IMPROVED KNOWLEDGE DISTILLATION WITH DYNAMIC NETWORK PRUNING

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

EREN ŞENER

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
COMPUTER ENGINEERING

AUGUST 2019

Approval of the thesis:

IMPROVED KNOWLEDGE DISTILLATION WITH DYNAMIC NETWORK PRUNING

submitted by **EREN ŞENER** in partial fulfillment of the requirements for the degree of **Master of Science in Computer Engineering Department, Middle East Technical University** by,

Prof. Dr. Halil Kalıpçılar
Dean, Graduate School of **Natural and Applied Sciences** _____

Prof. Dr. Halit Oğuztüzün
Head of Department, **Computer Engineering** _____

Assist. Prof. Dr. Emre Akbaş
Supervisor, **Computer Engineering, METU** _____

Examining Committee Members:

Assoc. Prof. Dr. Sinan Kalkan
Computer Engineering, METU _____

Assist. Prof. Dr. Emre Akbaş
Computer Engineering, METU _____

Assoc. Prof. Dr. Nazlı İkizler Cinbiş
Computer Engineering, Hacettepe University _____

Date:

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Surname: Eren Şener

Signature :

ABSTRACT

IMPROVED KNOWLEDGE DISTILLATION WITH DYNAMIC NETWORK PRUNING

Şener, Eren
M.S., Department of Computer Engineering
Supervisor: Assist. Prof. Dr. Emre Akbaş

August 2019, 46 pages

Deploying convolutional neural networks to mobile or embedded devices is often prohibited by limited memory and computational resources. This is particularly problematic for the most successful networks, which tend to be very large and require long inference times. In the past, many alternative approaches have been developed for compressing neural networks based on pruning, regularization, quantization or distillation. In this thesis, we propose the "Knowledge Distillation with Dynamic Pruning" (KDDP), which trains a dynamically pruned compact student network under the guidance of a large teacher network. In KDDP, we train the student network with supervision from the teacher network, while applying L_1 regularization on the neuron activations in a fully-connected layer. Subsequently, we prune inactive neurons. Our method automatically determines the final size of the student model. We evaluate the compression rate and accuracy of the resulting networks on image classification datasets, and compare them to results obtained by Knowledge Distillation (KD). Compared to KD, our method produces better accuracy and more compact models.

Keywords: knowledge distillation, model compression, deep learning

ÖZ

DİNAMİK AĞ BUDAMA YÖNTEMİYLE GELİŞTİRİLMİŞ BİLGİ DAMITMA

Şener, Eren
Yüksek Lisans, Bilgisayar Mühendisliği Bölümü
Tez Yöneticisi: Dr. Öğr. Üyesi. Emre Akbaş

Ağustos 2019 , 46 sayfa

Evrişimli sinirsel ağların mobil veya gömülü cihazlara yerleştirilmesi çoğu zaman sınırlı bellek ve hesaplama kaynakları tarafından kısıtlanır. Bu kısıtlama, özellikle çok büyük olma eğiliminde olan ve uzun çıkarım süreleri gerektiren başarılı ağlar için önemli bir sorundur. Geçmişte sinir ağlarını sıkıştırmak için budama, düzenlileştirme, nicemleme veya damıtma temelli birçok alternatif yaklaşım geliştirilmiştir. Bu tez çalışmasında, büyük bir öğretmen ağının rehberliğinde küçük bir öğrenci ağını dinamik bir şekilde sıkıştırarak eğiten "Dinamik Budama ile Bilgi Damıtma" (DBBD) yöntemini öneriyoruz. DBBD'de, öğrenci ağını öğretmen ağının denetiminde eğitirken, tam bağlantılı bir katmanın nöron aktivasyonlarına L_1 düzenlileştirmesi uyguluyoruz. Daha sonra aktif olmayan nöronları buduyoruz. Metodumuz, öğrenci modelinin son boyutunu kendisi otomatik olarak belirliyor. Ortaya çıkan ağların görüntü sınıflandırma veri setleri üzerindeki sıkıştırma oranını ve doğruluğunu inceleyip bunları Bilgi Damıtma (BD) metodundan elde edilen sonuçlarla karşılaştırıyoruz. Yöntemimizi BD ile karşılaştırdığımızda BD'den daha kompakt ve daha iyi doğruluk derecesine sahip modeller ürettiğini gözlemliyoruz.

Anahtar Kelimeler: bilgi damıtma, model sıkıştırma, derin öğrenme

To my family

ACKNOWLEDGMENTS

I would like to thank my thesis advisor Asst. Prof. Dr. Emre Akbaş for his endless patience. Without his patience and encouragement, this work couldn't be finished. I am so glad to have him as my thesis advisor. I want to thank Assoc. Prof. Dr. Nazlı İkizler Cinbiş and Assoc. Prof. Dr. Sinan Kalkan for accepting to be in my thesis committee.

Special thanks go to my ultimate research associate, İlke. I worked with him in every machine learning project I got involved in. He endured every failure and annoyance in those projects with me.

I would like to thank my mother Zehra, my father Ali, my sister Fadime, my brother-in-law Sebastian and my brother Ender for providing a peaceful and supportive environment.

Finally, I would like to thank the love of my life Canan for her absolute trust and support during the last year.

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

LIST OF ABBREVIATIONS

ABBREVIATIONS

KDDP	Knowledge Distillation with Dynamic Pruning
KD	Knowledge Distillation
CNN	Convolutional Neural Network
SN	Student Network
BN	Batch Normalization
FRL	Final Response Layer
SVD	Singular Value Decomposition
FER	Facial Expression Recognition

CHAPTER 1

INTRODUCTION

1.1 Motivation and Problem Definition

With the rise of general purpose graphical processing units, their increasing computational power recently enabled the use of neural networks in a wide range of fields such as computer vision, speech recognition or machine translation, amongst others. Human-computer interaction has heavily benefited from the success of the application of neural networks in these fields. Similar developments in the mobile phone industry require the deployment of such applications to mobile or embedded devices. Although they are very powerful, large neural networks consume high amounts of energy, storage, and computational resources, which are limited in mobile devices. Hence, creating small and fast models is crucial.

The central question that motivates the work in this thesis is how much the size limits of neural networks can be pushed so that one ends up with a compact image classification model that still works reasonably well. To achieve this, the first thing that comes to mind is making a large network smaller by removing redundant structures (weights, neurons, blocks, etc.). LeCunn *et al.* proposed one of the pioneering network compression approaches, the Optimal Brain Damage [7] method, which was followed by many magnitude-based network pruning methods [8, 9]. These approaches work by removing weights that are close to zero. To be able to prune more structures and make the models smaller, regularization can be used to enforce sparsity. Han *et al.* [1] proposed one of the first regularization-based model compression methods. Following this work, some other methods [10, 11, 12, 13] that use regularization on different structures were also proposed. For convolutional networks, researchers have

designed novel convolutional filters to save parameters which decrease redundancy [14, 15, 16]. Research on low-rank factorization methods([17, 18, 19]) tries to find informative parameters by using matrix or tensor decomposition. Some other works [20, 21, 22] try to reduce the number of bits that represent each weight.

Some works aim to train a compact network with the guidance of a "cumbersome" model. In the Knowledge Distillation (KD) method [2] (see Figure 1.1), there is a large, cumbersome model called the *teacher* and a much smaller model called the *student*. The student is trained to mimic the softmax values of the teacher via a hyper-parameter called *temperature* (see Eq. 1.1.1). Typically, neural networks produce probabilities by using a softmax output layer that converts the logits computed for each class into a probability by comparing with the other logits. In KD, temperature term helps neural networks to produce soft predictions to train the student network. KD based methods [23, 24] have good performances on computer vision tasks and have a big impact on model compression. However, a major disadvantage of KD is having a fixed size student model during or after training. In this work, we target removing this disadvantage and having more compact student networks by dynamic pruning based on neuron activations.

$$q_i = \frac{e^{z_i/T}}{\sum_j e^{z_j/T}}, \tag{1.1.1}$$

where q_i denotes the class probability, z_i is the logit and T is temperature.

1.2 Proposed Methods and Models

Knowledge Distillation was introduced by Hinton *et al.* [2] in 2015. The main idea is to have a cumbersome network called the teacher to supervise the training of a much smaller network called the student via soft outputs. The aim is to increase the information about the target classes by introducing uncertainty into probability distributions. Since these distributions contain similarity information on different classes, Hinton *et al.* further used this similarity information coming from the teacher to correctly classify a target class intentionally removed from the training set of the student. After training, the student correctly classified samples of this class despite never having

