Efficient Convolutional Neural Networks for Image Classification and Regression

Muhammad Tayyab
University of Central Florida

EFFICIENT CONVOLUTIONAL NEURAL NETWORKS FOR IMAGE CLASSIFICATION AND REGRESSION

by

MUHAMMAD TAYYAB
M.S Lahore University of Management Sciences, 2014
B.S Pakistan Institute of Engineering & Applied Sciences, Nilore, Islamabad, 2012

A dissertation submitted in partial fulfilment of the requirements
for the degree of Doctor of Philosophy
in the Department of Computer Science
in the College of Engineering and Computer Science
at the University of Central Florida
Orlando, Florida

Summer Term
2023

Major Professor: Abhijit Mahalanobis

ABSTRACT

Neural networks have been a topic of research since 1970s and the Convolutional Neural Networks (CNNs) were first shown to work well for hand written digits recognition in 1998 [56]. These early networks were however still shallow and contained only a few layers. Moreover these networks were mostly trained on a small amount of data in contrast to the modern CNNs which contain hundreds of convolution layers and are trained on millions of images. However, this recent shift in machine learning comes at a cost. Modern neural networks have extremely large number of parameters and require huge amount of computations for training and inference. A 2018 study by OpenAI [4] estimated that the compute requirements for training large models have been doubling every 3-4 months. By comparison Moore's law has a 2 year doubling period. Thus, computational requirements have been surpassing the hardware capabilities very quickly. To address this issue, we have not only developed methods for reducing the computational cost for convolution neural networks, but also to enable them to train and continually learn in this framework.

Specifically we have developed a new approach for compression based on spectral decomposition of filters, which replaces each convolution layer in the model with compact low-rank approximations. In doing so, we are motivated by the observation that the original filters in a convolution layer can be represented as weighted linear combinations of a set of 3D *basis filters* with one-dimensional *weight* kernels. While compression is achieved by using fewer basis filters, we show that these basis filters can be jointly *finetuned* along with the weight kernels to compensate for any loss in performance due to truncation, and to thereby achieve state of the art results on both classification and regression problems. We then, proposed using a minimum L1-norm regularizer to simultaneously train and compress the convolutional neural networks from scratch. Two popular neural network compression methods are pruning and low rank filter factorization. Since both of these methods depend on filter truncation, it is important for their success that the original filters are compact in the first place and the discarded filters contain little information. However conventional methods do not explicitly train filters for this purpose so any deletion of filters also discards useful information learned by the model. To address this problem we propose to train the model specifically for compression from scratch. We show that unlike conventionally trained networks, models trained with our approach can learn the same information in a much more compact fashion. Moreover the minimum L1 norm regularizer enables us to train the model for subsequent compression by either filter factorization or by pruning, thereby unifying these two compression strategies.

We also show that our compression framework naturally extends to allow *continual learning*, where the model needs to learn continuously from new data as it becomes available. This introduces a problem referred to as *catastrophic forgetting* where the model fails to preserve previously learned information as it being trained on new data. We propose a solution which eliminates this problem altogether while also needing significantly less FLOPs as compared to other continual learning techniques.

To my beloved son Mustafa

ACKNOWLEDGMENTS

I would like to express my deepest gratitude to my advisor, Dr. Abhijit Mahalanobis, who transformed my time at UCF into an unforgettable experience, even amidst some challenging periods. His guidance, expertise, and patience have been invaluable.

To the esteemed members of my committee, Dr. Gita Sukthankar, Dr. Niels Lobo, and Dr. Robert Muise, I extend my sincere appreciation for their invaluable insights, time, and constructive feedback throughout the journey of this dissertation.

I also wish to extend my sincere gratitude to Dr. Mubarak Shah and Dr. Sohaib Ahmed Khan, without whose guidance and encouragement, my journey at UCF would not have been possible. Their belief in my abilities and their willingness to provide me with the opportunity to join the esteemed PhD program at UCF have been instrumental in shaping my academic career.

A heartfelt thanks goes out to my close friends, particularly Dr. Fahad Khan, Dr. Mahboob ur Rehman, Sardar Mehran, and Umar Khalid. Their unflagging support, encouragement, and camaraderie have been pillars of strength during this process.

Lastly, but most certainly not least, I owe a profound debt of gratitude to my family. My parents and siblings have provided unwavering love and support throughout my academic journey. Their faith in my potential has been a beacon of hope and motivation. Above all, my deepest and heartfelt appreciation is reserved for my wife. During the most challenging times of my PhD journey, her strength and resilience have been a source of great comfort and resolve. She has shared in both the highs and lows of this demanding pursuit, standing by my side, and for this, I am forever thankful.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1: INTRODUCTION

Motivation

It is commonly accepted that the first deep Convolutional Neural Network (CNN) was AlexNet [54] released in 2012, and demonstrated significant performance improvement on Large Scale Visual Recognition Challenge (ILSVRC), also referred to as ImageNet [47]. AlexNet contains more convolution layers and parameters than previous shallow networks, and was trained on 1 million images in ImageNet dataset. However AlexNet changed the trend of using ”engineered” or ”hand-crafted” features, and what followed was a revolution in machine learning. Over the last decade researchers have proposed ever larger and deeper neural networks for speech, vision and language problems, mostly based on data-driven learning using very large data sets.

This recent progress in machine learning comes at a cost. Modern neural networks have very large number of parameters and need tremendous amounts of computations for training and inference. A 2018 study by OpenAI [4] estimated that the compute requirements for training large models have been doubling every 3-4 months. By comparison Moore’s law has a 2 year doubling period. Although computing requirements are rapidly surpassing the hardware capabilities, the popularity of large machine learning models has been driven by custom hardware that allows more operations to be performed per second (GPUs and TPUs) and by researchers having to deal with financial cost of doing so.

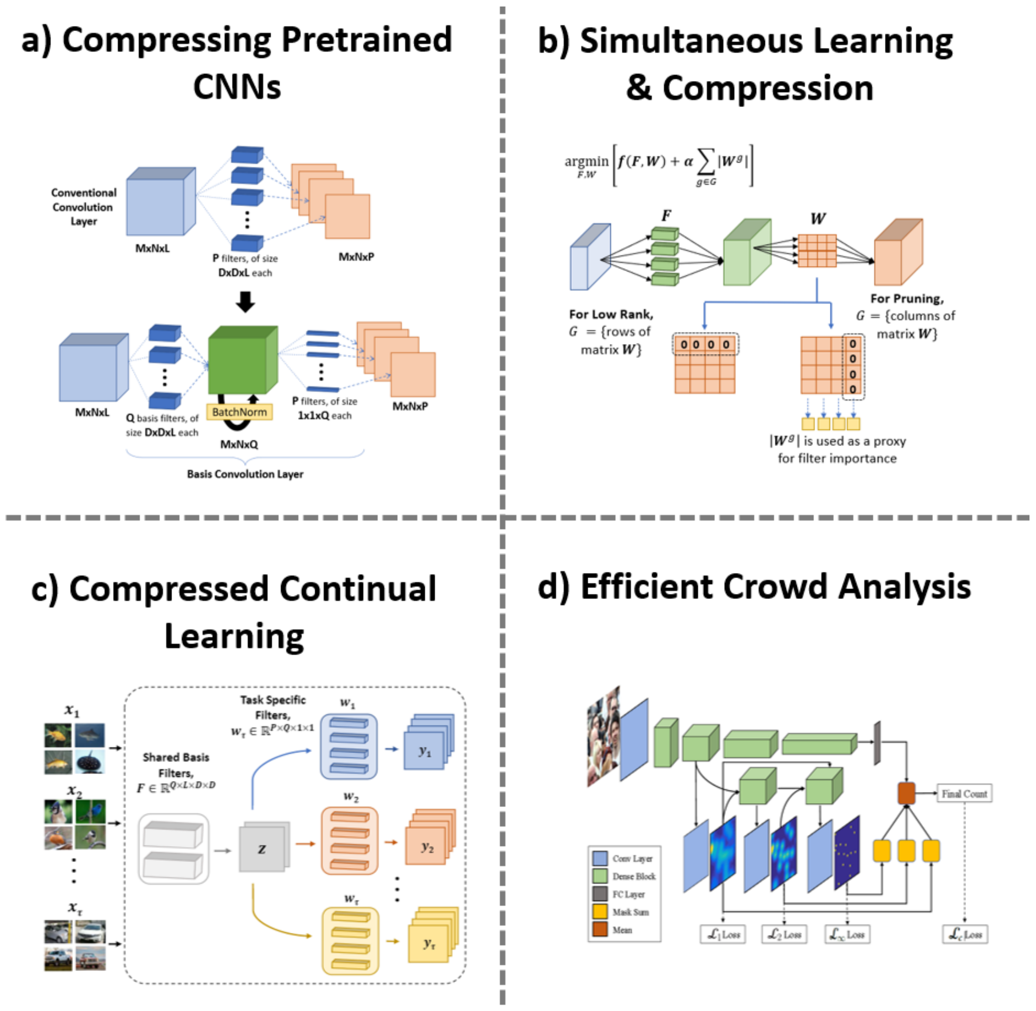

a) Compressing Pretrained CNNs
Conventional Convolution Layer
MxNxL
P filters, of size DxDxL each
MxNxP
MxNxL
Q basis filters, of size DxDxL each
BatchNorm
P filters, of size 1x1xQ each
MxNxP
MxNxQ
Basis Convolution Layer
b) Simultaneous Learning & Compression
$\underset{F,W}{\text{argmin}}\left[f(F,W)+\alpha\sum_{g\in G}|W^g|\right]$
F
W
For Low Rank, G = {rows of matrix W}
For Pruning, G = {columns of matrix W}
$|W^g|$ is used as a proxy for filter importance
c) Compressed Continual Learning
x_1
x_2
x_τ
Shared Basis Filters, $F \in \mathbb{R}^{Q\times L\times D\times D}$
Task Specific Filters, $w_\tau \in \mathbb{R}^{P\times Q\times 1\times 1}$
w_1
w_2
w_τ
z
y_1
y_2
y_τ
d) Efficient Crowd Analysis
Conv Layer
Dense Block
FC Layer
Mask Sum
Mean
Final Count
$\mathcal{L}_1$ Loss
$\mathcal{L}_2$ Loss
$\mathcal{L}_{fc}$ Loss
$\mathcal{L}_c$ Loss

Figure 1.1: Summary of contributions in each chapter in this dissertation. In Chapter 3 we introduce our filter decomposition based method for compressing pretrained ConvNets. In Chapter 4 we show how this method can be extended for simultaneous learning and compression using group L1 regularization. In Chapter 5 we apply filter decomposition to continual learning problem for compressed continual learning and finally in Chapter 6 we describe our efficient crowd analysis work.

Contributions

To address the computational cost of modern deep neural networks, we have developed a network compression framework based on spectral decomposition of filters which replace each convolution layer in the model with compact operations. Throughout this work we have measured the performance of our methods using following metrics.

- Computation requirements: As discussed above reducing computations is our primary objective. We measure the number of Floating Point Operations (FLOPs) needed to do inference on a single image for each model.
- Storage capacity: Another limitation of resource constrained devices is the storage capacity. So we also attempt to reduce the number of parameters that need to be stored.
- Accuracy: We aim to achieve high accuracy with our compressed models. In fact we compress the model such that its accuracy remains comparable to the original model.
- Energy consumption: Reducing computation and storage automatically reduces the energy consumption of the model. So where possible we have measured the energy saving achieved by our compression method.

Other than a few exceptions we have not measured the inference time of our compressed models. This is because inference time depends on external factors (e.g hardware, choice of development framework and exploitation of parallelism and so forth) which varies considerably from one method to another, and therefore not comparable across all methods reported in the literature.

Rest of this dissertation is organized as follows. We review the literature on network compression and previous works by other researchers in Chapter 2.

In Chapter 3 we explain our filter decomposition based neural network compression approach. We are motivated by the observation that the original filters in a convolution layer can be represented as weighted linear combinations of a set of 3D *basis filters* with one-dimensional *weight* kernels. The coefficients of these weight kernels represent the *spectra* of the filters in the basis in which they are represented. While compression is achieved by using fewer basis filters, we show that these can be jointly *finetuned* along with the weight kernels to compensate for any loss in performance due to truncation, thereby achieving state of the art results.

In Chapter 4 we propose a minimum L1-norm regularizer based approach to simultaneously train and compress the convolutional neural networks from scratch. Two popular neural network compression methods are pruning and low rank filter factorization. Since both of these methods depend on filter truncation, it is important for their success that the original filters are compact in the first place and the discarded filters contain little information. However conventional methods do not explicitly train filters for this purpose so any deletion of filters also discards useful information learned by the model. To address this problem we propose to train the decomposed model specifically for compression from scratch. We show that our approach allows networks to ”self compress” while learning the same information as conventional networks in a much more compact fashion. Moreover the choice of regularizer enable us to train the model for subsequent compression by either filter factorization or by pruning.

In Chapter 5 we show that our compression framework naturally extends to the continual learning problem. In continual learning, the learning algorithm does not have access to all training data at once. Therefore, the model needs to learn continuously from data as it becomes available. This introduces a problem referred to as *catastrophic forgetting* where model fails to preserve previously learned information as it being trained on new data. We propose a solution which eliminates catastropic forgetting while also requiring significantly less FLOPs as compared to other methods.

In Chapter 6 we study an application of our compression method for regression based crowd analysis framework. First, we propose a novel approach to crowd counting, density map estimation and localization of people in a given crowd image. Our approach for crowd-counting stems from the observation that these three problems are very interrelated - in fact, they can be decomposed with respect to each other. Counting provides an estimate of the number of people / objects without any information about their location. Density maps, which can be computed at multiple levels, provide weak information about location of each person. Localization provides accurate position information, nevertheless, it is extremely difficult to estimate directly due to its very sparse nature. Therefore, we propose to estimate all three tasks simultaneously, while employing the fact that each is special case of another one. Density maps can be 'sharpened' till they approximate the localization map, whose integral should equal to the true count. Subsequently, we employ a compression scheme based on low-rank filter decomposition, aimed at reducing the Floating Point Operations (FLOPs) without compromising the model's accuracy.

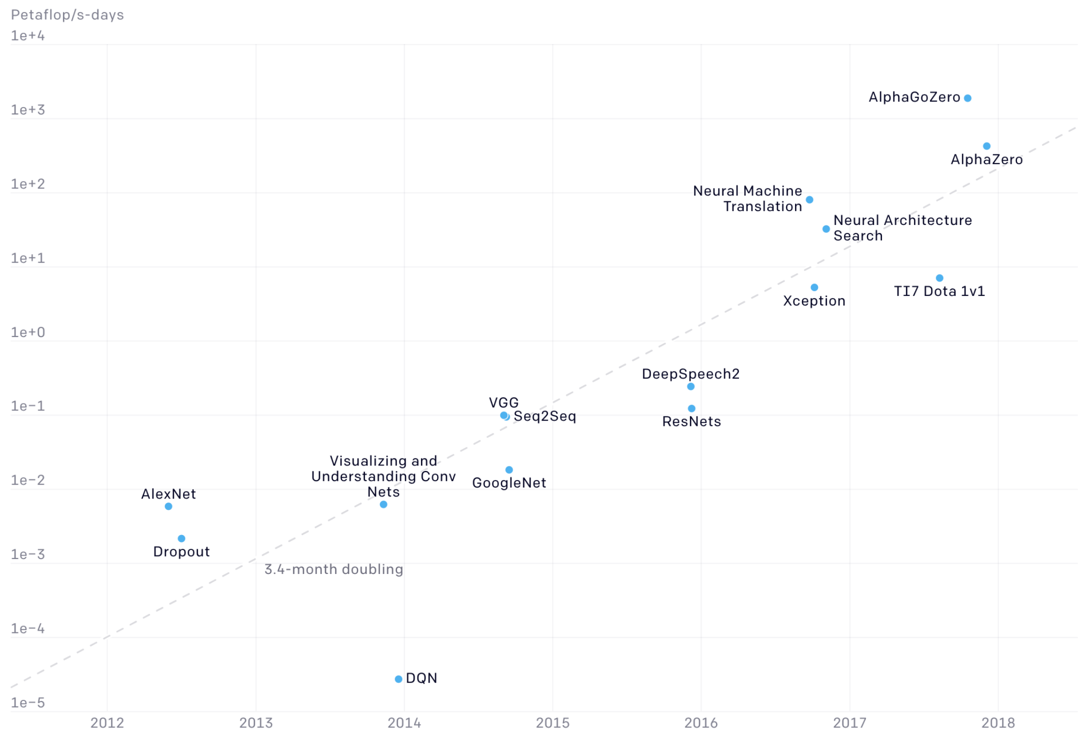


Figure 1.2: AlexNet to AlphaGo Zero: A 300,000x increase in compute. Y-axis denotes the number of operations required to train a model and account for the number of examples in the dataset as well as the the number of epochs model is trained for. A petaflop/s-day consists of performing 10^{15} neural net operations per second for one day, or a total of about 10^{20} operations. [4]

CHAPTER 2: LITERATURE REVIEW

Neural Network Compression

Low-rank approximations: Our work relates to a class of techniques that rely on low-rank approximations to represent the convolution filters. Previous works [45, 20, 81, 118, 50] have demonstrated excellent gains by exploiting linear subspace representation, but with relatively shallow networks. Wen et. al. [105] add extra gradients in back-propagation step to learn correlated filters, while Alvarez et. al. [3] train the regular CNNs to have low rank by constraining the nuclear norm of the filter matrix. Li et. al. [59] directly optimize the multiply-accumulate (MAC) operations via constrained optimization. Network weights are decomposed and then a binary masking variable is used to denote whether a particular singular value and the corresponding singular vectors are used in low-rank approximation. CaP [75] combines low rank factorization and pruning, by cascading the low rank projections of filters in the current layer to those in the next layer. Idelbayev et. al. [41] compresse the CNN by constraining the rank of filters. Trained Rank Pruning (TRP) [109] combines the channel wise low rank compression with spatially separable compression. Idelbayev et. al [41] solve rank selection problem with a mixed discrete-continuous optimization jointly over the rank and numerical values of the filters.

Pruning: Although not directly related to our work, filter pruning is probably the earliest explored research directions for compression and efficient implementation of CNNs. L. Cun et al. [18] and Hassibi et al. [27] showed that second derivative of the loss can be used to reduced the number of connections in a network. This strategy not only yields an efficient network but also improves generalization. These early pruning techniques were mostly applicable for training the network from scratch. More recently however there has been growing interest in pruning redundancies from a pre-trained network. Han et al. [26] proposed a compression method which aims to learn not only weights but also the connections between neurons from the training data. In contrast, Srinivas et. al. [98] proposed a data-free method to prune neurons instead of the whole filters. Chen et al. [15] proposed a hash based parameter sharing strategy which in turn reduces storage requirements.

Li et. al. [60] propose removing entire filters instead of individual weights using L1 norm to determine the unimportant filters. Channel Pruning [33] uses a LASSO regression based method to reduce parameters and accelerate CNNs. Zhao et. al. [120] uses variational Bayesian scheme to estimate channel saliency which is then used for pruning. Neuron Importance Score Propagation (NISP) [114] applies feature ranking techniques to measure the importance of each neuron in the final response layer, and formulates network pruning as a binary integer optimization problem for removing neurons in earlier layers. Discrimination-aware Channel Pruning (DCP) [121] choose channels which contribute to discriminative power of the model. AutoML for Model Compression (AMC) [34] propose a framework for model compression which leverage reinforcement learning to provide the model compression policy. Molchanov et. al. [77] [76] use Taylor series expansion to estimate the contribution of a neuron (filter) to the final loss and iteratively removes those with smaller scores.

A number of methods scale the output of specific structures, such as neurons, groups or residual blocks, to zero. Among these methods, Data Driven Sparse Structures (DDS) [40] introduces sparsity regularizations on scaling factors, and solve this optimization problem by a modified stochastic Accelerated Proximal Gradient (APG) method. Slimming [69] imposes L1 regularization on the scaling factors in batch normalization (BN) layers to force their output to zero. Gate Decorator (GDN) is a global filter pruning method [112] which scales factors for the filters and use Taylor expansion to estimate the change in the loss function caused by setting the scaling factor to zero. This estimation is then for filter importance ranking

Learning filter pruning criteria (LFPC) [31] introduces a differentiable pruning criteria to adaptively select the appropriate pruning criteria for different layers. Filter Pruning using High-Rank feature map (HRank) [64] proposes to prune filters to produce the low-rank feature maps, and to show that such filters contain less information. In local feature compression (LFC) [96], the technique identifies pairs of highly correlated filters and makes then maximally correlated before discarding one of them. ThiNet [70] observes that only a subset of channels in a layers input are needed to approximate its output. They used this observation to compress the previous layer.

Quantization: Filter quantization has been also used for network compression. These methods aim to reduces the number of bits required to represent the filters which can in turn lead to efficient CNN implementation. Quantization using k-means clustering has been explored by Gong et al. [24] and Wu et al. [107]. Similarly Vanhoucke et al. [101] also showed that 8-bit quantization of the parameters can result in significant speed-up with minimal loss of accuracy. In contrast Han et. al. [25] combined quantization with pruning. A special case of quantized networks are binary networks, which use only one bit to represent the filter values. Some of the works which explore this direction are BinaryConnect [16], BinaryNet [17] and XNORNetworks [82]. Their main idea is to directly learn binary weights or activation during the model training.

Knowledge Distillation: Knowledge Distillation methods train a smaller network to mimic the output(s) of a larger pre-trained network. Bucila et. al. [10] is one of the earliest works exploring this idea. They trained a smaller model from a complex ensemble of classifiers without significant loss in accuracy. More recently Hinton et. al. [35] further developed this method and proposed a knowledge distillation framework, which eased the training of networks. Ba et. al. [6] is an adaption of [10] which aims to compress deep and wide networks into shallower ones. Belagiannis et. al. [8], Lin et. al. [66] and GAL [67] also used this idea to transfer knowledge from larger networks to much shallower ones using adversarial loss.

Lifelong Learning

A lot of research has been done on lifelong learning in recent years. We have followed Delange et al. [19] to divide these methods into following groups based on the way they tackle forgetting of the previously learned knowledge.

Network Expansion methods are most relevant to our proposed approach. These methods prevent catastrophic forgetting by adding dedicated parameters for each new task to calculate the task specific feature maps. However unconstrained parameter growth can very quickly overwhelm the memory resources, so the rate of parameter growth is a matter of concern here.

Rusu et al. [87], Yoon et al. [111] and Jerfel et al., 2019 [46] proposed incrementally adding parameters to the model. Mallaya et al. [72] prune the previous tasks parameters before introducing new task. Similarly, Wortsman et al., 2020 [106] presented a masking mechanism to train separate sub-network for each task. Recently, Vinay et. al. [102] suggested adding several task specific feature map transformation layers to the model which add a small number of additional parameters. Miao et. al. [74] proposed a low rank sub-space based approach which decompose original filters into a set of task specific atoms and shared coefficients. Task specific filters are obtained at inference time by multiplying the shared coefficients with corresponding filter atoms.

Replay Methods assign a small memory to store a subset of previous task samples or train a generator model to synthesise pseudo-samples. These samples are then used to train the model along with new task data to make sure that previously learned knowledge is retained. Storing samples from previous tasks however raises privacy concerns which is a drawback of these methods.

Shin et al. [92] proposed training a generative model to produce samples for previous tasks. Rebuffi et al. [83] stores a subset of exemplars per class, selected to best approximate class means in the learned feature space. Rolnick et al. [86] suggest a sampling strategy to limit size of the memory buffer. Yan et al. [110] proposed combining the network expansion with a memory buffer to store previous task samples.

Regularization-based methods avoid storing any samples and instead add a regularization term to the loss function meant to prevent the drift in previous task's loss landscape. These methods need to carefully balance the plasticity vs stability of the model to make sure the new information is ingested properly while also preventing catastrophic forgetting.

Li et al. [62] propose a knowledge distillation based technique, where previous task outputs are used as the soft labels to mitigate forgetting and transfer knowledge. Kirkpatrick et al. [51] estimate the Fisher information matrix which is used to identify the important parameters for previous tasks. Training algorithm then selectively penalizes changes to these parameters. Similarly, Aljundi et al. MAS [2] suggest unsupervised importance estimation using gradient magnitude. Finally, Titsias et al. [100] introduce Bayesian functional approach which avoids forgetting by constructing an approximate posterior belief of previous tasks.

Crowd analysis

Crowd analysis is active an area of research with works tackling the three aspects of the problem: counting-by-regression [88], [58], [43], [11], [103], density map estimation [58], [23], [116], [80], [119] and localization [71], [85].

