activation sum voting and its comparison with the PF based method is shown here. Suppose there are  $F$  feature maps with  $w$ ,  $h$  dimensions and rectified linear unit as the activation function. We compute the sum for each feature map across the whole mini-batch of size  $n$  on the validation set. Experimental results are shown in the plot on the right side of the figure with the MNIST [7] dataset. The network architecture is :  $1 \times 16(C5) - MP2 - 32(C5) - MP2 - 64(C5) - 120 - 10$ . It can be observed that for similar pruning ratios, the particle filter based method selects less adversarial pruning candidates than the simple method. We propose a simple and low

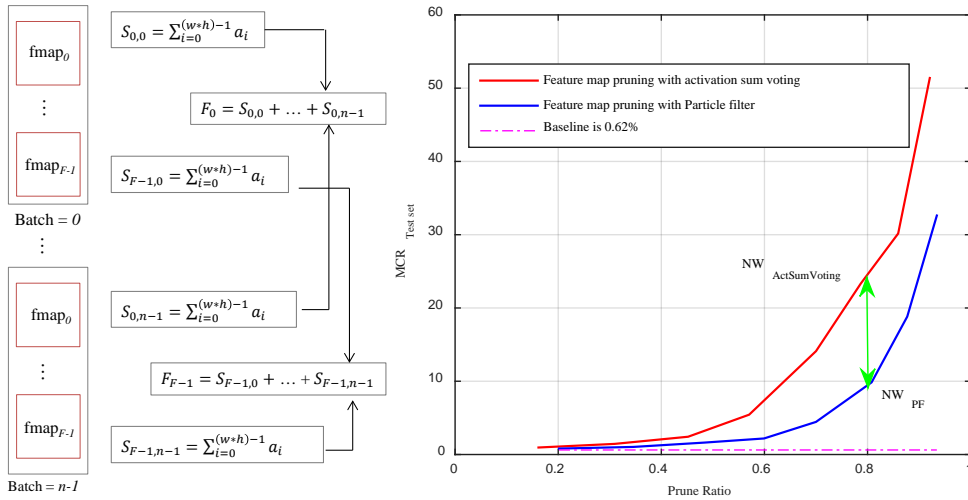


Figure 2.3: Activation sum voting

complexity scheme called the activation sum voting for feature map pruning. This criterion is inspired from max pooling where the magnitude of a neuronal output determines its importance. To the best of our knowledge this method has not been reported earlier and is depicted in Fig. 2.3. Each feature map is summed across the whole mini-batch on the validation set and the minimum summation feature maps are pruned. Experimental evaluation for the two methods are reported for the MNIST dataset [7] in Fig. 2.3. The network is first trained to the baseline MCR of 0.69%. We then select pruning candidates for various pruning ratios. This plot does not consider the effect of retraining the pruned networks. The  $NW_{PF}$  and  $NW_{ActSumVoting}$  shows the pruned network obtained with PF and the activation sum voting method respectively. The pruning plot shows that the particle filter based method performs better especially at higher pruning ratios by selecting less adversarial pruning candidates. Further in the plot, a green arrow shows the difference between the classification performance of  $NW_{PF}$  and  $NW_{ActSumVoting}$ . Network retraining may recover some of the pruning

Table 2.1: Specifications of the three networks

Network	Architecture	Baseline MCR(%)	Data Augmentation
$CNN_{MNIST1}$	$16(C5) - 32(C5) - 64(C5) - 120 - 10$	0.62	NO
$CNN_{MNIST2}$	$6(C5) - 16(C5) - 120(C5) - 84 - 10$	0.79	NO
$CNN_{CIFAR10.small}$	$2 \times 128C3 - MP2 - 2 \times 128C3 - MP2 - 2 \times 256C3 - 256FC - 10Softmax$	16.6	NO
$CNN_{CIFAR10.large}$	$2 \times 128C3 - MP2 - 2 \times 256C3 - MP2 - 2 \times 256C3 - 1 \times 512C3 - 1024FC - 1024FC - 10Softmax$	9.41	YES
$CNN_{SVHN}$	$(2 \times 64C3) - MP2 - (2 \times 128C3) - MP2 - (2 \times 128C3) - 512FC - 512FC - 10Softmax$	3.5	NO

losses. However,  $NW_{PF}$  starts better than the  $NW_{ActSumVoting}$ . This way we show that the higher computational complexity of the PF based method is justified as it selects better pruning candidates.

### 2.2.3 Absolute Weight Sum Voting

Figure 2.4 explains the idea presented in [31] and shows three layers,  $L1$ ,  $L2$  and  $L3$ . All the filters/kernels from previous layer to a feature map constitute one group which is shown with similar color. The  $S1, S2$  and  $S3$  is computed by summing the absolute value of all the weights in this group. In the next sectionSection ??, the comparison of the proposed method with the absolute weight sum method is shown for two networks.

### 2.2.4 Best of $N$ Random Masks

In this Section, we outline one of our main contributions regarding pruning candidate selection. We have proposed a simple strategy for the selection of pruning masks. Generally, in the literature granularity specific pruning strategies are reported [2, 31]. In the previous Section, we demonstrated a guided multi-step pruning method using a particle filtering approach. This method selects the best pruning mask through  $N$  random pruning evaluations. This approach enables one to select pruning mask in one step and is simpler than the multi-step technique. Our proposed algorithm generalizes



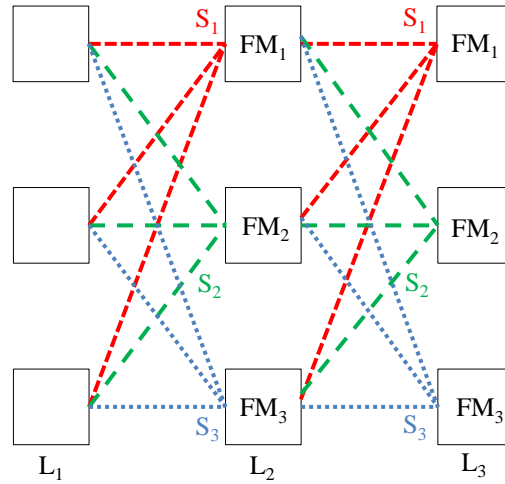


Figure 2.4: Absolute weight sum criterion

well and can select near-optimal candidates for feature map, kernel and intra-kernel pruning. Further, the proposed approach is not computationally expensive as it involves  $N$  random evaluations on the small sized validation set.

We evaluate  $N$  random combinations and compute the MCR for each one. We then choose the best pruning mask which causes the least degradation to the network performance on the validation set. However, this further raises the question of how to approximate  $N$ . We report the relationship between pruning ratio and  $N$  in Fig. 2.5 and 2.7. This analysis is conducted for feature map pruning but is also applicable to other pruning granularities. From Fig. 2.5, we can observe that for higher pruning ratios, high value of  $N$  is beneficial as it results in better pruning candidate selection. For the pruning ratio of no more than 40%,  $N = 50$  random evaluations generate good selections. For lower pruning ratios, retraining is also more likely to compensate the losses as the non-pruned parameters may still be in good numbers. The computational cost of this technique is not much as the evaluation is conducted on the small sized

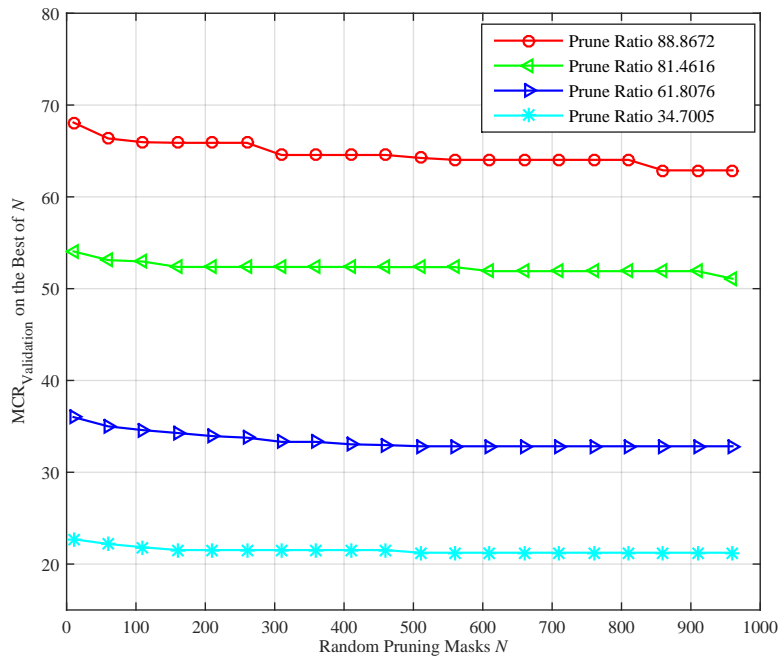


Figure 2.5: MCR with the best of  $N$  random masks for  $CNN_{CIFAR-10}$

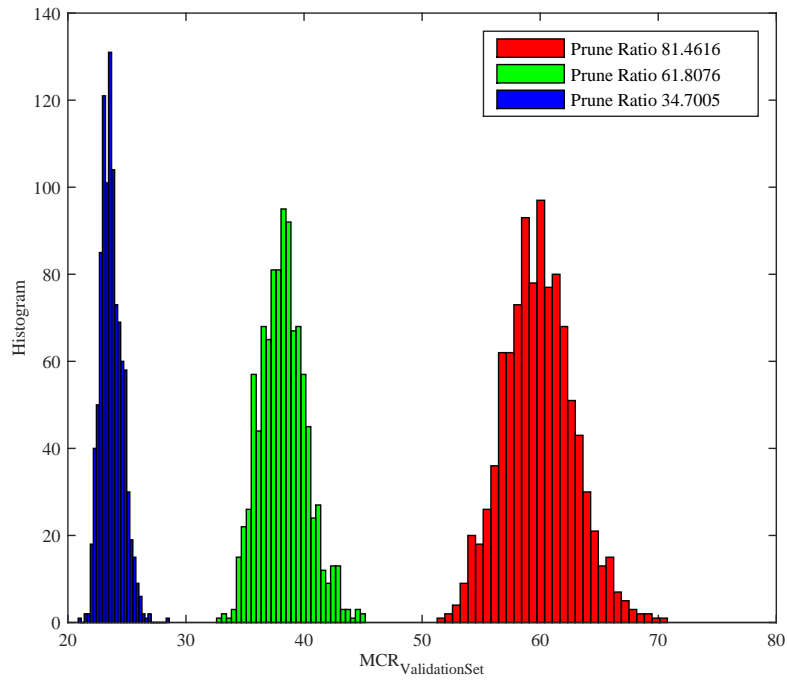


Figure 2.6: Distribution of  $N$  random evaluations for  $CNN_{CIFAR-10}$

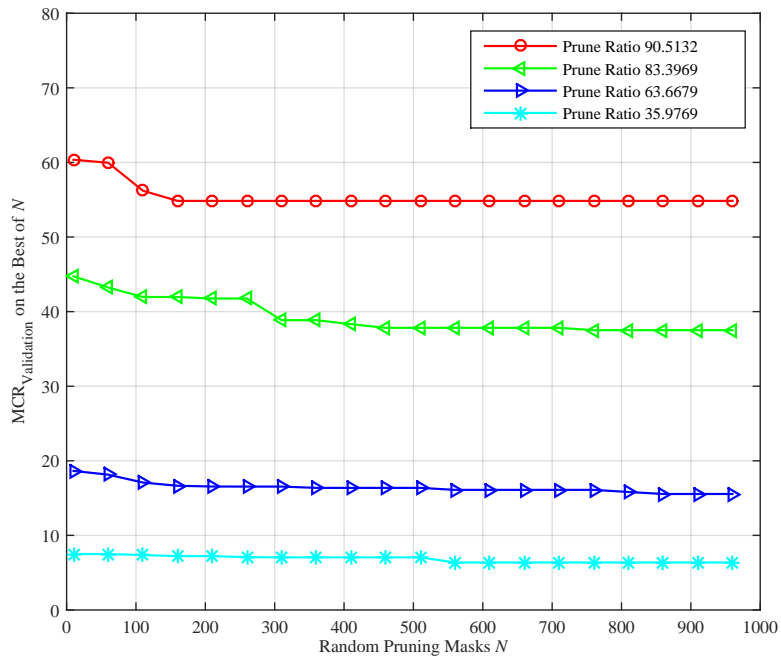


Figure 2.7: MCR with the best of  $N$  random masks for  $CNN_{SVHN}$

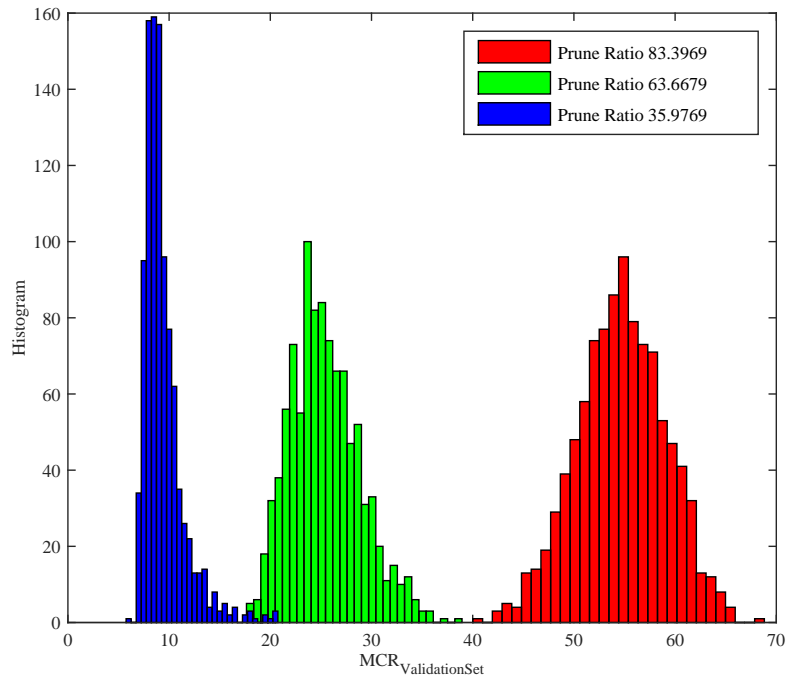


Figure 2.8: Distribution of  $N$  random evaluations for  $CNN_{SVHN}$

validation set. By observing Fig. 2.5 and 2.7, we propose that the value of  $N$  can be estimated initially and later used in several pruning passes. Consider that for the depicted architecture in Fig.2.4, we need to select feature map pruning candidates in layer  $L_2$  and  $L_3$  with  $1/3$  pruning ratio. If  $N = 4$ , the following  $N$  ordered pairs of feature maps may be randomly selected for  $(L_2, L_3) : (1, 2), (2, 3), (3, 1),$  and  $(1, 1)$ . These combinations generate random paths in the network and we evaluate the validation set MCR through these routes in the network.

We further explain and compare this method with the weight sum criterion proposed in [31] and shown in Fig. 2.4. The set of filters or kernels from the previous layer constitute a group. This is shown with the similar color in Fig. 2.4. According to [31], the absolute sum of weights determines the importance of a feature map. Suppose that in Fig.2.4, the Layer  $L_2$  undergoes feature map pruning. The weight sum criterion computes the absolute weight sum at  $S_1, S_2$  and  $S_3$ . If we further suppose that the pruning ratio is  $1/3$ , then the  $\min(S_1, S_2, S_3)$  is pruned. All the incoming and outgoing kernels from the pruned feature map are also removed. We argue that the sign of a weight in kernel plays important role in well-known feature extractors and therefore ignoring it may not be a good criterion. The network reported in Table 2.1 as  $CNN_{SVHN}$  is pruned to generate the pre-retraining plots in 2.10. Figure 2.7 compares the best candidate selected out of  $N$  random combinations for various feature map pruning ratios.