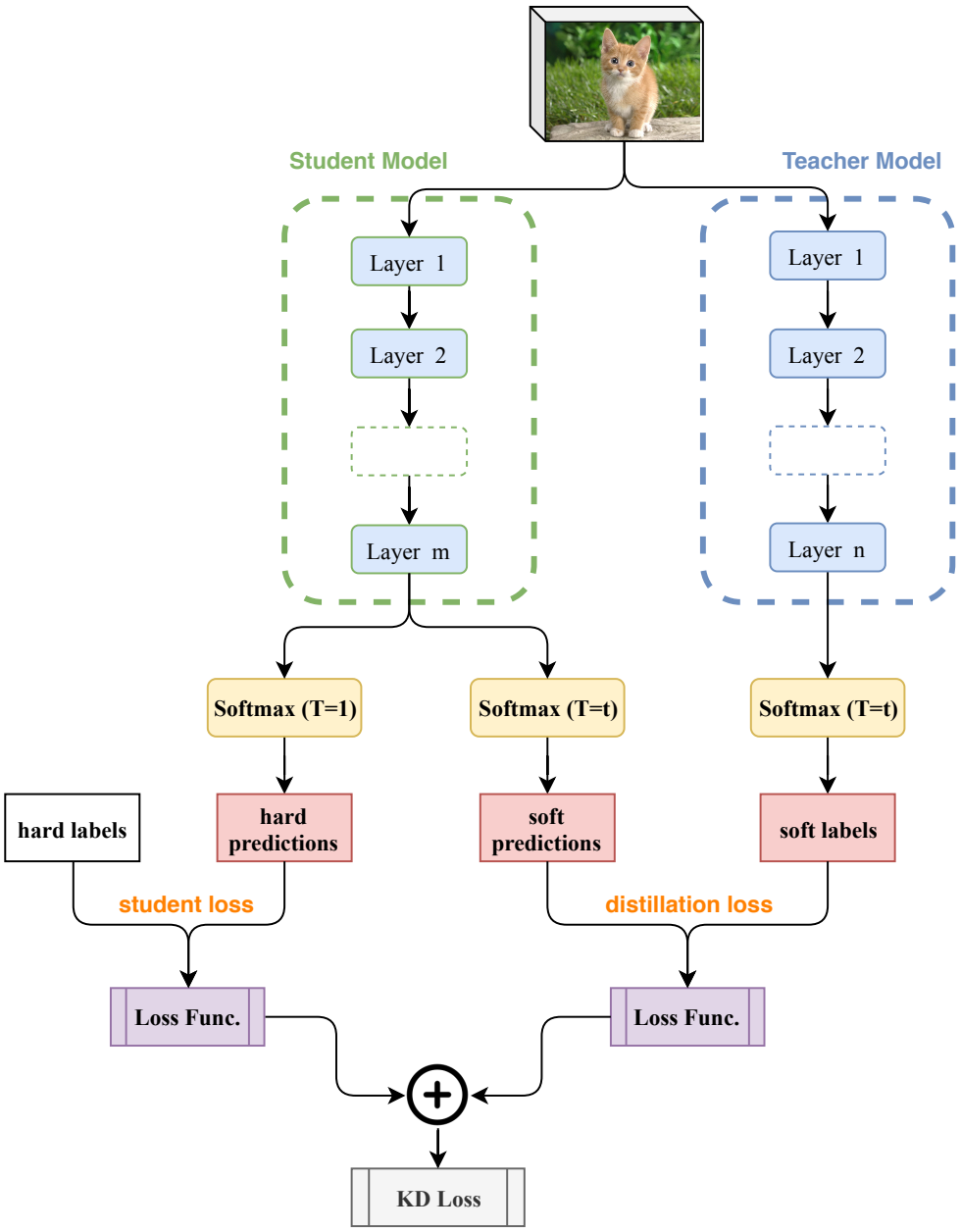


Figure 1.1: Illustration of the standard Knowledge Distillation method by Hinton *et al.* [2]. In KD, student model is trained with multiobjective loss that is composed of distillation loss and student loss. Former comes from soft labels, and the latter comes from hard labels. The student model in KD is given as input and its size does not change during or after training. However, in our method, we prune fully-connected layers in the student network with dynamic pruning based on neuron activations to get a compact model.

seen in training. Additionally, in order to prevent the teacher's strong predictions to dominate the similarity information, softmax logits of the teacher are softened using a hyper-parameter called temperature denoted as T in Eq. 3.2.1. The algorithm of KD is as follows: first, a large teacher network is trained for the task. Then, a much smaller student network is trained using both one-hot vectors of the true labels and the softened predictions (Eq. 3.2.2) of the teacher network. Thus, the student model is trained with multiobjective loss that is composed of distillation loss and student loss. Former is the cross-entropy loss between soft predictions of the student and soft predictions of the teacher network. Latter is the cross-entropy loss between true labels and predictions of the student. (see Figure 1.1)

In this thesis, we propose a dynamic network pruning method based on Knowledge Distillation. During traditional training of KD, we apply L_1 regularization on the activations of the neurons in a selected fully-connected layer in order to impose sparsity. Then we directly kill neurons with respect to the magnitude of their activations observed in the training set without making any assumption on the weights with respect to their numerical values. To the best of our knowledge, our compression technique is the first method that combines KD and regularization in this way. We name our method as Knowledge Disttilltion with Dynamic Pruning, or KDDP for short. Since our method can only prune fully-connected layers (and not convolutional layers), its typical targets are Multilayer Perceptrons (MLP) or CNNs with large fully-connected (`fc`) layers.

In this work, we extensively analyze both KD and our proposed method. On Cifar-10 dataset, we train standard KD networks, vanilla networks which are trained from scratch without any teacher guidance and our KDDP models. After training these models, we can conclude that our method performs better than the baselines (KD and Vanilla networks). Furthermore, we find that setting hyper-parameters is crucial for KD based methods. Temperature, T, distillation weight, α, and L_1 regularization penalty should be tuned to find a good balance between the model size and the classification performance. In summary, when the hyper-parameters are chosen carefully, our method works well.

1.3 Contributions and Novelties

Our contributions are as follows:

- We propose a new dynamic compression method based on KD. It dynamically prunes inactive neurons in selected fully-connected layers from the network. Our method does not require the final size of the compressed model as input; it is determined dynamically.
- We experimentally analyze our method and compare against KD. We make extensive experiments on our hyper-parameters to find meaningful relations with the accuracy of the compressed model.
- We test our method on the Cifar-10 dataset. We get better accuracies than the KD method with much fewer parameters.

1.4 The Outline of the Thesis

We first summarize the neural network model compression literature in Chapter 2. We describe our proposed method and its implementation details in Chapter 3. Then, we analyze the effectiveness of our approach with experiments performed on the Cifar-10 dataset in Chapter 4, and finally conclude in Chapter 5.

CHAPTER 2

RELATED WORK AND BACKGROUND

In this chapter, we review the literature on model compression in deep neural networks. We split these works into 5 categories (see Table 2.1). We give more detailed information about the parameter pruning and sharing due to its direct relation to our method.

2.1 Parameter Pruning and Sharing

Parameter pruning attracted many researchers since the early development of neural networks due to its effectiveness on reducing model complexity and over-fitting. It is also shown that pruning redundant parameters from the network improves generalization, which is an important side-effect.

Early works to prune parameters are Optimal Brain Damage [7] (OBD) and Optimal Brain Surgeon [8]. In their work, the authors remove redundant paramaters after sorting them by their saliencies. Saliency is measured based on the Hessian of the objective function with respect to parameters. Recently, Srinivas and Babu [9] also showed how similar neurons, which have similar weight sets, are redundant. Since Hessian computation is heavy, they propose a more systematic way than OBD and data-free method to remove them.

Most of the following works to above use sparsity constraints (L_0, L_1*-norm, etc.*) in the optimization problems to obtain redundancy. Researchers use these constraints on different elements (e.g. weights, blocks, etc.). Han *et al.* [1] are one of the first to propose a regularization-based method on model compression. They apply L_2 re-

Table 2.1: Summary of the approaches in model compression literature.

	Description	Application on	Training Strategy
Parameter Pruning and Sharing	Find redundant structures in models and prune them	Convolutional and fully connected layers	From scratch or pre-trained model
Knowledge Distillation	Train a compact network with the guidance of a cumbersome network	Convolutional and fully connected layers	Only from scratch
Low-rank Factorization	Find informative parameters by using matrix/tensor decomposition	Convolutional and fully connected layers	From scratch or pre-trained model
Compact Convolutional Filters	Design new convolutional filters to save parameters	Convolutional layers	Only from scratch
Quantization and Binarization	Try to reduce the number of bits required to represent each weight	Convolutional and fully connected layers	From scratch or pre-trained model

gularization during the training phase in order to have near zero-valued parameters. Then they prune all low-weight connections from the network. The deep compression method [21] uses the same procedure as Han *et al.* [1] for removing redundant connections. The authors also add quantization and Huffman coding on top of the pruned network to have a more compact one.

Recently, redundancy in convolutional networks also has been explored. Lebedev and Lempitsky [25] apply the idea of Optimal Brain Damage[7] to convolutional filters. They remove entries of $L_{2,1}$-norm regularization applied convolution filters, which are below a threshold, in a group-wise fashion. Similarly, work by Zhou *et al.* [10] enforce low-rank constraints on tensors and $L_{2,1}$-norm regularization on the objective function during the training stage to achieve compact CNNs with reduced neurons. Another study which uses $L_{2,1}$-norm is Wen *et al.* [11]'s work. They apply regularization to big baseline models to learn more compact CNNs. With their structured sparsity method, they regularize filters, channels, filter shapes and layer depth of CNNs. Huang and Wang [26] improve the method of Wen *et al.* [11] and propose a more general end-to-end method for network pruning. Their method contains a factor to scale the outputs of some specific structures (neurons, groups or blocks). They apply L_1-norm sparsity regularization to the scaling factors. Structures having scaling factors below a threshold are removed from the network while training. Unlike the previous works, Ullrich *et al.* [27] base their regularization on the soft weight-sharing method [28]. They compress weights of the pre-trained model into clusters

by fitting mixtures of Gaussian models. After retraining the model with new weights concentrated on the cluster means, they obtain a layer-wise-pruned compact network.

There are also methods which focus on sparsity in batch-normalization (BN) layers. Liu *et al.* [12] add a scaling factor after BN layers. L_1 regularization is imposed on these scaling factors during training to automatically identify redundant filters. Then, they prune channels with near-zero scaling factors. Another recent study [29] uses the method proposed by Beck and Teboulle [30] to enforce sparsity on the γ-parameter in BN operator. During training, this method makes some γ values zero and helps these channels to block sample-wise (for each sample in the training set) information flow. After the training is completed, they remove these constant-valued channels from the original network. The study MorphNet [13] uses a combination of three ideas above: first, an L_1-norm-based regularization of the neurons, second, the idea of multipliers of Howard *et al.* [31] for reducing the floating point operations and model size, and third, the paradigm introduced by Han *et al.* [1] for retraining of the pruned network.