Earlier regression-based approaches mapped global image features or a combination of local patch features to obtain counts [52], [13], [43], [14]. Since these methods only produce counts, they cannot be used for density map estimation or localization. The features were hand-crafted and in some cases multiple features were used [11], [43] to handle low resolution, perspective distortion and severe occlusion. On the other hand, CNNs inherently learn multiple feature maps automatically, and therefore are now being extensively used for crowd counting and density map estimation.

CNN based approaches for crowd counting include [55], [116], [119], [79], [5]. Zhang et. al. [116] train a CNN alternatively to predict density map and count within a patch, and then average the density map for all the overlapping patches to obtain density map for the entire image. Lebanoff and Idrees [55] introduce a normalized variant of the Euclidean loss function in a deep network to achieve consistent counting performance across all densities. The authors in [119] use three-column CNNs, each with different filter sizes to capture responses at different scales. The count for the image is obtained by summing over the predicted density map. Sindagi and Patel [95] presented a CNN-based approach that incorporates global and local contextual information in an image to generate density maps. The global and local contexts are obtained by learning to classify the input image patches into various density levels, later fused with the output of a multi-column CNN to obtain the final density map. Similarly, in the approach by Sam *et al.* [89], image patches are relayed to the appropriate CNN using a switching mechanism learnt during training. The independent CNN regressors are designed to have different receptive fields while the switch classifier is trained to relay the crowd scene patch to the best CNN regressor.

For localization in crowded scenes, Rodriguez *et al.* [85] use density map as a regularizer during the detection. They optimize an objective function that prefers density map generated on detected locations to be similar to predicted density map [58]. This results in both better precision and recall. The density map is generated by placing a Gaussian kernel at the location of each detection. Zheng *et al.* [71] first obtain the density map using a sliding window over the image [58], and then use integer programming to localize objects on the density maps. Similarly, in the domain of medical imaging, Sirinukunwattana *et al.* [97] introduce spatially-constrained CNNs for detection and classification of cancer nuclei.

CHAPTER 3: LEARNING SPARSE REPRESENTATIONS

Introduction

While there has been a tremendous surge in convolutional neural networks and their applications in computer vision, relatively little is still understood about how information is learned and stored in the network. This is evidenced by the fact that many approaches exist for compressing a network after it has been trained such as pruning weights [18, 27], and low rank approximations of filters [20, 45, 81]. It is clear that CNNs do not need to explicitly use a large number of coefficients, and can be substantially compressed before they are deployed.

Low-rank approximation methods often represent a bank of filters in a given layer using the decomposition shown in Figure 3.1. The basic idea is that *P* original filters can be approximated as weighted linear combinations of *Q* intermediate filters, where $Q < P$. Instead of applying the original filters, the input data is first processed using the intermediate filters, followed by a set of 1D convolutions that recombines their outputs to obtain the final result. We refer to this structure in Figure 3.1 as the *BasisConv* layer. Compression results from the fact that the number of intermediate filters Q required for this approximation is less than the number of original filters P. Various different cost functions have been proposed for optimizing the intermediate filters and the weights of linear combination to prevent loss of performance due to compression, including formulations that directly approximate the original filters and their outputs at each layer.

One of the key challenges is determining the optimum choice of Q, which is the rank for each layer of the network. Existing methods either select Q arbitrarily, or perform exhaustive searches to find values of Q that enhance both compression and performance [41]. We are motivated by observation that if the intermediate filters are linearly independent, then the rank of the decomposition can be

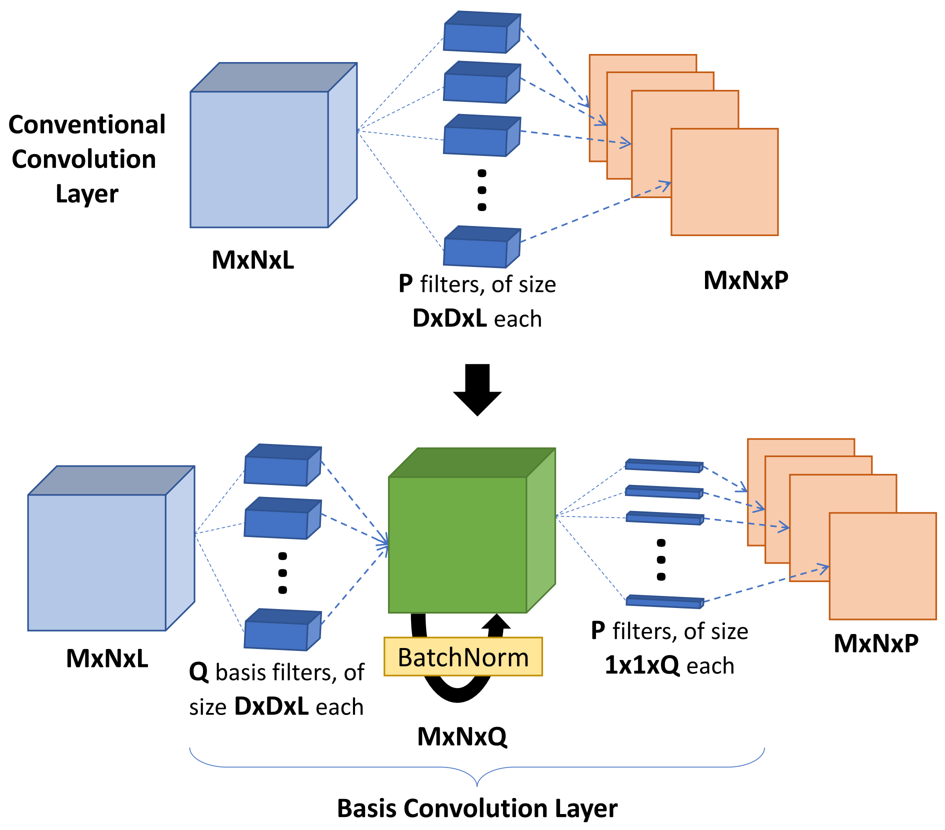

Conventional Convolution Layer
MxNxL
P filters, of size DxDxL each
MxNxP
MxNxL
Q basis filters, of size DxDxL each
BatchNorm
MxNxQ
P filters, of size 1x1xQ each
MxNxP
Basis Convolution Layer

Figure 3.1: We propose to compress the network by replacing each convolution layer (top) in a CNN with the basis convolution layer (bottom). Q basis filters are initialized with the top ranked eigen vectors of P filters in original convolution layer, with Q < P.

automatically reduced by simply minimizing the L1-norm of the 1D weight vectors to encourage sparsity in their values. We will show that this decreases the eigen-values associated with the singular value decomposition of the equivalent original filters, and Q simply emerges as the number of non-zero eigen-values.

Based on this observation, an optimum low-rank version of the BasisConv layer is obtained by using the Q non-redundant eigen-vectors (i.e. those corresponding to non-zero eigenvaleus) as the intermediate filters, and the corresponding coefficients as the 1D weight vectors. If the intermediate filters form a *basis set*, then the weights of linear combination represent the *spectra* of the spatial filters in that basis. Furthermore, these basis filters and spectral weights can be easily fine-tuned to recover any performance loss due to compression. Our approach does not seek to approximate the original filters or their responses, but directly finds a sparse set of basis filters and spectral weights that maximizes overall performance of the compressed network.

Figure 3.2 shows the main steps of the proposed approach. First, the original filters are extracted from the layers of the pretrained network (also referred to as *ConvNet*). The eigen-decomposition of these filters are used to initialize the *BasisConv* layers. To reduce the rank of the decomposition, the model is then finetuned while simultaneously minimizing the L1-norm of the 1D weight vectors. After convergence, the intermediate filters may no longer form a basis, but the sparsity of the weights directly reduces the rank of the approximation. This is exploited in Step 2 where the intermediate filters and sparse weights are transformed again into a new *basis set* and *spectral weights*. The drop in the corresponding eigen-values of the decomposition enable us to retain fewer basis filters thereby making Q small. Finally, the truncated basis set and the spectral weights are jointly fine-tuned to maximize the performance of the compressed network. The details of the proposed method will be described in Section 3.

We refer to our proposed method as *Sparse Spectral Representation and Fine Tuning* (SSRFT). In addition to its algorithmic and structural simplicity, a key difference between conventional methods for compression using low-rank approximation and ours is that filters are *learned* via sparse representation in a compact basis set. This is reminiscent of *compressive sensing* where the idea of sparse representation is key to the recovery of signals from compressive measurements. As we will show, SSRFT dramatically reduces the number of filtering operations and filter storage requirements, without notable drop in performance. To be clear, our method does not sacrifice the 4D tensor structure of the filter banks. While linear combination using spectral weights is cleverly implemented as a 1D convolution, this is not a rank-1 assumption, nor is it approximating a higher

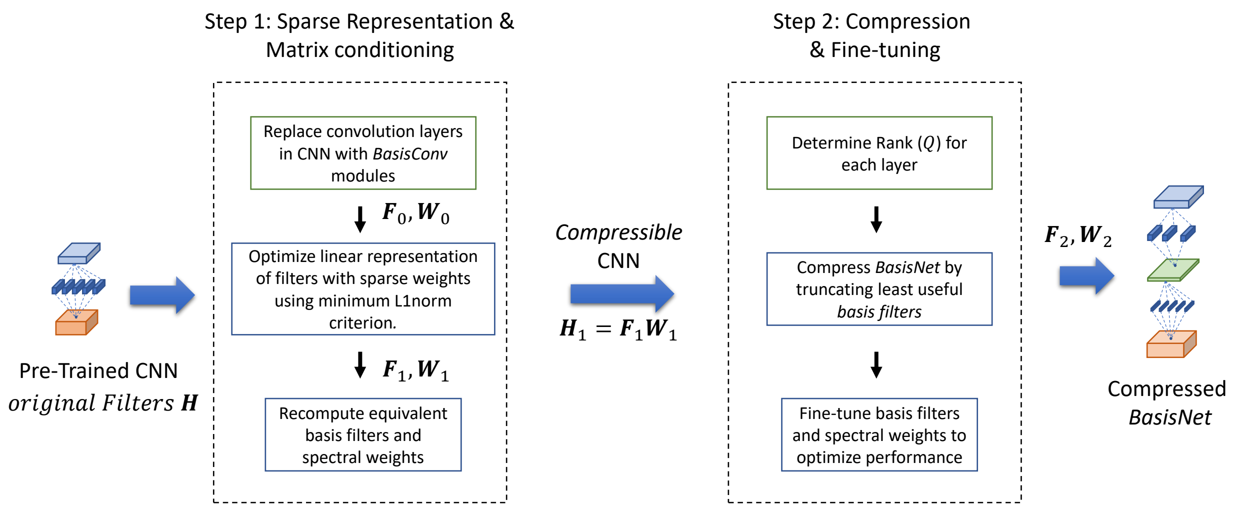

Step 1: Sparse Representation & Matrix conditioning
Replace convolution layers in CNN with BasisConv modules
F_0, W_0
Optimize linear representation of filters with sparse weights using minimum L1norm criterion.
F_1, W_1
Recompute equivalent basis filters and spectral weights
Pre-Trained CNN
original Filters H
Compressible CNN
$H_1 = F_1 W_1$
Step 2: Compression & Fine-tuning
Determine Rank (Q) for each layer
Compress BasisNet by truncating least useful basis filters
Fine-tune basis filters and spectral weights to optimize performance
F_2, W_2
Compressed BasisNet

Figure 3.2: Schematic diagram of our compression framework.

order tensor by a 1D quantity.

Our contributions in this chapter are as follows. i) We introduce a novel method, Sparse Spectral Representation and Fine Tuning (SSRFT), for optimizing and compressing convolutional neural networks (CNNs) by learning filters via sparse representation in a compact basis set. ii) We address a key challenge in CNN compression – determining the optimal rank for each layer of the network – by automatically reducing the rank of decomposed layer through minimization of the L1-norm of the 1D weight vectors. iii) We evaluate our method for compression and accuracy on both classification and object detection problems.

Method

As depicted in Figure 3.2, SSRFT has two key steps: i)learning a sparse representation for the original filters that inherently favors rank reduction and compression, and ii) joint optimization of the basis and spectral weights to maximize the performance of the compressed network. Given a pretrained model, we replace its convolution layers with *BasisConv* modules. The components of these modules, (i.e. *intermediate filters*) are then fine-tuned while minimizing the L1 norm of the linear combination weights. We will show that this decreases the non-zero eigen-values of the decomposition, which ultimately favors rank reduction and compression.

In step 2, we re-transform the intermediate filters and weights into a new set of *basis filters* and *spectral weights*, and compress the network by eliminating eigen-vectors with eigen-values close to zero. Based on the optimization in step 1, most of the network's knowledge is concentrated in a few basis filters and rest can be truncated without significant loss in performance. Finally we fine-tune the basis space and the corresponding spectral weights to optimize the performance of the compressed network.

Consider the fundamental convolution operation in any given layer of a convolutional neural network depicted on the top in Figure 3.1. Assume that an input block of data $x(m, n, l)$ (such as the activations or output of the previous layer) is convolved with a set of *P* filters $h_k(m, n, l),\ k = 1 \ldots P$, where each filter is of size $D \times D \times L$. The output $y_k(m, n)$ can be expressed as

$$y_k(m, n) = x(m, n, l) \ * \ h_k(m, n, l), \qquad 1 \leq k \leq P \tag{3.1}$$

where $*$ represents the convolution operation. The bottom of Figure 3.1 shows how the same output can be obtained using two successive convolution stages. Here, we assume that the filters can be expressed as a weighted sum of *Q* linearly independent functions $f_i(m, n, l)$, such that

$$h_k(m, n, l) = \sum_{i=1}^{Q} w_k(i) \cdot f_i(m, n, l) \tag{3.2}$$

where $w_k(i)$ are the weights of linear combination. Using this representation, the output can be expressed as

$$y_k(m,n) = \sum_{i=1}^{Q} w_k(i) \cdot [x(m,n,l) * f_i(m,n,l)], \quad 1 \leq k \leq P \tag{3.3}$$

The key observation is that the *Q* intermediate convolution terms $z_i(m,n) = x(m,n,l) * f_i(m,n,l)$ need to be computed only once, and they are common to all P outputs $y_k(m,n)$. These can be stacked together to form the 3D intermediate result $z(m,n,i)$ while the weight can be treated as $1 \times 1 \times Q$ filter $w_k(i)$. Therefore, the outputs $y_k(m,n)$ are simply the convolution of the two terms, i.e

$$y_k(m,n) = w_k(i) * z(m,n,i), \qquad 1 \leq k \leq P \tag{3.4}$$

We refer to this construct as *BasisConv* when $f_i(m,n,l), 1 \leq i \leq Q$, form a basis, and the model obtained by replacing all convolution layers with this module is called *BasisNet*. In practice, we also introduce a batch normalization operation in Eq. 3.4 to improve convergence so that the output is given by

$$y_k'(m,n) = w_k(i) * BN[z(m,n,i)], \qquad 1 \leq k \leq P \tag{3.5}$$

Here BN is the batch normalization layer which is applied to the batch dimension of z.

Step 1: Sparse Representation and Matrix Conditioning

Our goal is to learn a *sparse* representation of the original filters in a linear subspace that favors rank reduction and compression. To achieve this, we define the $LD^2 \times 1$ dimensional vector $\mathbf{h_k}$ as a vectorized representation of $h_k(m,n,l)$, and the matrix $\mathbf{H} = [\mathbf{h}_1 \ \mathbf{h}_2 \ ... \ \mathbf{h}_P]$ with $\mathbf{h}_k$ as its columns. We also define the $LD^2 \times 1$ dimensional vector $\mathbf{f}_i$ as a vectorized representation of the intermediate filters $f_i(m,n,l)$. We then construct the matrix $\mathbf{F} = [\mathbf{f}_1\ \mathbf{f}_2\ ...\ \mathbf{f}_{Q_0}]$, where Q_0 is the initial number of intermediate filters prior to compression. Finally, we define the 1D weight vectors $\mathbf{w_k} = [w_k(1)\ w_k(2)\ ...\ w_k(Q_0)]^T, 1 \leq k \leq P$. The relation between the original and intermediate filters is simply given by $\mathbf{H} = \mathbf{FW}$, where $\mathbf{W} = [\mathbf{w}_1 \mathbf{w}_1\ ...\ \mathbf{w}_P]$. Thus, the columns of $\mathbf{H}$ are represented as linear combinations of the columns of $\mathbf{F}$, weighted by the elements of $\mathbf{W}$

To promote sparsity during learning, the cost function is modified as follows. Assume that there are *B* layers in BasisNet. Denoting the layer by the superscript ^{j}, we introducing sparsity in $\mathbf{W}^j$ while fine-tuning both $\mathbf{F}^j$ and $\mathbf{W}^j$, $1 \leq j \leq B$ by minimizing

$$\mathcal{L}_{task} + \frac{\alpha}{B}\sum_{j=1}^{B}\left(\frac{1}{P^j \cdot Q_0^j}\sum_{k=1}^{P^j}|\mathbf{w}_k^j|\right) \tag{3.6}$$

Here $\mathcal{L}_{task}$ is the task specific loss function that the network was originally trained with. Furthermore, P^j and Q_0^j represent the number of original and intermediate filters, respectively, in the *j-th* layer. Essentially, the second term in Eq. 3.6 computes the average L1 norm of all 1D weight vectors in the model and α is the weight given to this term in the overall loss function. After convergence, many elements $\mathbf{w}_k^j$ will zero or very small. This increases the condition number of the matrix of filters, and thus favor rank reduction in Step 2.

For the purposes of simplicity, we drop the supercript j with the understanding the following hold for all layers. For each layer, we initialize $\mathbf{F}$ by setting it columns equal to the eigen-vectors of $\mathbf{HH}^T$ denoted by $\phi_i, 1 \leq i \leq LD^2$. Although $\mathbf{F}$ is not required to be a orthonomal matrix, this provides a good choice of linearly independent vectors at initialization. We arbitrarily choose the first Q_0 eigen-vectors, $P \leq Q_0 \leq LD^2$, so that at initialization $\mathbf{F}_0 = [\phi_1, \phi_2, ..., \phi_{Q_0}]$, and $\mathbf{W}_0 = \mathbf{F}_0^T\mathbf{H}$. After convergence, we denote the intermediate filters and sparse weights obtained at the end of Step 1 by $\mathbf{F}_1$ and $\mathbf{W}_1$ (for each layer).

Effect of Sparsity on Matrix rank and eigen-values

To show that introducing sparsity into $\mathbf{W}$ can lower the rank of $\mathbf{H}$, we write $\mathbf{H}\mathbf{H}^T = \Phi\Delta\Phi^T$, where the columns of Φ are the eigenvectors ϕ_i, and the Δ is a diagonal matrix whose diagonal elements are the corresponding eigenvalues λ_i. Since $\mathbf{H}\mathbf{H}^T = \mathbf{F}\mathbf{W}\mathbf{W}^T\mathbf{F}^T$, it follows that

$$rank(\mathbf{F}\mathbf{W}\mathbf{W}^T\mathbf{F}^T) = rank(\Phi\Delta\Phi^T) = rank(\Delta) \tag{3.7}$$

Using the fact that $rank(\mathbf{F}\mathbf{W}\mathbf{W}^T\mathbf{F}^T) = rank(\mathbf{F}\mathbf{W})$, we obtain

$$rank(\Delta) = rank(\mathbf{F}\mathbf{W}) = min[rank(\mathbf{F}), rank(\mathbf{W})] \tag{3.8}$$

Since the rank of **F** and **W** are Q_0 and P respectively, and since $P \leq Q_0$, we arrive at

$$rank(\Delta) = rank(\mathbf{W}) \tag{3.9}$$

Ideally, the goal is to minimize $rank(W)$. However, it has been shown [22] that for symmetric positive definite matrices, surprisingly good results are obtained by minimizing the trace in lieu of the rank. Heuristically, the non-convex rank objective can be replaced with the sum of the eigen-values (which is the dual of the spectral norm) and minimized. Now let $\gamma_i, 1 \leq i \leq n$, be an eigenvalue of $\mathbf{W}^T\mathbf{W}$. Since the trace of symmetric matrix is equal to the sum of its eigenvalues, we note that

$$\sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij}^2 = \sum_{i=1}^{n} \gamma_i \tag{3.10}$$

Since $\sqrt{\sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij}^2} \leq \sum_{i=1}^{n}\sum_{j=1}^{n} |w_{ij}|$, we obtain

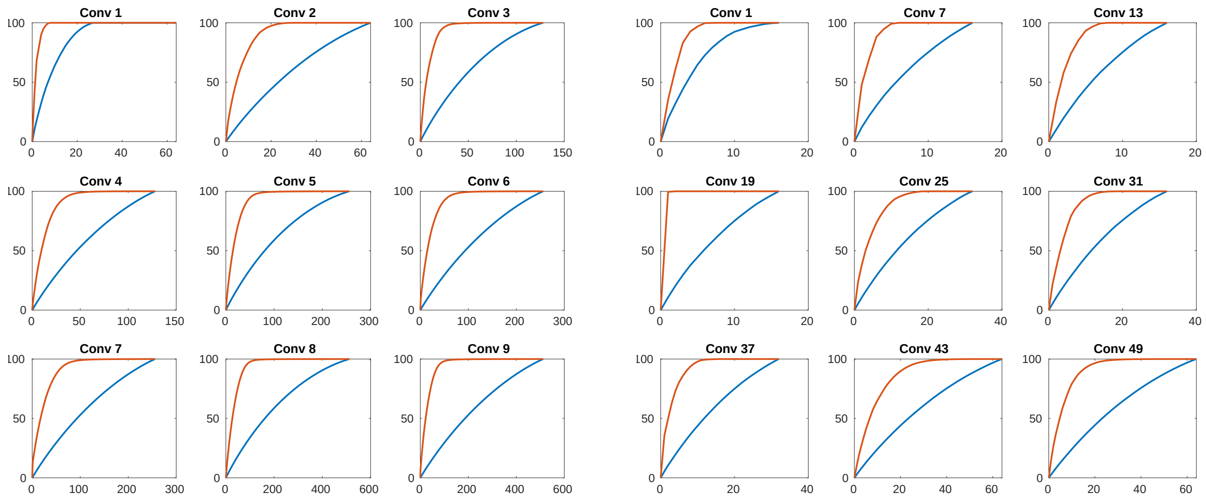
$$\sqrt{\sum_{i=1}^{n} \gamma_i} \leq \sum_{i=1}^{n}\sum_{j=1}^{n} |w_{ij}| \tag{3.11}$$

Therefore, minimizing the right hand side of Eq 3.11 minimizes an upperbound on the sum of the eigen-values of $\mathbf{W}\mathbf{W}^T$. This forces many of the eigen-values γ_i to become very small or go to zero, which also increases the condition number of $\mathbf{W}$. Since the rank of $\mathbf{W}$ is now easily reduced by simply eliminating the eigenvectors associated with these trivial eigenvalues, we conclude (based on Eq 3.9) that this process also reduces an upper bound on the rank of $\mathbf{H}$ as well as its decomposition $\mathbf{FW}$.

Step 2: Compression and Spectral Fine-tuning

Given the intermediate filters and sparse weights obtained in Step 1, our goal is now to compress the network by discarding redundant basis filters, thereby making Q small. This is a key step in the process which governs the trade-off between compression and accuracy. We can either retain smaller number of intermediate filters in each layer to achieve more compression at the cost of lower model accuracy, or we can sacrifice the compression in favour of accuracy by allowing Q to be larger. Fortunately, the optimization in Step 1 makes this trade easier by inducing a sharper roll-off in the eigenvalues, and increasing the number of eigenvectors that can be safely discarded.

