A Study on Knowledge Transfer Techniques to Support Deep Learning on Edge Devices

by

Ragini sai sri lakshmi sistla

A Thesis Presented in Partial Fulfillment
of the Requirements for the Degree
Master of Science

Approved March 2018 by the
Graduate Supervisory Committee:

Ming Zhao, Chair
Baoxin Li
Hanghang Tong

ARIZONA STATE UNIVERSITY

May 2018

ABSTRACT

With the emergence of edge computing paradigm, many applications such as image recognition and augmented reality require to perform machine learning (ML) and artificial intelligence (AI) tasks on edge devices. Most AI and ML models are large and computational heavy, whereas edge devices are usually equipped with limited computational and storage resources. Such models can be compressed and reduced in order to be placed on edge devices, but they may loose their capability and may not generalize and perform well compared to large models. Recent works used knowledge transfer techniques to transfer information from a large network (termed teacher) to a small one (termed student) in order to improve the performance of the latter. This approach seems to be promising for learning on edge devices, but a thorough investigation on its effectiveness is lacking.

The purpose of this work is to provide an extensive study on the performance (both in terms of accuracy and convergence speed) of knowledge transfer, considering different student-teacher architectures, datasets and different techniques for transferring knowledge from teacher to student.

A good performance improvement is obtained by transferring knowledge from both the intermediate layers and last layer of the teacher to a shallower student. But other architectures and transfer techniques do not fare so well and some of them even lead to negative performance impact. For example, a smaller and shorter network, trained with knowledge transfer on Caltech 101 achieved a significant improvement of 7.36% in the accuracy and converges 16 times faster compared to the same network trained without knowledge transfer. On the other hand, smaller network which is thinner than the teacher network performed worse with an accuracy drop of 9.48% on Caltech 101, even with utilization of knowledge transfer.

ACKNOWLEDGMENTS

I would first like to thank my thesis advisor Dr. Ming Zhao, Associate Professor of ASU School of Computing, Informatics, and Decision Systems Engineering. The door to Prof. Zhao office was always open whenever I ran into a trouble spot or had a question about my research. He consistently allowed this paper to be my own work, but steered me in the right direction whenever he thought I needed it.

I would also like to thank the experts who were involved in the validation of this research project: Saman Biookaghazadeh and Yitao Chen who are pursuing Ph.D. in Computer Science. Without their passionate participation and input, the validation of my research work could not have been successfully conducted.

I would also like to acknowledge Saman Biookaghazadeh, Ph.D. in Computer Science at Arizona State University as the second reader of this thesis, and I am gratefully indebted for his very valuable comments on this thesis.

TABLE OF CONTENTS

CHAPTER Page

LIST OF TABLES

LIST OF FIGURES

CHAPTER Page

CHAPTER Page

PREFACE

Recently one of my colleagues came up with an idea, which is retraining a new neural network from scratch while receiving mentorship from an already trained network. One can use this technique to retrain a neural network from scratch in a much easier manner, compared to non-mentored version. Now what if the new network could maintain a smaller size than the original network, but at the same time be able to represent the same knowledge? This may be a great idea, since it makes it possible to adapt the knowledge of a heavy neural network, while make training easier and faster. This is basically the idea I have extended in my thesis in order to bring training into mobile phones.

Chapter 1

INTRODUCTION

Enabling AI, such as deep neural networks on embedded and mobile devices is an on-going effort in the literature. Howard *et al.* (2017) and Chollet (2016) demonstrate the effectiveness of operation replacement with their faster counterparts. Based on their results, convolutions can be replaced with less computational intensive ones, without significant loss of accuracy. LeCun *et al.* (1990) show that certain number of parameters learn the same representations, and by eliminating such parameters, the size of the model can be reduced while preserving the accuracy. Sau and Balasubramanian (2016), Romero *et al.* (2014) and, Venkatesan and Li (2016) are utilizing knowledge distillation techniques to improve the accuracy of smaller models (termed students) by taking supervision from larger models (termed teachers). Han *et al.* (2015a) demonstrates pruning and compression techniques on trained model. Their results are showing feasibility of model size reduction, while preserving the similar accuracy. Kang *et al.* (2017) studies separation of a model into two pieces by placing one section on a server and another on an embedded device.