There has also been some research for measuring the redundancy in the networks. Guo *et al.* [32] present a feedback mechanism named splicing which re-establishes mistakenly removed parameters after the pruning operation. With this work, they show that measuring the redundancy of the parameters is an extremely difficult task. Researchers use different techniques for measuring redundancy. In [33] L_1-norm of kernels are calculated. After sorting kernels by their L_1-norm values, small valued kernels and corresponding feature maps were pruned. ThiNet [34] does filter-level pruning based on filter statistics computed from the following layer, not the current layer. In spite of their success, the compression rate of the filters had to be predefined, which is another difficult problem for pruning methods. Moreover, He *et al.* [35] exploit feature maps for redundancy. The authors select the most representative channels of the feature maps and prune the redundant ones. After pruning, in order not to damage accuracy, they reconstruct the outputs with the remaining channels using linear least squares.

Recently, several methods proposed to measure the importance of structures. Yu *et al.* [36] propose that layer-by-layer network pruning leads to significant reconstruction error propagation. They introduce a global neuron importance measuring algorithm

which uses information at the Final Response Layer (FRL, the second-to-last layer before classification). The algorithm obtains the importance of all neurons in the network with a single backward pass after a feature ranking operation on the FRL. Subsequently, the trimming of the whole network is performed considering the pruning ratio per layer as a pre-defined hyper-parameter. Prakash *et al.* [37] propose a novel inter-filter orthogonality metric for ranking filter importance and a new training strategy. Their method consists of temporarily dropping (some) of the least important convolutional filters (ranked by their metric), and reintroducing dropped filters with new weights. They repeat this process cyclically. With this strategy, they improve generalization and reduced overlap of learned features. Unlike the traditional deterministic methods, Wang *et al.* [38] approach pruning weights of convolutional layers in a probabilistic manner. They specify a pruning probability for each weight group. At each iteration, these probabilities are updated with the L_1 norm as an importance criterion of each weight group. The pruning is guided by sampling from the pruning probabilities. He *et al.* [39] use a novel pruning method instead of norm-based pruning approaches. They calculate the geometric median [40] of the filters within the same layer, and prune the filter(s) near to the geometric median. In addition to above works, Dong *et al.* [41] improve the idea in previous works [7, 8]. Their pruning method is based on second order derivatives of a layer-wise error function.

There are also some recent and novel compression techniques used for pruning. In SplitNet[42], the goal is to find a tree-structured network that contains a set or a hierarchy of subnetworks, where the leaf-level subnetworks are associated with a specific group of classes. Since each group uses a subset of features that are completely disjoint from the ones used by other groups, the splitting algorithm prunes out inter-group connections while optimizing the cross entropy loss and the group regularization. At the end, the weight matrix can be explicitly split into block diagonal matrices to reduce the number of parameters. Similarly, Yang *et al.* [43] approach the network compression from energy consumption of the network. They sort layers by their energy consumption, and pruned weights, which have small magnitudes, of the layers that consume the most energy first. A very recent work, similar to the work of Liu *et al.* [12], Zhao *et al.* [44] modify the BN layer and add a new parameter called channel saliency to the BN layer. They try to find approximate gamma distributions

over these channel saliency parameters. They then remove redundant channels with mean and variance of their gamma distributions less than predefined thresholds.

2.2 Knowledge Distillation

Knowledge Distillation is a simple way to have compact deep learning models. In this method, we train a large and cumbersome (teacher) network or an ensemble model which can extract important features from the data and can produce better predictions. Then, we train a small (student) network with the guidance of the cumbersome model. This small network will be able to produce comparable results with its teacher, in some cases.

In 2006, Caruana *et al.* [45] approach the idea of knowledge transfer from a different point of view. Instead of training a neural network on an original small set, they use an ensemble of base-level classifiers to label a large unlabeled dataset and then train the network on this much larger dataset. Ba and Caruana[46] propose using an L_2 loss on the logits to mimic the teacher network. Hinton *et al.* [2] show that a student network could imitate the soft output of a larger teacher network or ensemble of networks. In other words, the student model can be trained with the distilled knowledge obtained from the teacher network.

FitNets *et al.* [23] are rooted in the knowledge distillation method proposed by Hinton *et al.* [2] to produce deep and thin student networks with comparable or better performance than the teacher. They achieve this by training some layers of the student beforehand with the teacher's supervision for better initialization. Luo *et al.* [24], show that instead of using soft targets in the output layer, the knowledge of the teacher can be obtained from the top hidden layer. They use L_2 loss while mimicking the teacher's feature space. Yim *et al.* [47] distill knowledge from the teacher by generating a matrix from feature maps at each layer. Then, they transfer the knowledge from teacher to student, which has the same depth as the teacher, by applying L_2 loss to these matrices.

Different from the standard classification tasks, Chen *et al.* [48] introduce a novel end-to-end trainable framework for multi-class object detection problem through

knowledge distillation. They propose three ideas: a weighted cross-entropy loss to prevent the impact of background domination during classification, a teacher bounded regression loss for knowledge distillation to avoid the damage that may arise from contradiction bounding-box regression outputs of the teacher with the ground truth, and adaptation layers for hint learning which is proposed by Romero *et al.* [23].

2.3 Low-rank Factorization

Low-rank factorization methods aim at finding informative parameters by using matrix/tensor decomposition. Denton *et al.* [17] exploit redundancy present in convolutional filters by deriving approximations to reduce the required computation. They apply several decomposition methods based on SVD to compress weight matrices. Yu *et al.* [49] propose a unified deep compression framework that decomposes weight matrices into their low-rank and sparse components. They show that their method helps the models to achieve higher compression rates than models using SVD. Recently, Minnehan and Savakis [19] present a data-driven approach that compresses both the present layer and inputs of the next layer by projecting them to the same low dimensional space based on a low-rank projection. In another recent work, Kim*et al.* [18] propose novel accuracy metrics to describe the relationship between the accuracy and complexity of a network. They use these metrics to find the right rank configuration of the whole network which satisfies the compression constraints. Since they obtain this configuration in a non-iterative and fast way, they move ahead of SVD-based network compression methods which decide the right rank for every layer of the network separately.

2.4 Compact Convolutional Filters

Designing new convolutional filters and training models with compact filters is another way of model compression. Jin *et al.* [14] create CNNs with 1D filters which are constructed from 3D convolutional filters. They show that they can do such substitution without a loss in accuracy and the new CNN provides two times speed-up during feed-forward pass when it is compared to the baseline model. Fire modules are intro-

duced in SqueezeNet [15]. This module is composed of a squeeze convolution layer, which has only 1×1 filters, and an expand layer that has a mix of 1×1 and 3×3 filters. With this module, they presented a CNN architecture that has 50x fewer parameters than AlexNet[50] and maintains AlexNet-level accuracy on ImageNet [51]. Recently, Li *et al.* [16] proposed a factorized convolutional filter which consists of a standard convolution filter and a binary scalar, together with a dot-product operator between them. After training CNN models with these convolutions, they only keep filters corresponding to the 1-valued scalars to obtain a compact model.

2.5 Quantization and Binarization

Works related to quantization and binarization try to reduce the number of bits required to represent each weight. In BinaryConnect [20] proposes a method for training a DNN with binary weights ($\{+1, -1\}$) during the forward and backward propagations. This gives a great advantage to specialized deep learning hardware because many multiply-accumulate operations are replaced with simple accumulations. Hubara *et al.* [22] proposes a method based on this idea in which they train Binarized Neural Networks - neural networks with binary weights and activations at run-time. Zhou *et al.* [52] presents a method where they convert the weights of a pre-trained full-precision CNN model into a low-precision one with weights which are constrained to be powers of two or zero. In a recent work, Son *et al.* [53] proposes a novel and more structured quantization method. They cluster 3×3 kernels and replace redundant ones with their centroids. The final compressed model is represented with a set of centroids and a corresponding cluster index per each kernel.

CHAPTER 3

METHOD

In this thesis, we target image classification using convolutional neural networks. We assume an input set of images, each annotated with a class label. Our goal is to design an architecture that both includes much fewer parameters than a large network and can perform comparably.

In this chapter, we present our model architecture and implementation details of our method.

3.1 Knowledge Distillation

The key challenge in the design of our model is the way of decreasing the number of parameters in a convolutional neural network while maintaining accuracy as high as possible. In the Knowledge Distillation (KD) [2] method, the aim is to distill the knowledge from a teacher network to a student network. The teacher network is much larger than the student network in methods using KD. As large and deep models have more capacity than smaller networks, they can extract more features of higher complexity, and therefore perform better. The motivation is to train the small network with the help of the large network. If the model used as a teacher generalizes well, the student model can be trained to generalize in the same way with KD methods.

In our work, we use two convolutional neural networks. Our teacher network is deep and large, and our student network is shallow and small. With this choice, our goal is to show that it is possible to transfer knowledge from the large teacher to a much smaller student network, which is then pruned to an even more compact network.

3.1.1 Teacher Network

After AlexNet[50] won the ILSVRC[54] image classification challenge in 2012 by a large margin, creating models with deep layers has gained a lot of attention. First, VGG[55] explored the capabilities of a network with 19 layers and became the state of the art by winning the ILSVRC challenge in 2013. The year after GoogLeNet[56] won the challenge with a network of 22 layers. Finally, Resnet[3] achieved the top performance with 34, 101 and 152 layered very deep architectures.

Although it sounds trivial to continuously increase the depth of a network for better performance, it poses a very important problem: increasing the depth of the network comes at the price of vanishing gradients. In ResNet, the authors successfully solved this problem by adding identity-mappings to their network.

ResNet and its variants proved their success on many computer vision tasks. As the core visual processing module, we use a ResNet variant for our teacher model due to its state-of-the-art performance on the ImageNet challenge. We use ResNet-56 which has 6.97% error rate on Cifar-10 and 850k parameters, which is sufficient for us to use it as the teacher.