After Step 1, the eigenvectors ϕ_i and eigenvalues λ_i of $\mathbf{H}_1 = \mathbf{F}_1\mathbf{W}_1$ are recomputed for each layer. The eigen-values are then sorted are in descending order, and the first Q_t eigenvectors are selected



(a) VGG16 (b) Resnet56

Figure 3.3: Comparison of the normalized cumulative sum of eigenvalues (denoted by S in Equation 3.12) for the filters trained with sparsity in Step 1 (red curve), and those of the original baseline network (blue curve). X-axis represents the eigenvalue number, while the y-axis the corresponding value of S. It is clear that imposing sparsity on $\mathbf{W}$ results in a more compact representation, and fewer eigenvectors are needed for the red curve to reach 100%. compared with the blue curve.

such that the normalized cumulative sum

$$S = \frac{\sum_{i=1}^{Q_t} \lambda_i}{\sum_{i=1}^{LD^2} \lambda_i} \tag{3.12}$$

is equal to a preset threshold *t* between 0 and 1, while the remaining $LD^2 - Q_t + 1$ eigenvectors are discarded. The value of Q_t in each layer depends on the the decay of the eigenvalues in that layer. The intermediate filters of the compressed network are re-initialized as $\mathbf{F}_2 = [\phi_1, \phi_2, ..., \phi_{Q_t}]$, and the 1D weight vectors as $\mathbf{W}_2 = \mathbf{F}_2^T \mathbf{H}_1$

Having thus compressed the network, our goal is to now re optimize its performance by fine-tuning $\mathbf{F}_2$ and $\mathbf{W}_2$. However, to ensure that the intermediate filters remains a basis set, the network is fine-tuned to minimize

$$\mathcal{L}_{task} + \eta \sum_{l=1}^{B} \mathcal{G}_l \tag{3.13}$$

where $\mathcal{L}_{task}$ is the task specific loss the original network was trained with, and $\mathcal{G}_l$ for the l-th layer is defined as

$$\mathcal{G}_l = \frac{\delta}{Q_{tl}} \sum_{i=1}^{Q_{tl}} (1 - \mathbf{f}_{il}{}^T \mathbf{f}_{il})^2 + \frac{2(1-\delta)}{Q_{tl}(Q_{tl}-1)} \sum_{i=1}^{Q_{tl}} \sum_{j=i+1}^{Q_{tl}} (\mathbf{f}_{il}{}^T \mathbf{f}_{jl})^2 \tag{3.14}$$

Here, Q_{tl} is the number of intermediate filters $\mathbf{f}_{il}$ in the l-th layer, and δ is a positive number between 1 and 0. The first term in $\mathcal{G}_l$ causes the L2 norm of $\mathbf{f}_{il}, i = 1...Q_{tl}$ to be as close to unity as possible, while the second term ensures the filters remain orthogonal. The parameter δ can be set to 0.5 to equally emphasize both term, or selected as necessary to balance the trade-off between the two terms. Finally η is a constant that balances the two components of the overall cost function in Equation 3.13. The final *optimized* basis filters and spectral weights obtained after the convergence of Step 2 are denoted by $\mathbf{F}_{opt}$ and $\mathbf{W}_{opt}$

It is easy to show that if the columns of **F** form a basis, then $\mathbf{W}\mathbf{W}^T = \mathbf{\Delta}$ is a diagonal matrix and

$$\sum_{k=1}^{P} w_k(i)^2 = \lambda_i, \text{ and } \sum_{k=1}^{P} w_k(i) w_k(j) = 0 \tag{3.15}$$

Therefore the convergences of each element of $\mathbf{w_k}$ is statistically uncorrelated with the behavior of the other elements, which implies that the corresponding eigen vector also contributes to the overall learning process in an uncorrelated manner.

Estimated Computational Reductions

Depending on the values of P and Q_t, the proposed compression scheme lead to substantial reduction in the number of multiplication operations. Specifically, let O represents the number of multiplications for one convolution operation (between $x(m,n,l)$ and either $h_k(m,n,l)$ or $f_i(m,n,l)$). If the size of the filters is $D \times D \times L$, and the size of the input data is $M \times N \times L$, It is easy to show that $O = LD^2(M-D+1)(N-D+1)$. Therefore, multiplications required in Eq. (3.1) is

$$A = PO = PLD^2(M-D+1)(N-D+1) \tag{3.16}$$

while multiplications required to obtain output of Eq (3.4) is

$$\begin{aligned} B &= Q_tLD^2(M-D+1)(N-D+1)+ \\ &PQ_t(M-D+1)(N-D+1) \\ &= Q_t[L^2+P](M-D+1)(N-D+1) \end{aligned} \tag{3.17}$$

We see that the ratio of the two is

$$\begin{aligned} \frac{A}{B} &= \frac{PLD^2(M-D+1)(N-D+1)}{Q_t[LD^2+P](M-D+1)(N-D+1)} \\ &= \frac{PLD^2}{Q_t[LD^2+P]} \end{aligned} \tag{3.18}$$

Thus, as long as $LD^2 >> P$, the number of multiplications will be reduced by a factor close to P/Q_t (i.e. the ratio of the the original number of filters and the number of basis filters used). This condition is generally true since the dimensionality of the filters is greater than the total number of filters. As an example, consider a typical case where $D = 5$, $L = 100$, and $P = 50$. Then LD^2 =2500, which is much larger than P. Therefore for most realistic case, the condition $LD^2 >> P$

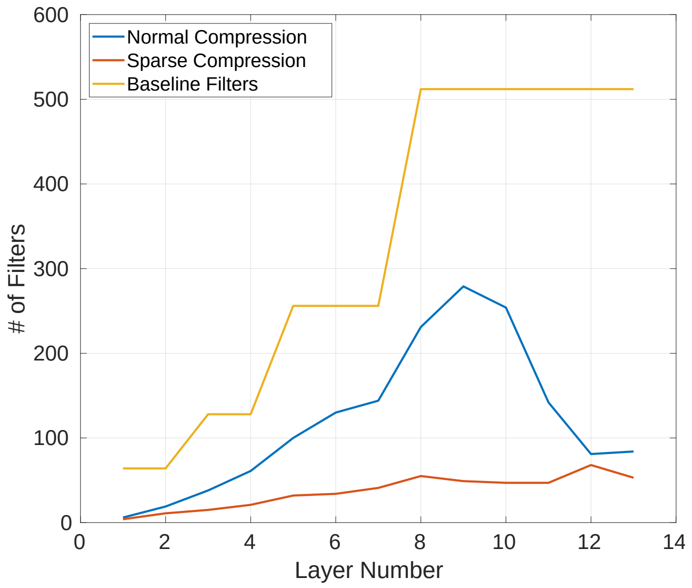
is readily satisfied, and the compression ratio can be predicted by P/Q_t. If in some rare instances this is not satisfied, compression ratio will depend on the collective rank of the filter set, and is therefore a data dependent quantity.

The architecture shown in Figure 3.1 not only reduces the filter storage and multiplications required for each convolution layer, but is also amenable to spectral fine-tuning where the number of *learnable parameters* is substantially reduced. Recall that number of learnable parameters in the original filters is LD^2. Since there are P such filters, the total number of original learnable parameters is PLD^2. However, the total number of ”learnable” parameters for BasisConv is $Q(LD^2+P)$ (depicted in Figure 3.1 as a Q basis filters with LD^2 parameters and P one-dimensional filters of length Q). Therefore, the reduction in the number of learnable parameters is $PLD^2/Q(LD^2+P)$. If $LD^2 >> P$, the number of learnable parameters are also reduced by a factor of P/Q.

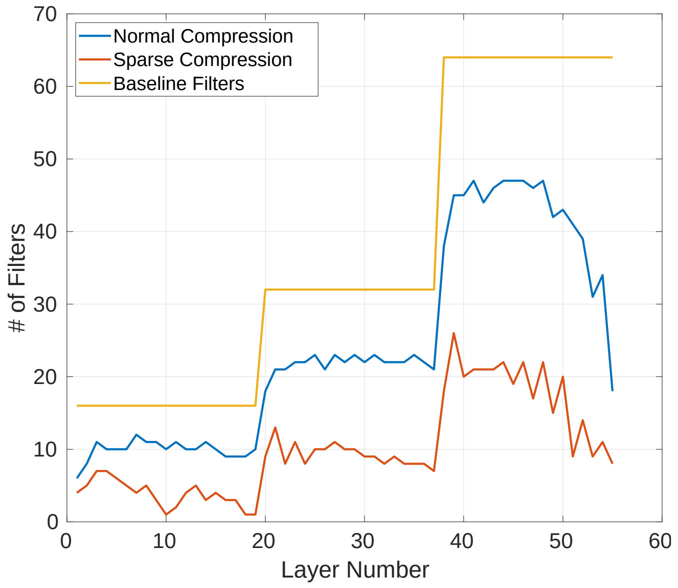
Experiments

Data Sets and CNN Architectures: We evaluated our method’s performance by compressing various CNN models designed for image classification and detection. We used three well known image classification architectures; *VGG* [93], *Resnet* [28] and *Densenet* [38]. For object detection we used *YOLOv3* [84]. These network architectures have little in common from number of layers and filters to the use of skip connections and loss function, but still our method applies to them without any special modification. Additionally these architectures and their variants are widely used for several applications across different industries so making them efficient have real world benefits.

For evaluating classification models we performed most of our experiments on *CIFAR10*[53] and *ImageNet*[47] datasets, while for some comparissons we have used *MNIST*[56], *SVHN*[78] and



(a) VGG16



(b) Resnet56

Figure 3.4: The proposed approach drastically reduces the number of filters required to achieve $S = 0.8$ for the various layers of the VGG16 (a) and RenNet56 (b) networks. The yellow curve shows the original number of filters, while the red and blue curves show the number of filters obtained with and without the optimization in Step 1.

CIFAR100[53]. For object detection we have used MS-COCO [68] dataset. Following is a brief description of these datasets.

- *MNIST* [56] contains 28×28 gray scale images divided two sets of 60,000 training and and 10,000 test samples.
- *SVHN* [78] contains 32×32 colored images divided two sets of 73,257 training and and 26,032 test samples.
- *CIFAR10 and CIFAR100* [53] are both image classification datasets and contain 60,000 colored images each, of 32×32 pixel resolution. CIFAR10 has 10 object classes while CIFAR100 has 100 classes.
- *ImageNet* [47] contains 1.28 million training images and 50,000 test images for 1000 classes.
- *MS-COCO* [68] is an object detection dataset. It contains 117263 training and 5000 validation images.

Evaluation Criteria: The widely used metrics for comparing model's compression are number of trainable parameters and number of Floating Point Operations (FLOPs). As noted in [9] there are some ambiguities in how these metrics are used because different papers have their own definitions. In an effort to remove any ambiguity we define FLOPs as number of multiply operations in the convolution layers of the model, while the number of parameters include all trainable parameters in the network. Finally the percent reduction in both metrics (denoted by %↓) is defined as $100 * (1 - \frac{Compressed}{Original})$.

Table 3.1: Results on CIFAR10.

Method	**FLOPs ↓%**	**Params ↓%**	**Acc. %**
ResNet-56			
Zhao et. al. [120]	20.30	20.49	92.26
Li et. al. [60]	27.60	13.70	93.06
NISP [114]	43.61	42.60	93.01
DCP [121]	47.01	70.32	93.81
CP [33]	50.00	-	91.80
AMC [34]	50.00	-	91.90
CaP [75]	50.20	-	93.22
GBN [112]	60.10	53.50	93.43
LFPC [31]	52.90	-	93.24
FPGM [30]	52.60	-	93.49
HRank [64]	50.00	42.40	93.17
GAL [67]	60.20	65.90	91.58
BasisNet	61.00	56.39	93.34
ResNet-110			
Zhao et. al. [120]	36.44	41.27	92.96
Li et. al. [60]	38.60	32.40	93.30
NISP [114]	43.78	43.25	93.35
CaP [75]	50.10	-	94.14
LFPC [31]	60.30	-	93.79
FPGM [30]	52.30	-	93.85
HRank [64]	58.20	59.20	93.36
GAL [67]	48.50	44.80	92.74
Idelbayev et. al. [41]	62.69	-	93.38
BasisNet	61.18	57.95	93.56
VGG-16			
Zhao et. al. [120]	39.10	73.34	93.18
Li et. al. [60]	34.20	64.00	93.40
DCP [121]	65.03	93.58	94.57
HRank [64]	65.30	82.10	92.34
GAL [67]	45.20	82.20	93.42
Slimming [69]	51.00	88.50	93.80
Idelbayev et. al. [41]	65.63	-	93.88
BasisNet	67.99	83.96	93.81
DenseNet-40			
Zhao et. al. [120]	44.78	59.67	93.16
HRank [64]	61.00	53.80	93.68
GAL [67]	71.40	75.00	93.23
Slimming [69]	55.00	65.20	94.35
BasisNet	60.74	55.35	93.59

Results on CIFAR10

For CIFAR10 dataset we evaluated our method on four different models. These are ResNet-56, ResNet-110, VGG-16 and DenseNet-40. These networks were trained for 60 epochs in Step 1, and for approximately 200 epochs in Step 2 with an initial learning rate of 0.01. As shown in Table 3.1, for ResNet-56 and ResNet-110 our method achieves almost 61.0% FLOPs reduction with 93.34% and 93.56% classification accuracy respectively, which is better then recent methods like GAL [67] and Hrank [64] both in terms of network compression and accuracy. Comparing with other methods our framework achieves highest FLOPs reduction with slightly worse accuracy. Similarly for VGG-16 out method outperforms both [67] and Hrank [64] by compressing the model to achieves 68% FLOPs reduction with 93.8% classification accuracy. Finally for DenseNet-40 our compressed model achieves performance and compression comparable to the other pruning methods.

As discussed in Section 3 minimizing the L1-norm of the weights of linear combination promotes sparsity in their values, which in turn forces the model to learn a compressible and compact representations. This is demonstrated in Figure 3.3(a) for the different layers of the VGG16 network used for obtaining the results in Table 3.1, while Figure 3.3(b) shows the same for some of the layers of the ResNet56 network (also from Table 3.1). Specifically, the red curves show the rapid rise of S as a function of the number of eigenvalues in the numerator of Equation 3.12 obtained at the end of Step 1. Similarly, the blue curves represents the behavior of S based on the eigenvalues associated with the original filters for the same layer of the network. Figure 3.4 compares the number of filters in the different layers of the original and compressed versions of the same VGG16 and ResNet56 networks. The number of filters in the compressed versions Q_t was automatically selected such that the ratio S was equal to 0.8. The red line represents the number of filters required to meet this criteria in each layer at the end of Step 1, while the blue line shows the same prior to optimization. The yellow line provides the original number of filters for comparison. The reduction in the number of filters required in each layer is evident from these plots. In step 2, the basis for these filters is further optimized for maximizing overall performance.

Results on ImageNet

VGG16: To evaluate our method's performance on VGG16 for ImageNet the pre-trained model is optimized in Step 1 for 60 epochs. The initial learning rate is 0.01 with step scheduler to reduce it by a factor of 10 every 20 epochs. This sparsely trained model attained 71.9% and 91.7% top1 and top5 accuracy respectively while the baseline accuracy is 71.6% top1 and 90.4% top5. We then discard the least important basis filters to obtain obtain three compressed models with 3x, 4x and 5x speedup ratios, and optimize their performance in Step 2. The results of this experiment are presented in Table 3.2. Compared with other methods our basis VGG16 achieves better Top-5 accuracy on all speedup ratios.

Table 3.2: Top-5 accuracy of VGG16 pretrained on ImageNet and compressed to 3x, 4x and 5x speedup ratios.

Method	**3×**	**4×**	**5×**
Jaderberg et. al. [45]	87.6	80.2	60.2
Zhang et. al. (asym.) [118]	89.5	89.0	87.9
Zhang et. al. (asym. fine-tuned) [118]	89.9	89.6	88.9
CP (fine-tuned) [33]	-	88.9	88.2
ELR [104]	-	89.4	88.8
GDP [65]	-	88.0	-
Li et. al. [60]	-	81.3	67.9
DDS [40]	-	88.2	-
Molchanov et. al. [77]	-	84.5	-
Kim et. al. [50]	-	-	89.4
BasisNet	**91.3**	**90.8**	**90.3**

Resnet50: A pretrained Resnet50 was optimized in Step 1 with a starting learning rate of 0.1. This

Table 3.3: Results for Resnet-50 pretrained on ImageNet.

Method	**FLOPs** (%↓)	**Params.** (%↓)	**Top-1 Accuracy** Baseline	Compressed	**Top-5 Accuracy** Baseline	Compressed
Alvarez et. al. [3]	27.00	-	74.70	75.20	-	-
SFP [32]	41.80	-	76.15	74.61	92.87	92.06
HRank [64]	43.77	36.67	-	74.98	-	92.33
TRP [109]	44.44	-	75.90	74.06	92.7	92.07
IMP [76]	44.99	44.53	76.18	74.5	-	-
LFC [96]	49.60	-	-	-	92.2	91.4
CP [33]	50.00	-	-	-	92.2	90.8
ELR [104]	50.00	-	-	-	92.2	91.2
GDP [65]	51.30	-	75.13	71.89	92.3	90.71
DDS [40]	52.35	38.82	76.12	71.82	92.86	90.79
FPGM [30]	53.50	-	76.15	74.83	92.87	92.32
GBN [112]	55.06	53.4	75.85	75.18	92.15	92.41
DCP [121]	55.56	51.45	76.01	74.95	92.93	92.32
ThiNet [70]	55.83	51.56	72.88	71.01	91.14	90.02
AFD [37]	56.00	52.73	76.1	75.42	-	-
LFPC [31]	60.80	-	76.15	74.46	92.87	92.04
GAL [67]	61.36	24.27	76.15	71.80	92.87	90.82
BasisNet-50	51.27	43.94	76.13	**76.34**	92.86	**93.05**
BasisNet-60	60.64	52.78	76.13	75.79	92.86	92.83

model with compact representation achieves 77.0% and 93.4% top1 and top5 accuracy respectively while the baseline accuracy is 76.1% top1 and 92.9% top5. We then compress this model by discarding the least useful filters to obtain two compressed models with 51.3% and 60.6% FLOPs reduction. We refer to these models as *BasisNet-50* and *BasisNet-60* respectively. The performance of these two version after Step 2 exceeds that of all others methods as shown in Table 3.3.

Comparisons with the well received previous methods like ThiNet[70], IMP[76], CP[33], FPGM[30], GBN[112], LFPC[31] and HRank[64] show that our method achieves more network compression with the better classification accuracy.

Resnet18: We train Resent18 with the same learning parameters as Resnet50 for the sparse-training phase. This compact model achieves 70.6% and 89.65% top1 and top5 accuracy respec-

Table 3.4: Results for Resnet-18 pretrained on ImageNet.

Method	FLOPs (%↓)	Params. (%↓)	Top-1 Accuracy		Top-5 Accuracy	
			Baseline	Compressed	Baseline	Compressed
SFP [32]	41.80	-	70.28	67.10	89.63	87.78
LCNN [21]	34.60	-	69.98	66.33	89.24	86.94
AFD [37]	42.50	33.10	69.76	68.91	-	-
AFD [37]	46.10	47.10	69.76	68.15	-	-
FPGM [30]	41.80	-	70.28	68.41	89.63	88.48
DCP [121]	50.00	47.10	69.64	67.35	88.98	87.60
Sampling [63]	29.30	43.80	69.74	67.38	89.07	87.91
TRP [109]	44.44	-	69.10	65.46	88.94	86.48
BasisNet-40	43.45	41.77	70.66	**69.58**	89.65	**89.25**
BasisNet-60	62.10	59.11	70.66	67.57	89.65	88.21

Table 3.5: Results of network compression on object detection problem: We compressed 2 predefined variants of YOLOv3 [84] architecture on MS-COCO [68] dataset.

Model	FLOPs (Billions)			Params (Millions)			MAP@0.5		
	Baseline	*Ours*	↓%	Baseline	*Ours*	↓%	Baseline	*Ours*	↓
YOLOv3-tiny	5.56	2.71	51.26	8.85	2.86	67.73	32.90	29.58	3.32
YOLOv3-416-2x	65.86	32.35	50.88	61.95	22.22	64.13	55.40	55.34	0.06
YOLOv3-416-3x	65.86	22.27	66.18	61.95	15.97	74.22	55.40	53.83	1.57

tively while the baseline models has 69.8% top1 and 89.1% top5 accuracy. We then compress this model to obtain two models with FLOPs reduction of 43.45% and 62.10%. Results of this experiment are summarised in Table 3.4.

Results on Object Detection

To demonstrate our method's effectiveness irrespective of the model architecture, dataset and the objective function, we tested it on object detection problem. We used 2 predefined variants of YOLOv3 [84] object detector pretrained on MS-COCO [68] dataset. These variants are YOLOv3-tiny [84], which contains 13 convolution layers and a much deeper model YOLOv3-416 [84],

Table 3.6: VGG16 inference time, power and energy consumption on NVIDIA Jetson AGX Xavier, with various compression factors of 2x, 3x, 4x and 5x

Method	**Baseline**	**2×**	**3×**	**4×**	**5×**
Inference time (ms)	40	30	26	25	23
Power (W)	6.01	5.55	5.15	5.08	4.94
Energy (Joule)	0.24	0.17	0.14	0.12	0.11
Top5 Accuracy (%)	90.4	-	91.3	90.8	90.3

which contains 75 convolution layers. We used code and pretrained models from [48] to serve as baselines. For YOLOv3-tiny our method was able to achieve 51.26% reduction in FLOPs and 67.73% reduction in model parameters.

We also compressed YOLO-416 to 2x and 3x speedup ratios. These compressed models attain 53.83 and 55.34 MAP score respectively, as compared to the baseline model which has 55.40 MAP. Detailed results are presented in Table 3.5.

Implementation On Edge Processors

We also measured the GPU speed up of our method on Nvidia Jetson Nano by compressing the VGG16 to reduce the FLOPs by 50% and 66%. Our method achieves 35.6% and 45.3% speedup respectively. Table 3.6 shows the inference time, power and energy consumption of VGG16 on the NVIDIA Jetson AGX Xavier (trained on ImageNet) using compression factors of 2x, 3x, 4x and 5x. We see that total energy use can be reduced in half with less than 1% loss in accuracy. Thus, we verified that the compressed model not only runs faster, but also requires less energy when implemented in edge processors that operate with limited resources such as GPU memory and power. It should be noted that we did not implement our method in C++ but used the python based Pytorch instead. We believe that an optimized C++ cuda implementation can achieve even better performance on GPUs.

Table 3.7: Comparison with DCFNet [81] (Fourier-Bessel bases and K=3): Our method significantly outperforms DCFNet in reducing the number of FLOPs and number of parameters, while also achieving better accuracy.

Model	**Dataset**	**FLOPs ↓%**		**Param ↓%**		**Accuracy (%)**	
		DCFNet	*Ours*	DCFNet	*Ours*	DCFNet	*Ours*
DCFNet (Conv-2)	MNIST	84.86	**88.40**	87.74	**98.30**	**99.40**	98.34
DCFNet (Conv-3)	SVHN	87.04	**89.40**	87.96	**99.20**	92.84	**93.12**
VGG16	CIFAR10	65.82	**82.10**	66.59	**99.10**	88.21	**91.07**

Additional Discussions and Insights

In this section, we provide additional discussions and insights into how basis representation and SSRFT enables compression with minimal performance loss. As described in section 2, several previous works have employed similar filter decomposition for the neural network compression. In DCFNet [81], authors assume that the filters can be represented using 2D Fourier Bessel functions, and optimize the corresponding spectral weights for this basis. To compare against DCFNet we first implemented and trained Conv-2, Conv-3 and VGG16 network architectures as described in their paper [81]. Next we compressed those baselines using our compression and SFT. In Table 3.7 we compare the compression achieved by our method to that of Fourier-Bessel bases variant of DCFNet, with K=3. It can been seen that our method significantly outperform DCFNet both in terms of reduction in FLOPs and reduction in number of parameters. For example, with VGG16 and the CIFAR10 data set, our method outperforms DCFNet by achieving greater reduction in FLOPs (82.10% vs. 65.82%) and number of parameters (99.10% vs. 66.59%) while offering greater accuracy (91.07% vs. 88.21%).

Another similar low-rank approximation technique has been proposed by Zhang et. al in [118]. To understand the key difference between our method and Zhang's, consider the convolution of an input image x with the filter kernel h which produces the output $y = h * x$. Zhang et. al

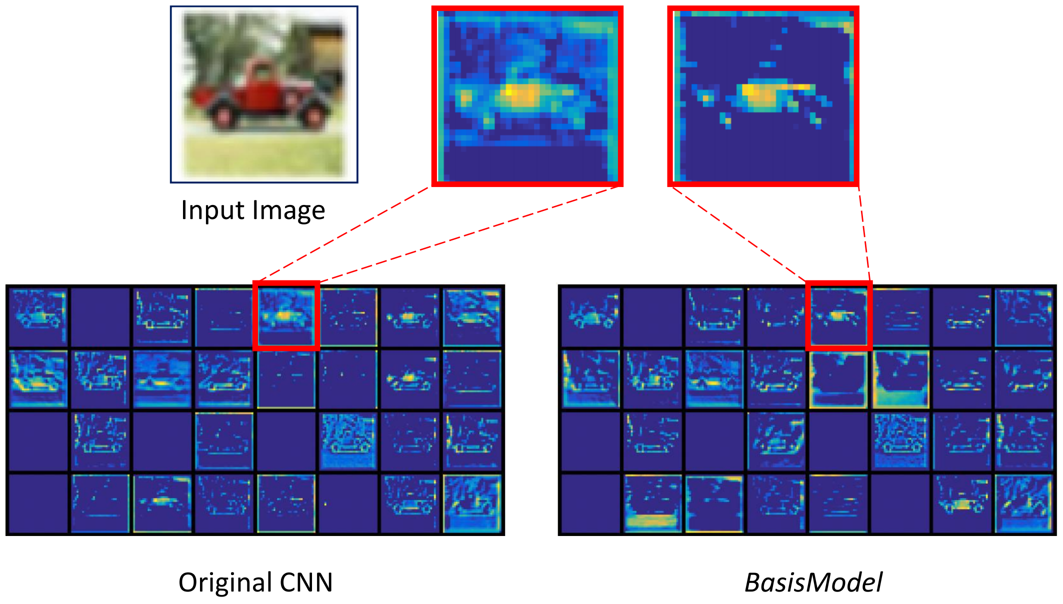


Figure 3.5: Comparison of first 32 activations for the first layer of orignal convNet and basisNet: Despite significant differences in reconstruction, basisNet achieves 80% compression in FLOPs with less then 1% performance difference.