Summarizing all the above approaches, they are categorized as follows;***Pretrained on Cloud***, where the model is pre-trained on cloud with a large set of data, service providers will host the model needed for AI application on the cloud. As shown in Fig. 1.1, Mobile applications send a request along with the input data, to ask for models suggestion. This approach is widely being used by current applications such as Siri, Google Now, Cortana, and etc.

Pretrained on Cloud with User Dataset, where models are pre-trained specifically for the user using their customized data set. Mobile applications still

Figure 1.1: Model on Cloud with Service-Provider Dataset

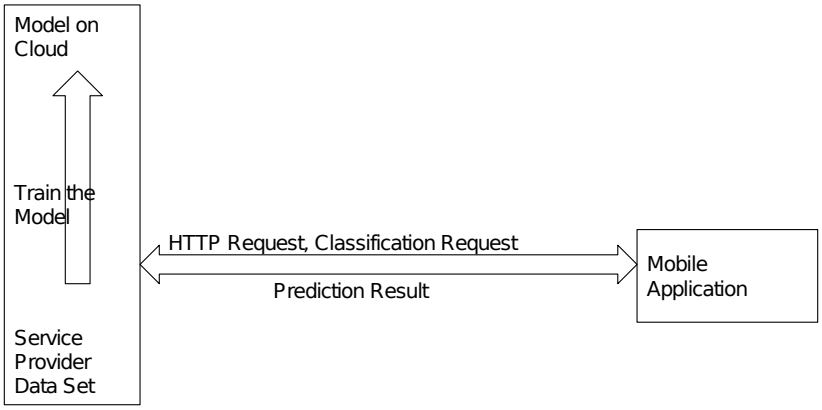
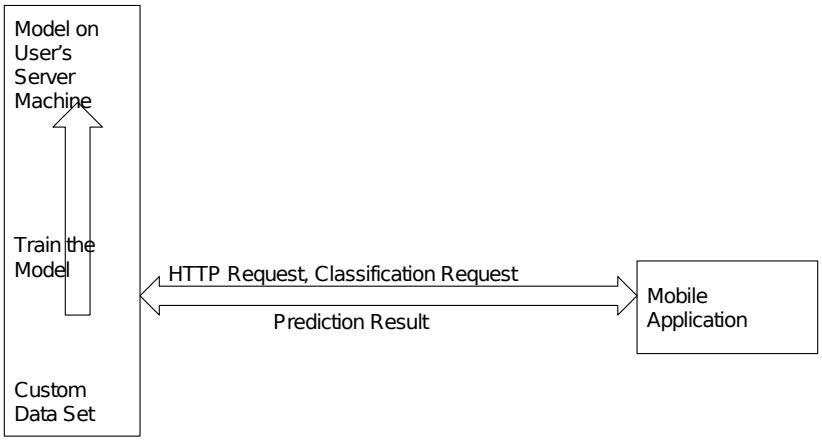


Figure 1.2: Model on User's Server Machine with Custom Dataset

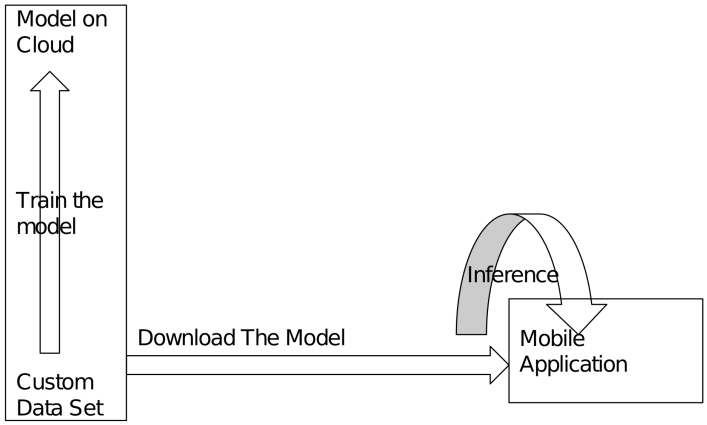


send a request along with the input data to ask for the predictions, as being demonstrated in Fig. 1.2. The second approach can be chosen when the data set on which the user wants to train the model is highly customized, as a result the generic pre-trained model cannot fulfill the users specific requirements.