3.1.2 Student Network

We choose our student network to have a very simple architecture in order to efficiently analyze the performance of our method.

Our student network architecture starts with an input layer for $32 \times 32 \times 3$ sized images. It is followed by a convolutional layer with a kernel size of 7×7 with a stride of 1, with 64 convolution filters. This result in an output of size $16 \times 16 \times 64$. The convolutional layer is followed by a Batch Normalization (BN) layer and a non-linear activation function, ReLU [57]. We later use a max-pooling layer which has a window size of 3×3 with stride 2 that produces an output of size $7 \times 7 \times 64$. This layer is followed by an identity-block of the ResNet architecture [3]. ResNet's identity block is composed of 3 convolutional layers, each followed by a BN and a ReLU layer. The first and third convolutional layers have a kernel size of 1×1, and the middle layer

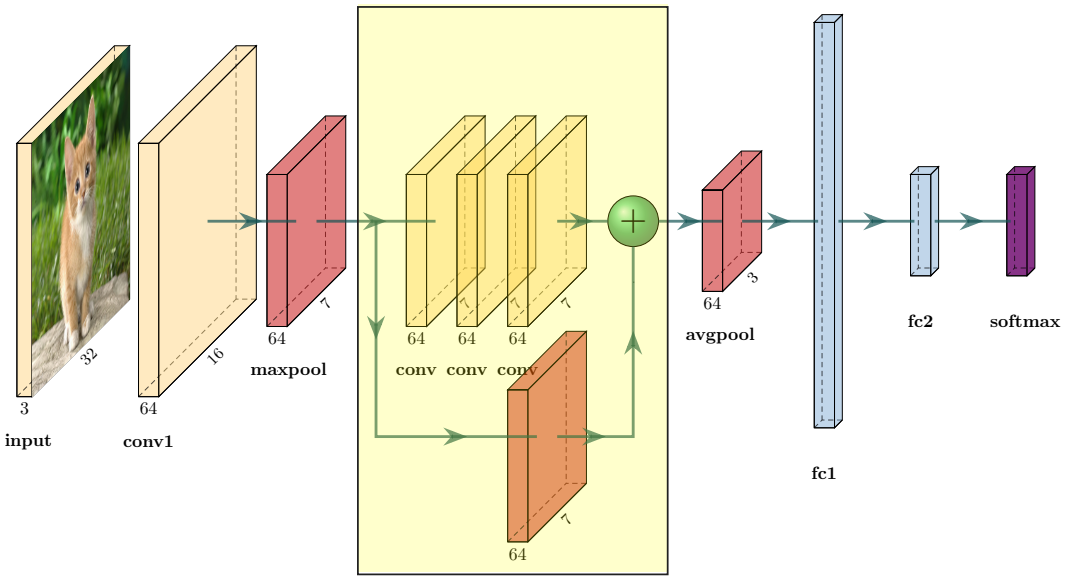


Figure 3.1: Student Network overview. The network takes an image and outputs a class label. It is composed of an input layer followed by a convolutional layer, max pooling, an identity-block of ResNet[3], average pooling, two fully connected layers, and a softmax layer. ResNet's identity block is highlighted with the yellow rectangle. This layer is composed of three convolutional layers and a skip connection which adds the input of the identity block and the output of the last convolutional layer in the identity block. (Best viewed in color.)

has a kernel size of 3×3. The stride of all convolutions of the identity blocks is 1 and the number of filters used in each layer is 64. Before the ReLU layer of the last convolutional layer inside the residual block, there is a skip connection that allows the flow of information from the initial layers to the last layers by adding the input of the identity block and the output of the ReLU layer. The identity-block is followed by an average pooling layer which outputs a $3 \times 3 \times 64$-dimensional tensor. This layer is followed by a fully-connected layer, `fc1`, and a ReLU layer. Finally, the ReLU layer is followed by another fully connected layer, `fc2`, as a bridge to a softmax layer at the end.

In our experiments, we use three different networks with different neuron counts in the first fully-connected layer, `fc1`. We use 50, 100 and 500 neurons for this layer to explore the effect of the increasing number of neurons. We set the number of neurons in the second fully-connected layer, `fc2`, to the number of classes in the classification task at hand. In Figure 3.1, we present the architecture of our student network visually.

3.2 Method

Knowledge Distillation [2] successfully showed a way of transferring the generalizations of a complex model to a much lighter model. Let p_t be the softened output of the teacher network's softmax layer and z_i be the logits of the teacher network. Furthermore, let p'_s be the hard and p_s be the soft output of the student network's softmax layer and v_i be the logits of the student network. Knowledge Distillation is parameterized by weight α and temperature T. p_t, p_s and p'_s are defined as follows:

$$p_t = \frac{e^{z_i/T}}{\sum_j e^{z_j/T}}, \quad p_s = \frac{e^{v_i/T}}{\sum_j e^{v_j/T}}, \quad p'_s = \frac{e^{v_i}}{\sum_j e^{v_j}}, \tag{3.2.1}$$

The objective function is composed of a weighted average of two loss functions and is computed as follows, given N training images with ground truth labels y_n:

$$\mathcal{L} = \alpha \left(\frac{1}{N} \sum_{n=1}^{N} \mathcal{H}(p_t, p_s) \right) + (1 - \alpha) \left(\frac{1}{N} \sum_{n=1}^{N} \mathcal{H}(y_n, p'_s) \right). \tag{3.2.2}$$

The first term is the cross-entropy $\mathcal{H}$ with the soft targets. It helps the student network to mimic the teacher's softened outputs. This term is the building block of KD. The temperature T in here is used for selecting a suitable set of soft targets from the logits of the teacher. During training of the student network, it tries to match these soft targets. The second term is the cross-entropy with the correct labels. The weight α sets the contribution of these two objective functions, i.e. the weight of distillation. We experimentally analyze these two hyper-parameters in Chapter 4.

We build our model on the KD [2] method. To decrease the number of neurons further and explore our limits we apply regularization and pruning methods to our model. We call our model "*Knowledge Distillation with Dynamic Pruning*", or *KDDP* for short, as our pruning process changes the number of neurons at the end.

Typically, fully-connected layers contain most of the parameters in a convolutional neural network (except the fully-convolutional networks [58, 59, 60]). Our proposed method reduces the number of parameters in the `fc`-layers of a network by removing neurons based on their average activations observed on the training set. We use L_1 regularization to induce sparsity on the activations of the `fc`-layers.

In our method, we train our student networks with Eq. 3.2.2 and apply L_1 regularization to the fully connected layer with the higher amount of neurons, the `fc1` layer in our case (see Table 3.1). Then, for each neuron at `fc1`, we calculate the activation of the neuron for all samples in the training set. We take the average of the activations. If the activation of the neuron is below 10^{-6}, we kill that neuron and delete the corresponding weight set in `fc2`. After testing all neurons at `fc1`, we retrain the pruned network with Eq.3.2.2 and without L_1 regularization on `fc1`.

KDDP, Student Network (SN): We present the pseudo-code of our method in Algorithm 1. At the first stage, the network is trained with Eq. 3.2.2 and an L_1 regularization penalty on the dynamic `fc1` layer. After neurons are pruned from the `fc1` layer, the network is trained with Eq. 3.2.2 and without penalty.

Table 3.1: Student networks differ only in the number of neurons in the `fc1` layer. Percentages in parenthesis indicate the ratio of the parameters in `fc1` to the total number of parameters.

Model	# of neurons in `fc1`	# of parameters at `fc1`	total # of parameters
SN_{50}	50	29k (34%)	85k
SN_{100}	100	58k (50%)	114k
SN_{500}	500	289k (83%)	349k

Algorithm 1 Dynamic Model Size Reduction

1: Train KDDP model with L_1 reg. on `fc1` using Eq.3.2.2
2: **Inputs:** N training samples (x_j, y_j)
3: **for** each neuron n_i in `fc1` **do**
4: // calculate the avg. activation
5: $o_i = \frac{1}{N} \sum_{j=1}^{N} ReLu(w_i \cdot x_j + b_i)$
6: **if** $o_i \leq 10^{-6}$ **then**
7: // prune low activity neurons
8: Kill the neuron n_i and delete the corresponding w_i in `fc2`
9: **end if**
10: **end for**
11: Train KDDP model without L_1 reg. on `fc1` using Eq. 3.2.2

3.2.1 Baseline methods

We compare our method with the following models.

- **Vanilla SN:** We train the student network from scratch without any teacher guidance and regularization penalty. We use this model to find out the baseline performance of our student networks.
- **Vanilla-KD SN:** We train student network with Knowledge Distillation [2] at different temperature values (T) but without regularization penalty.

3.3 Implementation Details

Teacher Network (TN): We train a ResNet-56 model from scratch. The base learning rate is set as 10^{-4} and remained constant through iterations, mini-batch size is 64, and the optimization algorithm is Adam [61].

Student Networks (SN): We use the same hyper-parameters while training all student models. All models are trained from scratch. Weights and biases are initialized with Xavier's initialization [62]. Network architectures are implemented via the Keras framework [63]. Adam [61] is used for training. The learning rate is set to 10^{-4}, and the mini-batch size is 64. An L_1 regularization penalty is applied on `fc1` during the training of the KDDP student networks. The training is stopped early if there is no improvement in the accuracy on the validation set for 50 epochs.

CHAPTER 4

EXPERIMENTS

In this chapter, first, we evaluate our method in Section 4.2 by comparing against other methods and provide extensive experiments on the hyper-parameters in Section 4.3.

4.1 Dataset

Cifar-10 [64]: We use the Cifar-10 dataset in our experiments. It contains 60000 32x32 color images in 10 classes, with 6000 images per class (see example images in Figure 4.1). There are 50000 training images and 10000 test images. We select 10000 images of the training set as a validation set using a stratified sampling strategy [65]. We report our results after seeing no improvement on the validation set for 50 epochs during training. To augment data, we only flip images horizontally. We, also, subtract pixel mean from all images. We use this setup in all experiments.