[118] focus on the subspace of the filter output y. Assuming that y lie in a low-rank subspace, Zhang et. al apply SVD to the matrix $\mathbf{YY}^T$ (where each column of $\mathbf{Y}$ is a response of the filter to a particular training image), and find a linear combination of the eigen vectors of $\mathbf{YY}^T$ whose output approximates y. On the other hand, we pay no attention to the output y, but apply the basis representation to the filters h directly.

Table 3.2 compares our method with Zhang's, and shows the Top-5 accuracy on the ImageNet dataset using VGG-16 network compressed for 3x, 4x and 5x speedup ratios. Our compression and optimization technique outperform Zhang's method for all speedups ratios.

Whereas [118] and [45] (among others) optimize the spatial convolution filters to approximate the responses of the filters of the original network, we optimize the underlying basis and the represen-

Table 3.8: Effect of orthonormality constraint (eq. 3.14) on test accuracy for VGG-16 on CIFAR100. Higher value of **t** indicates less compression.

t	**Params.**	**Accuracy (%)**	
		$\mathcal{L}_{CE}$	$\mathcal{L}_{CE} + \mathcal{G}$
0.15	0.42M	31.05	34.05
0.25	0.80M	47.35	48.63
0.35	1.28M	55.06	56.34
0.45	1.90M	64.39	64.46
0.55	2.69M	66.51	66.59

tation of the filters in this basis. In fact, there is no expectation that the activations produced by the basisConv layer will be the same as those produced by the convolution layer of the original network. To see how the reconstructions of intermediate outputs is different for the compressed network and the original network we visualized the first 32 activations of first convolution layer and first basisConv layer in Figure 3.5. The zoomed insets show the input image used to generate these activations, and 5th activation of both networks are clearly visually quite different. In fact, careful comparison of corresponding activations show subtle yet noticeable differences between how responses of the BasisConv and conventional convolution layers respond. It is evident that each emphasizes and learn different image features in the manner necessary to optimize the overall performance of the network.

Effect of Orthogonality term in Loss Function: The purpose of the loss term in Equation 3.14 is to ensure the basis filters remain orthogonal and as close to unit norm as possible when SFT is applied. Figure 3.6 shows the covariance matrices for the basis filters for three of the thirteen basisConv layers in the compressed network, with the orthogonality constraint included on the left, and without it on the right. We can see that introducing this term does indeed decorrelates the optimized filters, thereby ensuring that they continue to form an orthogonal basis. As a result, the corresponding 1D filters continue to represent the *spectra* of equivalent spatial convolution filters

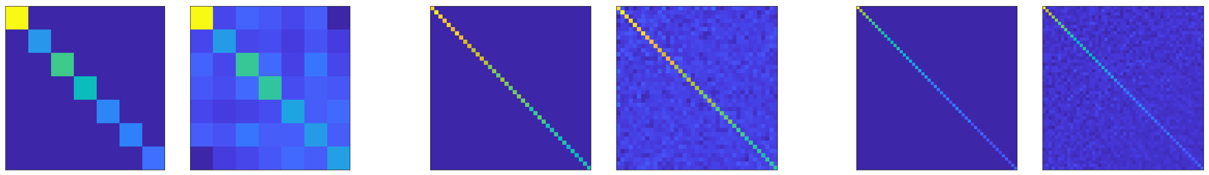

(a) Conv 1 (b) Conv 5 (c) Conv 10

Figure 3.6: Basis filter covariance matrices with (left) and without the normalization (right) term in loss: We have displayed covariance matrices for 3 of the 13 basisConv layers in compressed VGG16 pretrained on CIFAR100.

in this basis.

To better understand the impact of the orthogonality term in Equation 3.14, we compressed VGG16 pretrained on CIFAR100 with five different values of the threshold **t** applied to S (Equation 3.12) to obtain corresponding values of Q_t. We trained these compressed network with and without the orthogonality term $\mathcal{J}_l$ in the loss function in Equation 3.13. Results of this experiment are summarised in Table 3.8. We can see that the improvement due to the inclusion of the orthogonality term increases as we increase the compression. Therefore, imposing the orthogonality condition improves the performance of the compressed network and enables greater compression.

Conclusion

In summary, we have introduced a very general method for CNN compression that is conceptually straightforward, effective, and easy to implement, and yet achieves very competitive results compared to other (more intricate) state of the art techniques. As noted in [41], the real difficulty in compressing a network using low rank approximation in determining what the optimal rank of each layer is–effectively, an architecture search problem with one hyperparameter per layer. We have shown that this can be solved by i)learning a linear representation for the original filters with sparse weights that automatically favors rank reduction and compression, and ii) jointly optimizing the low-rank basis and filter spectra to maximize the performance of the compressed network. Referred to as SSRFT, the proposed two-step method first imposes sparsity on the linear representation of the filters to make the decomposition more amenable to compression and rank reduction. Specifically, we showed that minimizing the L1 norm of the weights of linear combination reduces a bound on trace of the weight matrix, forcing many of the corresponding eigenvalues to become small (or zero) in value. In turn, this makes it easy to compress the network by simply selecting a small number of basis filters using a fixed threshold, and then optimizing the low-rank basis and filter spectra to maximize performance . We used various network architectures (VGG16, ResNet, DenseNet, and YOLOV3) and different datasets (MNIST, SVHN, CIFAR-10, CIFAR-100, ImageNet, and MS-COCO) to show that our method consistently achieves significant reduction in the learnable parameters, storage size, and FLOPs while outperforming many other well-received state of the art methods for network compression. The results of these direct comparison with other state of the art methods show that the proposed method often achieves greater compression at competitive performance levels.

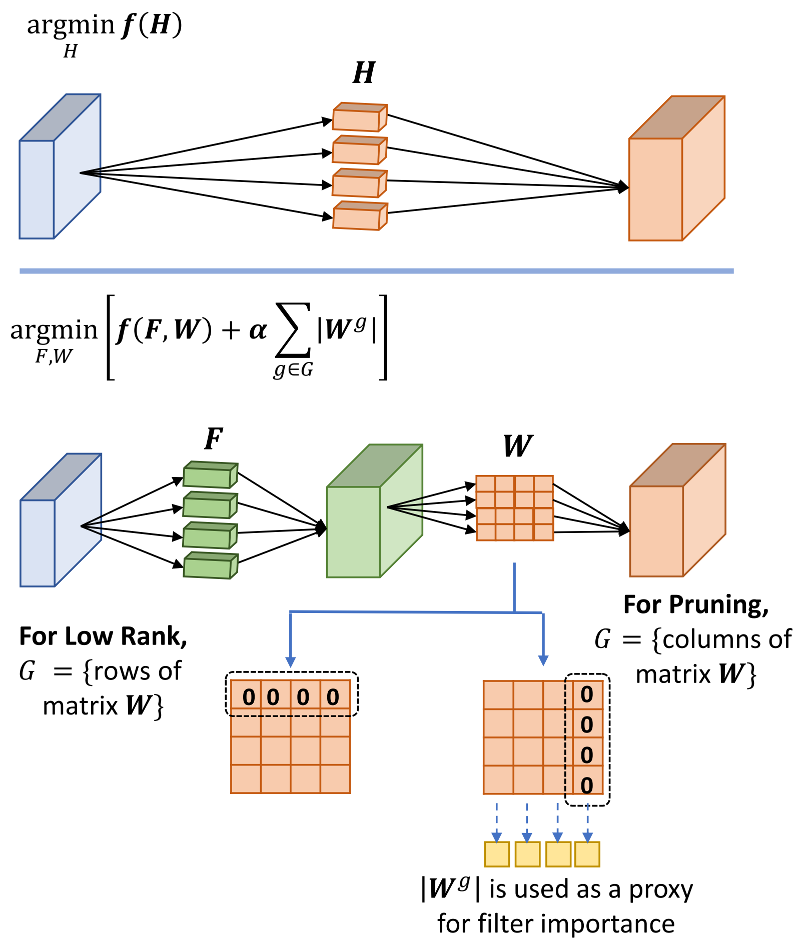
CHAPTER 4: SIMULTANEOUS LEARNING AND COMPRESSION

Introduction

Two popular neural network compression methods are filter pruning and low rank approximation. In this work we propose a unified framework for tackling both kinds of compression methods while the network is being trained. Conventional pruning typically starts with a filter importance score estimation, where a dedicated algorithm estimates the importance score of each pretrained filter in the model. There are several ways to estimate filter importance, some methods (i.e. [76, 114, 70]) estimate the contribution of each filter to some loss and use it as a proxy for importance while others estimate filter importance based on the statistics of the output feature map [64]. Least important filters are then removed from the model, either iteratively or all at once and the resulting compressed model then is fine-tuned to recover any loss of performance.

Low rank approximation compression methods [20, 3, 99, 118] start with the assumption that the pretrained convolutions filters form a rank deficient matrix. They exploit this fact by representing the filters in a given convolution layer by the weighted linear combination of a set of basis filters [118, 99]. The network layer then operates by applying the basis filters first on the input followed by the weighted linear recombination of the channels of intermediate output. Compression results from the fact that the decomposition represents the same information as the original filters in a much more compact fashion by discarding basis filters corresponding to small eigen values. This truncation of filters introduces some error in the model's prediction which is mitigated by subsequent fine-tuning.

Since both of these methods depend on filter truncation, it is important for their success that the original filters are compactly represented in the first place and the discarded filters contain little



$\underset{H}{\operatorname{argmin}} \boldsymbol{f}(\boldsymbol{H})$
$\boldsymbol{H}$
$\underset{F,W}{\operatorname{argmin}} \left[\boldsymbol{f}(\boldsymbol{F}, \boldsymbol{W}) + \boldsymbol{\alpha} \sum_{g \in G} |\boldsymbol{W}^g| \right]$
$\boldsymbol{F}$
$\boldsymbol{W}$
For Low Rank, G = {rows of matrix $\boldsymbol{W}$}
For Pruning, G = {columns of matrix $\boldsymbol{W}$}
0 0 0 0
0 0 0 0
$|\boldsymbol{W}^g|$ is used as a proxy for filter importance

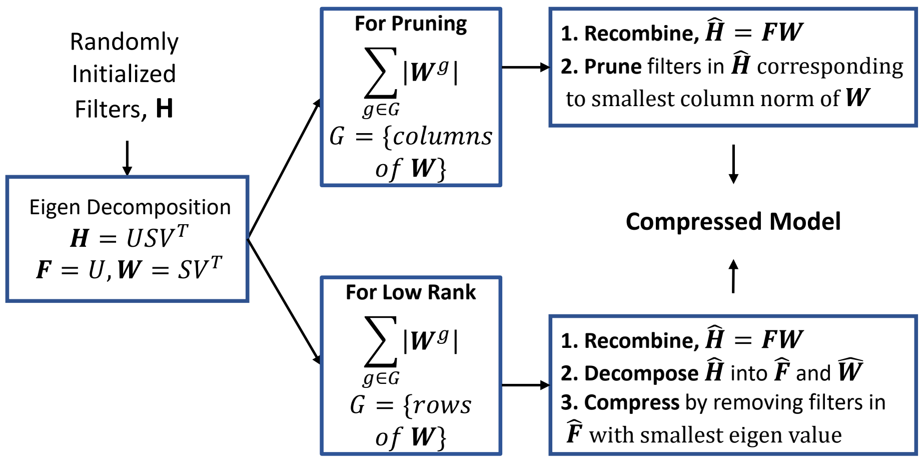
Figure 4.1: Randomly initialized filters are trained in eigen decomposed form with a weighted sum of task specific loss and group L1 constraint. Set G can be rows of $\mathbf{W}$ to obtain optimally compact recombined filters for low rank approximation or the columns for pruning based methods.

information. However conventional methods do not explicitly train filters for this purpose so any deletion of filters also discards useful information learned by the model. Recently Alvarez et. al. [3] have attempted to address this issue by training a compression aware model from scratch. However their method is only applicable to the compression methods based on low rank approximation and doesn't take pruning methods into consideration.

We show that our basisConv structure can be used for both low-rank approximation and pruning. Assume that the recombined filters are the columns of the matrix $\hat{\mathbf{H}}$, the basis filters are the columns of $\mathbf{F}$ and coefficient of linear combination are the columns of $\mathbf{W}$. The relation between these quantities is given by $\hat{\mathbf{H}} = \mathbf{FW}$. Taking inspiration from [61], we train the randomly initialized filters in eigen decomposed form with group L1 regularization on linear combination matrix $\mathbf{W}$. This in turn encourages sparsity in $\mathbf{W}$ and enables us to obtain optimally compact filters for low rank approximation and pruning based compression methods. The use of group L1 regularization enables us to force either entire rows or columns of $\mathbf{W}$ to be zero (depending on the choice of group). We will show that minimizing the L1 norm of the rows leads to low-rank approximation, while minimizing the column-wise L1 norm removes entire filters, and is eqivalent to pruning. This is different from vanilla L1 which does not impose structural constraints, but achieves overall smaller L1 norm with sub-optimal compactness.

Finally, our method also eliminates the need for a dedicated filter importance metric which is at the core of network pruning method. Instead, our method uses the column norm of $\mathbf{W}$ as a proxy for filter importance and deletes the filters corresponding to the smallest norms.

In this chapter, our contributions are as follows: (i) we present a framework for simultaneous training and compression which enables training, pruning, and rank reduction of CNNs filters in eigen space as the network learns from scratch. (ii) We show that group L1 regularization on recombination coefficients enables learning a compact filter representation suitable for network compression. (iii) We show that the L1 norm of the columns of the matrix $\mathbf{W}$ can be used a proxy for the filter importance score which can then be used to prune (delete) unimportant filters.



Randomly Initialized Filters, H
Eigen Decomposition
$H = USV^T$
$F = U, W = SV^T$
For Pruning
$\sum_{g \in G} |W^g|$
$G = \{columns\ of\ W\}$
For Low Rank
$\sum_{g \in G} |W^g|$
$G = \{rows\ of\ W\}$
1. Recombine, $\widehat{H} = FW$
2. Prune filters in $\widehat{H}$ corresponding to smallest column norm of W
1. Recombine, $\widehat{H} = FW$
2. Decompose $\widehat{H}$ into $\widehat{F}$ and $\widehat{W}$
3. Compress by removing filters in $\widehat{F}$ with smallest eigen value
Compressed Model

Figure 4.2: Schematic diagram of our compression framework.

Method

In this section, we describe our approach which aims to learn compact representation favourable for network compression. As depicted in Figure 4.2, our method first replaces randomly initialized filters denoted by **H** with a set of basis filters **F** and recombination coefficients matrix **W**. The model is then trained with group L1 regularization on **W** along with any task specific loss function (e.g. cross entropy loss for classification). Following [61] we applied group L1 regularization to the columns of **W** to train for pruning and rows of **W** for low rank approximation based compression. The addition of group L1 term to the overall loss function encourages entire groups of the elements **W** to become sparse. Finally we multiply the learned **W** and **F** matrices to obtain a compact approximation for the original filters **H**. These filters are finally compressed using standard pruning or low rank approximation techniques.

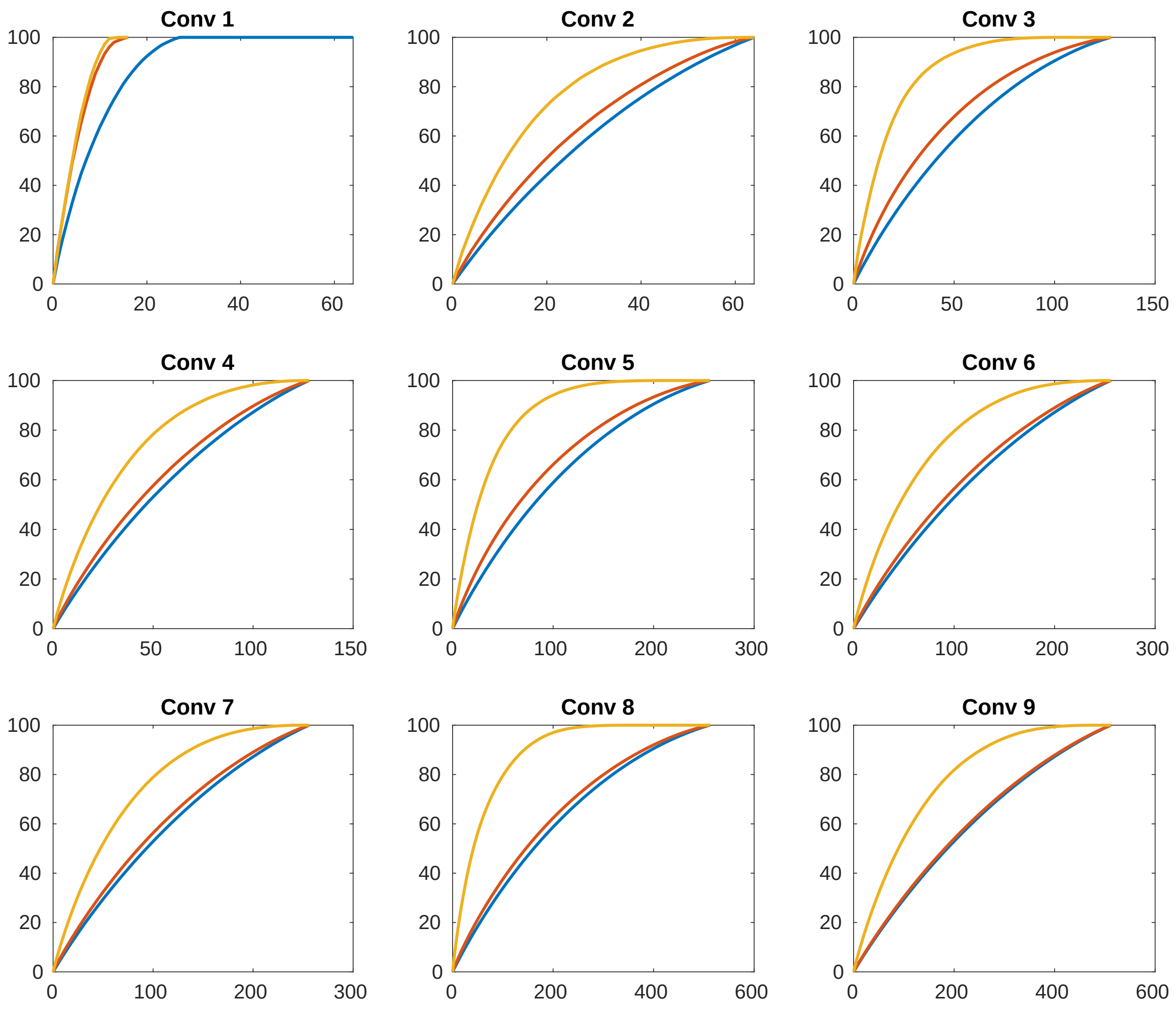


Figure 4.3: Comparison of the normalized cumulative sum of the eigen-values associated with the different layers of Vgg16 trained on CIFAR10. The model is trained with row wise L1 norm (yellow curve), vanilla L1 norm (red curve) and without any regularizer (blue curve). X-axis represents the eigen-filter number, while the y-axis is the normalized cumulative sum of eigen-values. Imposing groups wise sparsity on w results in a more compact filters, so fewer eigen-filters are needed for the yellow curve to reach 100%.

Training from scratch

To train form scratch we first need to calculate **F** and **W** from the randomly initialized filters. To understand how we do it, consider the convolution layer of a CNN, whose filters are of size

$M \times N \times D \times D$, where N and M are number of input channels and output channels respectively and the spatial dimension of the filters is $D \times D$. These filters are flattened and arranged as columns of the matrix $\mathbf{H} \in \mathbb{R}^{A \times M}$, where $A = ND^2$. Following [20, 99] we compute the singular value decomposition of $\mathbf{H}$. This lets us write it as $\mathbf{H} = \mathbf{USV}^T$ where $\mathbf{U} = \mathbf{F}$ and $\mathbf{SV}^T = \mathbf{W}$.

After decomposition, **F** and **W** are optimized to minimize the following cost function to train the model:

$$\underset{F,W}{\operatorname{argmin}} \left(f(\mathbf{F}, \mathbf{W}) + \frac{\alpha}{L} \sum_{j=1}^{L} \left(\sum_{g \in G} |\mathbf{w}_j^g| \right) \right) \tag{4.1}$$

Here $f(\mathbf{F}, \mathbf{W})$ is the task specific loss function (i.e. cross entropy loss for classification). L is the number of layers, and $|\mathbf{w}_j^g|$ is the L2 norm of the vector that belongs to G, the set of either rows or columns of **W**. Since the L2 norm is used for each member of the group, and then we minimize the L1 norm of these values collectively for the whole group, this is referred to as group L1 regularization or alternatively mixed norm [90]. So, the second term in Eq. 4.1 computes the average of group L1 norm of **W** in every layer in the model and α is the weight given to this term in the overall loss function.

After training, **F** and **W** are finally recombined to obtain the filter matrix $\hat{\mathbf{H}} = \mathbf{FW}$, which is now suitable for compression using with either pruning or low rank approximation based techniques.

Low rank approximation

Following the methods proposed by [20, 118] we again decomposed the filters $\hat{\mathbf{H}}$ with singular value decomposition to obtain the final basis filters $\hat{\mathbf{F}}$ and coefficient matrix $\hat{\mathbf{W}}$. As depicted in Figure 4.1 the original convolution layer is replaced with two convolutions, the first with $\hat{\mathbf{F}}$

followed by another with $\hat{\mathbf{W}}$, after appropriate reshaping of the filters. The model is now easily compressed by discarding the columns of $\hat{\mathbf{F}}$ (i.e. the basis filters) which correspond to the smallest eigen-values of the decomposition.

Pruning

For pruning, we first need to estimate the *importance* of each filter so that the least important filters can be identified and deleted. While other pruning methods [76, 64] propose complicated techniques to estimate the importance of each filter, we make use of the the linear combination matrix $\mathbf{W}$. Since $\hat{\mathbf{H}}$ is computed as the matrix product of $\mathbf{F}$ and $\mathbf{W}$, we can see that each filter in $\hat{\mathbf{H}}$ is the linear combination of basis filters in $\mathbf{F}$, with the columns of $\mathbf{W}$ acting as the weights of this linear combination. Therefore, the importance of any filter in $\hat{\mathbf{H}}$ can be estimated by just taking the column wise L1 norm of the $\mathbf{W}$ matrix. Once the filter importance is estimated, pruning becomes trivial by simply eliminating the least important columns of $\hat{\mathbf{H}}$ corresponding to the columns of $\mathbf{W}$ with smallest L1 norm.

Training from Scratch: Recall that our method replaces the convolution layers in each of the networks with the decomposition shown in Figure 4.1 prior to training. We then train the decomposed models **from scratch** as described in Section 4 for same number of epochs as the original models. We have observed that the decomposed model easily attains an accuracy within $\pm 1\%$ of the original model with some minor tweaking of the hyper-parameter α. To show that our method encourages compact representation with basis filters, in Figure 4.3 we compare the normalized cumulative sum of the eigen-values associated with the VGG-16 filters learned with group wise L1 norm, vanilla L1 norm and those without any regularizer. Thus, the faster the plot reaches 100%, the smaller the number of required filters is. It is clear that imposing group wise sparsity on $\mathbf{W}$ yields the most compact representation, with less eigen-filters required for any given value of the cumulative sum of the eigen-values.