We compare the performance of the two algorithms and Fig. 2.9 and 2.10 shows the experimental results. These results present the network status before any retraining is conducted. We report the performance degradation in the network classification against the pruning ratio. From Fig. 2.10 and 2.9, we can observe that our proposed method outperforms the weight sum method particularly for higher pruning

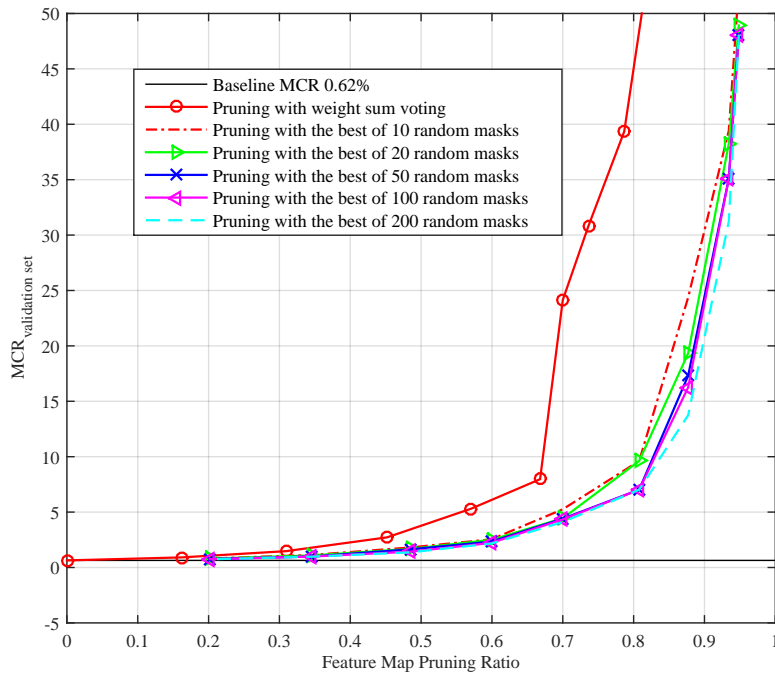


Figure 2.9: Weight sum vs. best of  $N$  random masks for  $CNN_{MNIST}$

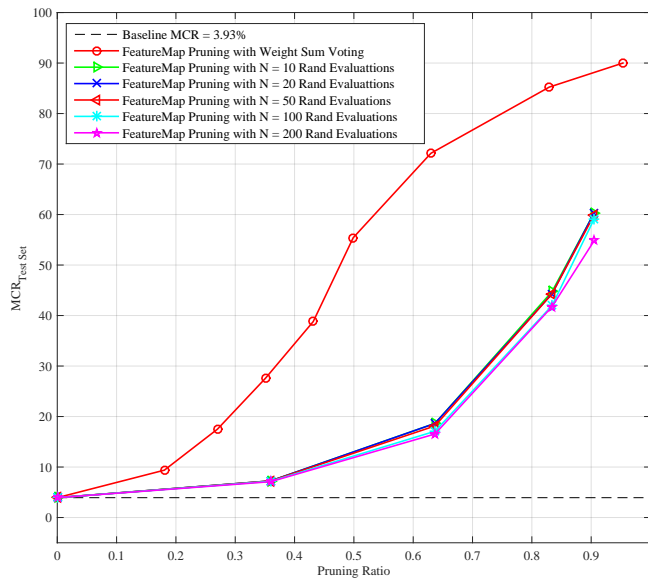


Figure 2.10: Weight sum vs. best of  $N$  random masks for  $CNN_{SVHN}$

ratios. The *best of N Pruning masks* strategy evaluates pruning candidates in combinations and provides a holistic view. The criterion in [31] evaluates the importance of a pruning unit in the context of a single layer while our proposed approach evaluates several paths through the network and selects the best one. The combinations work together and matter more instead of individual units. Further, our proposed technique is generic and can be used for any pruning granularity: feature map, kernel and intra-kernel pruning.

### 2.2.5 The Effect of Retraining on Pruning Masks

We further analyze the effect of retraining on the pruning mask selection. We prune a network with several masks and retrain each pruned network. As several networks need to be pruned and retrained many times, we experiment with a small network where the architecture is reported like this:  $32(C5) - MP2 - 64(C5) - MP2 - 64(C5) - 64FC - 10Softmax$ . The network is trained with the CIFAR-10 dataset (40,000 training samples) without any data augmentation and batch normalization. The network achieves the baseline performance of 26.7% on the test set. The results are reported in Fig. 2.11, where the pre and post-retraining network performance is shown on the  $x$  and  $y$  axis, respectively. Further, we superimpose a least-squares (LS) line fit to each of the scatter plot. It can be observed that the slope of the LS line decreases for higher pruning ratios. We infer that for high pruning ratios, the final network performance is dictated by the surviving number of effective parameters. It can be observed that the overall distribution is noisy. However, in general, the pre-retraining least adversarial pruning masks perform better after retraining. In the rest of this work, we therefore use the pre-retraining best mask for pruning the network.

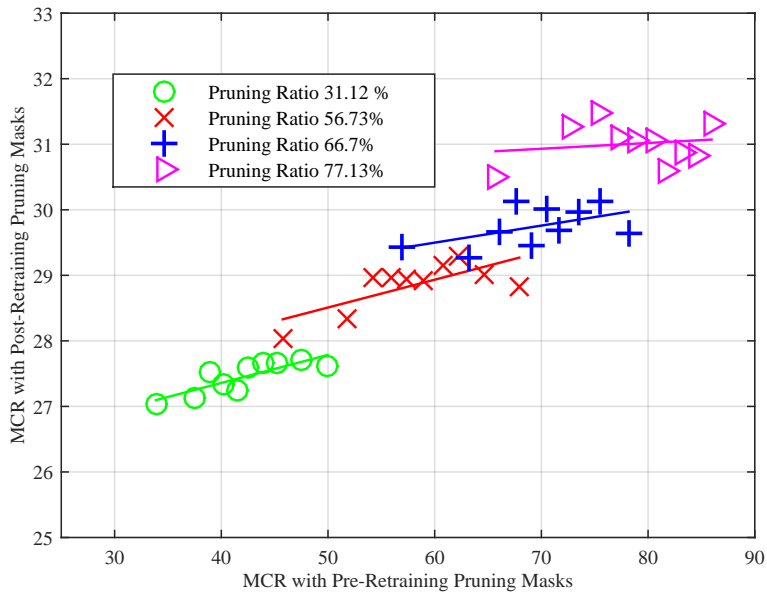


Figure 2.11: In this plot, we prune a CNN network with various masks and compare their pre and post retraining performance. It can be observed that on the average, pre-retraining masks perform better after retraining.

## 2.3 Fixed-Point Optimization

The proposed work lowers the hardware complexity by constraining the learned convolutional kernels to be separable and also reducing the word-length of these kernels and other weights in the fully connected layers. To compensate for the effect of direct quantization, a retraining scheme that includes filter separation and quantization inside of the adaptation procedure is developed in this work. The filter separation reduces the number of parameters and arithmetic operations by 60% for a 5x5 kernel, and the quantization further lowers the precision of storage and arithmetic by more than 80 to 90% when compared to a floating-point algorithm. Experimental results on MNIST and CIFAR-10 datasets are presented. Components in a digital sig-

nal processing system may exhibit varying level of sensitivity to quantization noise. Therefore we use different quantization bits for each layer in accordance to sensitivity analysis [10] [13]. The results drawn in [10] shows that the rear layers of CNN are comparatively more sensitive to quantization noise. Each convolution kernel is treated independently and has its own quantization step size. However each fully connected layer has one quantization step size. Biases are kept in high precision. Convolution layers that incur higher computational complexity are hot spots for optimizations. The separability constraint greatly reduces the computational cost and memory requirement. The constraint-less initial learning with floating point weights enables the network to achieve the baseline performance while the late introduction of separability constraint and re-training ensures resource efficiency. The fixed-point optimization further reduces the word-length and results in power efficiency which may be crucial for embedded systems. Our technique is generic and is useful for both software and hardware implementations. We will discuss in detail in Chapter 5 about fixed-point optimization.



## Chapter 3

# Structured Pruning

Real time application of deep learning algorithms is often hindered by high computational complexity and frequent memory accesses. Network pruning is a promising technique to solve this problem. However, pruning usually results in irregular network connections that not only demand extra representation efforts but also do not fit well on parallel computation. We introduce structured sparsity at various scales for convolutional neural networks, which are feature map wise, kernel wise and intra-kernel strided sparsity. This structured sparsity is very advantageous for direct computational resource savings on embedded computers, parallel computing environments, and hardware based systems. To decide the importance of network connections and paths, the proposed method uses a particle filtering approach. The importance weight of each particle is assigned by assessing the misclassification rate with a corresponding connectivity pattern. The pruned network is retrained to compensate for the losses due to pruning. While implementing convolutions as matrix products, we particularly show that intra-kernel strided sparsity with a simple con-

straint can significantly reduce the size of the kernel and feature map tensors. The proposed work shows that when pruning granularities are applied in combinations, we can prune the CIFAR-10 network by more than 70% with less than 1% loss in accuracy.

### **3.1 Introduction**

Large sized deep convolutional neural networks (CNN) have been successfully applied to diverse classification problems including speech and image recognition [32, 11, 33]. These networks can produce state of the art results at a high computational cost. For resource limited machines and real time applications, it is important to learn the unknown function with a reduced complexity network. Large networks are capable of learning complicated problems but may overfit on the training set. On the other hand, small networks that demand low computational cost usually have limited learning capabilities. It is therefore of prime importance to design high performance low complexity neural networks. This goal can be achieved by applying the pruning technique to high performance large sized networks, where the pruning reduces the computational cost. Further, these lightweight networks can be implemented using only on chip memory for energy savings as frequent DRAM accesses consume much energy. Pruning induces sparsity in a network and can be categorized as structured and unstructured. Unstructured pruning does not follow a specific geometry or constraint. In most cases, this technique needs extra information to represent sparse locations. It leads to irregular sparsity which is difficult to exploit for efficient computation. On the other hand, structured sparsity places non-zero parameters at well-defined locations. This kind of constraint enables modern CPUs and graphics

processing units (GPUs) to easily exploit computational savings.

Network pruning has been studied by several researches [2, 3, 15, 16, 17, 18]. The works of Han et al. [2, 3] have shown that a much bigger portion of weights can be set to zero with minimum or no loss in performance. They train a network with an additional L1/L2 loss function on the weights and gradually prune it. If the weight of a connection is less than a threshold, the connection is dropped. The authors in [2] further extend this work by quantizing the finally pruned network [3]. However, both works have to explicitly locate non-zero weights with sparse representation. Conventionally sparse representation uses the compressed sparse row/compressed sparse column (CSR/CSC) format which represents  $m$  non-zero numbers with  $2m + n + 1$  numbers where  $n$  represents the number of rows or columns. The work of [3] shows that half of the AlexNet memory space is required for storing the indices of the non-zero parameters. This also doubles memory accesses as each weight fetch now becomes an *(index, weight)* pair. Our proposed work does not demand such extra representation.

In this work, we explore feature map and intra-kernel sparsities as a means of structured pruning. In the feature map pruning, all the incoming and outgoing weights to/from a feature map are pruned. The intra-kernel sparsity prunes weights in a kernel. The kernel level pruning is a special case of intra-kernel sparsity with 100% pruning. These pruning granularities can be applied in various combinations and different orders. The proposed work introduces structured pruning at various granularities for maximum pruning benefits. Feature map pruning reduces the width of a convolution layer and directly produces a low complexity network. We propose intra-kernel stride sparsity (IKSS) which reduces the size of matrices in convolution unrolling [13, 14].

The rest of the paper is organized as follows. Section 3.2 briefly introduces CNN and discusses the pruning granularities. Experimental results are discussed in Section

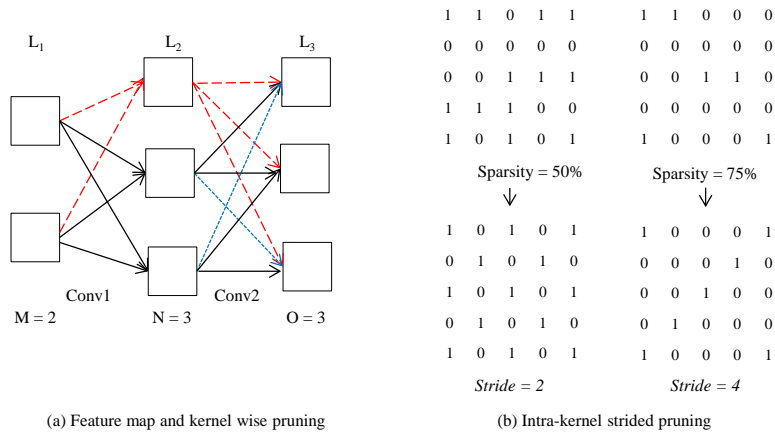


Figure 3.1: Intra-kernel strided sparsity

3.3 for each pruning granularity and their combinations. Section 3.4 compares this work with the related works while Section 3.5 discusses the future research directions and concludes this article.

## 3.2 Feature Map and Intra-Kernel Pruning

In this section, the CNN network is briefly introduced in the context of pruning. The network can be pruned at various granularities which are outlined here. Further, we present the proposed intra-kernel structured (IKSS) sparsity and its impact on reducing the dimensions of matrices in convolution unrolling.

### 3.2.1 Intra Kernel Strided Sparsity

Taking computational advantages using randomly scattered unstructured sparsity in a network is very difficult. It demands many conditional operations and extra representation to denote the indices of zero or non-zero parameters. In this subsection,

we describe three levels of structured pruning, which are feature map, kernel and intra-kernel strided sparsity (IKSS) pruning. Generally, the convolution layers of a non-pruned network employ fully connected convolution connections. Figure 3.1 depicts the pruning granularities and intra-kernel strided sparsity. The red dashed line shows feature map pruning. When we prune all the incoming kernels to a feature map, all the outgoing kernels are also pruned. The blue dotted line depicts pruning  $k \times k$  kernels. Figure 3.1(b) shows intra-kernel sparsity for both structured and unstructured cases. Kernel level pruning (blue dotted) is a special case of intra-kernel pruning, when the kernel level sparsity rate is 100%. In Fig. 3.1(a), the layers  $L1$ ,  $L2$  and  $L3$  contain 2, 3, and 3 feature maps, respectively. The number of convolution connections between  $L1$  and  $L2$  is  $2 \times 3 = 6$  and that between  $L2$  and  $L3$  is  $3 \times 3 = 9$ . Each feature map in  $L2$  has a  $k \times k$  convolution connection from each feature map in  $L1$ . Thus, the pruning exploiting the largest granularity is deleting one or more feature maps. The next level pruning is intra-kernel pruning where each kernel represents one whole convolution. The kernel level pruning is depicted with blue dotted lines in Fig. 3.1(a). Figure 3.1(b) shows examples of intra-kernel sparsity for both the structured and unstructured cases. Kernel level pruning (blue dotted) is a special case of intra-kernel pruning, when the sparsity ratio is 100%.

The finest pruning granularity is the intra-kernel sparsity, which forces some weights into zero. In the previous works, the intra-kernel level pruning is usually conducted by zeroing small valued weights [2, 3]. We particularly explore the intra-kernel level pruning using the sparsity at well-defined locations, which is called the intra-kernel strided sparsity (IKSS). Figure 3.1 (b) depicts this idea. The starting index (offset) for the first non-zero element is randomly assigned in the range of  $[0, stride - 1]$ . Therefore the IKSS associates an offset as the starting index and the

stride size with each kernel. The stride size is also randomly sampled and depends on how well pruning proceeds. In our experiments, we start with a stride size of two. If, during the gradual iterative pruning, the pruned network has similar or better performance when compared with the unpruned network, we use bigger strides and prune the connections more aggressively (use higher pruning ratios). We monitor how well the pruning proceeds after each prune-retrain step. If the last pruning increases the misclassification rate (MCR) by more than a small tolerance number, then we decrease the intra-kernel pruning rate and stride size. In normal pruning mode, majority of the kernels use small strides. The evolutionary particle filter evaluates several possible combinations of the offset, stride, and feature maps and selects the best combination based on likelihood. This way we induce sparsities in the network. The retraining procedure uses the finally selected pruning mask. We found this mask with an evolutionary approach and the validation dataset.

The computational complexity of a network is mainly decided by its depth, width and connectivity pattern. The proposed work does not affect the depth of the network. However, both IKSS and feature map alter the connectivity pattern. Further, feature map pruning directly reduces the width of a convolution layer. Therefore, it is quite straightforward to exploit the computational savings of this technique. This lighter network can then be implemented with convolution unrolling [13], FFTs [4] or conventional ways. The IKSS can further increase the pruning ratio. As earlier mentioned, the IKSS associates an offset and a stride with each kernel. These two numbers are enough to locate the non-zero weights. We further show that if we constrain each outgoing convolution connection from a source feature map to have similar stride and offset, it will result in only two extra parameters (*offset, stride*) per feature map. This way the IKSS can be computationally exploited to avoid MAC