Figure 4.1: Example images from the Cifar-10 dataset

4.2 Analysis of Proposed Method

We present our KDDP model's results in Table 4.1. We use the same teacher logits in for all experiments (see Eq. 3.2.2). We train our teacher once. We use the same initial weights for all student network trainings in Table 4.1 with hyper-parameters: $L_1 = 1e^{-4}$ and $\alpha = 0.5$. We also train Vanilla SNs and Vanilla-KD SNs for each model to explore the capacity of these networks and compare with our model.

In Table 4.1, our teacher network has an accuracy of 88.08% with 1.7M parameters on the test set. This score is much lower than the current state-of-the-art performances, which is 97.92 [66]. This is because we are using 10K samples of our training set as validation data to have a solid early stopping criterion. With data augmentation and 40k samples, our teacher does its best. However, we don't aim to have state-of-the-art performances on Cifar-10. We want to explore the weaknesses and strengths of our knowledge distillation method. Thus, the current performance of the teacher network suffices our expectations.

The Vanilla SN shows $80.48\%, 80.75\%, 81.28\%$ accuracies on the test set for our networks $SN_{50}, SN_{100}, SN_{500}$, respectively. We can also see that increasing the number of neurons in the `fc1` layer has a positive effect on model performance. However, this causes an increase in the number of parameters, as well. Compared to the Vanilla SNs, in Table 4.1, we observe that the Vanilla-KD improves accuracies approximately by 1% for all models. This improvement tells us that the teacher network does its job and successfully increases performance.

Our method, KDDP, shows $81.93\%, 82.19\%, 82.81\%$ accuracies on the test set for our networks $SN_{50}, SN_{100}, SN_{500}$, respectively. We observe that our method works better than Vanilla-KD and improves the accuracies around 0.5% for $SN_{50}, SN_{100}, SN_{500}$ with $3\%, 9\%, 40\%$ fewer parameters than their original networks. It removes 10%, $17\%, 48\%$ of the parameters at `fc1`.

We also do further detailed experiments to compare our method against Vanilla-KD to provide fair comparisons based on the total number of neurons in the network. We train smaller Vanilla-KD SNs (having the same neuron count at `fc1` with final KDDP SNs) using the same weights. We refer to these experiments 'KD' in Table 4.1 and

Table 4.1: Results for models $SN_{50}, SN_{100}, SN_{500}$. Hyper-parameters: L_1 penalty= $1e^{-4}$, $\alpha = 0.5$

			KD	KDDP	
		T	**Acc.**	**Acc.**	**Final fc1 size**
50 Neurons		2	**0.8141**	**0.8193**	**45**
		4	0.8117	0.8123	40
		8	0.8050	0.7891	41
		12	0.7930	0.7913	38
	Model **Acc.** **Params** Teacher 0.8808 1,673,738 Vanilla SN_{50} 0.8048 85,104 Vanilla-KD SN_{45} 0.8145 82,169	16	0.7900	0.7902	33
		20	0.7960	0.8011	37
		32	0.8030	0.8010	37
		64	0.8050	0.8078	37
		100	0.8072	0.8088	32
		200	0.8007	0.8053	41
		1000	0.8065	0.8065	35
		5000	0.7947	0.8032	40
100 Neurons		2	**0.8197**	**0.8219**	**83**
		4	0.8183	0.8156	79
		8	0.8083	0.8037	61
		12	0.8039	0.7952	59
	Model **Acc.** **Params** Teacher 0.8808 1,673,738 Vanilla SN_{100} 0.8075 114,454 Vanilla-KD SN_{83} 0.8146 104,475	16	0.7942	0.7922	59
		20	0.7988	0.7978	59
		32	0.8055	0.8085	61
		64	0.8090	0.8095	59
		100	0.8134	0.8077	54
		200	0.8074	0.8003	54
		1000	0.8066	0.8171	53
		5000	0.8104	0.8087	54
500 Neurons		2	0.8225	**0.8281**	**264**
		4	**0.8264**	0.8186	156
		8	0.8206	0.8106	113
		12	0.8141	0.8079	118
	Model **Acc.** **Params** Teacher 0.8808 1,673,738 Vanilla SN_{500} 0.8128 349,254 Vanilla-KD SN_{264} 0.8267 210,722	16	0.8159	0.7952	109
		20	0.8064	0.8003	91
		32	0.8062	0.8147	111
		64	0.8096	0.8121	109
		100	0.8147	0.8114	101
		200	0.8088	0.8146	112
		1000	0.8069	0.8140	116
		5000	0.8097	0.8093	105

we present our results over different Temperatures and the number of neurons. For example, we train the Vanilla-KD models with $\{45, 83, 264\}$ neurons at their `fc1` layer for our networks $SN_{50}, SN_{100}, SN_{500}$, with a Temperature of 2 in the first rows. We observe that the higher temperatures result in better performances and our method outperforms the Vanilla-KD models with the same amount of parameters in the `fc1` layer.

We conclude that pruning after training with Knowledge Distillation helps in increasing the performance with much fewer parameters.

4.3 Hyper-parameter Analysis

4.3.1 L_1 Regularization Penalty Analysis

We use L_1 regularization on the activations of `fc1` layer neurons to increase sparsity. L_1 regularization penalizes the absolute value of the activations of the neurons. We present results for different L_1 penalties in Table 4.2. We set α to 0.5 in these experiments.

We observe that larger values of L_1 penalty result in fewer active neurons at the `fc1` layer and therefore decreases the performance of the models. For example, when L_1 is $1e^{-3}$ and $T = 32$ at KDDP SN_{100} experiment, the model gets stuck at some local minima and cannot even reach the vanilla model's performance. However, when there are fewer parameters it helps the model to get acceptable performances. For example, for hyper-parameters $L = 1e^{-3}, T = 2$, our KDDP SN_{50} model achieves better performance than the other SN_{50} models. We also observe that using smaller values for L_1, e.g. $L_1 = 1e^{-5}$, does not work for our pruning method in all student models. Therefore, L_1 penalty should be tuned to strike a good balance between the model size and the classification performance. In our experiments, we set the L_1 regularization to $1e^{-4}$.

Table 4.2: Results for models $SN_{50}, SN_{100}, SN_{500}$. Hyper-parameters: $\alpha = 0.5$

		$L_1 = 1e^{-3}$		$L_1 = 1e^{-4}$		$L_1 = 1e^{-5}$	
	T	**Acc.**	**FC1 size**	**Acc.**	**FC1 size**	**Acc.**	**FC1 size**
50 Neurons	2	**0.8198**	**15**	**0.8193**	**45**	**0.8162**	**49**
	4	0.8054	10	0.8123	40	0.8146	48
	8	0.7638	7	0.7891	41	0.7968	47
	12	0.7707	7	0.7913	38	0.7918	47
	16	0.7761	8	0.7902	33	0.7948	46
	20	0.7802	8	0.8011	37	0.7961	47
	32	0.7607	5	0.8010	37	0.8049	48
	64	0.7776	7	0.8078	37	0.8034	44
	100	0.7871	6	0.8088	32	0.8038	49
	200	0.7966	7	0.8053	41	0.8077	47
	1000	0.7877	8	0.8065	35	0.8078	46
	5000	0.7616	5	0.8032	40	0.8069	43
100 Neurons	2	**0.8132**	**18**	**0.8219**	**83**	**0.8219**	**97**
	4	0.7913	9	0.8156	79	0.8192	93
	8	0.7655	7	0.8037	61	0.8099	92
	12	0.7931	10	0.7952	59	0.7941	92
	16	0.7970	76	0.7922	59	0.7931	92
	20	0.7802	7	0.7978	59	0.7935	89
	32	0.6900	3	0.8085	61	0.8001	93
	64	0.7779	6	0.8095	59	0.8012	91
	100	0.7594	4	0.8077	54	0.8121	91
	200	0.7800	6	0.8003	54	0.8055	90
	1000	0.7609	4	0.8171	53	0.8058	95
	5000	0.8063	89	0.8087	54	0.8010	88
500 Neurons	2	0.8211	103	**0.8281**	**264**	0.8185	467
	4	**0.8213**	**330**	0.8186	156	**0.8196**	**453**
	8	0.8175	400	0.8106	113	0.8101	441
	12	0.8017	140	0.8079	118	0.8110	432
	16	0.8010	235	0.7952	109	0.8089	429
	20	0.7991	151	0.8003	91	0.8075	425
	32	0.8047	98	0.8147	111	0.8063	425
	64	0.8050	17	0.8121	109	0.8025	417
	100	0.8040	90	0.8114	101	0.8030	431
	200	0.8062	185	0.8146	112	0.8076	423
	1000	0.8059	15	0.8140	116	0.8040	425
	5000	0.8123	17	0.8093	105	0.8070	423

4.3.2 α Analysis

α in Eq. 3.2.2 sets the contribution of the two objective functions (i.e. the weight of distillation). In other words, using bigger α values means giving more importance to soft targets in the objective function.

We present results for different α values in Table 4.3. We can see that too small and large α values don't lead to good performances. α also should be tuned carefully to have a good balance between the model size and classification performance. In our experiments, we observe that setting α to 0.5 gives the best results.