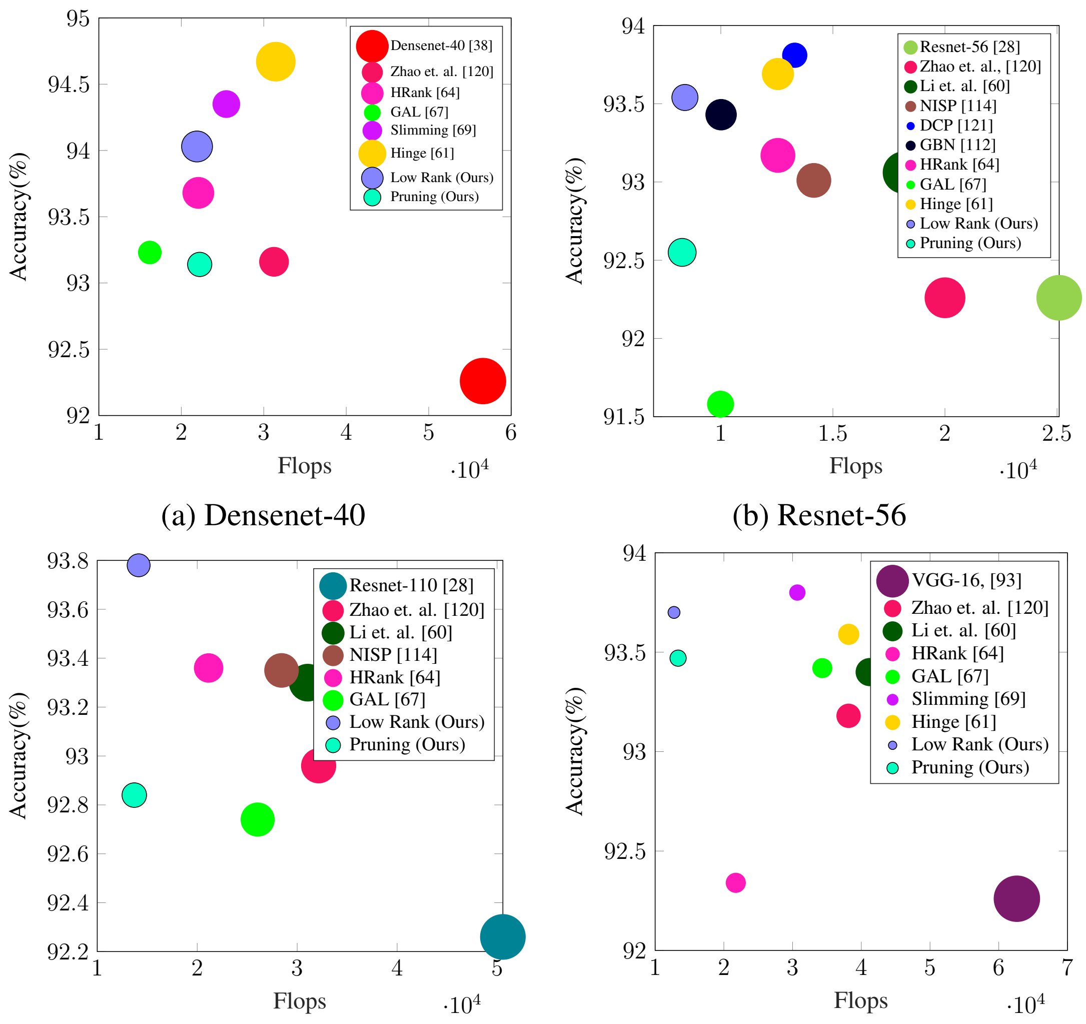
Dynamic Compression: As previously discussed, most neural network compression techniques are applied to pretrained models, which have not been designed with subsequent compression in mind. This approach typically results in a distribution of learned filters where information is almost uniformly dispersed. In contrast, our strategy of compression-aware training from the beginning facilitates the learning of compact filters. One significant advantage of this approach is that it allows us to compress the model without the need for any further fine-tuning, while still preserving high accuracy.

We apply dynamic compression in the low rank decomposed form. Once the model is trained from scratch using the method outlined in 4 we achieve dynamic compression by simply truncating the basis filters in $\hat{\mathbf{F}}$ and coefficients in $\hat{\mathbf{W}}$ corresponding to the smallest eigen values.

Experiments

In this section we present the evaluation of our method for learning compact model from scratch and compression with low rank and pruning methods. We evaluated our method on on CIFAR10 and ImageNet classification datasets. The network architectures investigated for compression are ResNet-18, ResNet-56, ResNet-110, VGG-16 and DenseNet-40.

Compression: After training, the models are compressed with both pruning (dropping recombined filters) and low rank methods (dropping basis filters), and the final fine-tuning step is performed to mitigate any performance loss due to filter deletion. All compression results and are summarised in Figures 4.4 for CIFAR10, and in 4.5 for ImageNet, and compared to other state of the art metods. Each method is represented with a colored circle, centered around the accuracy (y axis) and FLOPs (x axis) while the size of circle is proportional to the number of parameters. Since our



(a) Densenet-40 (b) Resnet-56

(c) Resnet-110 (d) VGG-16

Figure 4.4: Results on CIFAR-10

pre-training procedure enables learning of compact representations, we can discard large number of filters containing less information and achieve higher compression compared to other methods. In each case, we see that our low-rank method (labeled "Low Rank (Ours)") achieves the ***lowest*** flops with very competitive performance accuracy compared to other state of the art methods. Our pruned models (”Pruning (Ours)”) achieves even greater compression and reduction in FLOPS with slightly less accuracy compared to other methods, while also avoiding costly filter importance

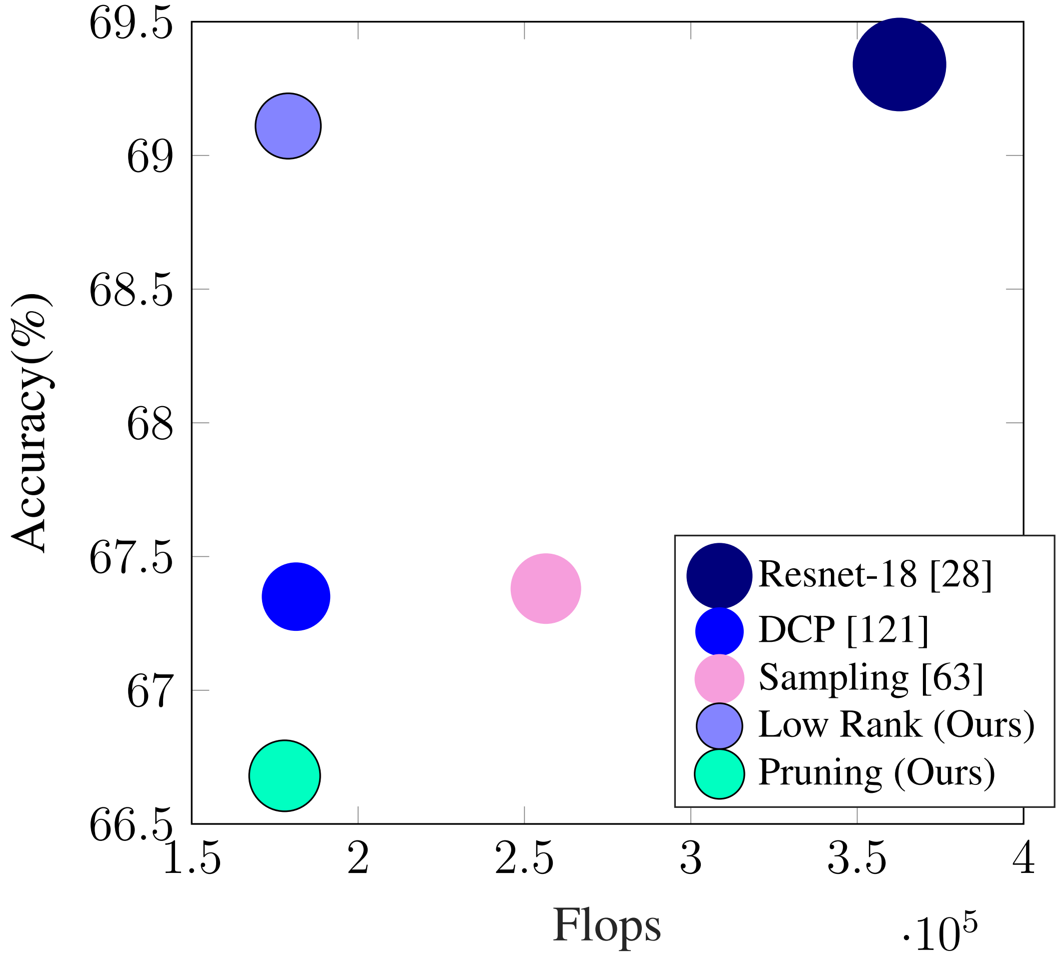


Figure 4.5: Results on ImageNet for Resnet-18.

estimation proposed in [64, 76, 114].

Dynamic Compression: As previously discussed, our training approach allows us to compress the model without the need for any further fine-tuning, while still preserving high accuracy. Conventional models, when subjected to filter truncation, often exhibit significant performance degradation. However, our method maintains high accuracy even when a substantial number of filters are removed from the model, as demonstrated in Figure 4.6.

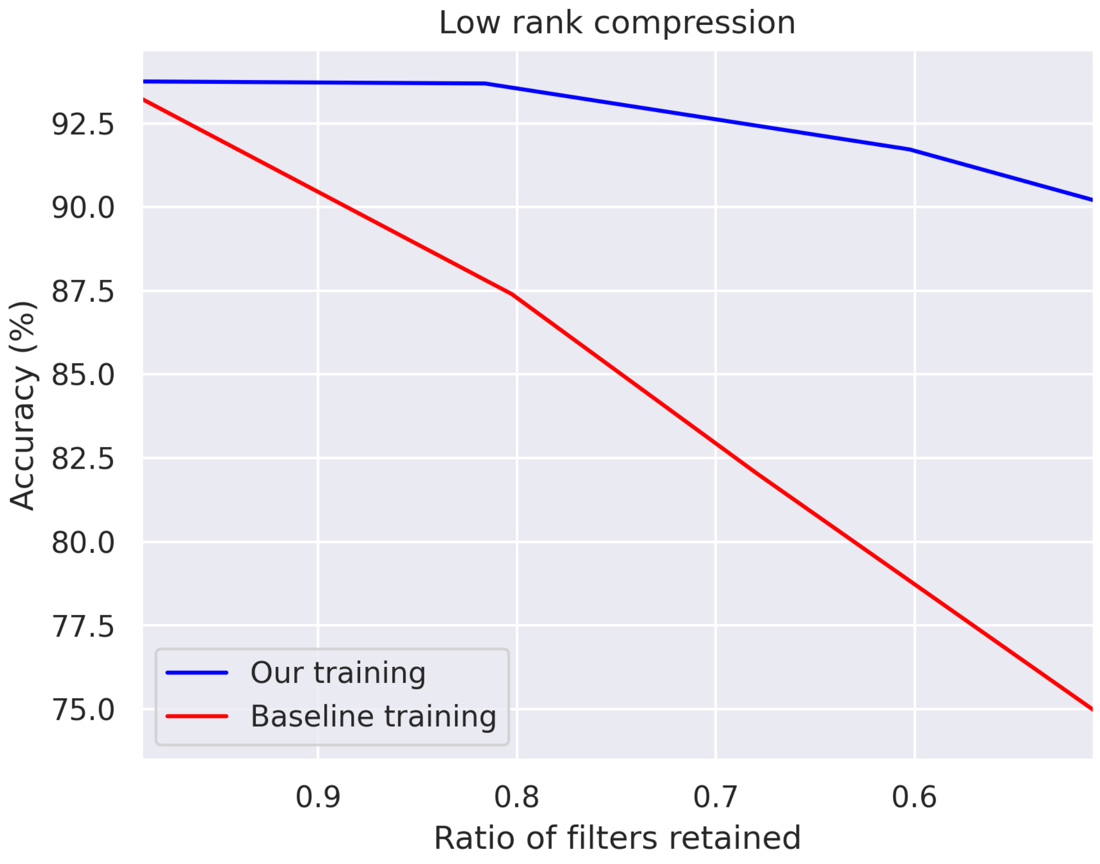


Figure 4.6: Result of dynamically compressing the model without any finetuning. Compact filters, which were learned through our training process, enable the model to maintain high accuracy level even when a significant number of filters are removed.

This unique attribute of our training scheme allows us to deploy various compressed versions of the model without requiring any additional fine-tuning on the data.

Conclusion

Traditional methods for network compression deal with conventionally trained networks that have a lot of redundant parameters and filters. Our method essentially allows a network to *self-compress* as it undergoes training. Essentially, a minimum group L1 norm criterion is imposed on a decomposed structure for the convolutional layers. By defining the group L1 norm either along the columns or along the rows of the coefficient matrix, the network can “self compress” by either deleting unimportant filters (pruning) or reducing their rank (compact representation). Finally, using the CIFAR-10 and ImageNet data sets along with several different network architectures, we showed that our method provides very competitive accuracies at **the lowest** number of FLOPs as compared to several other state of the art methods.

CHAPTER 5: COMPRESSED CONTINUAL LEARNING

Introduction

Recent progress in machine learning research has led to impressive advances and has enabled many practical applications of the deep learning models [28, 84]. These conventional methods however assume that all training data is available at once which is hardly the case in real world applications. For instance, imagine a warehouse robot tasked with visually scanning items. If traditional training pipeline is used, every time a set of new items classes are added to the inventory the model will need to be trained again from scratch, with all previous and newly available data. This process is not only time consuming but also needlessly computationally expensive. Additionally, previous training data may not be available now. Naively training the model on new dataset alone makes the model suffer *catastrophic forgetting*, where model forgets the previously learned representations and adapt to new data only. This limitation has hampered the widespread adaption of deep learning methods.

The strategies for addressing this problem are referred to as *continual learning* or *lifelong learning* in the literature and are an active area of research. Lifelong learning aims to train a model incrementally, as new data becomes available while also preserving the previously learned representations. To solve this problem several methods have been introduced, and among them, network expansion based methods have shown potential. These methods add dedicated parameters to the model for each new task which can be used to calculate the task specific feature maps, thus avoiding forgetting. For example Rusu et al. [87], Yoon et al. [111] and Jerfel et al., 2019 [46] have proposed incrementally adding parameters to the model for each new task. Vinay et. al. [102] suggested adding several task specific feature map transformation layers to the model which require a small number of additional parameters. Similarly, Miao et. al. [74] proposed a low rank sub-space based approach which decompose original filters into a set of task specific atoms and shared coefficients. Task specific filters are obtained at inference time by multiplying the shared coefficients with corresponding filter atoms.

We propose a filter decomposition based approach to address the problem of life-long learning while avoiding catastropic forgetting. Our method enables knowledge sharing between tasks with a set of shared basis filters while task specific coefficients enable the model to compute task specific feature maps. We leverage the low rank approximation of convolution filters $\mathbf{H}$ to decompose them into compact basis filters $\mathbf{F}$ and coefficients $\mathbf{w}$. We share the filters $\mathbf{F}$ among all tasks while new coefficients are added to the model for each additional task. In contrast to Miao et al. [74], task specific feature maps are computed by a sequence of convolutions with $\mathbf{F}$ followed by $\mathbf{w_t}$, as depicted in Fig. 5.2. Finally choosing compact basis to represent $\mathbf{H}$ lets us get away with using a small number of basis filters. This enables significant reduction in FLOPs and number of parameters in the model.

While the formulation introduced above performs well compared to similar methods, it has one notable drawback: since the basis filters $\mathbf{F}$ are trained exclusively on the first task's data, this creates a bias towards that specific task. To address this issue, we also propose a progressive growth approach for the basis filters. In this proposed method, basis filters can be seen as a continuously expanding knowledge base, capable of learning from all subsequent tasks.

To demonstrate efficiency of the proposed approach, we evaluate our model on a variety of datasets and network architectures. With Resnet18[28] based architecture, we report performance improvement on CIFAR100 [53] with significantly low FLOPs and parameters as compared to other methods. For the ImageNet [47] data set, our method achieves comparable performance to other recently reported methods.

Our contributions in this chapter are as follows. i) We propose a filter decomposition-based ap-

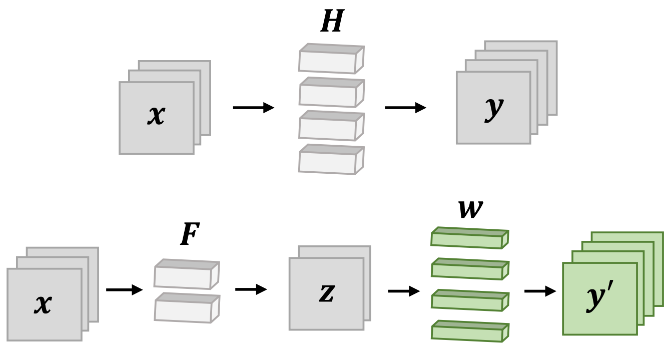


Figure 5.1: Illustration of low rank filter decomposition. On top is the 2D convolution with filters $\mathbf{H}$, which can be decomposed into basis filters $\mathbf{F}$ and coefficients $\mathbf{w}$. A convolution with $\mathbf{F}$ followed by $\mathbf{w}$ can be used to approximate the original output.

proach to address life-long learning in convolutional neural networks, which effectively mitigates catastrophic forgetting. Our method facilitates knowledge sharing across tasks using a set of shared basis filters, while employing task-specific coefficients to compute distinct feature maps for each task. This not only ensures adaptability but also results in a significant reduction in FLOPs and the number of model parameters, thanks to the compact basis filters. ii) We introduce a progressive growth strategy for the basis filters to counter the bias towards the first task. In our approach, the basis filters act as an evolving knowledge base, capable of assimilating learnings from all preceding tasks, thereby ensuring a more balanced and comprehensive model. iii) We provide empirical evidence demonstrating the efficiency of our approach through evaluations on multiple datasets and network architectures. For instance, with a Resnet18-based architecture, our model demosntrates performance improvements on CIFAR100 with substantially lower FLOPs and parameters compared to alternative methods. Additionally, our method delivers competitive performance on the ImageNet dataset relative to other state-of-the-art approaches.

Method

Our method for lifelong learning has been inspired by low rank filter decomposition-based techniques [20, 45]. We show that this formulation can not only be used to compress the pretrained convolution layer but also naturally extends to adapt the model for learning new tasks. It has been observed by [20] that task parameters lie in low dimensional sub space of the convolutional filters. We have exploited this fact for CNN compression by substituting the trained convolutional filters (for the currently known tasks) with compact basis filters $\mathbf{F}$ and coefficients $\mathbf{w}$, as shown in Fig. 5.1. For the lifelong learning problem setting, the basis filters are shared (and are kept fixed) while separate new coefficients are added and trained for each new task. The combination of shared basis filters and task specific coefficients enables the model to preserve previously learned knowledge and avoid forgetting previous tasks.

Low Rank Approximation of Filters

To find these compact basis filters and coefficients, consider the fundamental 2D convolution operation in a CNN. Assume that an input tensor x is convolved with a set of filters $\mathbf{H} \in \mathbb{R}^{P \times L \times D \times D}$, where *P* is the number of filters in H and with each filter of size $L \times D \times D$. The output y, of this convolution operation is expressed as

$$y = x * \mathbf{H} \tag{5.1}$$

We know that eigen decomposition results in a compact basis that minimizes the reconstruction error achieved by a linear combination of basis functions. We therefore choose $\mathbf{F}$ as the eigen filters that represent the sub-space in which the original filters $\mathbf{H}$ lie. The method for obtaining these is also well-known and straightforward. To approximate the compact sub-space these filters are flattened and arranged as columns of the matrix $\tilde{\mathbf{H}} \in \mathbb{R}^{A \times P}$, where $A = LD^2$. We then

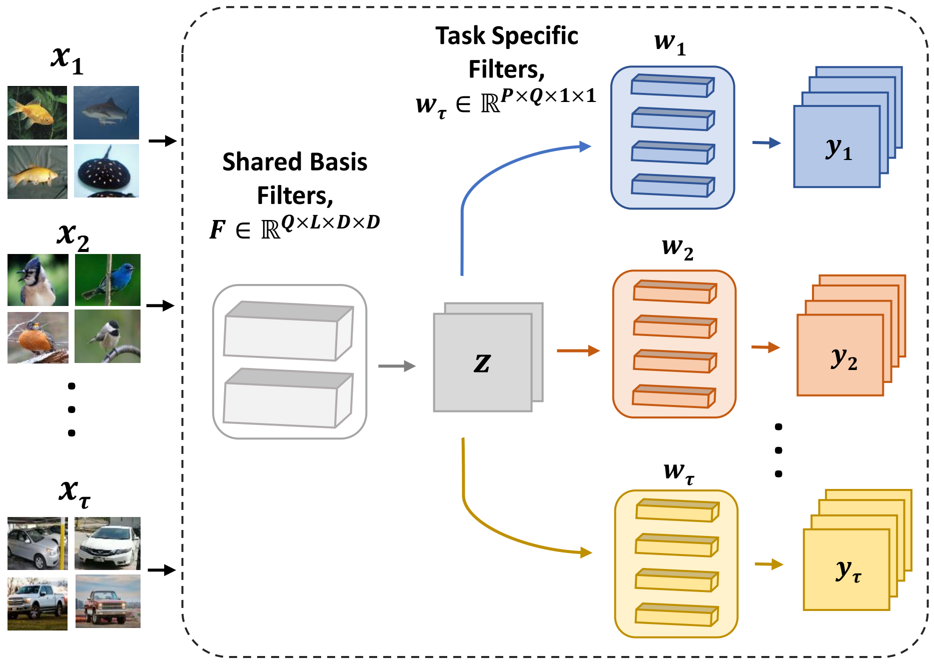

x_1
x_2
x_τ
Task Specific Filters, $w_\tau \in \mathbb{R}^{P\times Q\times 1\times 1}$
Shared Basis Filters, $F \in \mathbb{R}^{Q\times L\times D\times D}$
w_1
w_2
w_τ
z
y_1
y_2
y_τ

Figure 5.2: Our Efficient Lifelong Learning (ELL) method. We substitute the original filters $\mathbf{H}$ with shared basis filters $\mathbf{F}$ and a set of task specific coefficients $\mathrm{w_t}$. Combination of these two lets us compute task specific features throughout the network.

compute its singular value decomposition as $\tilde{\mathbf{H}} = \mathbf{USV}^T$ and initialize $\tilde{\mathbf{F}}$ with the columns of $\mathbf{U}$ corresponding to non zero eigen values. This leads to $\tilde{\mathrm{w}} = \mathbf{SV}^T$ and finally appropriate reshaping of $\tilde{\mathbf{F}}$ and $\tilde{\mathrm{w}}$ lets us rewrite the convolution operation defined in Eq. 5.1 as a sequence of of two successive convolutions:

$$y' = (x * \mathbf{F}) * \mathbf{w} \tag{5.2}$$

where $\mathbf{F} \in \mathbb{R}^{P\times L\times D\times D}$ and $\mathbf{w} \in \mathbb{R}^{P\times P\times 1\times 1}$. We refer to the construct introduced in Eq. 5.2 as *decomposed convolution*. This representation is typically used to compress the model in terms of FLOPs and number of parameters however in its current form it actually increases FLOPs and parameters. To compress the model we discard $k\%$ the basis filters in $\mathbf{F}$ corresponding to the smallest eigen values. This results in $\mathbf{F} \in \mathbb{R}^{Q \times L \times D \times D}$ and correspondingly $\mathbf{w} \in \mathbb{R}^{P \times Q \times 1 \times 1}$, where $Q = \lceil P - kP/100 \rceil$.

Efficient Lifelong Learning

The objective of lifelong learning algorithm (Algo. 1) is to incrementally train the model on a set of disjoint classes with some data availability constraints. Formally, the model observes data $(\mathcal{X}_t, \mathcal{Y}_t)$ randomly drawn from distribution $\mathcal{D}_t$, corresponding to task t, where $\mathcal{X}_t$ and $\mathcal{Y}_t$ represents the input images and corresponding ground truth labels respectively. The goal of a continual learning algorithm is to train model on $\mathcal{D}_t$ while also preserving the previously learned knowledge. However, this data availability constraint introduces the problem of catastrophic forgetting [62, 51] where the model *forgets* the previous knowledge and performs poorly on previous tasks.

We seek to solve this catastrophic forgetting problem by using the *decomposed convolution* structure. We initially train a vanilla neural network on the first task with convolution filters $\mathbf{H}$, given the data samples $(\mathcal{X}_1, \mathcal{Y}_1)$ via the optimization

$$\underset{\mathbf{H}}{\operatorname{argmin}} \sum_{\mathcal{D}_1} \ell(f(\mathcal{X}_1; \mathbf{H}), \mathcal{Y}_1) \tag{5.3}$$

here $f(:)$ represents the model function and l is the loss function. Once the model is trained we decompose and compress the trained filters as described in Sec. 5. Compact nature of basis filters allows us to discard up to 25% of the total filters corresponding to the smallest eigen values, without an adverse affect on model's performance. However with even higher levels of compression model encounters performance degradation. To counter that we fine tune $\mathbf{F}$ and $\mathbf{w_1}$ to minimize the same loss as before.