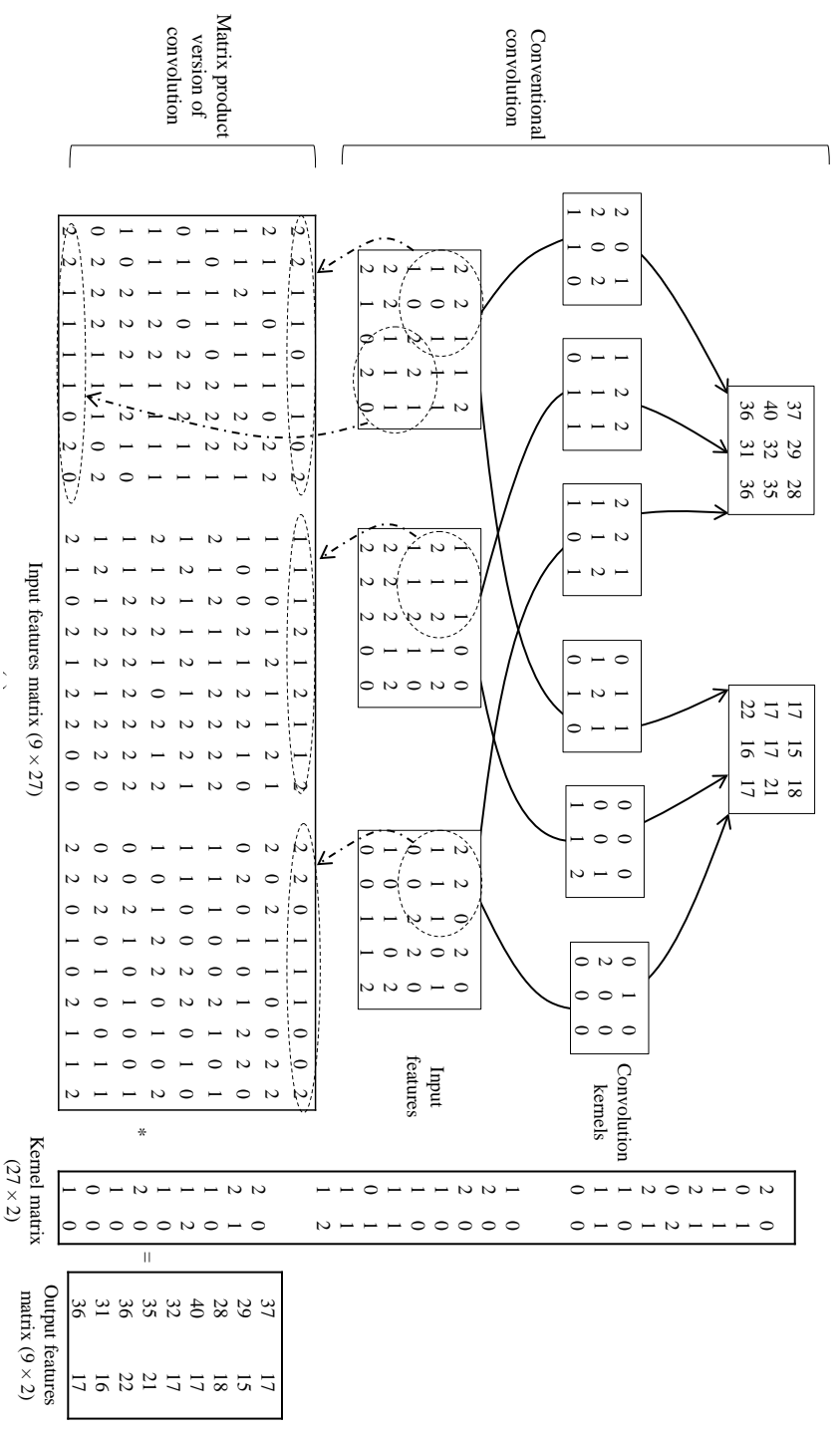


Figure 3.2: Convolution unrolling

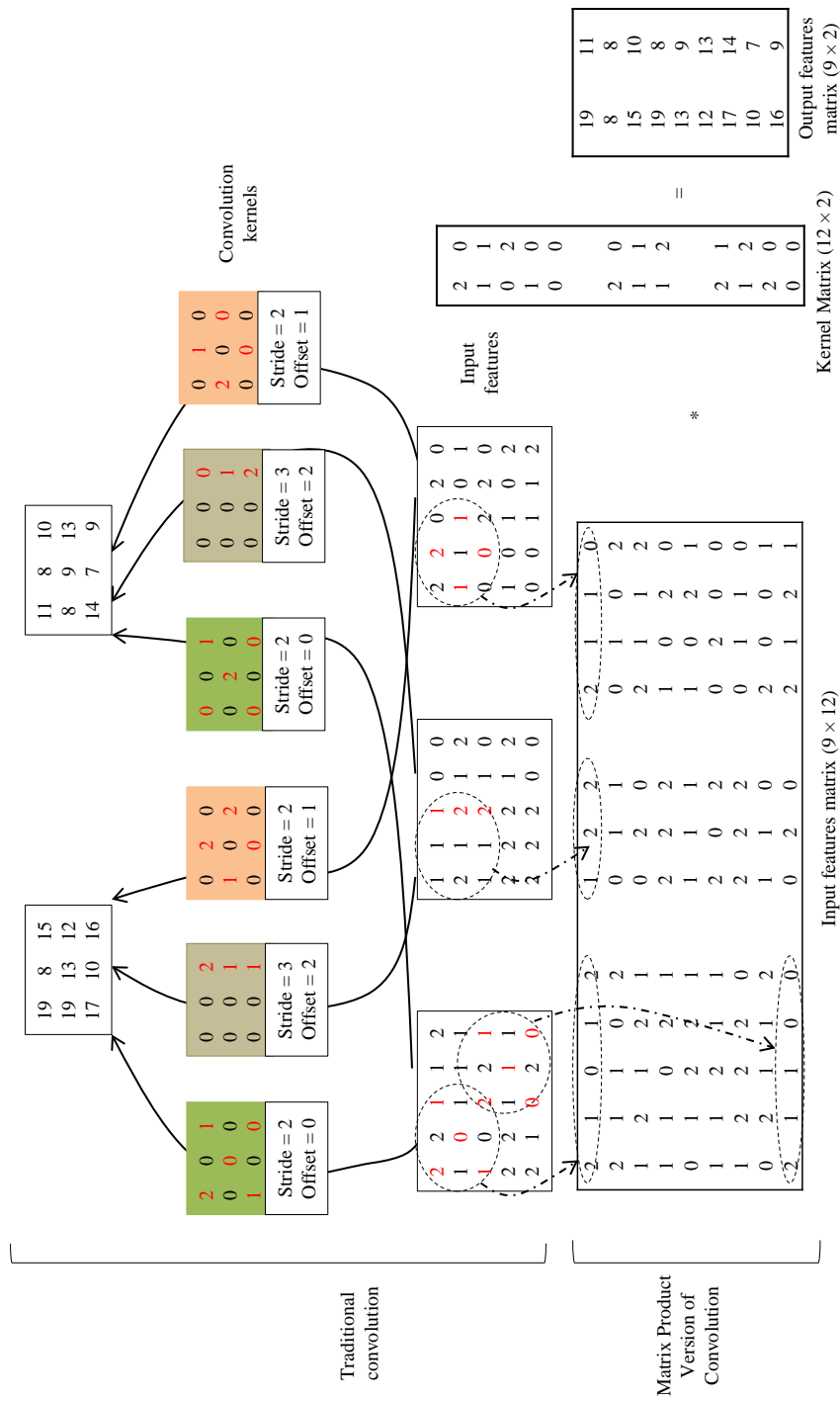


Figure 3.3: Convolution unrolling and IKSS



Table 3.1: Specifications of the networks

Network	Architecture	Baseline MCR(%)
$CNN_{small}$	$32C5-MP2-32C5-AP2-64C5-10Softmax$	25.71
$CNN_{large}$	$2 \times 128C3-MP2-2 \times 128C3-MP2-2 \times 256C3-256FC-10Softmax$	16.6
$CNN_{verylarge}$	$2 \times 128C3-MP2-2 \times 256C3-MP2-2 \times 256C3-512C3-1024FC-1024FC-10Softmax$	9.41

operations with zeroed weights. The convolution layer can be computed as matrix-matrix multiplication by unrolling the source layer feature maps and kernels [13, 14]. Figure 3.2 explains this idea with an example in much detail. Figure 3.3 shows how the proposed IKSS and a simple constraint can reduce the size of feature and kernel matrices. The constraint bounds each outgoing convolution kernel from the source feature maps to have the same stride and offset. The constraint is shown with similar background colors for kernels. Our proposed idea constrains each outgoing convolution connection for a source feature map to have the same stride and offset. The offset shows the index of the first non-zeroed weight. The constraint is shown with the similar colored background squares. This significantly reduces the size of both features matrix and kernel matrix. The first 9 columns in row 1 of the input feature matrix changes from  $[2 \underline{2} \ 1 \ \underline{1} \ 0 \ \underline{1} \ 1 \ 0 \ \underline{2}]$  to  $[2 \ 1 \ 0 \ 1 \ 2]$  with the underlined elements pruned. Only the red colored elements in the feature maps and kernels survive and the rest are pruned. For this example, the size of feature matrix is reduced from  $9 \times 27$  to  $9 \times 12$  and the kernel matrix size is reduced from  $27 \times 2$  to  $12 \times 2$ . These pruning granularities have the potential to bridge the gap between pruning and its computational advantages.

### 3.3 Experimental Results

The pruning process starts with a pre-trained network. The EPF guided pruning process may degrade the network performance which is compensated by retraining. Retraining is important to keep the network performance closer to the non-pruned network. In the spirit of [2], we train the network with L2 regularization in the absence of batch normalization [34]. We consider the pruning limit for each layer depending on its parameter count and learning capacity. Usually, the first convolution layer has fewer parameters than the following layers. Secondly it directly operates on the input layer and is therefore more sensitive to pruning [2]. We present experimental results with the CIFAR-10 and SVHN datasets [35], [36]. The CIFAR-10 dataset consists of a ten class classification problem [35]. The dataset includes samples from ten classes: airplane, automobile, bird, cat, deer, dog, frog, horse, ship and truck. The training set consists of 50,000 RGB samples. The test set contains 10,000 samples. Each sample has  $32 \times 32$  resolution. During training and pruning, we use the stochastic gradient descent (SGD) with a mini-batch size of 128 and RMSProp [37].

We present experimental results with three networks on the CIFAR-10 dataset:  $CNN_{small}$ ,  $CNN_{large}$  and  $CNN_{verylarge}$ . Details about these three networks are provided in Table 1 and the network architectures are represented with alphanumeric strings as reported in [38]. The  $(2 \times 128C3)$  represents two convolution layers with each having 128 feature maps and  $3 \times 3$  convolution kernels.  $MP2$  and  $AP2$  represent  $2 \times 2$  max and average pooling layers respectively. The  $CNN_{small}$  is trained with the original CIFAR-10 dataset without any pre-processing and data augmentation. The  $CNN_{large}$  is inspired from [38] and is trained with batch normalization [34]. We pre-process

the original CIFAR-10 dataset with global contrast normalization followed by ZCA whitening. The validation set uses 10, 000 training samples. Batch normalization accelerates training by normalizing each layer inputs and also reduces the impact of weight scale [34][38]. The latter property suits pruning as we prefer small sized weights [2].  $CNN_{small}$  has three convolution layers and is smaller than the  $CNN_{large}$  which has six convolution layers. Further,  $CNN_{small}$  applies a bigger receptive field of  $5 \times 5$  convolution kernels whereas  $CNN_{large}$  has  $3 \times 3$  kernels.

The pruning process is guided by EPF which locates and identifies pruning candidates. The EPF evaluates several possible pruning candidates and selects the one which has the least adversarial effect on the network when pruned. The count of potential pruning candidates is dependent on the pruning rate and granularity. Smaller pruning rates and the IKSS constraint of similar stride and offset limits this count. Further, retraining the network can compensate the loss in accuracy due to pruning.

### 3.3.1 Feature Map Pruning

This plot shows feature map pruning results for  $CNN_{small}$  and  $CNN_{large}$ . This pruning granularity can be induced in higher rates for  $CNN_{large}$  than  $CNN_{small}$  due to bigger width. The pruning ratios are computed for convolutional layers. It is important to mention that the pruned networks are retrained to compensate the loss in performance. In this section we present feature map pruning guided by EPF. We discuss experimental results with  $CNN_{small}$  and  $CNN_{large}$ . We do not prune the first convolution layer in both networks due to fewer convolution connections and higher pruning sensitivity. Our experiments show that 100 particles are enough to select the pruning candidates. The pruning plots are provided in Fig. 3.4. For feature map pruning and the  $CNN_{small}$ , under a tolerance of 1% increase in MCR, the numbers of

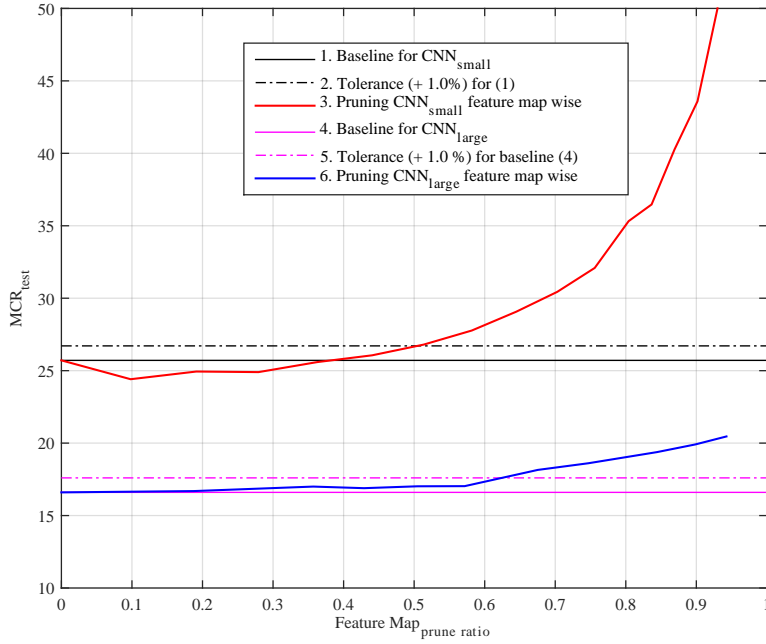


Figure 3.4: Feature map pruning

convolution connections are reduced to  $672(32 \times 21 = 672)$  and  $798 (21 \times 38 = 798)$  in the 2nd and 3rd convolution layers respectively. It means that  $35\% (= (1024 - 672)/(32 \times 32))$  and  $62\%(= (2048 - 798)/(32 \times 64))$  of the convolution connections are dropped with less than 1% increase in MCR. Increasing the feature map level sparsity beyond this point increase the MCR by more than 1%.

Next, the  $CNN_{large}$  is feature map pruned. At 57% pruning rate, the  $CNN_{large}$  is reduced to  $(128C3 - 83C3)$ -MP2- $(83C3 - 84C3)$ -MP2- $(166C3 - 166C3)$ -256FC-10Softmax with less than 0.6% increase in MCR. Comparing the feature map pruning of the two networks, we observe that  $CNN_{small}$  cannot be pruned as much as the  $CNN_{large}$  due to modest width. The  $CNN_{large}$  is wider and has more room for feature map pruning. Feature map pruning the  $CNN_{large}$  beyond 62% decreases the network's

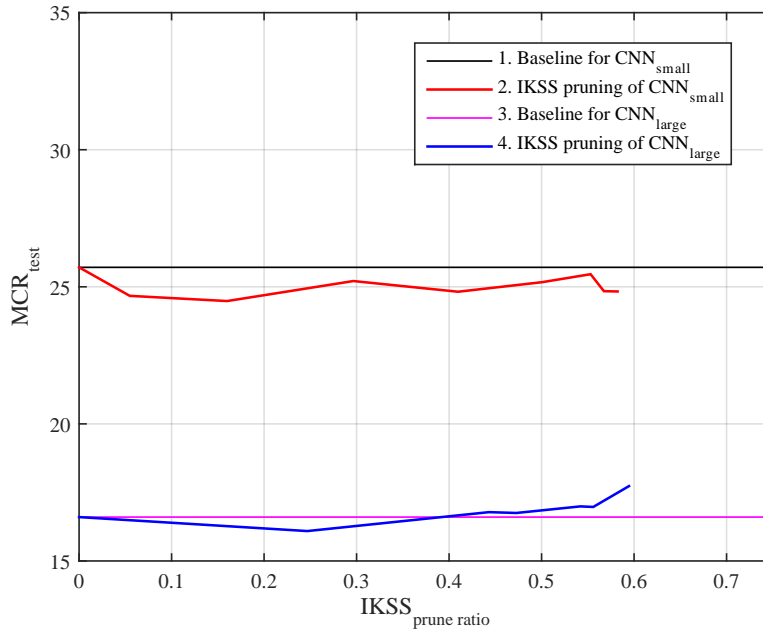


Figure 3.5: Intra-kernel pruning

baseline performance by more than 1%.

### 3.3.2 Intra-Kernel Pruning

This plot shows intra-kernel pruning results for  $CNN_{small}$  and  $CNN_{large}$ . The  $CNN_{small}$  uses  $5 \times 5$  kernels while  $CNN_{large}$  uses  $3 \times 3$  kernels. The IKSS can be induced in higher rates for  $CNN_{small}$  than  $CNN_{large}$  due to bigger size kernels. The pruning ratios are computed for convolutional layers only. It is important to mention that the pruned networks are retrained to compensate for the loss in accuracy.

In this section we present IKSS for  $CNN_{small}$  and  $CNN_{large}$  and discuss the experimental results shown in Fig. 3.5. It can be observed in Fig. 3.5 that the  $NW_{small}$  can achieve more intra-kernel sparsity than the  $NW_{large}$  which cannot be pruned with

higher rates. This is attributed to the difference in kernel sizes. Although  $NW_{large}$  is deeper and wider but it uses a smaller kernel size of  $3 \times 3$ . Therefore we infer that IKSS is more related to the kernel size than the width of the network.

Next, we discuss the selection of the stride and the offset for IKSS. We introduce a new term for each feature map, OKFF, which stands for the set of outgoing kernels from a feature map. Thus each feature map has one OKFF. As earlier mentioned, all the kernels in an OKFF has the same stride and offset. We further suppose that the feature maps in layer  $L$  has  $F$  non-pruned OKFFs and that the pruning rate is  $pr$ . The EPF has to select the pruning candidates with  $N$  particles. The state vector dimension is represented by  $d = F$  and contains the stride size for each OKFF. The OKFFs associated with  $\lfloor (F \times (1 - pr)) \rfloor$  are not pruned and are assigned zero offset and the stride of 1. If  $F$  is 10 and  $pr$  is 25%, seven OKFFs are not pruned while the remaining 3 undergoes pruning. A sample state vector for this example may look like this: [1 2 1 3 1 1 1 1 2 1]. The strides are assigned from a random distribution but takes feedback from how well the pruning proceeds. If pruning is proceeding well, we choose higher strides and induce IKSS more aggressively. Further, the earlier discussions suggest that the intra-kernel sparsity can be induced at higher rates in  $5 \times 5$  kernels compared to  $3 \times 3$  kernels. Therefore the stride assignment also considers the kernel sizes. We experimentally fine tune the hyper parameters with the validation set. Each particle's likelihood is also computed with the validation set. The offset represents the index of the first non-zero weight in a kernel and is randomly assigned in the range of  $[0, stride - 1]$ . Thus two OKFFs having similar stride size but different offsets result in two different intra-kernel pruning masks. This way the EPF guides the IKSS to approximately select the best pruning candidates.