4.3.3 T Analysis

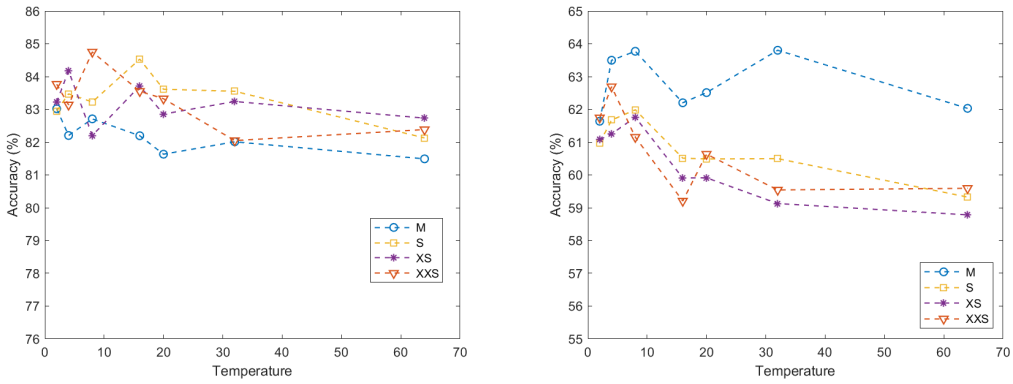
Temperature (T) is a tool to enforce the uncertainty of a teacher network to emerge. This uncertainty might be used as similarity information between different classes to enhance the training. There is no formulation for selecting the most effective T; it is set empirically. We did a grid search over the temperatures values of [2, 4, 8, 12, 16, 20, 32, 64, 100, 200, 1000, 5000]. If we keep increasing T, at some point, logits will be saturated and no information will flow from the teacher to the student network. We present our results in Table 4.3. We can see that when we train the network with 100 neurons solely with the loss coming from the soft targets (the first term in Eq. 3.2.2) with a temperature of 5000, we get an accuracy of 10%, which is equal to the random guess for Cifar-10.

In Figure 4.3, we present our results in Table 4.1 in plots to show the trends for different temperature values to see the unpredictable behavior of T easier. For all models, the accuracy fluctuates between low and high values as the Temperature is swept across a wide range. Therefore, we can say that the temperature parameter should also be tuned carefully.

In another Knowledge Distillation paper [4], this accuracy fluctuation due to T is also shown on Facial Expression Recognition (FER) datasets, CK+ [5] and Oulu-Casia [6]. These two datasets are widely used benchmark databases for FER problem. They contain face images of subjects each having different emotions such as anger,

Table 4.3: Results for models $SN_{50}, SN_{100}, SN_{500}$. Hyper-parameters: $L_1 = 1e^{-4}$

		$\alpha = 0.2$		$\alpha = 0.5$		$\alpha = 0.8$		$\alpha = 1$	
	T	**Acc**	**FC1 size**	**Acc**	**FC1 size**	**Acc**	**FC1 size**	**Acc**	**FC1 size**
50 Neurons	2	0.8046	48	**0.8193**	**45**	0.8093	49	0.7955	45
	4	0.8003	44	0.8123	40	**0.8147**	**39**	**0.8034**	**32**
	8	0.7987	42	0.7891	41	0.8031	26	0.7967	8
	12	0.7945	43	0.7913	38	0.7843	28	0.5061	25
	16	0.8093	43	0.7902	33	0.7726	23	0.5844	46
	20	0.8081	45	0.8011	37	0.7761	25	0.5882	45
	32	0.8058	39	0.8010	37	0.7872	23	0.5101	42
	64	0.8060	42	0.8078	37	0.7961	25	0.5850	42
	100	0.8021	44	0.8088	32	0.8069	24	0.5835	42
	200	0.8057	42	0.8053	41	0.8018	22	0.5526	39
	1000	0.8072	44	0.8065	35	0.8088	26	0.6277	40
	5000	**0.8106**	**42**	0.8032	40	0.8040	26	0.1000	12
100 Neurons	2	0.8077	76	**0.8219**	**83**	**0.8239**	**87**	0.8078	66
	4	0.8082	75	0.8156	79	0.8177	69	**0.8108**	**49**
	8	0.8042	66	0.8037	61	0.8041	33	0.7948	8
	12	0.8041	75	0.7952	59	0.7880	30	0.6685	88
	16	0.8005	69	0.7922	59	0.7733	26	0.5931	63
	20	0.8010	79	0.7978	59	0.7689	31	0.5886	40
	32	0.8075	69	0.8085	61	0.7889	25	0.5764	49
	64	0.8065	79	0.8095	59	0.7962	33	0.5835	51
	100	0.7963	72	0.8077	54	0.7985	33	0.5890	85
	200	0.8047	67	0.8003	54	0.8037	29	0.5748	85
	1000	0.8039	71	0.8171	53	0.8000	34	0.5864	86
	5000	**0.8096**	**75**	0.8087	54	0.8073	31	0.1000	14
500 Neurons	2	**0.8170**	**247**	**0.8281**	**264**	**0.8255**	**262**	**0.8158**	**212**
	4	0.8138	205	0.8186	156	0.8215	126	0.8157	57
	8	0.8100	204	0.8106	113	0.8133	44	0.6642	407
	12	0.8065	177	0.8079	118	0.7863	28	0.5768	256
	16	0.8092	181	0.7952	109	0.7872	27	0.5890	319
	20	0.8070	187	0.8003	91	0.7734	24	0.5843	397
	32	0.8112	184	0.8147	111	0.7925	29	0.6397	123
	64	0.8087	178	0.8121	109	0.8017	28	0.6458	394
	100	0.8072	183	0.8114	101	0.8068	27	0.7257	389
	200	0.8110	189	0.8146	112	0.8085	36	0.6410	135
	1000	0.8069	189	0.8140	116	0.8070	36	0.6374	395
	5000	0.8116	176	0.8093	105	0.8039	33	0.1001	120



(a) CK+ (b) Oulu-CASIA

Figure 4.2: Classification performances of the student networks in [4] across different temperatures on the CK+ [5] (left) and Oulu-CASIA [6] (right) datasets.

disgust, fear, happiness etc. In this work, they use an Inception_v3 [67] model, which has 21.8M parameters, as the teacher network and several custom architecture student networks having 900K, 232K, 121K and 65K parameters (Medium-size (M), Small-size (S), X-Small-size (XS), XX-Small-size (XXS) models respectively). They train student networks with traditional KD. Their results on above datasets are presented in Figure 4.2. Again, we can say that accuracy shows similar fluctuations to the method proposed in this thesis when the temperature is selected across a wide range.

4.3.4 Activity Threshold Analysis

In our method, we directly prune neurons which have an activity smaller than 10^{-6} (see Algorithm 1). We extensively analyzed different activity threshold values. Results of SN_{100} model are presented at Table 4.4. We observe that selection of activity threshold value is not too critical. For different small values, model trained with our method gives similar accuracy and compression rates. Since L_1 regularization forces the weights to be reduced to zero, we can set a small value to activity threshold parameter. We use 10^{-6} as activity threshold in our all experiments.

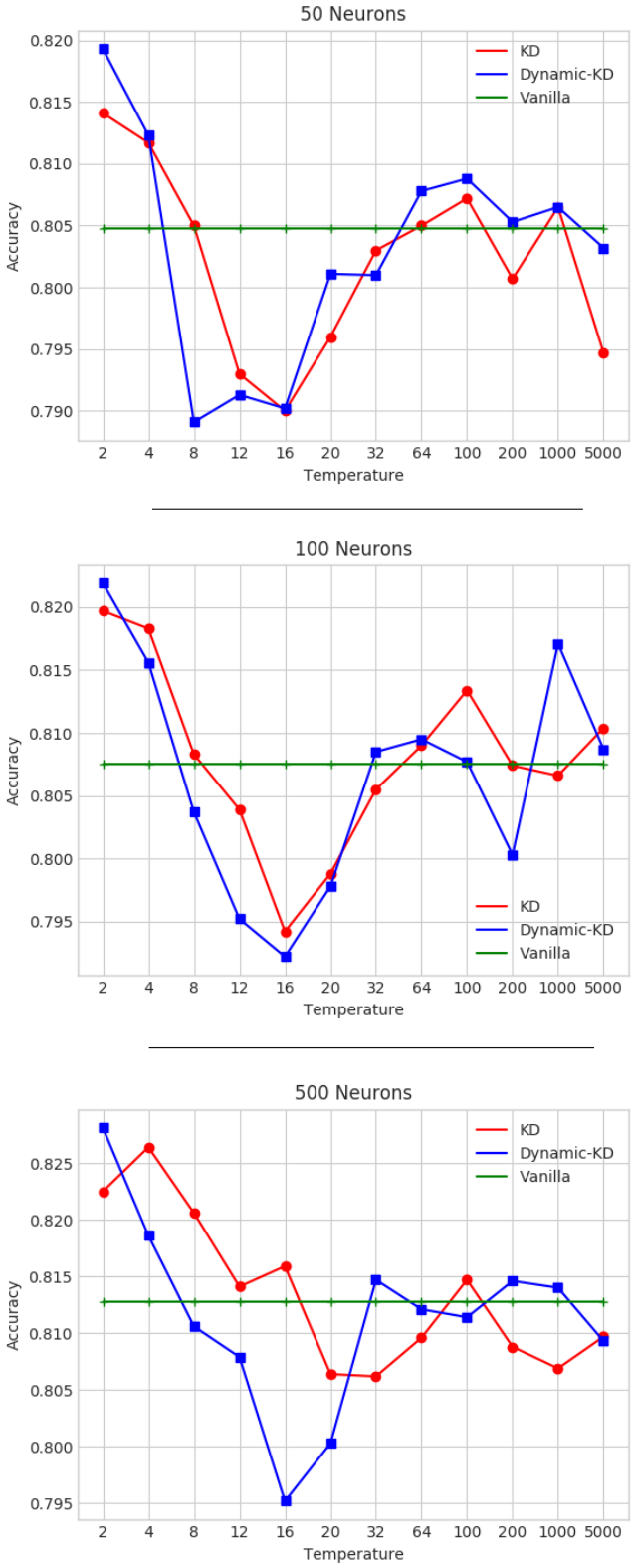


Figure 4.3: Plot of results presented in Table 4.1

Table 4.4: SN_{100} model results for different activity thresholds. Hyper-parameters: $L_1 = 1e^{-4}$ and $\alpha = 0.5$