Algorithm 1 Training procedure of the proposed method

```
1:  function TRAIN($\mathbf{H}$, $k$, $\mathcal{D}_1$ ... $\mathcal{D}_\mathcal{T}$)
2:      Train model using $\mathcal{D}_1$, according to Eq. 5.3
3:      Decompose $\mathbf{H}$ in each Conv2D layer into $\mathbf{F}$ and $\mathbf{w}$
4:      Truncate $k\%$ filters in $\mathbf{F}$ and update $\mathbf{w}$ accordingly
5:      for $t = 1$ to $\mathcal{T}$ do
6:          if $t == 1$ then
7:              Initialize $\mathbf{w}_1 = \mathbf{w}$
8:              Finetune $\mathbf{F}$ and $\mathbf{w}_1$ using $\mathcal{D}_1$, according to Eq. 5.4
9:          else
10:             Initialize $\mathbf{w}_t = \mathbf{w_1}$
11:             Train $\mathbf{w}_t$ using $\mathcal{D}_t$, according to Eq. 5.5
12:         end if
13:     end for
14:     return ($\mathbf{F}$, $\mathbf{w}_1$ .. $\mathbf{w}_\mathcal{T}$)
15: end function
```

$$\underset{\mathbf{F},\mathbf{w_1}}{\operatorname{argmin}} \sum_{\mathcal{D}_1} \ell(f(\mathcal{X}_1; \mathbf{F}, \mathbf{w_1}), \mathcal{Y}_1) \tag{5.4}$$

It is important to note that initial training and fine tuning is done using $\mathcal{D}_1$ only. For each new task $t > 1$, we add additional task specific coefficients $\mathbf{w_t}$ and initialize them with $\mathbf{w_1}$. This is followed by training the model using $\mathcal{D}_t$ to optimize

$$\underset{w_t}{\operatorname{argmin}} \sum_{\mathcal{D}_t} \ell(f(\mathcal{X}_t; w_t), \mathcal{Y}_t) \tag{5.5}$$

To compute the task specific feature maps, we first convolve the input tensor with the shared basis filters followed by a convolution with corresponding task specific coefficients $\mathbf{w_t}$, as depicted in Fig. 5.2. Since shared basis filters are only trained once, this formulation allows us to preserve the learned task specific representations perfectly through out the model and thus avoids the catastrophic forgetting problem entirely.

Expandable Basis Filters

While the formulation introduced in the previous section performs well compared to similar methods, it has one notable drawback: since the basis filters $\mathbf{F}$ are trained exclusively on the first task's data, this creates a bias towards that specific task. To address this issue, we propose a progressive growth approach for the basis filters. In this proposed method, basis filters can be seen as a continuously expanding knowledge base, capable of learning from all subsequent tasks.

Formally, let's assume we have convolution filters of the form $\mathbf{H} \in \mathbb{R}^{P\times L\times D\times D}$, where P represents the depth of the input tensor and L denotes the number of filters in the current layer. We can decompose $\mathbf{H}$ into shared basis filters $\mathbf{F} \in \mathbb{R}^{P\times L\times D\times D}$ and coefficients $\mathbf{w} \in \mathbb{R}^{P\times P\times 1\times 1}$, as discussed in the previous section. Following compression, the filters for first task $(t = 1)$ are configured as

$$\mathbf{F}_1 \in \mathbb{R}^{Q\times L\times D\times D}$$

$$\mathbf{w}_1 \in \mathbb{R}^{P\times Q\times 1\times 1}$$

where $Q < P$. To eliminate bias introduced due to freezing $\mathbf{F}_1$ basis filters, task specific basis filters are added such that for all subsequent tasks, $t > 1$

$$\mathbf{F}_t \in \mathbb{R}^{\alpha Q\times L\times D\times D}$$

$$\mathbf{w}_t \in \mathbb{R}^{P\times Q(1+\alpha(t-1))\times 1\times 1}$$

where α, also referred to as *growth rate*, is the hyper-parameter which governs the number of filters in each $\mathbf{F}_t$ and its value is constrained within range [0, 1]. It is important to highlight that when the *growth rate* parameter is set to zero, the method is reduced to its elementary form, wherein no task-specific basis filters are used. Finally we define forward pass for task t as

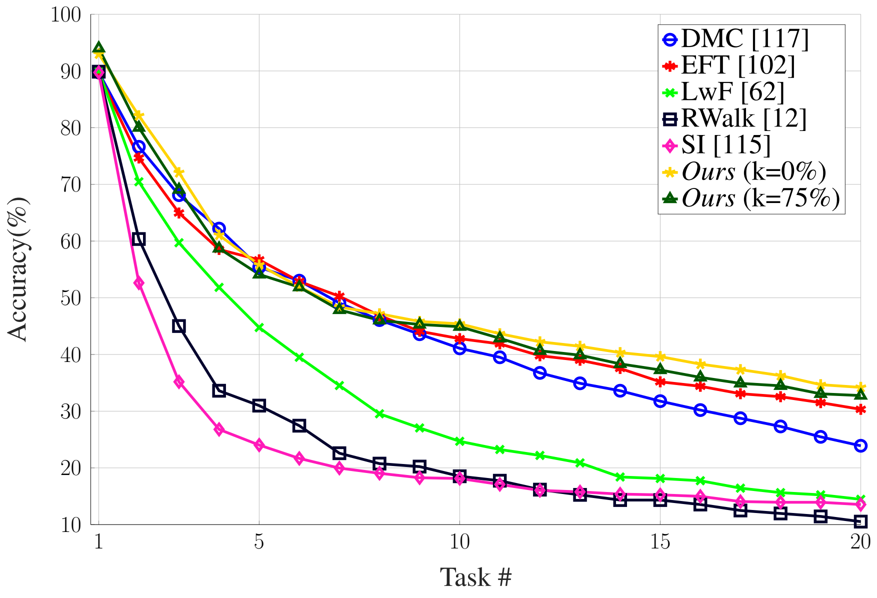
$$y = (x * cat(\mathbf{F}_1, \mathbf{F}_2..\mathbf{F}_t)) * \mathbf{w}_t \tag{5.6}$$

where $cat(.)$ is the concatenation function, which concatenates all previously learned basis filters. This ensures every subsequent task can not only benefit from representations learned with all previous task data, but also have more flexibility to learn their own. It should be noted that while training with task t data we only adapt $\mathbf{F}_t$ and $\mathbf{w}_t$ in each layer and all other filters remain fixed.

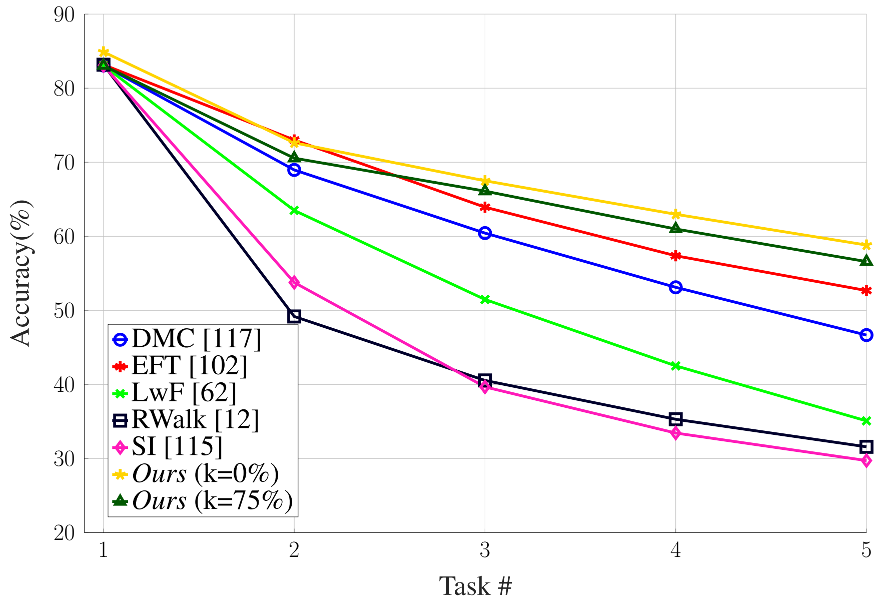
Experiments

We evaluated our method on two lifelong learning scenarios - Task Incremental Learning (TIL) and Class Incremental Learning (CIL). These two scenarios differ in the manner in with new task is treated at inference time. In TIL it is assumed that the task ID is known at inference time which can be used to select the corresponding $\mathbf{w_t}$ to calculate task specific representations. In CIL task ID is not provided and has be to predicted at inference time. We adopted a simple entropy based strategy to predict the task ID, where task ID of an unknown sample is chosen to be the ID of classification head with least entropy.

We also evaluated our method for compression (in Sec. 5) by calculating the FLOPs and parameters required for a given model. We opted FLOPs over wall clock time as a measure of model's efficiency, since FLOPs are machine and implementation independent and can be easily estimated in Pytorch.



(a) *20-split* CIFAR100



(b) *5-split* CIFAR100

Figure 5.3: Comparison of CIL accuracy with Resnet18 architecture on CIFAR100 dataset.

Class Incremental Learning (CIL)

Resnet18: To evaluate our method on CIL we first adopted Resnet18 network architecture and trained it on three uniform splits of CIFAR100. Specifically we divide the 100 classes into 5, 10

Table 5.1: Comparison of CIL accuracy with Resnet18 architecture on *10-split* CIFAR100 dataset.

Method / Task ID	**t1**	**t2**	**t3**	**t4**	**t5**	**t6**	**t7**	**t8**	**t9**	**t10**	**Avg.**
LwF [62]	88.5	70.1	54.8	45.7	39.4	36.3	31.4	28.9	25.5	23.9	44.5
EWC [51]	88.5	52.4	48.6	38.4	31.1	26.4	21.6	19.9	18.8	16.4	36.2
SDC [113]	88.5	78.8	75.8	73.1	71.5	60.7	53.9	43.5	29.5	19.3	59.5
SI [115]	88.5	52.9	40.7	33.6	31.8	29.4	27.5	25.6	24.7	23.3	37.8
MAS [2]	88.5	42.1	36.4	35.1	32.5	25.7	21.0	19.2	17.7	15.4	33.4
RWalk [12]	88.5	55.1	40.7	32.1	29.2	25.8	23.0	20.7	19.5	17.9	35.3
DMC [117]	88.5	76.3	67.5	62.4	57.3	52.7	48.7	43.9	40.1	36.2	57.4
EFT [102]	90.2	76.2	70.1	63.1	57.9	53.6	52.1	49.6	47.6	45.5	60.6
Ours (k=0%)	92.4	76.3	67.1	63.6	62.0	58.6	56.4	54.3	52.1	49.8	63.3
Ours (k=75%)	89.1	74.7	65.1	61.6	59.8	55.6	53.7	51.4	49.4	47.3	60.8

and 20 sets of tasks and refer to them as *5-split*, *10-split* and *20-split respectively*. These splits cover a wide range of problem difficulty as *20-split* CIFAR100 tests models ability to adapt to large number of new tasks while 5-split setting tests model's ability to train on a larger number of classes per task. For each one of these splits we trained Resenet18 at two compression levels with value of k set to 0% and 75% (k is the percentage of basis filters discarded). Results for the 10-split scenario are summarised in Table 5.1 while the results for the 5-split and 20-split cases are plotted in Figure 5.3. We notice that our method outperform others even with very high levels of compression. For example in Table 5.1, when compared with EFT [102] our Resnet18 with $k = 75\%$ performs slightly better with 60.8% average CIL accuracy as compared to EFT's 60.6%. Additionally truncation of large number of basis filters significantly reduces the FLOPs and number of parameters in the model (detailed comparison in Sec. 5).

Resnet32: For evaluating Resnet32 on CIFAR100 we followed the non uniform split used by FAS's[74]. This split is different form the previous ones because here the first tasks always contains 50 classes while the remaining 50 are divided into sets of 5 or 10. This experiment evaluates a significantly different and harder problem as now we have very uneven number of classes per task.

Table 5.2: Average CIL with Resnet32 architecture on non uniform *6-split* and *11-split* CIFAR100. Dataset is divided into tasks such that first task contains 50 classes and remaining are equally divided into 5 or 10 sets.

Method	***6-split***	***11-split***
LwF [62]	57.03	56.82
EWC [51]	56.28	55.41
iCaRL [83]	57.17	52.57
SDC [113]	57.10	56.80
BiC [108]	59.36	54.20
Rebalancing [36]	63.12	60.14
FAS-a [74]	60.23	55.54
FAS-b [74]	65.44	62.48
Ours (k=0%)	66.65	62.54
Ours (k=25%)	62.73	59.23

Table 5.3: Average TIL accuracy and task prediction accuracy for Rensnet32, trained on non-uniform split CIFAR100.

Method	***6-split***	***11-split***
TIL (K=0%)	88.6	93.4
TIL (K=25%)	87.5	93.0
Task Prediction (K=0%)	79.1	74.5
Task Prediction (K=25%)	76.7	73.2

Table 5.4: Comparison of TIL accuracy with AlexNet architecture on *10-split* ImageNet.

Method / Task ID	**t1**	**t2**	**t3**	**t4**	**t5**	**t6**	**t7**	**t8**	**t9**	**t10**	**Avg.**
LwF [62]	27.6	37.2	42.0	44.4	50.5	56.6	57.9	61.2	62.0	62.7	50.2
IMM [57]	68.5	53.6	52.1	51.7	52.5	55.5	54.7	53.5	54.2	51.8	54.8
EWC [51]	21.8	26.5	29.5	32.9	35.6	40.4	40.0	44.7	47.8	61.1	38.0
PackNet [72]	67.5	65.8	62.2	58.4	58.6	58.7	56.0	56.5	54.1	53.6	59.1
EFT [102]	69.0	63.2	60.1	62.5	53.6	57.2	55.1	52.8	55.7	62.5	59.4
Ours (k=0%)	65.6	63.3	60.7	63.9	56.5	57.4	55.0	52.8	55.7	64.1	59.5
Ours (k=25%)	65.0	63.0	59.7	62.8	55.5	56.3	54.5	52.4	55.0	63.5	58.8

As we can see in Table 5.2 our method with $k = 0$ out performs other methods. However due to non-uniformity of the splits our method does not perform equally well with high compression. This performance degradation mainly comes from the failure of class prediction mechanism as shown in Table 5.3.

Table 5.5: Comparison of TIL accuracy with VGG16 architecture on *10-split* TinyImageNet.

Method / Task ID	**t1**	**t2**	**t3**	**t4**	**t5**	**t6**	**t7**	**t8**	**t9**	**t10**	**Avg.**
LfL [49]	32.4	35.4	43.4	44.1	45.0	55.9	49.4	51.1	58.6	61.4	47.7
LwF [62]	45.1	45.5	53.5	57.6	56.2	65.7	63.5	58.4	59.6	58.5	56.4
IMM [57]	50.6	38.5	44.7	49.2	47.5	51.9	53.7	47.7	50.0	48.7	48.3
EWC [51]	33.9	35.4	43.6	46.7	49.5	52.5	47.8	50.2	56.6	61.4	47.8
HAT [91]	46.8	49.1	55.8	58.0	53.7	61.0	58.7	54.0	54.6	50.3	54.2
PackNet [72]	52.5	49.7	56.5	59.8	55.0	64.7	61.7	55.9	55.2	52.5	56.4
TFM [73]	48.2	47.7	56.7	58.2	54.8	62.2	61.5	57.3	58.5	54.8	56.0
EFT [102]	67.2	62.5	69.4	62.6	68.3	69.6	59.0	67.8	71.5	70.1	66.8
Ours (k=0%)	64.3	64.6	70.8	65.9	68.0	70.6	60.2	69.4	71.2	69.0	67.4
Ours (k=50%)	64.2	64.6	69.1	64.0	68.4	68.7	60.4	67.5	72.1	69.7	66.9

Task Incremental Learning (TIL)

We also tested our method on TIL with AlexNet and VGG16 architectures. We trained these on uniform *10-split* ImageNet [47] and TinyImageNet datasets respectively. ImageNet is a classification dataset containing 1000 classes, while TinyImageNet is a smaller subset of ImageNet dataset containing 200 classes, downsampled to 64×64 spatial resolution. The results of these experiments are shown in the Table 5.4 and Table 5.5. We can see that, for both ImageNet and TinyImageNet our uncompressed models $(k = 0\%)$ achieves better average accuracy compared to the other methods, while the compressed variants attain results comparable to the previous SOTA.

Expandable Basis Filters

Our proposed expandable basis filters scheme was evaluated on the Resnet18 architecture using the *5-split* CIFAR100 dataset. Initially, the model was compressed with k = 90%, resulting in the removal of 90% of filters in every convolution layer. The resulting model was then trained with four distinct values of α, specifically (0, 0.1, 0.3, 0.5). The results for Task Incremental Learning

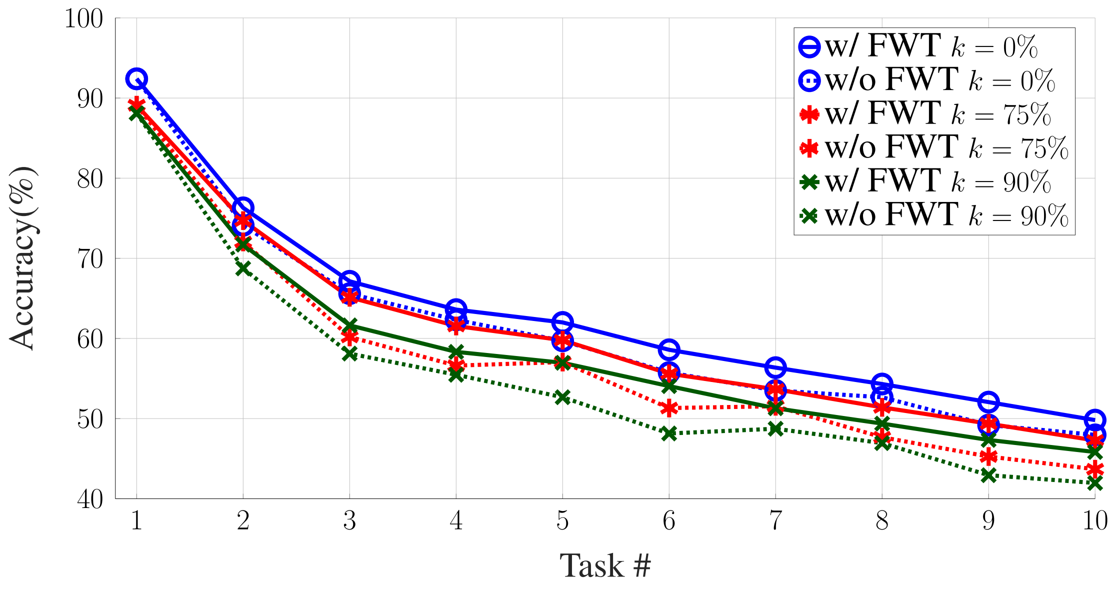
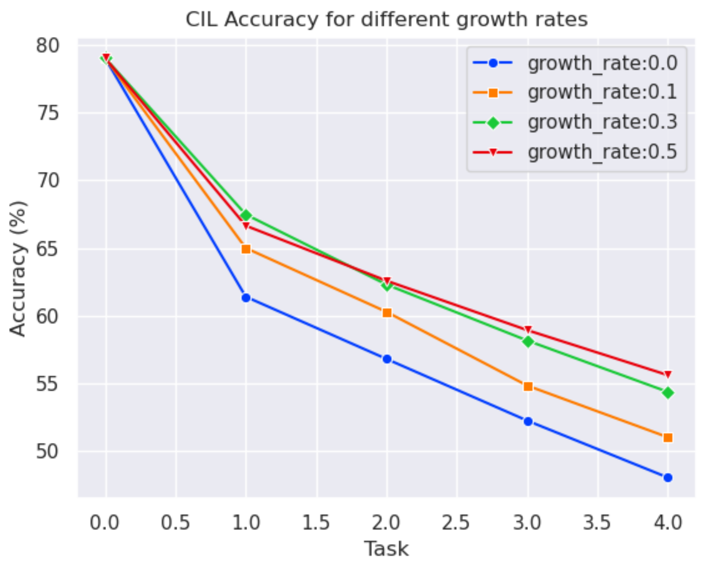


Figure 5.4: Evaluation of Forward Knowledge Transfer (FWT).

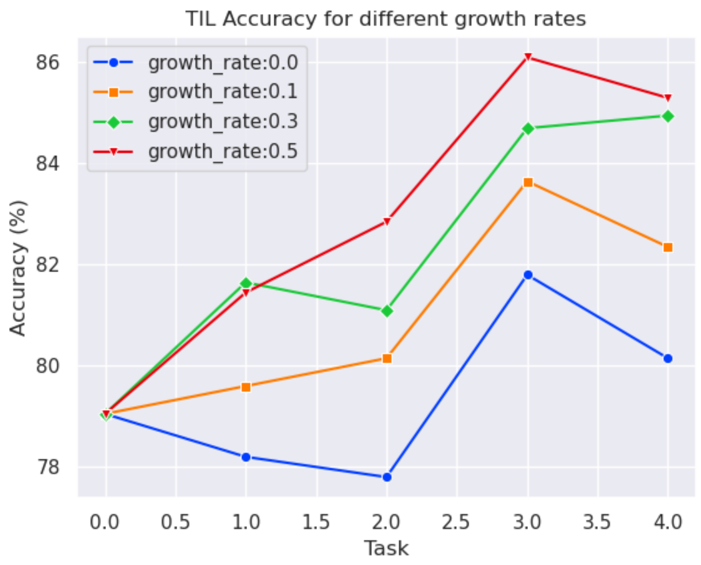
and Class Incremental Learning are displayed in Figure 5.5. As hypothesized, the addition of task-specific basis filters markedly enhances performance for all tasks where $t > 1$.

Forward Transfer (FWT)

FWT is an important metric of the quality of representations learned by a lifelong learning system. It measures the ability of the model to positively influence a future task's performance based on the existing representations. Our method achieves FWT mainly by sharing the basis $\mathbf{F}$ and by initializing task specific parameters $\mathbf{w_t}$ with $\mathbf{w_{t-1}}$. We evaluated our method for FWT on Resnet18 with *10-split* CIFAR100 by comparing this method of weight initialization to initializing $\mathbf{w_t}$ randomly for three levels of compression. Results of this experiment are presented in Fig. 5.4. We notice that for all compression levels initializing with $\mathbf{w_{t-1}}$ helps model achieve better performance. Secondly for higher levels of compression (when $k > 0$) model gets better at FWT.



(a) Class incremental learning.



(b) Task incremental learning.

Figure 5.5: Results of our proposed expandable basis filter for Resnet18 on *5-split* CIFAR100 dataset.

Table 5.6: Comparison of FLOPs and parameters for various models used in our experiments.

k	**FLOPs (Billions)**				**Params (Millions)**			
	Resnet18	*Resnet32*	*VGG16*	*AlexNet*	*Resnet18*	*Resnet32*	*VGG16*	*AlexNet**
Baseline	1.11	0.14	2.50	1.74	11.16	0.46	14.76	2.88
0%	1.25	0.15	2.84	2.00	26.84	0.80	33.24	9.77
25%	0.95	0.12	2.10	1.50	20.61	0.60	25.05	8.36
50%	0.64	0.08	1.43	1.00	14.38	0.41	16.86	6.94
75%	0.32	0.04	0.72	0.48	7.24	0.21	8.66	5.52

Analysis of FLOPs and Parameters

We now compare the FLOPs and parameters of various network architectures used in our experiments. For our method, number of total tasks does not impact the FLOPs of a single input image. As can be deduced from Eq. 5.2, the output at each layer is obtained by convolving the input tensor with the shared basis filters followed by the convolution with a single task specific $\mathbf{w_t}$. However as more tasks are added to the model, additional coefficients and classification heads increase the number of parameters in the model drastically. For comparative purposes we selected Resnet18, VGG16 and ALexNet, each containing ten task specific $\mathbf{w_t}$'s in every layer. Table 5.6 show the FLOPs and parameters at four levels of compression.

We can see that our Resnet18 trained for CIL, with $k = 75\%$ needs 0.32 Billion FLOPs and 7.24 Million parameters. In comparison EFT [102] performs equally well but needs 1.21 Billion FLOPs and 11.60 Million parameters. Similarly our Resnet32 with $k = 0\%$ outperforms FAS-b[74] on non-uniform split CIFAR100 and needs only 0.15 Billion FLOPs as compared to 0.28 Billion required by FAS-b[74].