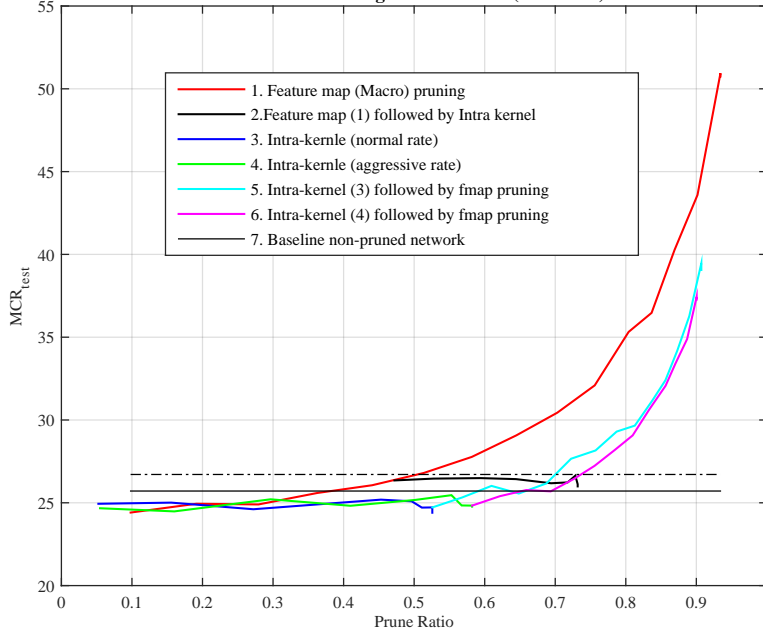


Figure 3.6: Feature map and IKSS,  $CNN_{small}$

### 3.3.3 Pruning Granularities Applied in Combinations

The plot in Fig. 3.6 shows pruning applied in various combinations to  $NW_{small}$ . It is observed that feature map pruning followed by intra-kernel pruning provides the best result. The pruning ratios are computed for convolutional layers only.

In this section, we evaluate several combinations of pruning granularities applied to  $CNN_{small}$  and  $CNN_{large}$ . We first present experimental results with  $CNN_{small}$  and the results are shown in Fig. 3.6. We observe that due to modest depth of the network, higher feature map pruning ratios cannot be achieved. Intra-kernel sparsities are induced in normal and aggressive rates. Choosing an aggressive rates increases the pruning ratio with bigger strides but also increases the MCR as shown in Fig. 3.6. We can find that the MCR does not increase much when IKSS follows the feature

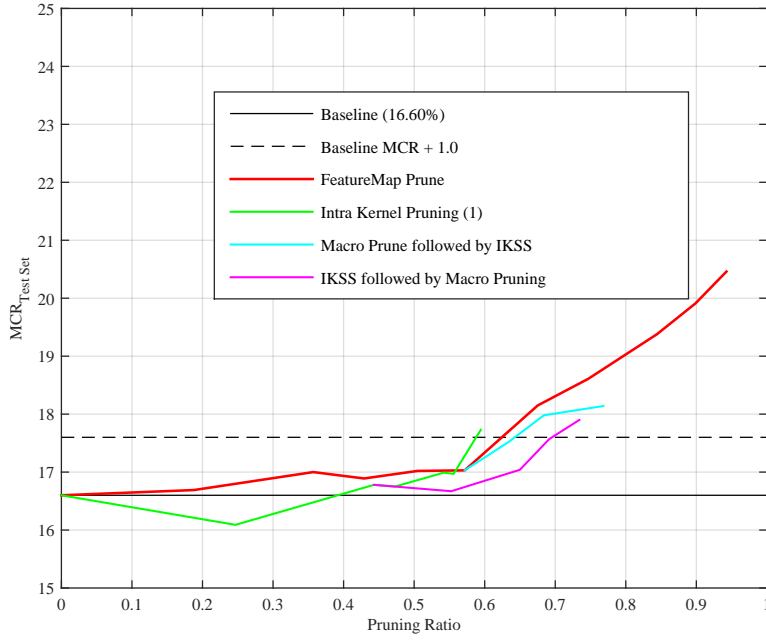


Figure 3.7: Feature map and IKSS,  $CNN_{large}$

map pruning. In Fig. 3.6, the solid black line extending from the red line shows the best pruning results. This is achieved with feature map pruning (46%) followed by IKSS (shown in Fig. 3.6 with a black solid line extending from the red solid line). Our experimental results show that we can reduce the size of the  $CNN_{small}$  network by 72% with less than 1% loss in accuracy.

The plot in Fig.3.7 shows different pruning granularities applied to  $CNN_{large}$  in various combinations. For this network, we achieve the best pruning result when intra-kernel pruning is followed by feature map pruning.

The pruning plots for  $CNN_{large}$  are provided in Fig. 3.7. It can be observed that inducing intra-kernel sparsities beyond 55% increases MCR by more than 1%. However, feature maps can be pruned by more than 62% in the same network. This is



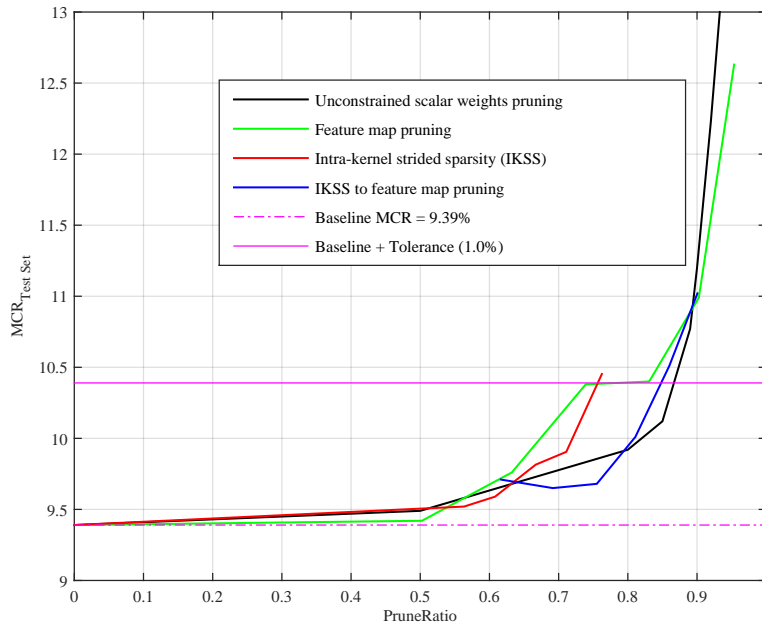


Figure 3.8: Constraint-less and constrained pruning,  $CNN_{verylarge}$

due to smaller kernel sizes of  $3 \times 3$  and higher width of the  $CNN_{large}$  network. From the figure, we obtain the best result when intra-kernel sparse network is followed by feature map pruning. From these results we can infer that wider networks have more chances of feature map level pruning whereas bigger kernels are suitable for intra-kernel pruning. In both networks, we achieve the maximum pruning ratio when the better pruning granularity is applied at the later stage.

The plot shows different pruning granularities applied to  $CNN_{verylarge}$  in various combinations. Further, we also show the case when we apply constraint-less unstructured pruning the network. The pruning ratios are computed for convolutional layers only.

We conduct another experiment with the  $CNN_{verylarge}$  and achieve better baseline

accuracy than the  $CNN_{large}$ . The  $CNN_{verylarge}$  has seven convolution layers and the architecture is reported like this:  $(2 \times 128C3)$ -MP2- $(2 \times 256C3)$ -MP2- $(2 \times 256C3)$ - $(1 \times 512C3)$ -1024FC-1024FC-10Softmax. The network is trained with data augmentation where we extract  $28 \times 28$  patches and apply random flipping, rotation, translation and scaling to the input samples. We randomly crop  $28 \times 28$  regions from the  $32 \times 32$  input image. At test time, we crop the sample from the four corners and the center. We further apply horizontal flips to each sample. This way we obtain 10 samples and take the average of their predictions to decide the final label [11]. This improves the prediction accuracy and the network obtains 90.61% accuracy on the test set. We train the network with batch normalization as reported earlier. The corresponding pruning plots are reported in Fig. 3.8.

We attribute the better baseline performance of  $CNN_{verylarge}$  to data augmentation and the bigger network. The pruning plots in Fig. 3.8 shows similar trends as reported in Fig. 3.7. For this network, we also demonstrate the pruning ratios with the constraint-free unstructured pruning. It can be observed that higher pruning ratios can be induced if there is no regularity constraint imposed on pruning. However, higher pruning ratios may not directly translate into reduced complexity. We need to identify the indices of non-zero weights with CSR/CSC sparse representation as mentioned earlier. Further, the unstructured pruning is not good for utilizing parallel architectures, such as SIMD. Thus the effectiveness and cost of sparse representation also needs to be taken into consideration. The good side of constrained pruning is that we need very simple sparse representation and we can achieve higher pruning ratios by pruning the network with mixed granularities as shown in Fig. 3.8. If we compare the pruning plots of Fig. 3.6, 3.7 and 3.8, it can be observed that higher pruning ratios can be induced with the increasing size of the network. Thus we show that the

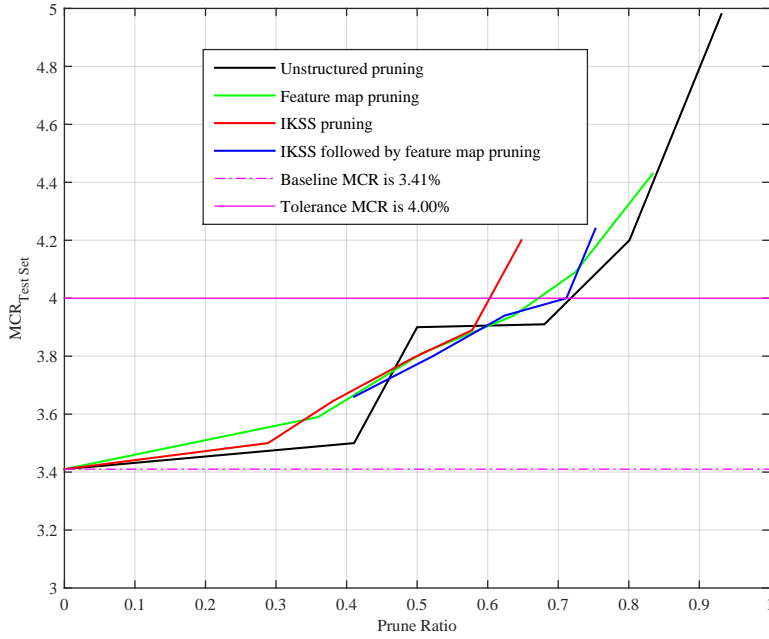


Figure 3.9: Pruning SVHN network

proposed pruning techniques have good scalability. In fact, we argue that a large size network is more resilient to pruning than a small one because the pruning followed by retraining makes the network learn the adversary effects [39].

### 3.3.4 SVHN Dataset

The plot shows different pruning granularities applied to SVHN dataset. Further, we also show the pruning granularities applied in different combinations.

The SVHN dataset consists of  $32 \times 32$  RGB images of house numbers [36]. This dataset bears similarity with the MNIST dataset [7] but is more challenging. The goal is to identify a digit in the center of a patch as there may be more than one digit in a sample. The dataset consists of 73,257 digits for training, 26,032 for testing and

53,1131 extra for training. The extra set consists of easy samples and is used along with the training set. We generate a validation set consisting of  $10 \times 400$  samples from the training set and  $10 \times 200$  samples from the extra [40]. This way we have 6,000 validation samples. This criterion is also used in [40]. The network architecture is reported like this:  $(2 \times 64C3)$ -MP2- $(2 \times 128C3)$ -MP2- $(2 \times 128C3)$  -512FC-512FC-10Softmax. This network is trained with batch normalization and RmsProp. This network achieves the baseline MCR of 3.41% on the SVHN test set. The corresponding pruning plots are reported in Fig. 3.9. Due to the small kernel size ( $3 \times 3$ ), we cannot achieve higher pruning ratios with IKSS. However, the constraint-less intra-kernel pruning can induce more than 70% pruning in the network. The network can be pruned with feature map granularity by more than 65% and thus the layer width is reduced. We achieve the best pruning ratio with structured pruning when we first prune the convolutional layers with IKSS followed by feature map pruning. This way we show that the network can be pruned by more than 70% with structured sparsity which requires very simple sparse representation. Thus we infer that the conclusions drawn in the previous sections generalize well to datasets other than the CIFAR-10.

### 3.3.5 Execution Time Savings

The plot shows the acceleration due to feature map and intra-kernel pruning granularities for second convolution layer of  $CNN_{small}$ . The  $x$ -axis shows the batch size whereas the  $y$ -axis shows the per image CPU time. The convolution connections are reduced from  $32 \times 32(C5)$  to  $32 \times 20(C5)$  with feature map pruning. Intra-kernel sparsity is then induced in the feature map pruned network which further accelerates the network.

In this section we report and compare the execution time of pruning granular-

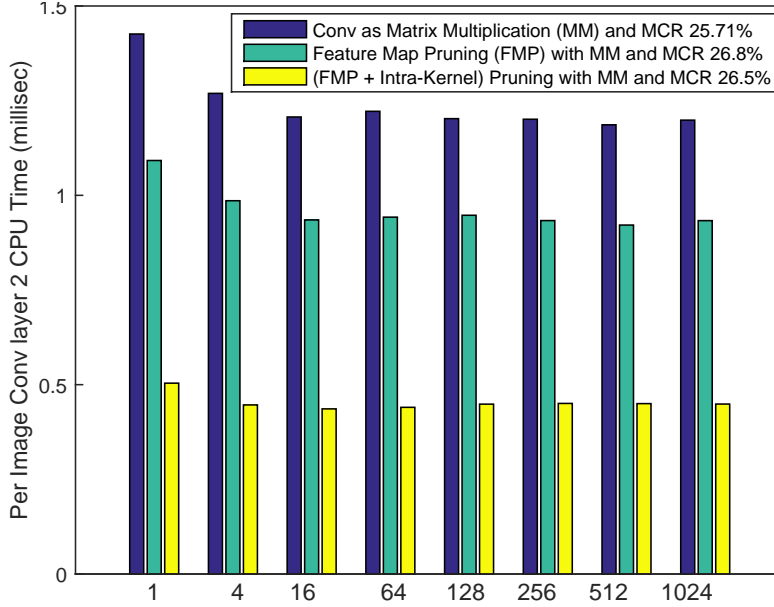


Figure 3.10: Profiling results

ities. For feature map pruning, we conduct convolutions as matrix multiplications with the BLAS library [13]. We first unroll the feature maps and kernels and conduct matrix multiplication on CPU. The execution time statistics are profiled for the aforementioned  $CNN_{small}$  network and reported in Fig. 3.10. The execution time for feed-forward path matters at real time and we profile it with various batch sizes (1, 4, 16, 64, 128, 256, 512 and 1024). We can observe from Fig. 3.10 that the macro pruning causes an acceleration of second convolution layer by 1.3 times where  $32 \times 32$  convolution connections are reduced to  $32 \times 20$ . This macro pruned network then undergoes IKSS. This further accelerates the macro pruned network by 2 times. Overall the matrix-matrix multiplication of the non-pruned network is accelerated by 2.67 times. Further, Fig. 3.10 shows that the speedups are in conformity with the theoretical numbers.

### 3.4 Comparison with the Previous Related Works

Several previous works have used pruning techniques to reduce the computational cost of deep neural networks. Computational complexity is reduced with sparse connectivity in convolution and fully connected layers in [16]. [17] pruned multi-layered feed-forward networks with genetic algorithm and simulated annealing [17]. A survey on pruning techniques is reported in [18]. These works [15, 16, 17, 18] utilize unstructured sparsity in feed-forward neural networks. A recently published work induces channel wise sparsity in a network [9]. Compared to [9], the proposed work explores sparsity at multiple levels using an evolutionary approach. Dropout [41] and Dropconnect [42] zeroes neuron outputs and weights only during training and the network architecture does not change at the evaluation time. Both techniques train different subsets of network parameters during training which results in better generalization. The proposed work drops parameters permanently and yields network with fewer parameters at test time. Convolutions are converted to matrix-matrix multiplication in [13], which follows from the same logic that two bigger sized matrix multiplications are better than several small sized ones [14]. Units in the hidden layers are pruned in [15] for a feed-forward deep neural network.

The reference work of [43] bears similarity with the proposed work for the case of intra-kernel sparsity. [43] is mainly aimed at implementing convolutions as matrix-matrix multiplications. In our proposed work, we explicitly apply various pruning granularities and the light weight network obtained from feature map pruning can be implemented in conventional way, convolution unrolling or convolution with FFTs [4]. Secondly the group wise sparsification in [43] is not necessarily strided. [43] learns the pruning mask with group-wise sparsification while our proposed approach

finds the pruning mask with an evolutionary particle filter. Further, the same work states that setting the regularization parameter is complicated.

### **3.5 Conclusions**

In this Chapter, we presented structured sparsity in deep convolutional neural networks as a means of reducing the computational complexity of the convolution layers. The sparsity in the feature map and the kernel levels is explained along with the intra-kernel strided one. The selection of the best pruning mask is guided by the evolutionary particle filtering algorithm. We found that the feature map level pruning is limited by the width of a layer while the intra-kernel sparsities are much affected by the kernel size. The proposed work has further showed that the IKSS along with convolution unrolling can significantly reduce the computational complexity of convolutions. Moreover, with three different CNN architectures and baseline performances, we showed that the proposed approach scales well to various network sizes.