		$1e^{-1}$		$1e^{-4}$		$1e^{-6}$		$1e^{-7}$	
	T	**Acc**	**FC1 size**	**Acc**	**FC1 size**	**Acc**	**FC1 size**	**Acc**	**FC1 size**
100 Neurons	2	**0.8203**	**85**	**0.8221**	**84**	**0.8219**	**83**	**0.8219**	**87**
	4	0.8125	74	0.8149	76	0.8156	79	0.8200	71
	8	0.7969	55	0.8019	58	0.8037	61	0.7935	55
	12	0.7952	54	0.7923	61	0.7952	59	0.7967	58
	16	0.7939	60	0.7958	54	0.7922	59	0.7939	58
	20	0.7924	61	0.7903	55	0.7978	59	0.7903	59
	32	0.8014	54	0.8095	57	0.8085	61	0.8048	52
	64	0.8080	59	0.8079	62	0.8095	59	0.8031	54
	100	0.8038	56	0.8046	61	0.8077	54	0.8084	57
	200	0.8105	60	0.8047	60	0.8003	54	0.8055	57
	1000	0.8022	62	0.8029	56	0.8171	53	0.8043	58
	5000	0.8049	58	0.8084	57	0.8087	54	0.8074	52

Table 4.5: Results for KD based weight killing methodology of models $SN_{50}, SN_{100}, SN_{500}$. Hyper-parameters: $L_1 = 1e^{-4}, \alpha = 0.5$ and weight killing threshold $= 1e^{-6}$,

	50 Neurons		**100 Neurons**		**500 Neurons**	
T	**Acc**	**Comp. Ratio**	**Acc**	**Comp. Ratio**	**Acc**	**Comp. Ratio**
2	0.8141	0.4725	**0.8265**	**0.4843**	**0.8322**	**0.5157**
4	0.8102	0.4676	0.8150	0.4838	0.8257	0.5123
8	0.8089	0.4660	0.8060	0.4875	0.8220	0.5228
12	0.7918	0.4661	0.8080	0.4989	0.8094	0.5229
16	0.7955	0.4795	0.8017	0.4957	0.8168	0.5191
20	0.8016	0.4826	0.8095	0.5021	0.8168	0.5204
32	0.8112	0.4865	0.8108	0.4933	0.8134	0.5283
64	0.8071	0.4833	0.8134	0.5030	0.8172	0.5217
100	0.8042	0.4904	0.8163	0.5034	0.8290	0.5249
200	0.8142	0.4822	0.8101	0.5072	0.8257	0.5247
1000	0.8127	0.4946	0.8144	0.4968	0.8204	0.5214
5000	**0.8177**	**0.4961**	0.8135	0.5027	0.8190	0.5247
Our Method	**0.8193**	**0.0344**	**0.8219**	**0.0871**	**0.8281**	**0.3966**

Table 4.6: Results for weight killing methodology [1] applied Vanilla models $SN_{50}, SN_{100}, SN_{500}$. Hyper-parameters: $L_1 = 1e^{-4}$ and weight killing threshold $= 1e^{-6}$,

	50 Neurons		**100 Neurons**		**500 Neurons**	
	Acc	**Comp. Ratio**	**Acc**	**Comp. Ratio**	**Ratio**	**Comp. Ratio**
[1]	0.8130	0.4899	0.8100	0.5029	0.8195	0.5156
Our Method	**0.8193**	0.0344	**0.8219**	0.0871	**0.8281**	0.3966

4.4 Comparison with the Literature

As we stated earlier, our work is the first KD based model compression work. In order to assess the performance of our method, we need to compare with other works in the literature. Although we do our experiments on Cifar-10 dataset, we cannot directly compare compression performances with them. Since we use customized neural networks, we have to implement those methods in order to compare with. We implemented compression method in [1]. In their work, Han *et al.* first trains the network from scratch without any guidance with L_1 regularization on the weights and then prunes weights below a threshold. We present our results for Vanilla SN models at Table 4.6. It is clearly seen that although the compression ratio of the method in [1] is appealing, we do better than this work in terms of accuracy on the test set for all models. And our method outputs compact neural network models at the end. We have to code extra structures for training and prediction of the model since killed weights are spread all over the model and our output model has different architecture. However, in our method, we prune the whole neurons and the compact output model has the same architecture with the initial one.

Moreover, we did some experiments by adding Han *et al.* 's method to our KD based model compression technique. We changed our neuron pruning strategy with pruning weights below a threshold from the network. We experimented on our SN_{50}, SN_{100}, SN_{500} models and results are presented at Table 4.5 We see that pruning based on weights gives better performances and better compression ratios. For example, for SN_{500} model, we get 83% accuracy with a network having just half of the parameters. We observe similar situations for the other models too. Therefore, our KD based

compression method can be improved with different pruning methods.

4.5 Statistical Analysis of the Results

We use Welch's T-test [68] to measure the significance of our method's results. We train our KDDP SN_{100} models with different initial weights 10 times. We set our hyper-parameters as $L_1 = 1e^{-4}, \alpha = 0.5$. We present our results Table 4.7. The t-test is used to determine if there is a significant difference between the means of two sets. In inferential statistics, it is assumed that dependent variable fits a normal distribution. When a normal distribution exists, the probability of a particular outcome can be calculated. The level of probability, ρ, we are willing to accept is pre-determined, and $\rho < .05$ is a commonly used value. A T-score is calculated using Eq.4.5.1. T-score is compared to a critical value in the T-Table to see whether the results fall within the acceptable level of probability. Based on this comparison one can determine whether the difference between the means occurred by chance or the sets have intrinsic differences.

$$t = \frac{\text{mean}_1 - \text{mean}_2}{\sqrt{\frac{{\text{var}_1}^2}{n_1} + \frac{{\text{var}_1}^2}{n_2}}}, \tag{4.5.1}$$

where n is the samples size and var is the variance.

KDDP & Vanilla Analysis:) We start with assuming a null hypothesis that the mean of the results of the KDDP is equal to the mean of the results of the Vanilla network. Then, we calculate the T-score of these sets (classification results) using Eq. 4.5.1. We get a T-score of 9.91 which is greater than the value in the T-table, 1.812 (one-tail, alpha=.05, degree of freedom=10). Therefore, it is safe to reject the null hypothesis that there is no difference between the means of results.

KDDP & Vanilla-KD Analysis:) We follow the same computations for comparing our KDDP with the Vanilla-KD. We get a T-score of 2.9730 which is greater than the value in the T-table 1.812 (one-tail, alpha=.05, degree of freedom=10). Therefore, it is again safe to reject the null hypothesis. We conclude that our KDDP model's performance has intrinsic differences from Vanilla and Vanilla-KD results, and they

Table 4.7: Table presents 11 run results of best performing KDDP SN_{100} models with hyper-parameters: $L_1 = 1e^{-4}, \alpha = 0.5$. (Model 0 is the model that we use its results throughout the paper.)

Model	**Vanilla Acc.**	**Vanilla-KD Acc.**	**KDDP Acc.**
0	0.8075	0.8197	0.8219
1	0.8041	0.8156	0.8226
2	0.8089	0.8182	0.8234
3	0.7975	0.8125	0.8166
4	0.8051	0.8172	0.8183
5	0.8101	0.8167	0.8175
6	0.7998	0.8210	0.8221
7	0.8080	0.8193	0.8212
8	0.8033	0.8163	0.8210
9	0.8098	0.8139	0.8152
10	0.7980	0.8169	0.8233
Mean	0.8047	0.8170	0.8203
Var	1.9522e^{-5}	5.6783e^{-6}	7.5033e^{-6}

KDDP & Vanilla T-Score	9.9177
KDDP & Vanilla-KD T-Score	2.9730

are strong and are not by chance.

CHAPTER 5

CONCLUSION AND FUTURE WORK

In this thesis, we target image classification using convolutional neural networks. Our goal is to design an architecture that both includes much fewer parameters than a large network and can perform comparably. We propose a new method based on Knowledge Distillation (KD) [2]. We use L_1 regularization on the activities of the neurons in a fully-connected layer and remove the inactive neurons. There is no need to provide the final size of the student model as input; our method determines it automatically. Our method performs better than the KD with much fewer parameters. After doing an extensive analysis of our method, we can say that we do better than KD.

In our experiments, we extensively analyze the strengths and weaknesses of our proposed method. In our experiments, we show that KD based methods including ours are highly hyper-parameters dependent. Temperature, T, and distillation weight, α selection determine the performance of the trained model. We observe that the accuracy fluctuates between low and high values for temperature values selected across a wide range. Moreover, α constrains us to decide to what extent we should rely on the teacher network's logits. However, when the hyper-parameters are chosen carefully, our method works well. It performs better than the baselines.

In conclusion, our method can be used when there is a need for a much smaller network that performs comparably. Moreover, considering the benefits such as comparable accuracy with fewer parameters, one should expect that the hyper-parameter selection is vital for the performance.

Lastly, we are curious about the potential of our algorithm for model size reduction for other layers such as convolutional layers in CNNs. We want to explore the effects

of different pruning strategies in more detail. We also want to explore the applicability of selecting the hyper-parameters dynamically. We consider these as future work.

REFERENCES

[1] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *Advances in neural information processing systems*, pp. 1135–1143, 2015.

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

[3] K. He, X. Zhang, S. Ren, and J. Sun, "Identity mappings in deep residual networks," in *European conference on computer vision*, pp. 630–645, Springer, 2016.

[4] İ. Çuğu, E. Şener, and E. Akbaş, "Microexpnet: An extremely small and fast model for expression recognition from frontal face images," *arXiv preprint arXiv:1711.07011*, 2017.