Conclusion

In this chapter we proposed a method for continual learning that draws upon principles from neural network compression to incrementally learn new tasks. This methodology stems from filter decomposition techniques and demonstrates a natural extension to the domain of continual learning. It is widely recognized that task parameters inhabit a low-dimensional subspace of the network's filters. This allows the filter kernels to be compactly represented using basis filters and remixing coefficients. In the context of continual learning, the basis filters are shared while new coefficients are introduced for each new task. This approach enables the model to perfectly retain previously acquired knowledge, thereby avoiding catastrophic forgetting. When applied to various continual learning problems for image classificatioin, our method consistently demonstrated performance improvements while maintaining significantly lower FLOPs and parameters. Additionally we also introduced a method to eliminate the bias inherent in earlier formulations for continual learning that stem from training by freezing the basis filters. Specifically, our approach incorporates task-specific basis filters into the model, coupled with an inference mechanism in which these filters functions as a knowledge reservoir. This knowledge reservoir provides a foundation of information that each subsequent task can utilize and expand upon.

CHAPTER 6: EFFICIENT CROWD ANALYSIS

Introduction

Hitherto fore, we have mainly discussed the compression of networks designed for classification problems. In this Chapter, we discuss the application of our basisConv compression methodology to a network designed to solve a regression problem. In particular, we develop a network for counting the number of people in a large crowd, and then study the effect of compression on this network.

Counting dense crowds is significant both from socio-political and safety perspective. At one end of the spectrum, there are large peaceful gatherings such as during pilgrimages that typically have large crowds occurring in known and pre-defined locations. At the other end are as expressive mobs in demonstrations and protests. Computer vision based crowd counting offers fast and objective estimation of the number of people in such events. Furthermore, crowd counting is extendable to other domains, for instance, counting cells or bacteria from microscopic images [58, 97], animal crowd estimates in wildlife sanctuaries [5], or estimating the number of vehicles at transportation hubs or traffic jams [79].

In this work, we propose a novel approach to crowd counting, density map estimation and localization of people in a given crowd image. Our approach stems from the observation that these three problems are very interrelated - in fact, they can be decomposed with respect to each other. Counting provides an estimate of the number of people / objects without any information about their location. Density maps, which can be computed at multiple levels, provide weak information about location of each person. Localization does provide accurate location information, nevertheless, it is extremely difficult to estimate directly due to its very sparse nature. Therefore, we propose to estimate all three tasks simultaneously, while employing the fact that each is special case of another one. Density maps can be 'sharpened' till they approximate the localization map, whose integral should equal to the true count.

Furthermore, we introduce a new and the largest dataset to-date for training and evaluating **dense** crowd counting, density map estimation and localization methods, particularly suitable for training very deep Convolutional Neural Networks (CNNs). Though counting has traditionally been considered the primary focus of research, density map estimation and localization have significance and utility beyond counting. In particular, two applications are noteworthy: initialization / detection of people for tracking in dense crowds [42]; and rectifying counting errors from an automated computer vision algorithm. That is, a real user or analyst who desires to estimate the exact count for a real image *without any error*, the results of counting alone are insufficient. The single number for an entire image makes it difficult to assess the error or the source of the error. However, the localization can provide an initial set of dot locations of the individuals, the user then can quickly go through the image and remove the false positives and add the false negatives. The count using such an approach will be much more accurate and the user can get 100% precise count for the query image. This is particularly important when the number of image samples are few, and reliable counts are desired. Table 6.1 summarizes the statistics of the multi-scene datasets for dense crowd counting. The proposed UCF-QNRF dataset has the most number of high-count crowd images and annotations, and a wider variety of scenes containing the most diverse set of viewpoints, densities and lighting variations.

Finally, to make crowd counting efficient we apply our filter decomposition based neural network compression method on our trained model. In doing so we are able to significantly reduce the FLOPs required to make an inference on the pretrained model, thus making the deployment more efficient.

Dataset	Number Images	Number Ann.	Avg Count	Max Count	Avg Resolution	Avg Density
UCF_CC_50 [43]	50	63,974	1279	4633	2101×2888	2.02×10^{-4}
WorldExpo'10 [116]	3980	225,216	56	334	576×720	1.36×10^{-4}
ShanghaiTech_A [119]	482	241,677	501	3139	589×868	9.33×10^{-4}
UCF-QNRF	**1535**	**1,251,642**	**815**	**12865**	$\mathbf{2013 \times 2902}$	$\mathbf{1.12 \times 10^{-4}}$

Table 6.1: Summary of statistics of different datasets. UCF_CC_50 (44MB); WorldExpo'10 (325MB); ShanghaiTech_A (67MB); and the proposed UCF-QNRF Dataset (4.33GB).

Our contributions in this work are manifold. i) We present a novel approach to crowd counting, density map estimation, and localization of individuals within an image. By recognizing the interconnectedness of these three problems, we simultaneously estimate all of them. Our method leverages the concept that density maps can be progressively sharpened to approximate localization maps, the integral of which corresponds to the true count. ii) We introduce the largest dataset to date for training and evaluating dense crowd counting, density map estimation, and localization methods. Our dataset, UCF-QNRF, is particularly suitable for training deep Convolutional Neural Networks (CNNs) and boasts an unparalleled diversity in terms of viewpoints, densities, and lighting conditions. iii) We enhance the efficiency of crowd counting by incorporating a filter decomposition-based neural network compression method on our trained model. This results in a significant reduction in the number of floating-point operations (FLOPs) needed for inference, making the model more efficient and practical for deployment in real-world scenarios.

Deep CNN with Composition Loss

In this section, we present the motivation for decomposing the loss of three interrelated problems of counting, density map estimation and localization, followed by details about the deep Convolutional Neural Network which can enable training and estimation of the three tasks simultaneously.

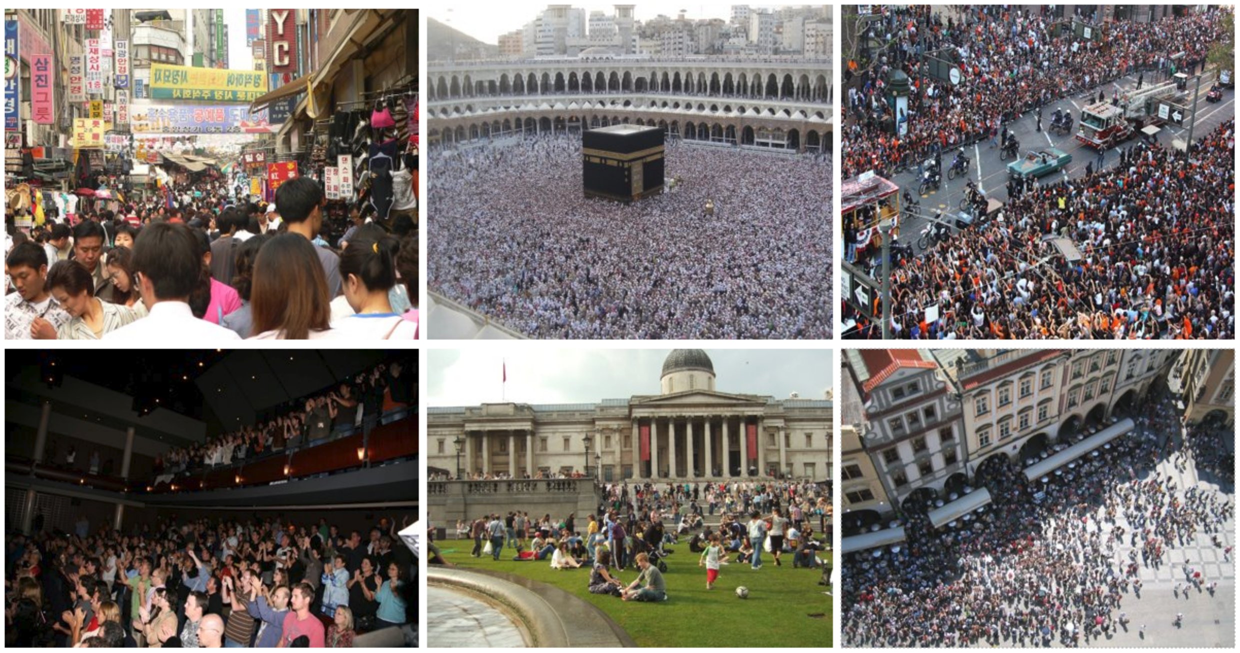


Figure 6.1: Six images from the dataset demonstrating the diversity of count and setting.

Composition Loss

Let $\mathbf{x} = [x, y]$ denote a pixel location in a given image, and N be the number of people annotated with $\{\mathbf{x}_i : i = 1, 2, \dots N\}$ as their respective locations. Dense crowds typically depict heads of people as they are the only parts least occluded and mostly visible. In localization maps, only a single pixel is activated, i.e., set to 1 per head, while all other pixels are set to 0. This makes localization maps extremely sparse and therefore difficult to train and estimate. We observe that successive computation of 'sharper' density maps which are relatively easier to train can aid in localization as well. Moreover, all three tasks should influence count, which is the integral over density or localization map. We use the Gaussian Kernel and adapt it for our problem of simultaneous solution for the three tasks.

Due to perspective effect and possibly variable density of the crowd, a single value of bandwidth, σ, cannot be used for the Gaussian kernel, as it might lead to well-defined separation between people

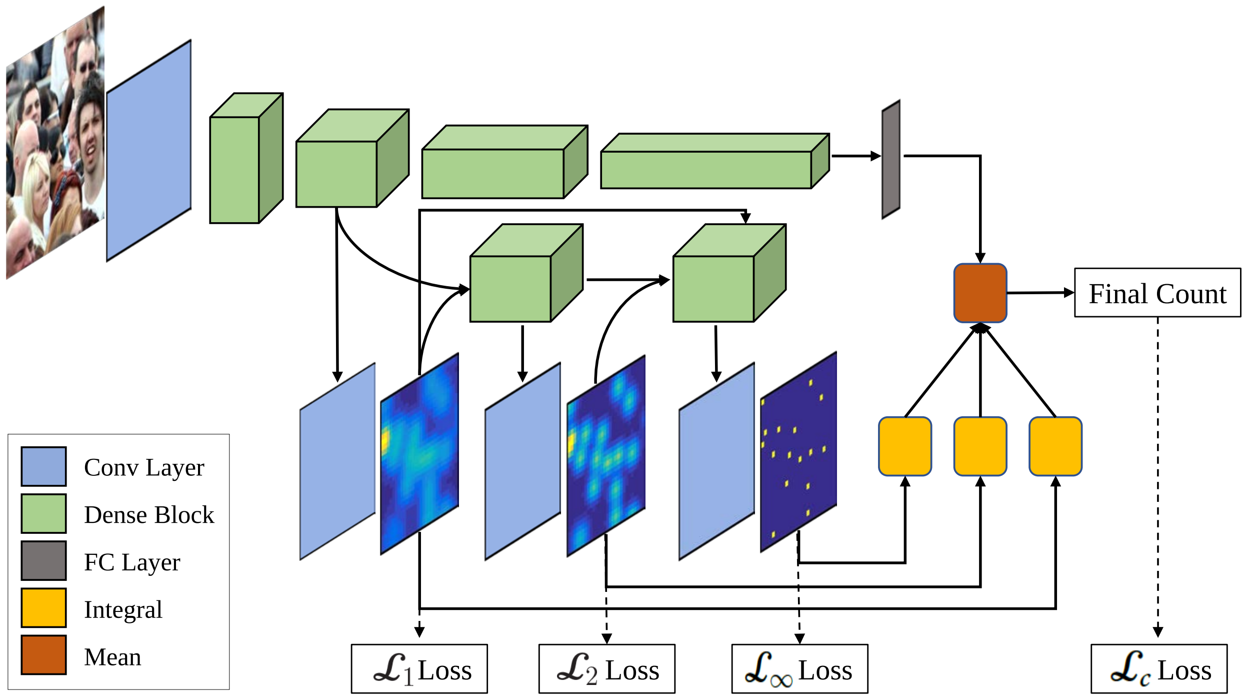


Figure 6.2: The figure shows the proposed architecture for estimating count, density and localization maps simultaneously for a given patch in an image. At the top is the base DenseNet which regresses only the counts. The proposed Composition Loss is implemented through multiple dense blocks after branching off the base network. We also test the effect of additional constraint on the density and localization maps (shown with amber and orange blocks) such that the count after integral in each should also be consistent with the groundtruth count.

close to the camera or in regions of low density, while excess blurring in other regions. Many images of dense crowds depict crowds in their entirety, making automatic perspective rectification difficult. Thus, we propose to define σ_i for each person i as the minimum of the ℓ_2 distance to its nearest neighbor in spatial domain of the image or some maximum threshold, τ. This ensures that the location information of each person is preserved precisely irrespective of default kernel bandwidth, τ. Thus, the adaptive Gaussian kernel is given by,

$$D(\mathbf{x}, f(\cdot)) = \sum_{i=1}^{N} \frac{1}{\sqrt{2\pi}f(\sigma_i)} \exp\left(-\frac{(x-x_i)^2 + (y-y_i)^2}{2f(\sigma_i)^2}\right), \tag{6.1}$$

where the function f is used to produce a successive set of 'sharper' density maps. We define $f_k(\sigma) = \sigma^{1/k}$. Thus, $D_k = D(\mathbf{x},\ f_k(\cdot))$. As can be seen when $k = 1$, D_k is a very smoothed-out density map using nearest-neighbor dependent bandwidth and τ, whereas as $k \longrightarrow \infty$, D_k approaches the binary localization map with a Dirac Delta function placed at each annotated pixel. Since each pixel has a unit area, the localization map assumes a unit value at the annotated location. For our experiments we used three density levels with last one being the localization map.

Hypothetically, since integral over each estimated $\hat{D}_k$ yields a count for that density level, the final count can be obtained by taking the mean of counts from the density and localization maps as well as regression output from base CNN. This has two potential advantages: 1) the final count relies on multiple sources - each capturing count at a different scale. 2) During training the mean of four counts should equal the true count, which implicitly enforces an additional constraint that $\hat{D}_k$ should not only capture the density and localization information, but that each of their counts should also sum to the groundtruth count. For training, the loss function of density and localization maps is the mean square error between the predicted and ground truth maps, i.e. $\mathcal{L}_k = \text{MSE}(\hat{D}_k, D_k)$, where $k = 1, 2,$ and ∞, and regression loss, $\mathcal{L}_c$, is Euclidean loss between predicted and groundtruth counts, while the final loss is defined as the weighted mean all four losses.

It is also interesting to note that various connections between density levels and base CNN also serve to provide intermediate supervision which can make the filters of base CNN towards counting and density estimation early on in the network.

DenseNet with Composition Loss

The architecture of our network is shown in Figure 6.2 We use DenseNet [39] as our base network. It consists of 4 Dense blocks where each block has a number of consecutive 1×1 and 3×3

Layer	**Output Size**	**Filters**
	$512 \times 28 \times 28$	
Density Level 1	$1 \times 28 \times 28$	1×1 conv
Density Level 2	$641 \times 28 \times 28$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 4$
	$1 \times 28 \times 28$	1×1 conv
Density Level ∞	$771 \times 28 \times 28$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 4$
	$1 \times 28 \times 28$	1×1 conv

Table 6.2: This table shows the filter dimensions and output of the three density layer blocks appended to the network in Fig. 6.2.

convolutional layers. Each dense block (except for the last one) is followed by a Transition layer, which reduces the number of feature-maps by applying 1×1 convolutions followed by 2×2 average pooling with stride 2. In our experiments we used DenseNet-201 architecture. It has $\{6, 12, 48, 32\}$ sets of 1×1 and 3×3 convolutional layers in the four dense blocks, respectively.

For density map estimation and localization, we branch out from DenseBlock2 and feed it to our Density Network. The density network introduces 2 new dense blocks and three 1×1 convolutional layers. Each dense block has features computed at the previous step, concatenated with all the density levels predicted thus far as input, and learns features aimed at computing the current density / localization map. We used 1×1 convolutions to get the output density map from these features. Density Level 1 is computed directly from DenseBlock2 features.

We used Adam solver with a step learning rate in all our experiments. We used 0.001 as initial learning rate and reduce the learning rate by a factor of 2 after every 20 epochs. We trained the entire network for 70 epoch with a batch size of 16.

Definition and Quantification of Tasks

In this section, we define the three tasks and the associated quantification measures.

Counting: The first task involves estimation of count for a crowd image i, given by $\mathbf{c}_i$. Although this measure does not give any information about location or distribution of people in the image, this is still very useful for many applications, for instance, estimating size of an entire crowd spanning several square kilometers or miles. For the application of counting large crowds, Jacob's Method [44] due to Herbert Jacob is typically employed which involves dividing the area $\mathbf{A}$ into smaller sections, finding the average number of people or density $\mathbf{d}$ in each section, computing the mean density $\bar{\mathbf{d}}$ and extrapolating the results to entire region. However, with automated crowd counting, it is now possible to obtain counts and density for multiple images at different locations, thereby, permitting the more accurate integration of density over entire area covered by crowd. Moreover, counting through multiple aerial images requires cartographic tools to map the images onto the earth to compute ground areas. The density here is defined as the number of people in the image divided by ground area covered by the image. We propose to use the same evaluation measures as used in literature for this task: the Mean Absolute Error (C-MAE), Mean Squared Error (C-MSE) with the addition of Normalized Absolute Error (C-NAE).

Density Map Estimation amounts to computing per-pixel density at each location in the image, thus preserving spatial information about distribution of people. This is particularly relevant for safety and surveillance, since very high density at a particular location in the scene can be catastrophic [1]. This is different from counting since an image can have counts within safe limits, while containing regions that have very high density. This can happen due to the presence of empty regions in the image, such as walls and sky for mounted cameras; and roads, vehicles, buildings and forestation in aerial cameras. The metrics for evaluating density map estimation are similar to counting, except that they are per-pixel, i.e., the per-pixel Mean Absolute Error (DM-MAE) and Mean Squared Error (DM-MSE). Finally, we also propose to compute the 2D Histogram Intersection (DM-HI) distance after normalizing both the groundtruth and estimated density maps. This discards the effect of absolute counts and emphasizes the error in distribution of density compared to the groundtruth.

Localization: The ideal approach to crowd counting would be to detect all the people in an image and then count the number of detections. But since dense crowd images contain severe occlusions among individuals and fewer pixels per person for those away from the camera, this is not a feasible solution. This is why, most approaches to crowd counting bypass explicit detection and perform direct regression on input images. However, for many applications, the precise location of individuals is needed, for instance, to initialize a tracking algorithm in very high-density crowd videos.

To quantify the localization error, estimated locations are associated with the ground truth locations through 1-1 matching using greedy association, followed by computation of Precision and Recall at various distance thresholds ($1, 2, 3, \ldots, 100$ pixels). The overall performance of the localization task is then computed through area under the Precision-Recall curve, L-AUC.

Compressing Crowd Counting Model

It is well known that large Convolutional Neural Networks (CNNs) are often over parameterized. This is underscored by the existence of various methodologies for compressing a pretrained model, including pruning-based techniques [18, 27] and low-rank factorization strategies [20, 45, 81]. In this work, we adopt a low-rank decomposition approach, disassembling the filters into a set of compact basis filters and with their corresponding coefficients. These compact basis filters enable us to compress the model by reducing the Floating Point Operations (FLOPs) required executing a forward pass through the model.

To find these compact basis filters and coefficients, consider the fundamental 2D convolution operation in a CNN. Assume that an input tensor x is convolved with a set of filters $\mathbf{H} \in \mathbb{R}^{P\times L\times D\times D}$, where *P* is the number of filters in H and with each filter of size $L \times D \times D$. The output y, of this convolution operation is expressed as

$$y = x * \mathbf{H} \tag{6.2}$$

We know that eigen decomposition results in a compact basis that minimizes the reconstruction error achieved by a linear combination of basis functions. We therefore choose $\mathbf{F}$ as the eigen filters that represent the sub-space in which the original filters $\mathbf{H}$ lie. The method for obtaining these is also well-known and straightforward. To approximate the compact sub-space these filters are flattened and arranged as columns of the matrix $\tilde{\mathbf{H}} \in \mathbb{R}^{P\times A}$, where $A = LD^2$. We then compute its singular value decomposition as $\tilde{\mathbf{H}} = \mathbf{USV}^T$ and initialize $\tilde{\mathbf{F}}$ with the columns of $\mathbf{V}^T$ corresponding to non zero eigen values. This leads to $\tilde{\mathbf{w}} = \tilde{\mathbf{H}}\tilde{\mathbf{F}}^T$ and finally appropriate reshaping of $\tilde{\mathbf{F}}$ and $\tilde{\mathbf{w}}$ lets us rewrite the convolution operation defined in Eq. 6.2 as a sequence of of two successive convolutions:

$$y' = (x * \mathbf{F}) * \mathbf{w} \tag{6.3}$$

where $\mathbf{F} \in \mathbb{R}^{P\times L\times D\times D}$ and $\mathbf{w} \in \mathbb{R}^{P\times P\times 1\times 1}$. To compress the model we discard $k\%$ the basis filters in $\mathbf{F}$ corresponding to the smallest eigen values. This results in $\mathbf{F} \in \mathbb{R}^{Q\times L\times D\times D}$ and correspondingly $\mathbf{w} \in \mathbb{R}^{P\times Q\times 1\times 1}$, where $Q = \lceil P - kP/100 \rceil$.

Method	C-MAE	C-NAE	C-MSE
Idrees et. al [43]*	315	0.63	508
MCNN [119]	277	0.55	426
Encoder-Decoder [7]	270	0.56	478
CMTL [94]	252	0.54	514
SwitchCNN [89]	228	0.44	445
Resnet101 [29]*	190	0.50	277
Densenet201 [39]*	163	0.40	226
Proposed	**132**	**0.26**	**191**
Compressed by 30%	134	0.26	206
Compressed by 50%	137	0.27	227

Table 6.3: We show counting results obtained using state-of-the-art methods in comparison with the proposed approach. Methods with '*' regress counts without computing density maps.

Experiments

Next, we present the results of experiments for the three tasks defined in Section 6. For each task, we also provide the performance metrics of two compressed variants of the base model. The compressed models have been optimized to operate with a reduction of 30% and 50% of the Floating Point Operations (FLOPs) in comparison to the original model.

Counting

For counting, we evaluated the proposed UCF-QNRF dataset using the proposed method which estimates counts, density maps and location of people simultaneously with several state-of-the-art deep neural networks [7], [29], [39] as well as those specifically developed for crowd counting [119], [94], [89]. To train the networks, we extracted patches of size $448, 224$ and 112 pixels at random locations from each training image. While deciding on image locations to extract patch from we assigned higher probability of selection to image regions with higher count. We used mean square error of counts as the loss function. At test time, we divide the image into a grid of $224 \times$

Method	DM-MAE	DM-MSE	DM-HI
MCNN [119]	0.006670	0.0223	0.5354
SwitchCNN [89]	0.005673	0.0263	0.5301
CMTL [94]	0.005932	0.0244	0.5024
Proposed	**4.4e-4**	**0.0017**	**0.9131**
Compressed by 30%	4.7e-4	0.0018	0.9185
Compressed by 50%	5.1e-4	0.0020	0.8503

Table 6.4: Results for Density map estimation: We show results on Histogram intersection (HI), obtained using existing state-of-the-art methods compared to the proposed approach.

224 pixel cells - zero-padding the image for dimensions not divisible by 224 - and evaluate each cell using the trained network. Final image count is given by aggregating the counts in all cells. Table 6.3 summarizes the results which shows the proposed network significantly outperforms the competing deep CNNs and approaches for crowd counting. Additionally, our compressed models demonstrate comparable performance to the original, uncompressed model.

Density Map Estimation

For density map estimation, we describe and compare the proposed approach with several methods that directly regress crowd density during training. Among the deep learning methods, MCNN [119] consists of three columns of convolution networks with different filter sizes to capture different head sizes and combines the output of all the columns to make a final density estimate. SwitchCNN [89] uses a similar three column network; however, it also employs a switching network that decides which column should exclusively handle the input patch. CMTL [94] employs a multitask network that computes a high level prior over the image patch (crowd count classification) and density estimation. These networks are specifically designed for crowd density estimation and their results are reported in first three rows of Table 6.4. The results of proposed approach and compressed models are shown in the bottom three row of Table 6.4.