## Chapter 4

# Kernel Pruning

### 4.1 Introduction

In this chapter, we target to sparsify the meshed convolution connectivity pattern. Wider convolution layers with dense kernel connectivity patterns increase the computational cost of inference. We propose feature map and kernel level pruning for reducing the computational complexity of a deep convolutional neural network. Figure 4.1 depicts this idea where feature map and kernel pruning are shown in Fig. 4.1(a) and (b) respectively. Pruning feature maps reduces the width of a layer and hence does not need any sparse representation. Further, kernel pruning changes the dense connectivity pattern into a sparse one. Due to coarse nature, these pruning granularities can be exploited by GPUs and VLSI based implementations. We propose a simple strategy to choose the least adversarial pruning masks. The proposed approach is generic and can select good pruning masks for feature map, kernel and intra-kernel pruning. The pruning masks are generated randomly, and the best performing one is



selected using the evaluation set. The sufficient number of random pruning masks to try depends on the pruning ratio, and is around 100 when 40% complexity reduction is needed. The pruned network is retrained to compensate for the loss in accuracy. We have extensively evaluated the proposed approach with the CIFAR-10, SVHN and MNIST datasets. Experiments with the CIFAR-10 dataset show that more than 85% sparsity can be induced in the convolution layers with less than 1% increase in the misclassification rate of the baseline network.

## 4.2 Kernel and Feature Map Pruning

In this section we discuss feature map and kernel pruning granularities. For a similar sized network, we analyse the achievable pruning ratios with feature map and kernel pruning. In terms of granularity, feature map pruning is coarser than kernel pruning. Feature map pruning does not need any sparse representation and the pruned network can be implemented in a conventional way, convolution lowering [13] or convolution with FFTs [4]. The main focus of the proposed work is analysing the unconstrained kernel pruning and feature map pruning. Pruning a feature map eliminates all the incoming and outgoing kernels because the outgoing kernels are no more meaningful.

Kernel pruning is comparatively finer. The dimension and connectivity pattern of 2D kernels determine the computing cost of a convolutional layer. The meshed fully connected convolution layers increases this cost and can hinder the real-time inference. The unconstrained kernel pruning converts this dense connectivity to sparse one. Kernel-pruning zeroes  $k \times k$  kernels and is neither too fine nor too coarse. Kernel level pruning provides a balance between fine-grained and coarse-grained pruning. It

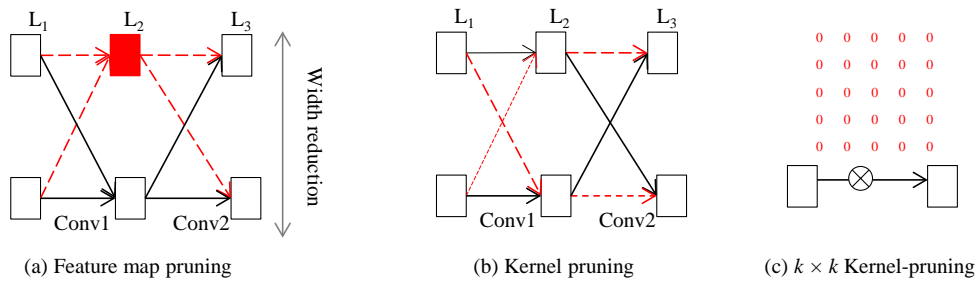


Figure 4.1: Unconstrained kernel pruning

is coarser than the intra-kernel sparsity and finer than the feature map pruning. That is why good pruning ratios can be achieved at very small sparse representation and computational cost. Each convolution connection represents one convolution operation which involves  $Width \times Height \times k \times k$  MAC operations. In LeNet [7], the second convolution layer has  $6 \times 16$  feature maps and the kernel connectivity has a fixed sparse pattern. With kernel pruning, we learn this pattern and achieve the best possible pruning ratios. The pruned network is then retrained to compensate for the losses incurred due to pruning. Figure 4.5 shows the feature map and kernel level pruning applied to MNIST [7] network. When pruning ratios increase beyond 60%, feature map pruning degrades the performance much. However the kernel level pruning can achieve higher pruning ratios due to finer scale granularity.

As the sparse granularities are coarse, a generic set of computing platform can benefit from it. One disadvantage of the unconstrained kernel pruning is that convolutions cannot be unrolled as matrix-matrix multiplications [13]. However, customized VLSI implementations and FFT based convolutions do not employ convolution unrolling. Mathieu et. al., have proposed FFT based convolutions for faster CNN training and evaluation [4]. The GPU based parallel implementation showed very good

speedups. As commonly known that the  $IFFT(FFT(kereneel) \times FFT(featuremap)) = kernel * featuremap$ , the kernel level pruning can relieve this task. Although the kernel size is small, massive reusability of the kernels across the mini-batch enables the use of FFT. The FFT of each kernel is computed only once and reused for multiple input vectors in a mini-batch. In a feed-forward and backward path, the summations can be carried in the FFT domain and once the sum is available, the IFFT can be performed [4]. Similarly, a customized VLSI based implementation can also benefit from the kernel level pruning. If the VLSI implementation imposes a constraint on the pruning criterion, such as the fixed number of convolution kernels from the previous to the next layer, the pruning criterion can be adapted accordingly. Figure 4.5 shows that the kernel pruning can be induced in much higher rates with minor increase in the MCR of the baseline MNIST network. In the next section, we report and discuss the experimental results in detail. As the commonly available libraries do not support masked convolutions, we therefore profile kernel pruning with customized GPU functions. The profiling results for kernel pruning are reported in Fig. 4.2 and 4.3. It can be observed that the kernel pruning reduces the execution time. The GPU function call scheduler shows that the call is only for non-masked kernels and passes the appropriate indices. The experiment is conducted with the CIFAR-10 CNN. In Fig. 4.3,  $F_i$  and  $F_o$  shows the input and output feature maps, while  $pr$  represents the pruning ratio. It can be observed that fewer number of convolutions will reduce the required number of GFLOPs. However, we conjecture that the true benefit of kernel pruning can be obtained with FFT based masked convolution.

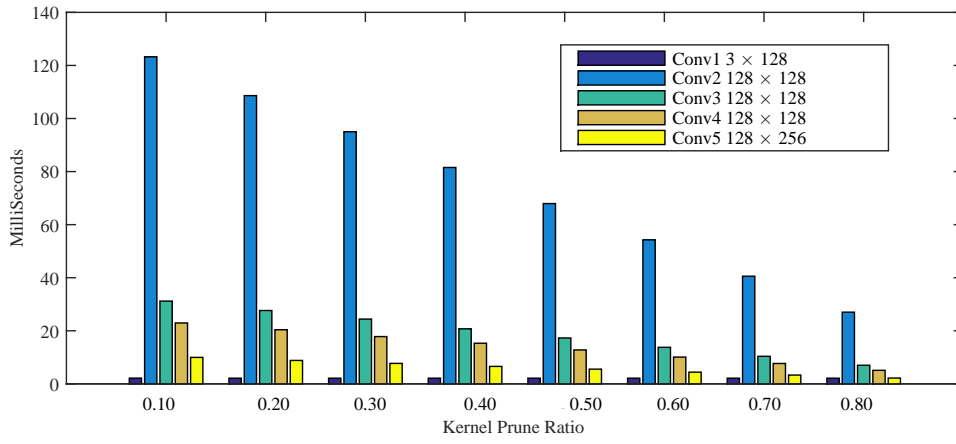


Figure 4.2: Profiling kernel pruning

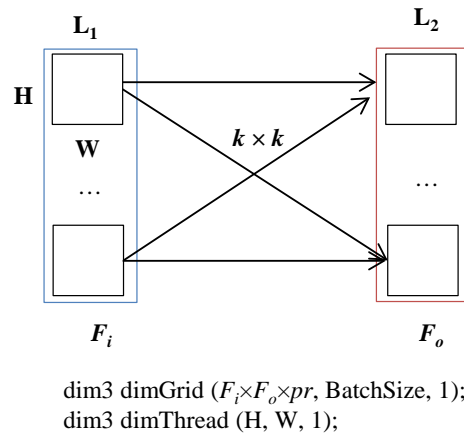


Figure 4.3: GPU function scheduler call

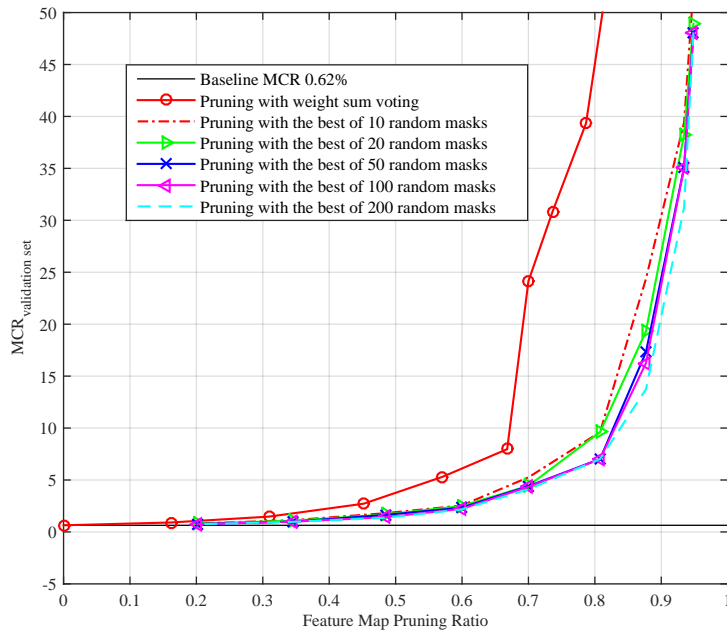


Figure 4.4: Best of  $N$  random masks vs absolute weight sum

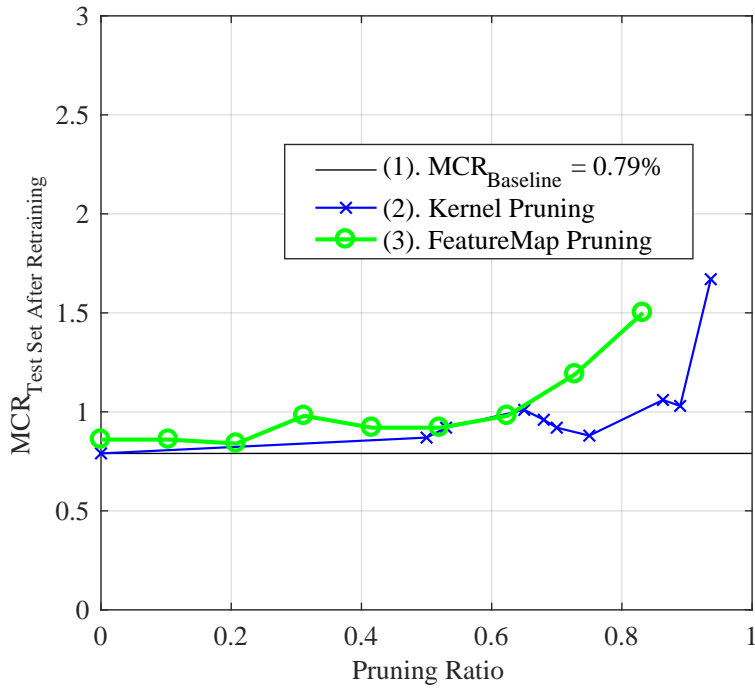


Figure 4.5: Feature map and constraintless kernel pruning

Table 4.1: The CIFAR-10 networks

Network	Architecture	Baseline MCR(%)	Data Augmentation
$CNN_{MNIST1}$	$16(C5) - 32(C5) - 64(C5) - 120 - 10$	0.62	NO
$CNN_{MNIST2}$	$6(C5) - 16(C5) - 120(C5) - 84 - 10$	0.79	NO
$CNN_{CIFAR10.small}$	$2 \times 128C3 - MP2 - 2 \times 128C3 - MP2 - 2 \times 256C3 - 256FC - 10Softmax$	16.6	NO
$CNN_{CIFAR10.large}$	$2 \times 128C3 - MP2 - 2 \times 256C3 - MP2 - 2 \times 256C3 - 1 \times 512C3 - 1024FC - 1024FC - 10Softmax$	9.41	YES
$CNN_{SVHN}$	$(2 \times 64C3) - MP2 - (2 \times 128C3) - MP2 - (2 \times 128C3) - 512FC - 512FC - 10Softmax$	3.5	NO

### 4.3 Experimental Results

In this section, we present detailed experimental results with the CIFAR-10 and SVHN datasets [35]. During training and pruning, we use the stochastic gradient descent (SGD) with a mini-batch size of 128 and RMSProp [37]. We train all the networks with batch normalization [34]. We do not prune the network in small steps, and instead one-shot prune the network for a given pruning ratio followed by retraining. The experimental results are reported in the corresponding two subsections.

#### 4.3.1 CIFAR-10

The CIFAR-10 dataset includes samples from ten classes: airplane, automobile, bird, cat, deer, dog, frog, horse, ship and truck. The training set consists of 50,000 RGB samples and we allocate 20% of these samples as the validation set. The test set contains 10,000 samples and each sample has  $32 \times 32 \times RGB$  resolution. We evaluate the proposed pruning granularities with two networks,  $CNN_{CIFAR10.small}$  and  $CNN_{CIFAR10.large}$ .  $CNN_{CIFAR10.small}$  has six convolution and two overlapped max pooling layers. We report the network architecture with an alphanumeric string as reported in [38] and outlined in Table 4.1. The  $(2 \times 128C3)$  represents two convolution layers with each having 128 feature maps and  $3 \times 3$  convolution kernels.  $MP2$  represents  $3 \times 3$  overlapped max-pooling layer with a stride size of 2. We pre-process the original CIFAR-10 dataset with global contrast normalization followed by zero compo-

ment analysis (ZCA) whitening.

The  $CNN_{CIFAR10.large}$  has seven convolution and two max-pooling layers. Further, online data augmentations are employed to improve the classification accuracy. We randomly crop  $28 \times 28 \times 3$  patches from the  $32 \times 32 \times 3$  input vectors. These cropped vectors are then geometrically transformed randomly. A vector may be flipped horizontally or vertically, rotated, translated and scaled. At evaluation time, we crop patches from the four corners and the center of a  $32 \times 32 \times 3$  patch and flip it horizontally. We average the evaluation on these ten  $28 \times 28 \times 3$  patches to decide the final label. Due to larger width and depth, the  $CNN_{CIFAR10.large}$  achieves more than 90% accuracy on the CIFAR-10 dataset. The  $CNN_{CIFAR10.small}$  is smaller than  $CNN_{CIFAR10.large}$  and trained without any data augmentation. The  $CNN_{CIFAR10.small}$  therefore achieves 84% accuracy.