[5] P. Lucey, J. F. Cohn, T. Kanade, J. Saragih, Z. Ambadar, and I. Matthews, "The extended cohn-kanade dataset (ck+): A complete dataset for action unit and emotion-specified expression," in *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition-Workshops*, pp. 94–101, IEEE, 2010.

[6] G. Zhao, X. Huang, M. Taini, S. Z. Li, and M. PietikäInen, "Facial expression recognition from near-infrared videos," *Image and Vision Computing*, vol. 29, no. 9, pp. 607–619, 2011.

[7] Y. LeCun, J. S. Denker, S. A. Solla, R. E. Howard, and L. D. Jackel, "Optimal brain damage.," in *NIPs*, vol. 2, pp. 598–605, 1989.

[8] B. Hassibi and D. Stork, "Second order derivatives for network pruning: Optimal brain surgeon," in *Advances in Neural Information Processing Systems 5,[NIPS Conference]*, pp. 164–171, 1992.

[9] S. Srinivas and R. V. Babu, "Data-free parameter pruning for deep neural net-

works," in *Proceedings of the British Machine Vision Conference 2015, BMVC 2015, Swansea, UK, September 7-10, 2015*, pp. 31.1–31.12, 2015.

[10] H. Zhou, J. M. Alvarez, and F. Porikli, "Less is more: Towards compact cnns," in *European Conference on Computer Vision*, pp. 662–677, Springer, 2016.

[11] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, "Learning structured sparsity in deep neural networks," in *Advances in neural information processing systems*, pp. 2074–2082, 2016.

[12] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang, "Learning efficient convolutional networks through network slimming," in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 2736–2744, 2017.

[13] A. Gordon, E. Eban, O. Nachum, B. Chen, H. Wu, T.-J. Yang, and E. Choi, "Morphnet: Fast & simple resource-constrained structure learning of deep networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1586–1595, 2018.

[14] J. Jin, A. Dundar, and E. Culurciello, "Flattened convolutional neural networks for feedforward acceleration," in *3rd International Conference on Learning Representations, ICLR, San Diego, CA, USA, May 7-9, 2015, Workshop Track Proceedings*, 2015.

[15] F. N. Iandola, M. W. Moskewicz, K. Ashraf, S. Han, W. J. Dally, and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <1mb model size," *CoRR*, vol. abs/1602.07360, 2016.

[16] T. Li, B. Wu, Y. Yang, Y. Fan, Y. Zhang, and W. Liu, "Compressing convolutional neural networks via factorized convolutional filters," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3977–3986, 2019.

[17] E. L. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus, "Exploiting linear structure within convolutional networks for efficient evaluation," in *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13, Montreal, Quebec, Canada*, pp. 1269–1277, 2014.

[18] H. Kim, M. U. K. Khan, and C.-M. Kyung, "Efficient neural network compression," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 12569–12577, 2019.

[19] B. Minnehan and A. Savakis, "Cascaded projection: End-to-end network compression and acceleration," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 10715–10724, 2019.

[20] M. Courbariaux, Y. Bengio, and J. David, "Binaryconnect: Training deep neural networks with binary weights during propagations," in *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems, Montreal, Quebec, Canada*, pp. 3123–3131, 2015.

[21] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding," in *4th International Conference on Learning Representations, ICLR*, 2016.

[22] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks," in *Advances in neural information processing systems*, pp. 4107–4115, 2016.

[23] A. Romero, N. Ballas, S. E. Kahou, A. Chassang, C. Gatta, and Y. Bengio, "Fitnets: Hints for thin deep nets," in *3rd International Conference on Learning Representations, ICLR, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[24] P. Luo, Z. Zhu, Z. Liu, X. Wang, and X. Tang, "Face model compression by distilling knowledge from neurons," in *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[25] V. Lebedev and V. Lempitsky, "Fast convnets using group-wise brain damage," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2554–2564, 2016.

[26] Z. Huang and N. Wang, "Data-driven sparse structure selection for deep neural networks," in *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 304–320, 2018.

[27] K. Ullrich, E. Meeds, and M. Welling, "Soft weight-sharing for neural network compression," in *5th International Conference on Learning Representations, ICLR, Toulon, France, April 24-26, 2017, Conference Track Proceedings*, 2017.

[28] S. J. Nowlan and G. E. Hinton, "Simplifying neural networks by soft weight-sharing," *Neural computation*, vol. 4, no. 4, pp. 473–493, 1992.

[29] J. Ye, X. Lu, Z. Lin, and J. Z. Wang, "Rethinking the smaller-norm-less-informative assumption in channel pruning of convolution layers," in *6th International Conference on Learning Representations, ICLR, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*, 2018.

[30] A. Beck and M. Teboulle, "A fast iterative shrinkage-thresholding algorithm for linear inverse problems," *SIAM journal on imaging sciences*, vol. 2, no. 1, pp. 183–202, 2009.

[31] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[32] Y. Guo, A. Yao, and Y. Chen, "Dynamic network surgery for efficient dnns," in *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems, Barcelona, Spain*, pp. 1379–1387, 2016.

[33] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient convnets," in *5th International Conference on Learning Representations, ICLR, Toulon, France, April 24-26, 2017, Conference Track Proceedings*, 2017.

[34] J.-H. Luo, J. Wu, and W. Lin, "Thinet: A filter level pruning method for deep neural network compression," in *Proceedings of the IEEE international conference on computer vision*, pp. 5058–5066, 2017.

[35] Y. He, X. Zhang, and J. Sun, "Channel pruning for accelerating very deep neural networks," in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1389–1397, 2017.

[36] R. Yu, A. Li, C.-F. Chen, J.-H. Lai, V. I. Morariu, X. Han, M. Gao, C.-Y. Lin, and L. S. Davis, "Nisp: Pruning networks using neuron importance score propagation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 9194–9203, 2018.

[37] A. Prakash, J. Storer, D. Florencio, and C. Zhang, "Repr: Improved training of convolutional filters," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 10666–10675, 2019.

[38] H. Wang, Q. Zhang, Y. Wang, and H. Hu, "Structured probabilistic pruning for convolutional neural network acceleration," in *British Machine Vision Conference 2018, BMVC, Northumbria University, Newcastle, UK, September 3-6, 2018*, p. 149, 2018.

[39] Y. He, P. Liu, Z. Wang, Z. Hu, and Y. Yang, "Filter pruning via geometric median for deep convolutional neural networks acceleration," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4340–4349, 2019.

[40] P. T. Fletcher, S. Venkatasubramanian, and S. Joshi, "Robust statistics on riemannian manifolds via the geometric median," in *2008 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1–8, IEEE, 2008.

[41] X. Dong, S. Chen, and S. Pan, "Learning to prune deep neural networks via layer-wise optimal brain surgeon," in *Advances in Neural Information Processing Systems*, pp. 4857–4867, 2017.

[42] J. Kim, Y. Park, G. Kim, and S. J. Hwang, "Splitnet: Learning to semantically split deep networks for parameter reduction and model parallelization," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 1866–1874, JMLR. org, 2017.

[43] T.-J. Yang, Y.-H. Chen, and V. Sze, "Designing energy-efficient convolutional neural networks using energy-aware pruning," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 5687–5695, 2017.

[44] C. Zhao, B. Ni, J. Zhang, Q. Zhao, W. Zhang, and Q. Tian, "Variational con-

volutional neural network pruning," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2780–2789, 2019.

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

[46] J. Ba and R. Caruana, "Do deep nets really need to be deep?," in *Advances in neural information processing systems*, pp. 2654–2662, 2014.

[47] J. Yim, D. Joo, J. Bae, and J. Kim, "A gift from knowledge distillation: Fast optimization, network minimization and transfer learning," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4133–4141, 2017.

[48] G. Chen, W. Choi, X. Yu, T. Han, and M. Chandraker, "Learning efficient object detection models with knowledge distillation," in *Advances in Neural Information Processing Systems*, pp. 742–751, 2017.

[49] X. Yu, T. Liu, X. Wang, and D. Tao, "On compressing deep models by low rank and sparse decomposition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 7370–7379, 2017.

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

[51] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A Large-Scale Hierarchical Image Database," in *CVPR09*, 2009.

[52] A. Zhou, A. Yao, Y. Guo, L. Xu, and Y. Chen, "Incremental network quantization: Towards lossless cnns with low-precision weights," *arXiv preprint arXiv:1702.03044*, 2017.

[53] S. Son, S. Nah, and K. Mu Lee, "Clustering convolutional kernels to compress deep neural networks," in *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 216–232, 2018.

[54] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.

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

[56] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9, 2015.

[57] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proceedings of the 27th international conference on machine learning (ICML-10)*, pp. 807–814, 2010.

[58] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3431–3440, 2015.

[59] J. Dai, Y. Li, K. He, and J. Sun, "R-fcn: Object detection via region-based fully convolutional networks," in *Advances in neural information processing systems*, pp. 379–387, 2016.

[60] L. Wang, W. Ouyang, X. Wang, and H. Lu, "Visual tracking with fully convolutional networks," in *Proceedings of the IEEE international conference on computer vision*, pp. 3119–3127, 2015.

[61] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

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

[63] F. Chollet *et al.*, "Keras." `https://keras.io`, 2015.

[64] A. Krizhevsky, G. Hinton, *et al.*, "Learning multiple layers of features from tiny images," tech. rep., Citeseer, 2009.

[65] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[66] H. Cai, L. Zhu, and S. Han, "Proxylessnas: Direct neural architecture search on target task and hardware," in *7th International Conference on Learning Representations, ICLR, New Orleans, LA, USA, May 6-9, 2019*, 2019.

[67] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2818–2826, 2016.

[68] B. L. Welch, "The generalization ofstudent's' problem when several different population variances are involved," *Biometrika*, vol. 34, no. 1/2, pp. 28–35, 1947.