Method	Av. Precision	Av. Recall	L-AUC
MCNN [119]	59.93%	63.50%	0.591
ResNet74 [29]	61.60%	66.90%	0.612
DenseNet63 [39]	70.19%	58.10%	0.637
Encoder-Decoder [7]	71.80%	62.98%	0.670
Proposed	**75.8%**	**59.75%**	**0.714**
Compressed by 30%	74.2%	58.45%	0.702
Compressed by 50%	73.9%	57.93%	0.685

Table 6.5: This table shows the localization results averaged over four distance thresholds for different methods. We show Average Precision, Average Recall and AUC metrics.

Localization

For the localization task, we adopt the same network configurations used for density map estimation to perform localization. To get the accurate head locations, we post-process the outputs by finding the local peaks/maximums based on a threshold, also known as non maximum suppression. Once the peaks are found, we match the predicted location with the ground truth location using 1-1 matching, and compute precision and recall. We use different distance thresholds as the pixel distance, i.e., if the detection is within the a particular distance threshold of the ground truth, it is treated as True Positive, otherwise it is a False Positive. Similarly, there is no detection within a ground truth location, it becomes a False Negative. The results of localization are reported in Table 6.5. This table shows that DenseNet [39] and Encoder-Decoder [7] outperform ResNet [29] and MCNN [119] while, the proposed methods significantly outperform all other methods for localization.

Conclusion

This work introduced a novel method to estimate counts, density maps and localization in dense crowd images. We showed that the observation that these three problems are very interrelated, and can be decomposed with respect to each other can be translated into Composition Loss which can then be used to train a neural network. We solved all three tasks simultaneously with the counting performance benefiting from the density map estimation and localization as well. We presented extensive set of experiments using several recent deep architectures, and show that the proposed approach is able give excellent performance. Finally we have also demonstrated the performance of two compressed variations of our proposed model. The results indicate that despite a reduction in Floating Point Operations (FLOPs) by 30% and 50%, the model continues to maintain high accuracy across all three tasks, and is able to outperform all other techniques evaluated in the comparisons.

CHAPTER 7: CONCLUSION

This dissertation's primary motivation arose from the paradigm shift in the field of machine learning, instigated by the introduction of AlexNet in 2012, which paved the way for increasingly larger and more complex neural networks. Despite the remarkable improvements these networks have brought to areas such as speech, vision, and language processing, their computational requirements and parameter counts have grown exponentially, outpacing the progression of hardware capabilities. Consequently, this has created a significant barrier to the broader adoption of neural networks, given the associated financial and computational costs.

Addressing these issues, our work has focused on the development of a set of neural network compression methods based on filter decomposition for compress convolutional neural networks (CNNs). We aimed to replace each convolutional layer with compact operations to minimize computational requirements, storage capacity, and energy consumption, all while maintaining or even enhancing model accuracy. Our methodology focused on three key performance metrics: computation requirements, storage capacity, and model accuracy.

This dissertation is organized into several chapters, each discussing a critical aspect of our research. We first reviewed previous works in Chapter 2, which set the stage for the development of our compression strategies and their application in the subsequent chapters.

Chapter 3 detailed our filter decomposition-based approach to the compression of pre-trained deep networks. This approach was premised on the observation that the original filters in a convolution layer can be represented as weighted linear combinations of a set of 3D basis filters, with one-dimensional weight kernels. We also discussed how these basis filters can be fine-tuned to offset any performance loss due to truncation.

In Chapter 4, we proposed a regularizer-based method for simultaneously training and compressing CNNs from scratch. We emphasized the importance of training filters to be compact and information-dense, to minimize the loss of useful information during filter truncation, a key step in popular compression methods such as pruning and low-rank filter factorization.

In Chapter 5, we demonstrated how our filter decomposition-based compression method can be effectively applied to the continual learning problem. We proposed a solution to the catastrophic forgetting issue, a common challenge in continual learning settings, and showed that our method significantly reduces the number of Floating Point Operations (FLOPs) required compared to other methods.

Finally, we applied the proposed compression strategy to a regression-based crowd analysis framework in Chapter 6. This framework introduced a novel method for integrating crowd counting, density map estimation, and localization of individuals in a crowd image into a singular, unified problem. We then employ a compression scheme based on low-rank filter decomposition to maintain high accuracy while reducing computational requirements.

In conclusion, this dissertation offers novel contributions to the field of neural network compression, presenting effective strategies for reducing the computational and financial costs of employing convolutional neural networks, thereby facilitating their broader adoption.

LIST OF REFERENCES

[1] "A history of hajj tragedies". In: *The Guardian* (2006). `http://www.guardian.co.uk/world/2006/jan/13/saudiarabia`. [Accessed: July 1, 2013].

[2] Rahaf Aljundi et al. "Memory Aware Synapses: Learning what (not) to forget". In: *The European Conference on Computer Vision (ECCV)*. Sept. 2018.

[3] Jose M. Alvarez and Mathieu Salzmann. "Compression-aware Training of Deep Networks". In: *Advances in Neural Information Processing Systems* (2017).

[4] Dario Amodei and Danny Hernandez. *AI and Compute*. 2018. URL: `https://openai.com/blog/ai-and-compute`.

[5] Carlos Arteta, Victor Lempitsky, and Andrew Zisserman. "Counting in the Wild". In: *European Conference on Computer Vision*. Springer. 2016, pp. 483–498.

[6] Lei Jimmy Ba and Rich Caruana. "Do Deep Nets Really Need to be Deep?" In: *Advances in Neural Information Processing Systems* (2013).

[7] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. "Segnet: A deep convolutional encoder-decoder architecture for image segmentation". In: *arXiv preprint arXiv:1511.00561* (2015).

[8] Vasileios Belagiannis, Azade Farshad, and Fabio Galasso. "Adversarial network compression". In: *ECCV Workshops*. Springer Verlag, 2019. ISBN: 9783030110178. DOI: `10.1007/978-3-030-11018-5_37`.

[9] Davis Blalock et al. "What is the State of Neural Network Pruning?" In: *Proceedings of Machine Learning and Systems* (2020).

[10] Cristian Bucilă, Rich Caruana, and Alexandra Niculescu-Mizil. “Model compression”. In: *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 2006. ISBN: 1595933395. DOI: 10.1145/1150402.1150464.

[11] A.B. Chan, Z. Liang, and N. Vasconcelos. “Privacy preserving crowd monitoring: Counting people without people models or tracking”. In: *CVPR*. 2008.

[12] Arslan Chaudhry et al. “Riemannian walk for incremental learning: Understanding forgetting and intransigence”. In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 532–547.

[13] K. Chen et al. “Feature Mining for Localised Crowd Counting”. In: *BMVC*. 2012.

[14] Ke Chen et al. “Cumulative attribute space for age and crowd density estimation”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2013, pp. 2467–2474.

[15] Wenlin Chen et al. “Compressing Neural Networks with the Hashing Trick”. In: *32nd International Conference on Machine Learning, ICML 2015* (2015).

[16] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. “BinaryConnect: Training Deep Neural Networks with binary weights during propagations”. In: *Advances in Neural Information Processing Systems* (2015).

[17] Matthieu Courbariaux et al. “Binarized Neural Networks: Training Deep Neural Networks with Weights and Activations Constrained to +1 or -1”. In: *https://arxiv.org/abs/1602.02830* (2016).

[18] Le Cun et al. *Optimal Brain Damage*. Tech. rep. 1989.

[19] Matthias Delange et al. “A continual learning survey: Defying forgetting in classification tasks”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2021), pp. 1–1. DOI: 10.1109/TPAMI.2021.3057446.

[20] Emily Denton et al. “Exploiting Linear Structure Within Convolutional Networks for Efficient Evaluation”. In: *Advances in Neural Information Processing Systems* (2014).

[21] Xuanyi Dong et al. “More is Less: A More Complicated Network with Less Inference Complexity”. In: *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017* (), pp. 1895–1903.

[22] Maryam Fazel, Haitham Hindi, and Stephen P. Boyd. “A rank minimization heuristic with application to minimum order system approximation”. In: *Proceedings of the American Control Conference* 6 (2001), pp. 4734–4739. DOI: 10.1109/ACC.2001.945730.

[23] Luca Fiaschi et al. “Learning to count with regression forest and structured labels”. In: *Pattern Recognition (ICPR), 2012 21st International Conference on*. IEEE. 2012.

[24] Yunchao Gong et al. “Compressing Deep Convolutional Networks using Vector Quantization”. In: *https://arxiv.org/abs/1412.6115* (2014).

[25] Song Han, Huizi Mao, and William J. Dally. “Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding”. In: *4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings*. International Conference on Learning Representations, ICLR, 2016.

[26] Song Han et al. “Learning both Weights and Connections for Efficient Neural Networks”. In: *Advances in Neural Information Processing Systems* (2015).

[27] Babak Hassibi and David G Stork. *Second order derivatives for network pruning: Optimal Brain Surgeon*. Tech. rep. 1992.

[28] Kaiming He et al. “Deep residual learning for image recognition”. In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. IEEE Computer Society, 2016. ISBN: 9781467388504. DOI: 10.1109/CVPR.2016.90.

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

[30] Yang He et al. “Filter Pruning via Geometric Median for Deep Convolutional Neural Networks Acceleration”. In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition* (2018).

[31] Yang He et al. “Learning Filter Pruning Criteria for Deep Convolutional Neural Networks Acceleration”. In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. 2020. DOI: `10.1109/CVPR42600.2020.00208`.

[32] Yang He et al. “Soft Filter Pruning for Accelerating Deep Convolutional Neural Networks”. In: *IJCAI International Joint Conference on Artificial Intelligence* (2018).

[33] Yihui He, Xiangyu Zhang, and Jian Sun. “Channel Pruning for Accelerating Very Deep Neural Networks”. In: *ECCV Workshops* (2017).

[34] Yihui He et al. “AMC: AutoML for model compression and acceleration on mobile devices”. In: *European Conference on Computer Vision (ECCV)*. 2018.

[35] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. “Distilling the Knowledge in a Neural Network”. In: *NIPS 2014 Deep Learning Workshop* (2015).

[36] Saihui Hou et al. “Learning a unified classifier incrementally via rebalancing”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 831–839.

[37] Zejiang Hou and Sun-Yuan Kung. “A Feature-map Discriminant Perspective for Pruning Deep Neural Networks”. In: *https://arxiv.org/abs/2005.13796* (2020).

[38] Gao Huang et al. “Densely Connected Convolutional Networks”. In: *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017* (2016).

[39] Gao Huang et al. "Densely connected convolutional networks". In: *arXiv preprint arXiv:1608.06993* (2016).

[40] Zehao Huang and Naiyan Wang. "Data-Driven Sparse Structure Selection for Deep Neural Networks". In: *ECCV* (2018).

[41] Yerlan Idelbayev and Miguel Carreira-Perpiñán. "Low-rank compression of neural nets: Learning the rank of each layer". In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. 2020. DOI: `10.1109/CVPR42600.2020.00807`.

[42] Haroon Idrees, Nolan Warner, and Mubarak Shah. "Tracking in dense crowds using prominence and neighborhood motion concurrence". In: *Image and Vision Computing* 32.1 (2014), pp. 14–26.

[43] Haroon Idrees et al. "Multi-source multi-scale counting in extremely dense crowd images". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2013.

[44] H. Jacobs. "To count a crowd". In: *Columbia Journalism Review* 6 (1967), pp. 36–40.

[45] Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. "Speeding up convolutional neural networks with low rank expansions". In: *BMVC 2014 - Proceedings of the British Machine Vision Conference 2014*. British Machine Vision Association, BMVA, 2014. DOI: `10.5244/c.28.88`.

[46] Ghassen Jerfel et al. "Reconciling meta-learning and continual learning with online mixtures of tasks". In: *Advances in Neural Information Processing Systems* 32 (2019).

[47] Jia Deng et al. "ImageNet: A large-scale hierarchical image database". In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. 2009. DOI: `10.1109/cvprw.2009.5206848`.

[48] Glenn Jocher et al. *ultralytics/yolov3: Rectangular Inference, Conv2d + Batchnorm2d Layer Fusion*. 2019. DOI: 10.5281/ZENODO.2672652.

[49] Heechul Jung et al. “Less-forgetting Learning in Deep Neural Networks”. In: *ArXiv* abs/1607.00122 (2016).

[50] Yong-Deok Kim et al. “Compression of Deep Convolutional Neural Networks for Fast and Low Power Mobile Applications”. In: *4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings* (2015).

[51] James Kirkpatrick et al. “Overcoming catastrophic forgetting in neural networks”. In: *Proceedings of the national academy of sciences* 114.13 (2017), pp. 3521–3526.

[52] Dan Kong, Douglas Gray, and Hai Tao. “A viewpoint invariant approach for crowd counting”. In: *Pattern Recognition, 2006. ICPR 2006. 18th International Conference on*. Vol. 3. IEEE. 2006, pp. 1187–1190.

[53] Alex Krizhevsky and G Hinton. *Learning multiple layers of features from tiny images*. Tech. rep. 2009.

[54] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. “ImageNet classification with deep convolutional neural networks”. In: *Advances in Neural Information Processing Systems* (2012). ISSN: 15577317. DOI: 10.1145/3065386.

[55] Logan Lebanoff and Haroon Idrees. *Counting in dense crowds using deep learning*. 2015.

[56] Yann LeCun et al. “Gradient-based learning applied to document recognition”. In: *Proceedings of the IEEE* (1998). ISSN: 00189219. DOI: 10.1109/5.726791.

[57] Sang-Woo Lee et al. “Overcoming catastrophic forgetting by incremental moment matching”. In: *Advances in neural information processing systems* 30 (2017).

[58] Victor Lempitsky and Andrew Zisserman. “Learning To Count Objects in Images”. In: *NIPS*. 2010.

[59] Chong Li and C. J.Richard Shi. “Constrained optimization based low-rank approximation of deep neural networks”. In: *ECCV*. 2018.

[60] Hao Li et al. “Pruning Filters for Efficient ConvNets”. In: *5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings* (2016).

[61] Yawei Li et al. “Group Sparsity: The Hinge Between Filter Pruning and Decomposition for Network Compression”. In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition* (2020), pp. 8015–8024. DOI: 10.1109/CVPR42600.2020.00804.

[62] Zhizhong Li and Derek Hoiem. “Learning without Forgetting”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 40 (12 2018), pp. 2935–2947. ISSN: 19393539. DOI: 10.1109/TPAMI.2017.2773081.

[63] Lucas Liebenwein et al. “Provable Filter Pruning for Efficient Neural Networks”. In: (2019). URL: https://arxiv.org/abs/1911.07412v2.

[64] Mingbao Lin et al. “HRank: Filter Pruning using High-Rank Feature Map”. In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition* (2020).

[65] Shaohui Lin et al. “Accelerating convolutional networks via global & dynamic filter pruning”. In: *IJCAI International Joint Conference on Artificial Intelligence*. 2018. ISBN: 9780999241127. DOI: 10.24963/ijcai.2018/336.

[66] Shaohui Lin et al. “Holistic CNN Compression via Low-Rank Decomposition with Knowledge Transfer”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2019). ISSN: 19393539. DOI: 10.1109/TPAMI.2018.2873305.

[67] Shaohui Lin et al. “Towards Optimal Structured CNN Pruning via Generative Adversarial Learning”. In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition* (2019), pp. 2785–2794.

[68] Tsung Yi Lin et al. “Microsoft COCO: Common objects in context”. In: *European Conference on Computer Vision (ECCV)*. 2014. DOI: 10.1007/978-3-319-10602-1_48.

[69] Zhuang Liu et al. “Learning Efficient Convolutional Networks through Network Slimming”. In: *Proceedings of the IEEE International Conference on Computer Vision*. Institute of Electrical and Electronics Engineers Inc., 2017, pp. 2755–2763. ISBN: 9781538610329. DOI: 10.1109/ICCV.2017.298. eprint: 1708.06519.

[70] Jian-Hao Luo, Jianxin Wu, and Weiyao Lin. “ThiNet: A Filter Level Pruning Method for Deep Neural Network Compression”. In: *Proceedings of the IEEE International Conference on Computer Vision* (2017).

[71] Zheng Ma, Lei Yu, and Antoni B Chan. “Small instance detection by integer programming on object density maps”. In: *CVPR*. 2015.

[72] Arun Mallya and Svetlana Lazebnik. “PackNet: Adding Multiple Tasks to a Single Network by Iterative Pruning”. In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2018), pp. 7765–7773.

[73] Marc Masana, Tinne Tuytelaars, and Joost van de Weijer. “Ternary feature masks: continual learning without any forgetting”. In: *arXiv preprint arXiv:2001.08714* 4.5 (2020), p. 6.

[74] Zichen Miao et al. “Continual Learning with Filter Atom Swapping”. In: *International Conference on Learning Representations*. 2022.

[75] Breton Minnehan and Andreas Savakis. “Cascaded Projection: End-to-End Network Compression and Acceleration”. In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition* (2019).

[76] Pavlo Molchanov et al. “Importance estimation for neural network pruning”. In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. IEEE Computer Society, 2019. ISBN: 9781728132938. DOI: `10.1109/CVPR.2019.01152`.

[77] Pavlo Molchanov et al. “Pruning Convolutional Neural Networks for Resource Efficient Inference”. In: *5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings* (2016).

[78] Yuval Netzer and Tao Wang. “Reading digits in natural images with unsupervised feature learning”. In: *Advances in Neural Information Processing Systems* (2011).

[79] Daniel Onoro-Rubio and Roberto J López-Sastre. “Towards perspective-free object counting with deep learning”. In: *European Conference on Computer Vision*. Springer. 2016.

[80] Viet-Quoc Pham et al. “Count forest: Co-voting uncertain number of targets using random forest for crowd density estimation”. In: *Proceedings of the IEEE International Conference on Computer Vision*. 2015.

[81] Qiang Qiu et al. “DCFNet: Deep Neural Network with Decomposed Convolutional Filters”. In: *35th International Conference on Machine Learning, ICML 2018* (2018).

[82] Mohammad Rastegari et al. “XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks”. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* (2016).

[83] Sylvestre-Alvise Rebuffi et al. “iCaRL: Incremental Classifier and Representation Learning”. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2017), pp. 5533–5542.

[84] Joseph Redmon and Ali Farhadi. “YOLOv3: An Incremental Improvement”. In: (2018). URL: `https://arxiv.org/abs/1804.02767`.

[85] M. Rodriguez et al. “Density-aware person detection and tracking in crowds”. In: *ICCV*. 2011.

[86] David Rolnick et al. “Experience replay for continual learning”. In: *Advances in Neural Information Processing Systems* 32 (2019).

[87] Andrei A Rusu et al. “Progressive neural networks”. In: *arXiv preprint arXiv:1606.04671* (2016).

[88] David Ryan et al. “Crowd counting using multiple local features”. In: *Digital Image Computing: Techniques and Applications, 2009. DICTA’09*. 2009.

[89] Deepak Babu Sam, Shiv Surya, and R Venkatesh Babu. “Switching convolutional neural network for crowd counting”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. Vol. 1. 3. 2017, p. 6.

[90] Mark Schmidt. *Group L1-Regularization, Proximal-Gradient*. URL: `https://www.cs.ubc.ca/~schmidtm/Courses/540-W17/L5.pdf`.

[91] Joan Serra et al. “Overcoming catastrophic forgetting with hard attention to the task”. In: *International Conference on Machine Learning*. PMLR. 2018, pp. 4548–4557.

[92] Hanul Shin et al. “Continual learning with deep generative replay”. In: *Advances in neural information processing systems* 30 (2017).

[93] Karen Simonyan and Andrew Zisserman. “Very deep convolutional networks for large-scale image recognition”. In: *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*. International Conference on Learning Representations, ICLR, 2015.

[94] Vishwanath A Sindagi and Vishal M Patel. “Cnn-based cascaded multi-task learning of high-level prior and density estimation for crowd counting”. In: *Advanced Video and Signal Based Surveillance (AVSS), 2017 14th IEEE International Conference on*. IEEE. 2017, pp. 1–6.

[95] Vishwanath A Sindagi and Vishal M Patel. “Generating high-quality crowd density maps using contextual pyramid cnns”. In: *IEEE International Conference on Computer Vision*. 2017.

[96] Pravendra Singh et al. “Leveraging Filter Correlations for Deep Model Compression”. In: *Proceedings - 2020 IEEE Winter Conference on Applications of Computer Vision, WACV 2020* (2018).

[97] Korsuk Sirinukunwattana et al. “Locality sensitive deep learning for detection and classification of nuclei in routine colon cancer histology images”. In: *IEEE transactions on medical imaging* 35.5 (2016), pp. 1196–1206.

[98] Suraj Srinivas and R. Venkatesh Babu. “Data-free Parameter Pruning for Deep Neural Networks”. In: *British Machine Vision Conference*. British Machine Vision Association and Society for Pattern Recognition, 2015. DOI: 10.5244/c.29.31.

[99] Muhammad Tayyab, Fahad Ahmad Khan, and Abhijit Mahalanobis. “Compressing Deep CNNs Using Basis Representation and Spectral Fine-Tuning”. In: *2021 IEEE International Conference on Image Processing (ICIP)*. 2021, pp. 3537–3541. DOI: 10.1109/ICIP42928.2021.9506128.

[100] Michalis K. Titsias et al. “Functional Regularisation for Continual Learning with Gaussian Processes”. In: *International Conference on Learning Representations*. 2020.

[101] Vincent Vanhoucke, Andrew Senior, and Mz Mao. “Improving the speed of neural networks on CPUs”. In: *Deep Learning and Unsupervised Feature Learning Workshop, NIPS 2011* (2011). ISSN: 9781450329569.

[102] Vinay Kumar Verma et al. “Efficient Feature Transformations for Discriminative and Generative Continual Learning”. In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition* (Mar. 2021), pp. 13860–13870. ISSN: 10636919. DOI: 10.1109/CVPR46437.2021.01365.

[103] Chuan Wang et al. “Deep people counting in extremely dense crowds”. In: *Proceedings of the 23rd ACM international conference on Multimedia*. ACM. 2015.

[104] Dong Wang et al. “Exploring Linear Relationship in Feature Map Subspace for ConvNets Compression”. In: *https://arxiv.org/abs/1803.05729* (2018).

[105] Wei Wen et al. “Coordinating Filters for Faster Deep Neural Networks”. In: *Proceedings of the IEEE International Conference on Computer Vision* (2017).

[106] Mitchell Wortsman et al. “Supermasks in superposition”. In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 15173–15184.

[107] Jiaxiang Wu et al. “Quantized Convolutional Neural Networks for Mobile Devices”. In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition* (2015).

[108] Yue Wu et al. “Large Scale Incremental Learning”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 374–382.

[109] Yuhui Xu et al. “TRP: Trained Rank Pruning for Efficient Deep Neural Networks”. In: *IJCAI International Joint Conference on Artificial Intelligence* (2020).

[110] Shipeng Yan, Jiangwei Xie, and Xuming He. “Der: Dynamically expandable representation for class incremental learning”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, pp. 3014–3023.

[111] Jaehong Yoon et al. “Lifelong Learning with Dynamically Expandable Networks”. In: *International Conference on Learning Representations*. 2018. URL: `https://openreview.net/forum?id=Sk7KsfW0-`.

[112] Zhonghui You et al. “Gate Decorator: Global Filter Pruning Method for Accelerating Deep Convolutional Neural Networks”. In: *Advances in Neural Information Processing Systems* (2019).

[113] Lu Yu et al. “Semantic Drift Compensation for Class-Incremental Learning”. In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2020), pp. 6980–6989.

[114] Ruichi Yu et al. “NISP: Pruning Networks using Neuron Importance Score Propagation”. In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition* (2017).

[115] Friedemann Zenke, Ben Poole, and Surya Ganguli. “Continual learning through synaptic intelligence”. In: *International Conference on Machine Learning*. PMLR. 2017, pp. 3987–3995.

[116] Cong Zhang et al. “Cross-scene crowd counting via deep convolutional neural networks”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2015.

[117] Junting Zhang et al. “Class-incremental learning via deep model consolidation”. In: *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*. 2020, pp. 1131–1140.

[118] Xiangyu Zhang et al. “Accelerating Very Deep Convolutional Networks for Classification and Detection”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2016). ISSN: 01628828. DOI: `10.1109/TPAMI.2015.2502579`.

[119] Yingying Zhang et al. “Single-image crowd counting via multi-column convolutional neural network”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016.

[120] Chenglong Zhao et al. “Variational convolutional neural network pruning”. In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition* (2019), pp. 2775–2784. ISSN: 10636919.

[121] Zhuangwei Zhuang et al. “Discrimination-aware Channel Pruning for Deep Neural Networks”. In: *Advances in Neural Information Processing Systems* (2018).