#### 4.3.1.1 Individual Pruning Granularities

After layer pruning, feature map pruning is the 2nd coarsest pruning granularity. Feature map pruning reduces the width of a convolutional layer and generates a thinner network. Pruning a single feature map, zeroes all the incoming and outgoing weights and therefore, higher pruning ratios degrade the network classification performance significantly. Feature map pruning for the  $CNN_{CIFAR10.small}$  is shown in Fig. 4.6 with a circle marked red colored line. The sparsity reported here is for Conv2 to Conv6. We do not prune the first convolution layer as it has only  $3 \times 128 \times (3 \times 3) = 3456$  weights. The horizontal solid line shows the baseline MCR of 16.26% whereas the dashed line shows the 1% tolerance bound. Training the network with batch normalization [34] enables us to directly prune a network for a target ratio, instead of taking small sized steps. With a baseline performance of 16.26%, the network

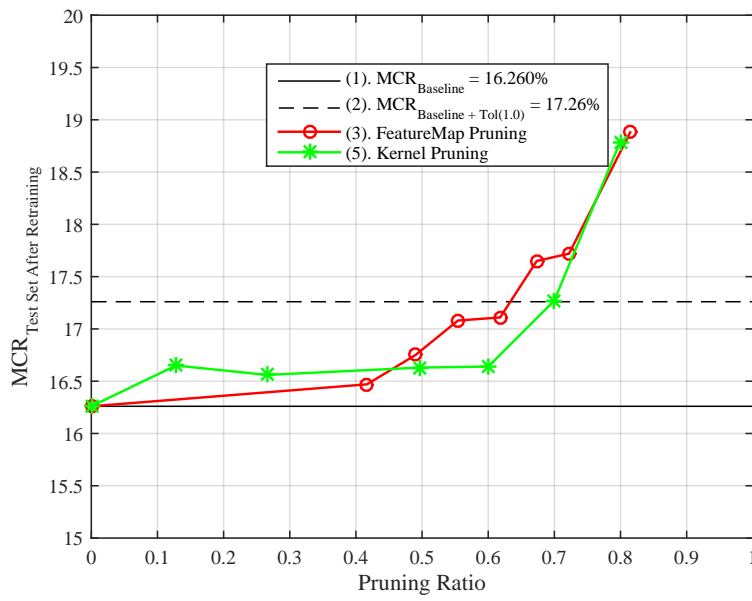


Figure 4.6: Feature map and kernel pruning, *CIFAR* – 10

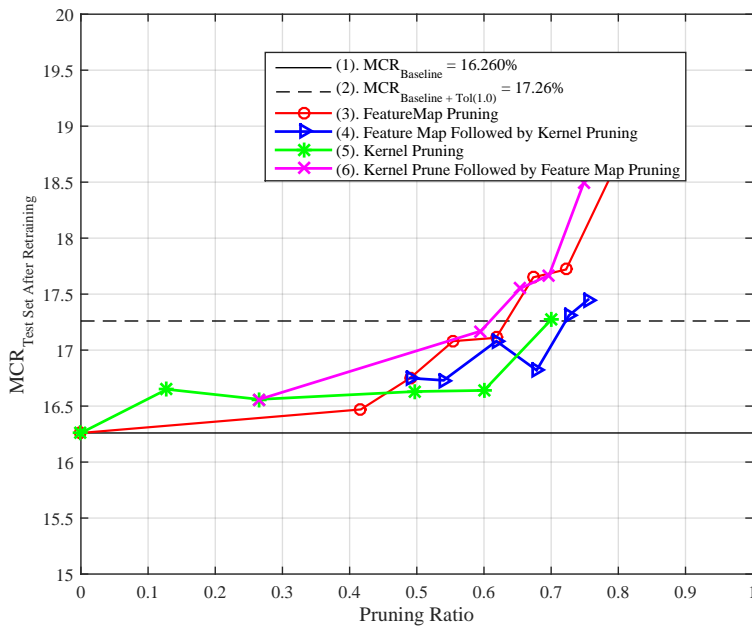


Figure 4.7: Combinations of feature map and kernel pruning



Table 4.2: Feature map and kernel level pruning (75%) in  $CNN_{CIFAR10.small}$

Feature Maps	Pruned Feature Maps	Feature Maps Prune Ratio	Pruned Kernels (%)	Conv Connections	Kernel Prune Ratio (%)
$Com2(128 \times 128)$	$128 \times 89$	30.5	$27306/9 = 3034$	11392	$3034/11392 = 26.6$
$Com3(128 \times 128)$	$89 \times 89$	51.5	$18702/9 = 2078$	7921	$2078/7921 = 26.2$
$Com4(128 \times 128)$	$89 \times 89$	51.5	$18702/9 = 2078$	7921	$2078/7921 = 26.2$
$Com5(128 \times 256)$	$89 \times 179$	51.4	$37881/9 = 4209$	15931	$4209/15931 = 26.4$
$Com6(256 \times 256)$	$179 \times 179$	51.1	$76851/9 = 8539$	32041	$8539/32041 = 26.6$

performance is very bad at 80% feature map pruning. We can observe that 62% pruning ratio is possible with less than 1% increase in MCR. The  $CNN_{CIFAR10.small}$  is reduced to  $(128C3 - 83C3)$ -MP3- $(83C3 - 83C3)$ -MP3- $(166C3 - 166C3)$ -256FC-10Softmax. As pruning is only applied in Conv2 to Conv6, therefore the Figure 4.6 pruning ratios are computed only for these layers.

For the same network, we can see that kernel level pruning performs better. We can achieve 70% sparsity with kernel level pruning. This is attributed to the fact that kernel pruning is finer and hence it achieves higher ratios. Further kernel pruning may ultimately prune a feature map if all the incoming kernels are pruned. However at inference time, we need to define the kernel connectivity pattern which can simply be done with a binary flag. So although the sparse representation is needed, it is quite simple and straightforward. Experimental results confirm that fine grained sparsity can be induced in higher rates. We achieved 70% kernel wise sparsity for Conv2 - Conv6 and the network is compressed with very simple sparse representation.

#### 4.3.1.2 Combinations of Kernel and Feature Map Pruning

In this section we discuss the various pruning granularities applied in different combinations. We first apply the feature map and kernel pruning to the  $CNN_{CIFAR10.small}$  network in different orders. With feature map pruning, we can achieve 60% sparsity under the budget of 1% increase in MCR. But at this pruning stage, the network learn-

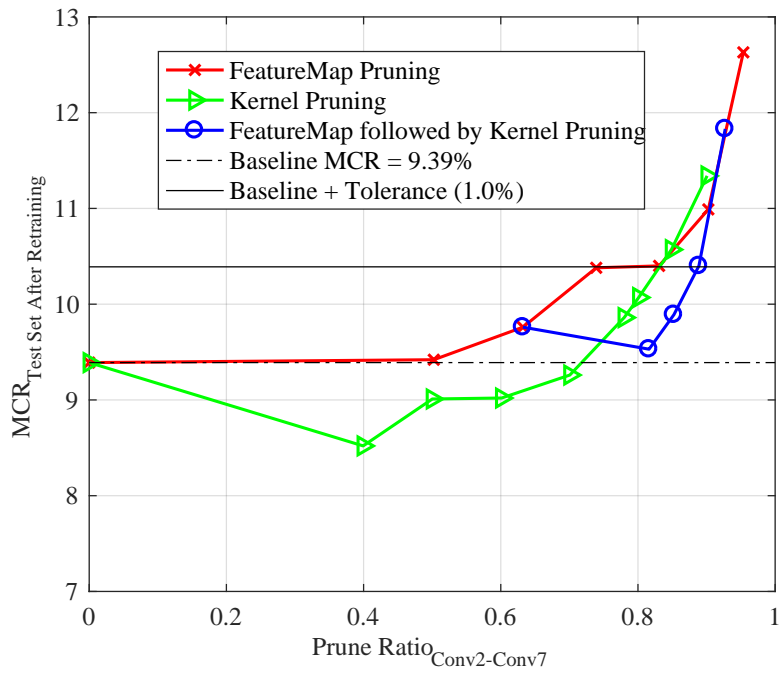


Figure 4.8: Pruning  $CNN_{CIFAR10.large}$

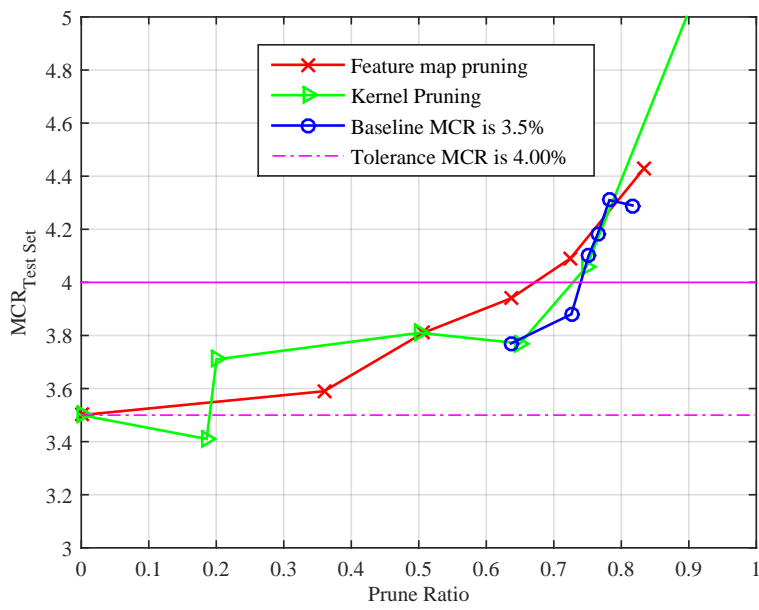


Figure 4.9: Pruning  $CNN_{SVHN}$

ing capability is affected much. So we take a 50% feature map pruned network, where the  $CNN_{CIFAR10.small}$  is reduced to (128C3 – 89C3)-MP3-(89C3 – 89C3)-MP3-(179C3 – 179C3)-256FC-10Softmax. As pruning is only applied to  $Conv2 – Conv6$ , therefore in Fig. 4.6, pruning ratios are computed only for these layers. This network then undergoes kernel level pruning. The blue rectangle line in Figure 4.7 shows the pruning results. We achieve the best pruning results in this case and the final pruned network is reported in detail in Table 4.2. Overall we achieve more than 75% pruning ratio in the final pruned network.

We further conducted experiments on the  $CNN_{CIFAR10.large}$  and the corresponding plots are shown in Fig. 4.8. The  $CNN_{CIFAR10.large}$  is much wider and deeper than the CNNsmall as reported in Table 1. Therefore there are more chances of redundancy and hence more room for pruning. Further we observe similar trends as  $CNN_{CIFAR10.small}$  where the kernel pruning can be induced in higher ratios compared to the feature map pruning. When the kernel pruning is applied to the feature map pruned network, we can achieve more than 88% sparsity in the  $Conv2 – Conv7$  of the  $CNN_{CIFAR10.large}$  network. This way we show that our proposed technique has good scalability. These results are in conformity to the resiliency analysis of fixed point deep neural networks [39].

### 4.3.2 CIFAR-100

The CIFAR-100 dataset has 50,000 images classified into 100 fine and 20 coarse labels. The dataset has 50,000 training and 10,000 test set images. The hundred class classification problem of CIFAR-100 has 500 images for each class. We construct a validation set for learning rate scheduling during training. The validation set is constructed with 100 samples for each class from the training set. This way we are left

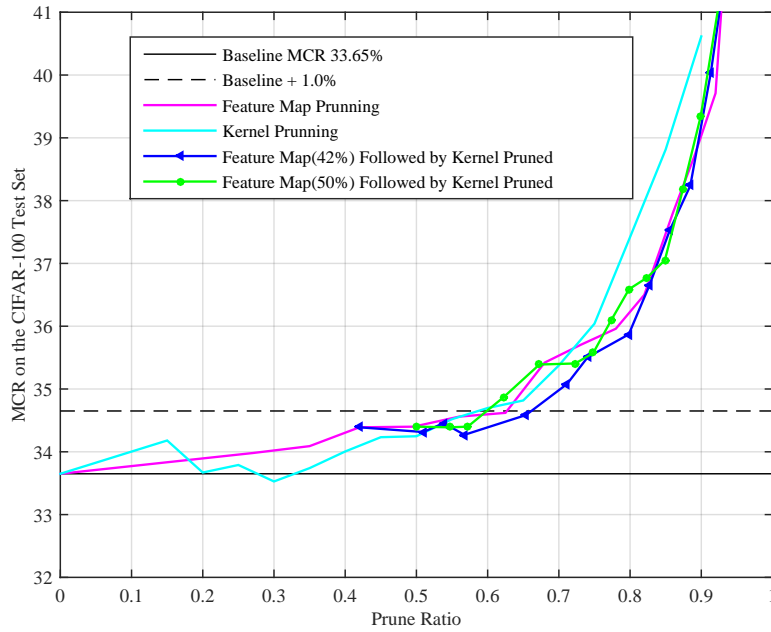


Figure 4.10: CIFAR-100 CNN

with 400 samples per class for training. We train the network with 40,000 images with data augmentation and batch normalization [34]. We obtain a baseline accuracy of 33.65% on the CIFAR-100 test set with a VGG styled network. The network architecture is reported in Table 2.1 as  $CNN_{CIFAR100}$ . The pruning plots for this dataset are provided in Fig. 4.10. It can be observed that around 60% of the network parameters can be pruned with less than 1% (absolute) increase in the network performance. Moreover, pruning in combinations further improve the pruning ratios. Thus the lessons learnt generalize well to other datasets.

### 4.3.3 SVHN

The SVHN dataset consists of  $32 \times 32 \times 3$  cropped images of house numbers [Netzer et al. 2011] and bears similarity with the MNIST handwritten digit recognition dataset [LeCun et al. 1998]. The classification is challenging as more than one digit may appear in sample and the goal is to identify a digit in the center of a patch. The dataset consists of 73,257 digits for training, 26,032 for testing and 53,1131 extra for training. The extra set consists of easy samples and may augment the training set. We generate a validation set of 6000 samples which consists of 4000 samples from the training set and 2000 samples from the extra [Sermanet et al. 2012]. The network architecture is reported like this:  $(2 \times 64C3)$ -MP2-  $(2 \times 128C3)$ -MP2- $(2 \times 128C3)$ -512FC-512FC-10Softmax. This network is trained with batch normalization and we achieve the baseline MCR of 3.5% on the test set. The corresponding pruning plots are reported in Fig. 4.9. We can observe a similar trend where kernels can be pruned by a bigger ratio compared to feature maps. More than 70% pruning ratio can be implemented in the reported network. Thus we show that the lessons learnt generalize well on various datasets.

There can be a concern that pruning may decrease the accuracy of the original network when it is deployed in the field for run time classification. For a specific problem domain, the test set is used as a proxy for the future unseen data. We argue that to some extent, this question can be answered by comparing the per class error for the original and pruned networks. This way we can see whether the pruned network is biased towards a specific class. To analyse this, we computed the per class error with the  $CNN_{SVHN}$  network as reported in Table 4.1. The results are reported in Fig. 4.11. It can be observed that the per class error for both validation and test set do not vary

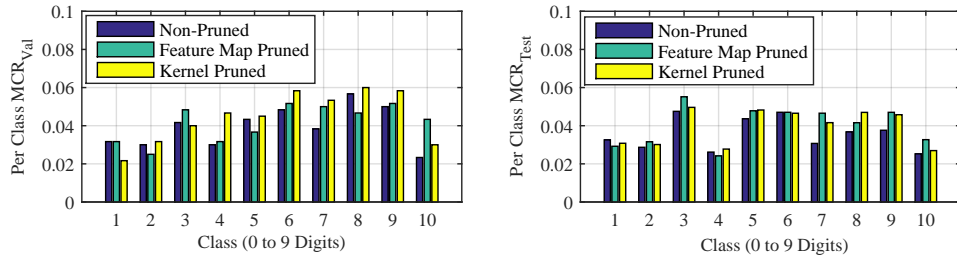


Figure 4.11: Per class error on the SVHN dataset

significantly. We therefore infer that the pruning and retraining process is a promising technique for complexity reduction.

This figure shows the per class MCR for the original, feature map, and kernel pruned networks. It can be observed that the per class error does not vary much in the pruned networks. This shows that the pruning method is not biased towards a specific class. The feature map pruned network has 63.67% sparsity with  $MCR_{Test} = 3.84\%$ ,  $MCR_{Val} = 4.16\%$ . The kernel pruned network has 65.01% sparsity with  $MCR_{Test} = 3.77\%$ ,  $MCR_{Val} = 4.45\%$ . The sparsity are computed for Conv2-Conv6.

#### 4.4 Related Works

In the literature, network pruning has been studied by several researches [2, 3, 1, 15, 16, 17, 18]. [16] have proposed a technique where irregular sparsity is used to reduce the computational complexity in convolutional and fully connected layers. However they have not discussed how the sparse representation will affect the computational benefits. The works of [2, 3] introduce fine-grained sparsity in a network by pruning scalar weights. If the absolute magnitude of any weight is less than a scalar threshold, the weight is pruned. This work therefore favors learning with

small valued weights and train the network with the L1/L2 norm augmented loss function. Due to pruning at very fine scales, they achieve excellent pruning ratios. However this kind of pruning results in irregular connectivity patterns and demand complex sparse representation for computational benefits. Convolutions are unrolled to matrix-matrix multiplication in [13] for efficient implementation. The work of [43] also induce intra-kernel sparsity in a convolutional layer. Their target is efficient computation by unrolling convolutions as matrix-matrix multiplication. Their sparse representation is not also simple because each kernel has an equally sized pruning mask. A recently published work propose sparsity at a higher granularity and induce channel level sparsity in a CNN network for deep face application [9]. The work of [15, 16, 17, 18] utilize unstructured fine grained sparsity in a neural network. Fixed point optimization for deep neural networks is employed by [19, 44, 39] for VLSI based implementations. The reference work of [5] analyzed feature map pruning with intra-kernel strided sparsity. To reduce the size of feature map and kernel matrices, they further imposed a constraint that all the outgoing kernels from a feature map must have the same pruning mask. In this work, we do not impose any such constraint and the pruning granularities are coarser. We argue that this kind of sparsity is useful for VLSI and FFT based implementations. Moreover we show that the best pruning results are obtained when we combine feature map and kernel level pruning.

## 4.5 Concluding Remarks

In this Chapter, we discussed feature map and kernel pruning for reducing the computational complexity of deep CNN. We have discussed that the cost of sparse

representation can be avoided with coarse pruning granularities. We demonstrated a simple and generic algorithm for selecting the best pruning mask from a random pool. We showed that the proposed approach adopts a holistic approach and performs better than the other methods. We conducted experiments with several benchmarks and networks and showed that the proposed technique has good scalability.



## Chapter 5

# Quantizing the Pruned Networks

### 5.1 Introduction

In this chapter, we introduce fixed-point optimization techniques for quantizing weights and signals. Generally, network parameters are stored in 32-bit floating point precision and reducing the word length is highly desired, especially for VLSI implementation. We start with a network pre-trained in high precision, prune it and then quantize it. This chapter explains the retraining based quantization algorithm and provides experimental results for fixed-point optimization of the pruned networks.

Deep convolutional neural networks have shown promising results in image and speech recognition applications. The learning capability of the network improves with increasing depth and size of each layer. However this capability comes at the cost of increased computational complexity. Thus reduction in hardware complexity and faster classification are highly desired. This work proposes an optimization method for fixed point deep convolutional neural networks. The parameters of a pre-

trained high precision network are first directly quantized using L2 error minimization. We quantize each layer one by one, while other layers keep computation with high precision, to know the layer-wise sensitivity on word-length reduction. Then the network is retrained with quantized weights. Two examples on object recognition, MNIST and CIFAR-10, are presented. Our results indicate that quantization induces sparsity in the network which reduces the effective number of network parameters and improves generalization.

In the literature, fixed point implementations of neural networks have been studied. A quantized feed-forward DNN and its VLSI implementation is introduced in [45, 44]. However, compared to DNN, CNN has more diverse layer types and hence quantization is more challenging. The work of [46] uses a directly quantized CNN. However it does not provide a retraining mechanism with low precision weights. In this Chapter, we outline our important contributions in fixed-point optimization and pruning. We provide a training mechanism with quantized weights which reduces the cost of VLSI hardware implementation. Weights with floating point precision are reduced to 3 and 4-bit precision, which yields more than 80% savings. The proposed work shows that quantizing the network to reduced word length resets 17.3% of network parameters to zero. When we only analyze the convolution layers, 19.2% parameters of the convolution layers are reset to zero. This leads to regularization impact on the network, which results in better generalization.

The rest of this Chapter is organized as follows. Section 5.2.1 presents direct quantization with L2 error minimization. Section 5.2.3 explains network retraining with quantized weights. Chapter 5.3 introduces separable fixed-point kernels. Quantization of the pruned networks is discussed in Section 5.4.2.

## 5.2 Retraining Based Quantization

The proposed work divides a network into layer-wise signal groups for quantization [47]. Throughout this chapter, high precision refers to single precision floating point. Each convolution kernel is treated independently and has its own quantization step size. For example, if the first convolution layer has  $1 \times 6$  feature maps and  $5 \times 5$  receptive field, then we have 6 convolutions in total. Each convolution operation shares a single quantization step size among  $5 \times 5 = 25$  weights. However each fully connected rear end layer has one quantization step size. In uniform quantization, the network is quantized with the same number of quantization levels for all layers. However our sensitivity analysis shows that better savings can be obtained with layer-specific quantization levels.

### 5.2.1 L2 Error Minimization and Direct Quantization

The L2 error minimization quantizes weights with an optimum quantization step size. The L2 error minimization criterion is the same as reported in [44] and Fig. 5.1. The approach is similar to Lloyd-Max quantization except that the quantizer is uniform [44].  $Q(x)$  represents the quantization function,  $\Delta$  shows the quantization step size,  $M$  represents the number of quantization levels and  $z$  shows integer membership. Results with direct quantization are reported in Table 5.2.1. The corresponding floating point misclassification rate (MCR) is 0.81%. We can find that misclassification rates with directly quantized network are not good. The network performance can be improved with retraining.

$$\begin{aligned}
Q(x) &= f(z) & (1) \\
z &= g(x) & (2) \\
f(z) &= \Delta \cdot z & (3) \\
g(x) &= \text{sign}(x) \cdot \min \left\{ \text{floor} \left( \frac{|x|}{\Delta} + 0.5 \right), \frac{M-1}{2} \right\} & (4) \\
E &= \frac{1}{2} \sum_{i=1}^N (Q(x_i) - x_i)^2 = \frac{1}{2} \sum_{i=1}^N (\Delta \cdot z_i - x_i)^2 & (5) \\
Z(t) &= \text{argmin}_z E(x, z, \Delta^{(t-1)}) & (6) \\
\Delta^{(t)} &= \frac{\sum_{i=1}^N x_i z_i^{(t)}}{\sum_{i=1}^N (z_i^{(t)})^2} & (7)
\end{aligned}$$

Figure 5.1: Direct quantization with L2 error minimization

Table 5.1: Direct quantization of the MNIST network

Conv1	Conv2	Conv3	FC1	FC2	MCR Test Set (%)
3	5	5	7	31	1.68
5	5	5	15	31	1.73
7	7	7	15	31	1.1
7	7	7	15	255	1.02

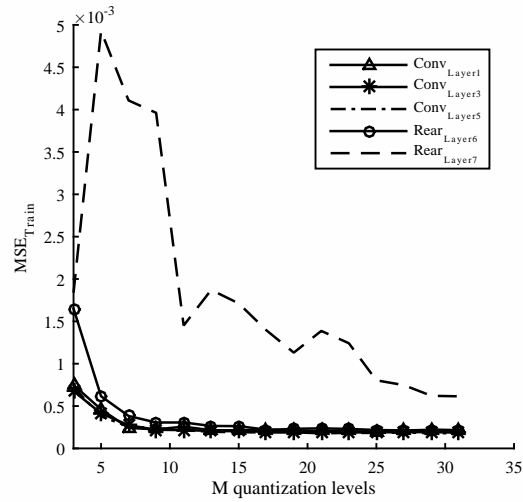


Figure 5.2: Layer-wise quantization sensitivity analysis

### 5.2.2 Layer Wise Sensitivity Analysis for Non-Uniform Quantization

In the current context, non-uniformity is layer wise and not in terms of selecting quantization levels. Uniform quantization means applying the same value of  $M$  to all layers,  $M_{conv} = M_{rear}$ . In order to squeeze the full benefits of quantized network, we conduct quantization sensitivity analysis for all convolution and rear layers. The sensitivity analysis procedure for a general signal processing system is outlined in [47]. The sensitivity analysis helps in computing the near-optimum value of  $M$  for each layer. We quantize one layer and keep the other layers in high precision. This process is conducted one by one for all the layers. The sensitivity analysis plot is shown in Fig. 2. The analysis is conducted for  $M = 3, 5, \dots, 31$ . Note that  $M$  of 31 corresponds to 5 bits quantization. The mean square error (MSE) on training set is recorded when the network converges. Figure 2 shows that the weights between the

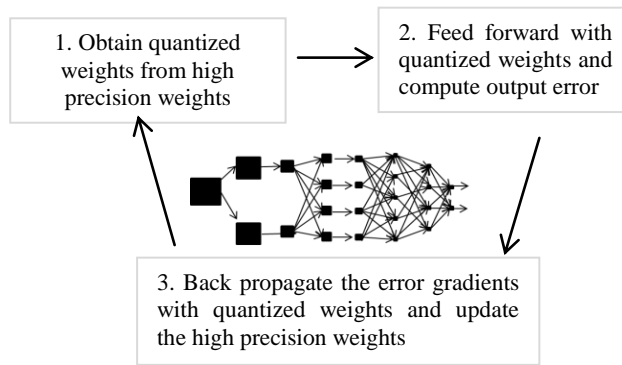


Figure 5.3: Retraining with quantized weights

penultimate and final layer,  $Rear_{layer7}$  are most sensitive to quantization. This is due to the addition of quantization noise closer to the network output. In our remaining experiments we keep this layer with 5 or 8 bit precision.  $Rear_{layer6}$  is the second most sensitive layer and we keep it with 4-bit precision. Convolution layers have proved to be robust to quantization and are kept in 3 bit precision. The next section discusses the retraining mechanism with quantized weights.

### 5.2.3 Retraining with the Quantized Weights

Direct quantization reduces the word length at the cost of degraded performance. Therefore retraining is desired to improve classification. As the theory of error back propagation [48] is well known, we focus here on explaining error back propagation in the context of our quantization framework. During training we keep parameters in both high and low precision. We set aside 5000 training samples for validation. This set is used to decide the network convergence criterion. We start with a high precision pre-trained network and obtain a quantized network using L2 error minimization. Then the input samples are fed forward via the network with the low precision

weights. This way the output error is indirectly driven by the quantization process. The output error is back propagated via low precision weights. The computed change in weights is added to the high precision weights. Thus we obtain new high precision weights. This process is iterated for several mini-batches and epochs. During training the selection of mini-batch size is important. Generally CNN employs the stochastic gradient descent (SGD) algorithm.

### 5.3 Separable Fixed-Point Kernels

Conceptually the convolution layers perform feature extraction while the rear end fully connected layers conduct classification. Usually most of the computations are performed in the convolution layers [7, 11]. Further, bigger sized convolution kernels may have improved representational capacity at the cost of increased processing. It is therefore highly desired to optimize these convolution layers. It was proposed by [49] that a 2D matrix can be approximated with the product of two 1D vectors. The singular value decomposition (SVD) was formally introduced in [50] to approximate a set of filters with linear combinations of a small number of basis functions.

A  $3 \times 3$  Sobel edge detector example is shown in Fig. 1. For a  $K \times K$  filter, the count of weights is reduced to  $K + K$  and the speedup is  $K^2/2K$ . The principal of separability is applied in [51] where two sets of separable and non-separable filters are first learnt. The non-separable filters are then approximated as linear combination of separable filters. Jaderberg et al. performs low rank decomposition not only for filters but also in the channel dimension [52]. In [16], the memory and runtime cost is reduced with sparse connectivity in convolution and fully connected layers. Fixed-point optimization of DNN and CNN is proposed in [45, 44, 19] for reduced memory and

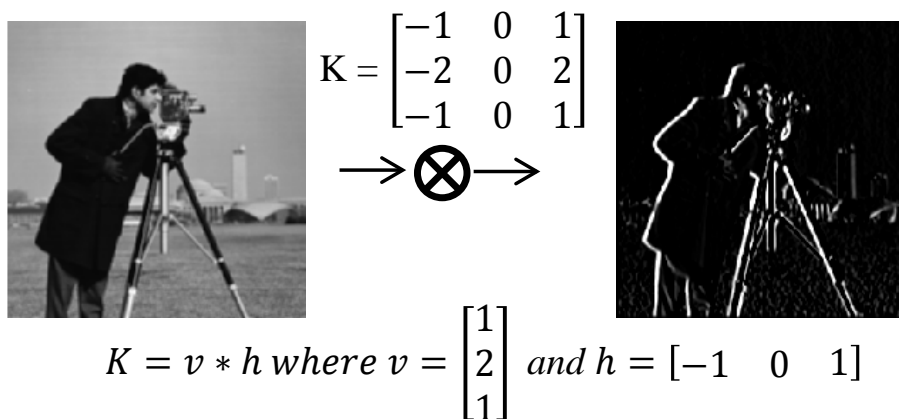


Figure 5.4: A  $3 \times 3$  separable kernel

runtime costs. However all these kernels are either non-separable or learnt in high precision (32 bit). In this Section, we outline our important contributions. We first approximate the separable kernels from non-separable ones using SVD. This is followed by quantization with reduced word-lengths for decreasing the computational and VLSI implementation cost. In order to compensate for the performance loss due to filter separation and weigh quantization, these operations are conducted inside of the retraining process so that the network learns the effect of these transforms and quantization. We first train the network with floating point weights using error back propagation and mini-batch gradient descent. The network has the freedom to learn any kinds of filters and no constraint is imposed on it at this stage. Each convolution kernel is then decomposed into 3 matrices using *SVD*. Eq. (5.1) and (5.2) show this relationship. Matrix  $U$  and  $V$  are orthogonal matrices and  $S$  is a diagonal matrix containing singular values. The reconstructed matrix  $W$  will be *rank* 1 if we only use  $S(1, 1)$  in Eq. (5.2).



$$[U, S, V] = \text{SVD}(W) \quad (5.1)$$

$$W = U \times S \times V^T \quad (5.2)$$

The SVD kernels and weights in the fully connected layer are then fixed-point optimized which is explained in the following two subsections. Separability and quantization can significantly reduce the required memory for storing convolutional weights. The kernels are only relevant for convolution layers. It is important to note that *SVD* computation does not demand any cost at the inference time since it is only computed during training. We retrain the directly quantized network with our algorithm outlined in Fig. 5.5. We modify the error back propagation algorithm in such a way that we can learn separable quantized kernels. During retraining we maintain both high and low precision weights. We initialize the network using the pre-trained floating point network. Each high precision 2D convolution kernel  $W$  is decomposed into two 1D vertical and horizontal filters  $(v, h)$  using SVD. At this stage both  $v$  and  $h$  are in high precision. We reconstruct the rank 1 high precision  $W^{(s)} = v \times h$ . The difference between  $W$  and  $W^{(s)}$  depends on the linear independence of columns or row vectors of  $W$ . In our analysis, smoother error gradients are obtained when we update  $W$  with  $W^{(s)}$ . The  $v$  and  $h$  vectors are then quantized with  $M$  quantization levels and we obtain  $v^{(q)}$  and  $h^{(q)}$ . These two 1D vectors now represent the 2D separable quantized kernel  $W^{(sq)}$ . The rear end fully connected layer weights are also quantized using L2 error minimization. These quantized kernels and weights are then used in the forward path of the network. This way the network output and the network error is driven by the quantization and separability constraints. The low precision weights

Table 5.2: Retraining the quantized MNIST network

Signal Quantization (bits)	M Quantization Levels					MCR Test Set (%)
	Conv1	Conv2	Conv3	FC1	FC2	
5	7	7	7	15	15	0.92
	7	7	7	15	31	0.88
	7	7	7	15	255	0.89
8	7	7	7	15	15	0.91
	7	7	7	15	31	0.88
	7	7	7	15	255	0.91
32	7	7	7	15	31	0.84
	7	7	7	15	255	0.77

and separable quantized kernels are used to propagate the output error backwards. The error is accumulated in high precision and floating point weights are updated. The newly obtained high precision weights are again made separable and quantized in the next iteration.

## 5.4 Quantizing the Pruned Networks

We first show our experimental results with non-pruned networks. MNIST is a handwritten digit recognition dataset consisting of 60,000 training and 10,000 test samples. Each sample has  $32 \times 32$  resolutions and is gray scale. We experiment with CNN architecture having  $6(C5) - MP2 - 16(C5) - MP2 - 120(C5) - 84FC1 - 10Softmax$  layer wise feature maps. We train the network with rectified linear units (ReLUs). Table 3 shows the classification results. We can find that higher precision for the rear layer results in better classification. The quantized network performs better, similar or comparable classification with only 10% memory space consumption. Signal quantization reduces the required hardware but slightly increases the misclas-

Separable quantized kernels  
(convolution layers):

$$W = \begin{bmatrix} w_{00} & w_{01} & w_{02} \\ w_{10} & w_{11} & w_{12} \\ w_{20} & w_{21} & w_{22} \end{bmatrix}$$

$$[U, S, V] = SVD(W)$$

$$v = U(:, 1) * \sqrt{S(1, 1)}$$

$$h = V(:, 1)^T * \sqrt{S(1, 1)}$$

$$W^{(s)} = v * h$$

Update:  $W = W^{(s)}$

$$W^{(sq)} = Q_w(v) * Q_w(h)$$

Quantized fully connected layer:

$$w_{ij}^{(q)} = Q_w(w_{ij})$$

Forward activation:

$$net_i = \sum_{j \in A_i} w_{ij}^{(q)} y_j^{(q)}$$

$$y_i^{(q)} = Q_s(f(net_i))$$

Backward step:

$$\delta_j = f'(net_j) \sum_{i \in P_j} \delta_i w_{ij}^{(q)}$$

Output error calculation:

$$\frac{\partial E}{\partial w_{ij}} = -\delta_i y_j^{(q)}$$

Weight update:

$$w_{ij, new} = w_{ij} - \alpha \left\langle \frac{\partial E}{\partial w_{ij}} \right\rangle$$

Figure 5.5: Network retraining with separable quantized kernels

Table 5.3: Quantizing the  $CNN_{SVHN}$  network

Conv1-Conv6	FC1-FC2	MCR Test Set (%)
3	3	3.59
3	7	3.81
7	7	3.77
7	15	3.78
15	15	3.70

Table 5.4: Quantizing the feature map pruned  $CNN_{SVHN}$  network

Conv1-Conv6	FC1-FC2	MCR Test Set (%)
3	3	4.03
3	7	4.12
7	7	4.05
7	15	4.14
15	15	3.98

sification rate. All results are obtained using  $2 \times 2$  max pooling. We also conducted experiments with the  $CNN_{SVHN}$  network described in Chapter 4.

#### 5.4.1 Feature Map Pruned Networks

In this Section, we report fixed-point optimization results for the feature map pruned networks. The pruned networks are obtained from the algorithms outlined in Chapter 3. The objective during pruning is to eliminate unimportant units in the network to reduce the complexity. Here we show that fixed-point optimization can further compress the network weights and signals. Table 5.4 shows the fixed-point optimization results for the feature map pruned  $CNN_{SVHN}$  network. The architecture of the feature map pruned network is  $64(C3) - 38(C3) - MP2 - 2 \times 76(C3) - MP2 - 2 \times 153(C3) - 512FC1 - 512FC2 - 10Softmax$  whereas the original network is:  $2 \times 128(C3) - MP2 - 2 \times 128(C3) - MP2 - 2 \times 256(C3) - 512FC1 -$

Table 5.5: Quantizing the kernel pruned  $CNN_{SVHN}$  network

Conv1-Conv6	FC1-FC2	MCR Test Set (%)
3	3	3.92
3	7	4.04
7	7	3.87
7	15	4.06
15	15	3.94

$512FC2 - 10Softmax$ . The pruned network has 3.84% MCR on the SVHN test set. From Table 5.4, we can observe that the pruned network can be compressed with 4 bits quantization. We are conducting experiments for fixed-point optimization of kernel pruned networks.

#### 5.4.2 Kernel Pruned Networks

In this Section, we report fixed-point optimization results for the kernel pruned networks. The pruned networks are obtained from the algorithms outlined in Chapter 4. The objective during pruning is to remove unimportant kernels in the network. The network architecture is:  $2 \times 128(C3) - MP2 - 2 \times 128(C3) - MP2 - 2 \times 256(C3) - 512FC1 - 512FC2 - 10Softmax$ . Table 5.5 shows the fixed-point optimization results for the kernel pruned  $CNN_{SVHN}$  network. The pruned network has 3.81% MCR on the SVHN test set. From Table 5.5, we can observe that 2-3 bits quantization can approximate the floating-point counterpart network.

### 5.5 Concluding Remarks

This work provides a training mechanism with quantized weights. We showed that the feature map and kernel pruned networks can be further compressed by quantiz-

ing the weights and signals to reduced precision. The resulting network can perform accurate classification with reduced word length. Further the induced sparsity helps the network to generalize well. The proposed work is good for efficient hardware and software implementations.

## Chapter 6

### Conclusion

In this dissertation, we have proposed techniques for reducing the computational complexity of convolution layers in a CNN. For this purpose, structured pruning, coarse pruning, and fixed-point optimization techniques are developed.

In Chapter 2, we introduced the pruning granularities and discussed the pruning candidate selection. We have proposed three techniques for pruning candidate selection; particle filter, best of  $N$  random masks, and activation sum voting. The best of  $N$  pruning masks is a simple and generic algorithm for selecting the best pruning mask from a random pool. We showed that the proposed approach adopts a holistic approach and performs better than the other methods. We analyzed the distribution of  $N$  random masks and discussed the reasonable range for  $N$ . We conducted experiments with several benchmarks and networks and showed that the proposed technique has good scalability. The proposed algorithm can be used to select pruning candidates for feature map, kernel, and intra-kernel pruning.

In Chapter 3, we explored structured sparsity in deep convolutional neural net-

works as a means of reducing the computational complexity of the convolution layers. The sparsity in the feature map and the kernel levels is explained along with the intra-kernel strided one. The selection of the best pruning mask is guided by the evolutionary particle filtering algorithm. We found that the feature map level pruning is limited by the width of a layer while the intra-kernel sparsities are much affected by the kernel size. The proposed work has further showed that the intra-kernel strided sparsity (IKSS) along with convolution unrolling can significantly reduce the computational complexity of convolutions. Moreover, with three different CNN architectures and baseline performances, we showed that the proposed approach scales well to various network sizes.

In Chapter 4 of this dissertation, we proposed feature map and kernel pruning for reducing the computational complexity of deep CNN. We discussed that FFT based implementations can particularly benefit from kernel level pruning. We further discussed that the cost of sparse representation can be avoided with coarse pruning granularities. We achieve the best pruning ratios when we prune a network with both pruning granularities successively.

In Chapter 5, we proposed that weights and signals in a deep neural network can be represented in 3-8 bits instead of 32-bits precision. The quantized weights are obtained using L2 error minimization. The network employs quantized weights in the feed-forward path while errors are collected in 32-bit precision. The resulting network can perform accurate classification with reduced word length. Moreover, we showed that fixed-point optimization can further compress the pruned networks.

In summary, the proposed IKSS and kernel pruning fits well on Graphics Processing Units (GPUs) while fixed-point optimized networks are helpful for VLSI based implementations. Thus techniques proposed in this dissertation, have wide applica-



bility.

# Bibliography

- [1] D. Yu, F. Seide, G. Li, and L. Deng, “Exploiting sparseness in deep neural networks for large vocabulary speech recognition,” in *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2012, pp. 4409–4412.
- [2] 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.
- [3] S. Han, H. Mao, and W. J. Dally, “A deep neural network compression pipeline: Pruning, quantization, huffman encoding,” *arXiv preprint arXiv:1510.00149*, 2015.
- [4] M. Mathieu, M. Henaff, and Y. LeCun, “Fast training of convolutional networks through ffts,” *arXiv preprint arXiv:1312.5851*, 2013.
- [5] S. Anwar, K. Hwang, and W. Sung, “Structured pruning of deep convolutional neural networks,” *arXiv preprint arXiv:1512.08571*, 2015.

- [6] D. H. Hubel and T. N. Wiesel, "Receptive fields and functional architecture of monkey striate cortex," *The Journal of physiology*, vol. 195, no. 1, pp. 215–243, 1968.
- [7] 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.
- [8] J. Nagi, F. Ducatelle, G. A. Di Caro, D. Cireşan, U. Meier, A. Giusti, F. Nagi, J. Schmidhuber, and L. M. Gambardella, "Max-pooling convolutional neural networks for vision-based hand gesture recognition," in *Signal and Image Processing Applications (ICSIPA), 2011 IEEE International Conference on*. IEEE, 2011, pp. 342–347.
- [9] A. Polyak and L. Wolf, "Channel-level acceleration of deep face representations," *Access, IEEE*, vol. 3, pp. 2163–2175, 2015.
- [10] M. Szarvas, A. Yoshizawa, M. Yamamoto, and J. Ogata, "Pedestrian detection with convolutional neural networks," in *IEEE Proceedings. Intelligent Vehicles Symposium, 2005*. IEEE, 2005, pp. 224–229.
- [11] 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.
- [12] D. Strigl, K. Kofler, and S. Podlipnig, "Performance and scalability of gpu-based convolutional neural networks," in *2010 18th Euromicro Conference on Parallel, Distributed and Network-based Processing*. IEEE, 2010, pp. 317–324.

- [13] K. Chellapilla, S. Puri, and P. Simard, "High performance convolutional neural networks for document processing," in *Tenth International Workshop on Frontiers in Handwriting Recognition*. Suvisoft, 2006.
- [14] S. Chetlur, C. Woolley, P. Vandermersch, J. Cohen, J. Tran, B. Catanzaro, and E. Shelhamer, "cudnn: Efficient primitives for deep learning," *arXiv preprint arXiv:1410.0759*, 2014.
- [15] G. Castellano, A. M. Fanelli, and M. Pelillo, "An iterative pruning algorithm for feedforward neural networks," *Neural Networks, IEEE Transactions on*, vol. 8, no. 3, pp. 519–531, 1997.
- [16] M. D. Collins and P. Kohli, "Memory bounded deep convolutional networks," *arXiv preprint arXiv:1412.1442*, 2014.
- [17] S. W. Stepniewski and A. J. Keane, "Pruning backpropagation neural networks using modern stochastic optimisation techniques," *Neural Computing & Applications*, vol. 5, no. 2, pp. 76–98, 1997.
- [18] R. Reed, "Pruning algorithms-a survey," *Neural Networks, IEEE Transactions on*, vol. 4, no. 5, pp. 740–747, 1993.
- [19] S. Anwar, K. Hwang, and W. Sung, "Fixed point optimization of deep convolutional neural networks for object recognition," in *Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on*. IEEE, 2015, pp. 1131–1135.

- [20] —, “Learning separable fixed-point kernels for deep convolutional neural networks,” in *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2016, pp. 1065–1069.
- [21] S. Anwar and W. Sung, “Compact deep convolutional neural networks with coarse pruning,” *arXiv preprint arXiv:1610.09639*, 2016.
- [22] D. Mishkin and J. Matas, “All you need is a good init,” *arXiv preprint arXiv:1511.06422*, 2015.
- [23] N. J. Gordon, D. J. Salmond, and A. F. Smith, “Novel approach to nonlinear/non-gaussian bayesian state estimation,” in *IEE Proceedings F-Radar and Signal Processing*, vol. 140, no. 2. IET, 1993, pp. 107–113.
- [24] M. S. Arulampalam, S. Maskell, N. Gordon, and T. Clapp, “A tutorial on particle filters for online nonlinear/non-gaussian bayesian tracking,” *Signal Processing, IEEE Transactions on*, vol. 50, no. 2, pp. 174–188, 2002.
- [25] K. Nummiaro, E. Koller-Meier, and L. Van Gool, “An adaptive color-based particle filter,” *Image and vision computing*, vol. 21, no. 1, pp. 99–110, 2003.
- [26] K. Nakamura, R. Yoshida, M. Nagasaki, S. Miyano, and T. Higuchi, “Parameter estimation of in silico biological pathways with particle filtering towards a petascale computing,” in *Pacific Symposium on Biocomputing*, vol. 14, 2009, pp. 227–238.
- [27] J. Carpenter, P. Clifford, and P. Fearnhead, “Improved particle filter for nonlinear problems,” in *Radar, Sonar and Navigation, IEE Proceedings-*, vol. 146, no. 1. IET, 1999, pp. 2–7.

- [28] J. Vermaak, A. Doucet, and P. Pérez, “Maintaining multimodality through mixture tracking,” in *Computer Vision, 2003. Proceedings. Ninth IEEE International Conference on*. IEEE, 2003, pp. 1110–1116.
- [29] T. Li, S. Sun, T. P. Sattar, and J. M. Corchado, “Fight sample degeneracy and impoverishment in particle filters: A review of intelligent approaches,” *Expert Systems with applications*, vol. 41, no. 8, pp. 3944–3954, 2014.
- [30] N. M. Kwok, G. Fang, and W. Zhou, “Evolutionary particle filter: re-sampling from the genetic algorithm perspective,” in *Intelligent Robots and Systems, 2005.(IROS 2005). 2005 IEEE/RSJ International Conference on*. IEEE, 2005, pp. 2935–2940.
- [31] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, “Pruning filters for efficient convnets,” *arXiv preprint arXiv:1608.08710*, 2016.
- [32] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” *arXiv preprint arXiv:1409.1556*, 2014.
- [33] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, *et al.*, “Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups,” *Signal Processing Magazine, IEEE*, vol. 29, no. 6, pp. 82–97, 2012.
- [34] S. Ioffe and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” *arXiv preprint arXiv:1502.03167*, 2015.

- [35] A. Krizhevsky and G. Hinton, “Learning multiple layers of features from tiny images,” 2009.
- [36] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, “Reading digits in natural images with unsupervised feature learning,” 2011.
- [37] T. Tieleman and G. Hinton, “Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude,” *COURSERA: Neural Networks for Machine Learning*, vol. 4, p. 2, 2012.
- [38] M. Courbariaux, Y. Bengio, and J.-P. David, “Binaryconnect: Training deep neural networks with binary weights during propagations,” in *Advances in Neural Information Processing Systems*, 2015, pp. 3105–3113.
- [39] W. Sung, S. Shin, and K. Hwang, “Resiliency of deep neural networks under quantization,” *arXiv preprint arXiv:1511.06488*, 2015.
- [40] P. Sermanet, S. Chintala, and Y. LeCun, “Convolutional neural networks applied to house numbers digit classification,” in *Pattern Recognition (ICPR), 2012 21st International Conference on*. IEEE, 2012, pp. 3288–3291.
- [41] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: A simple way to prevent neural networks from overfitting,” *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.
- [42] L. Wan, M. Zeiler, S. Zhang, Y. L. Cun, and R. Fergus, “Regularization of neural networks using dropconnect,” in *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, 2013, pp. 1058–1066.

- [43] V. Lebedev and V. Lempitsky, “Fast convnets using group-wise brain damage,” *arXiv preprint arXiv:1506.02515*, 2015.
- [44] K. Hwang and W. Sung, “Fixed-point feedforward deep neural network design using weights+ 1, 0, and- 1,” in *Signal Processing Systems (SiPS), 2014 IEEE Workshop on*. IEEE, 2014, pp. 1–6.
- [45] J. Kim, K. Hwang, and W. Sung, “X1000 real-time phoneme recognition vlsi using feed-forward deep neural networks,” in *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2014, pp. 7510–7514.
- [46] Y. Farabet, LeCun and E. Culurciello, “Hardware accelerated convolutional neural networks for synthetic vision systems,” in *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*. IEEE, 2010, pp. 257–260.
- [47] W. Sung and K.-I. Kum, “Simulation-based word-length optimization method for fixed-point digital signal processing systems,” *IEEE transactions on Signal Processing*, vol. 43, no. 12, pp. 3087–3090, 1995.
- [48] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning representations by back-propagating errors,” *Cognitive modeling*, vol. 5, no. 3, p. 1, 1988.
- [49] S. Treitel and J. L. Shanks, “The design of multistage separable planar filters,” *IEEE Transactions on Geoscience Electronics*, vol. 9, no. 1, pp. 10–27, 1971.
- [50] P. Perona, “Deformable kernels for early vision,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 17, no. 5, pp. 488–499, 1995.



- [51] R. Rigamonti, A. Sironi, V. Lepetit, and P. Fua, “Learning separable filters,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2013, pp. 2754–2761.
- [52] M. Jaderberg, A. Vedaldi, and A. Zisserman, “Speeding up convolutional neural networks with low rank expansions,” *arXiv preprint arXiv:1405.3866*, 2014.

## 국문 초록

최근 딥 러닝 알고리즘은 사물과 음성인식 등 다양한 분류 문제에서 인간 수준의 성능을 보이게 되었다. 그러나 딥 러닝은 많은 계산이 필요하기 때문에 리소스가 제한된 환경에서는 실행하기 어렵다. 따라서 이러한 계산 비용을 줄이기 위한 연구가 꾸준히 진행되고 있다. 본 논문에서는 인공 신경망의 계산 복잡도를 줄이기 위해 스푸아내기와 고정 소수점 최적화 기법을 제시한다. 스푸아내기는 큰 크기의 신경망에서 중요하지 않은 변수들을 제거하는 유망한 기술이다. 본 논문에서는 길쌈형 신경망 (convolutional neural network, CNN)의 스푸아내기 정도를 특징 맵 (feature map), 커널 (kernel), 커널 내부 (intra-kernel)의 세 단계로 나눈다. 특징 맵 스푸아내기는 많은 수의 커널을 제거하고 층의 너비를 직접 감소시킨다. 또한 추가적인 구조의 표현 (sparse representation)이 필요하지 않다. 따라서 신경망은 스푸아내기 전 신경망보다 크기가 작고 빠르게 계산될 수 있다. 하지만 특징 맵 스푸아내기는 연결된 모든 커널을 제거하기 때문에 많은 수의 변수에 영향을 준다. 따라서 특징 맵 스푸아내기는 높은 스푸아내기 비율을 얻지 못할 수 있다. 커널 스푸아내기는  $k \times k$  커널을 제거하는 중간 정도의 스푸아내기 방식이다. 밀도 높은 커널 연결 패턴을 성긴 것으로 바꿀 수 있다.  $W, H, k$ 가 각각 특징 맵 너비, 특징 맵 높이, 커널 크기를 나타낼 때 각 길쌈형 연결은  $W \times H \times k \times k$ 번의 곱셈과 덧셈 연산을 필요로 한다. 또한 커널 스푸아내기의 구조 표현은 간단하다. 한 개의 길쌈형 연결을 나타내는 데 한 개의 표시로 충분하다. 커널 내부 스푸아내기는 가장 미세한 정도의 스푸아내기 방식으로 스칼라 가중치들을 제거한다. 기존의 스푸아내기 방식은 불규칙적인 패턴으로 스칼라 가중치들을 0으로 만든다. 이러한 방식은 높은 스푸아내기 비율을 얻을 수

있지만 VLSI나 병렬처리 컴퓨터 구현을 위해서 추가적인 표현 방식을 필요로 한다. 거친 정도의 스페어네지는 간단한 표현 방식을 필요로 하지만 상대적으로 높은 스페어네지 비율을 얻기가 어렵다. 미세한 정도의 스페어네지는 높은 스페어네지 비율을 얻을 수 있는 반면 표현방식이 더 복잡하다. 본 논문에서는 언급한 세 개의 정도에 해당하는 스페어네지 기법을 제시한다. 또한 다양한 스페어네지 정도를 조합하여 신경망의 크기를 최대 한계까지 줄일 수 있음을 보인다. 커널 내부의 스칼라 가중치는 일반적으로 불규칙적인 패턴으로 스페어네지된다. 본 논문에서는 커널 내부의 건너뛰는 스페어네지 방식 (intra-kernel strided sparsity, IKSS)을 제시한다. IKSS는 일정 간격에 위치한 스칼라 가중치들을 스페어네지한다. 또한 IKSS를 적용할 때 한 특징 맵으로부터 나오는 모든 커널에 같은 간격과 출발 지점을 갖도록 제한하였다. 이는 길썬형 연산이 행렬-행렬 곱셈으로 나타내어질 때 행렬의 크기에 직접적인 영향을 준다. 위 조건을 가지는 IKSS의 구조 표현은 특징 맵 당 간격과 시작 지점에 해당하는 2개의 수로 나타내어진다. 본 논문에서는 파티클 필터, 임의의 스페어네지 마스크에서 최선을 선택하는 방법과 특징 맵 스페어네지를 위해 활성화 값의 합을 비교하는 방법을 사용하여 스페어네지 곳을 선택하는 방법을 제시한다. 임의의 마스크를 사용하는 기법에 대한 논의와 상세한 분석을 다룬다. 제시된 방법으로 다양한 정도의 스페어네지를 함께 사용하여 80% 이상의 스페어네지 비율을 얻었다.

또한, 스페어네지된 신경망은 가중치와 입출력 값을 양자화함으로써 추가적인 압축이 가능하다. 본 논문에서는 CNN의 가중치와 입출력 값을 3-8 비트로 나타내는 고정 소수점 최적화 알고리즘을 논의한다. 또한 CNN의 층 별 비트 수에 대한 민감도를 분석한다. 이와 같이 스페어네지와 고정 소수점 최적화를 적용하여 CNN의 계산 복잡도를 줄인다. 본 논문에서 제시된 방법은 graphics processing unit (GPU) 구현에 적합하다. IKSS는 행렬의 크기를 줄일 수 있으며 GPU는 행렬 곱셈에 좋은 성능을 보인다. 또한 FFT 기반 CNN 구현은 커널 수준의 스페어네지에서 이득을 얻는다. VLSI 구현의 경우 고정 소수점 최적화는 메모리 사용량을 줄여 신경망을

칩 내 메모리에 저장할 수 있게 해준다. 이와 같이 제시된 방법은 포괄적인 현대 컴퓨팅 플랫폼에 사용할 수 있다.

주요어 : CNN, Structured sparsity, Pruning, Fixed-point optimization

학번 : 2012-31288