In both the approaches user is bound to have a network connection. One drawback of this approach is significant space and computation usage by the service provider to store and process individual customized models.

Pretrained on Cloud & Inference on Mobile, as shown in Fig. 1.3, mod-

Figure 1.3: Download the Model from Cloud and Perform Inference on User's Mobile



els can be pre-trained by the service provider, specifically for each user input data. Furthermore, model alongside the application are moved into the user mobile device. In this approach, inference can be done on the mobile device which provides faster response time compared to previous approaches. It also eliminates the overhead of hosting the model on the cloud. Unfortunately, the model on the mobile phone cannot be continuously trained on the new input data, due to lack of power and storage on the device. This requires retraining of all the models on the cloud, which introduces same issues as mentioned in the second approach.

So far all the above related works can only support a single-shot prediction based on the input data. By growing popularity in AI on mobile applications, there would be a huge demand on customized evolution of models based on the individual user input data. For example, users may like their voice recognition application, such as Siri or Google now, to adaptively capture their unique accent continuously. Considering the great number of application users, it is almost impossible for application providers to host a single big and power consuming customized model per user, and train it over a long period of time. As a result, mobile applications can hardly provide customized AI models for each user.

I propose a solution to enable standalone AI applications on embedded devices. To overcome the challenges discussed above, below techniques are used:

- Pruning redundant parameters which learn similar representations to reduce model size and computation overhead.
- Replacing expensive operations, such as convolution, with their less expensive counterparts such as depthwise convolutions.
- Utilizing Dark Knowledge techniques to train significantly smaller networks by using supervision from original or larger networks.
- Cutting the model into two parts and placing the first part on the cloud and the later part on the embedded device. Using generalization techniques, later layers could be adaptively evolve based on the user input data.
- Utilizing available acceleration resources such as mobile GPU, in order to accelerate computations of certain operations such as Matrix Multiplication and Convolution.

So far, the feasibility of all the above techniques were proved to some extent. In current work, all the above methods are combined to enable continuous training of AI models, directly on the embedded device. All the above techniques are discussed in detail below:

1.1 Pruning Unnecessary Parameters

Increasing the size of neural networks, both in depth and width, can increase the capability of network to capture more complicated features, and as a consequence provide better accuracy. Unfortunately extending the size of network does not necessarily increase network capability for capturing new information. In another word,

redundancy is an outcome of greedy increase in network size. Pruning can help removing all redundant weights and keep the accuracy as the same level as original model. For example, after each training epoch, weights with small values can be eliminated from the model, since small weights have lowest contribution to the final accuracy. Several works have shown effectiveness of weight pruning in model size reduction. Unfortunately all reductions have happened after full training of a model. As a result, the same degree of weight elimination during training may cause severe damage accuracy. My solution is investigating the extent of parameter reduction in training phase. Preliminary results show effectiveness of pruning even during training phase.

1.2 Dark Knowledge Technique

Small neural networks (termed students) show a significant improvement in their performance provided they get sufficient knowledge from large neural networks (termed teachers). This transfer of knowledge from larger neural networks to smaller ones is termed as Dark Knowledge. The goal of the study is to train neural networks on mobile with customized data set and at the same time achieve good prediction results. Since its not feasible to train large neural networks on mobile due to its memory constraint, smaller networks are trained on mobile phones. But the drawback of training smaller networks is, it cannot learn and perform as good as larger networks. As the idea is to train the network on the mobile with customized user data the model should be able to learn all the features of the data efficiently and accurately. Since the network is small, it will not be able to extract all the features and will not be able to predict correctly. To overcome this, dark knowledge technique is adopted. Using this technique the knowledge that the network lacks due to its small size can be gained from the large and efficient network.

Till date, the dark knowledge technique was used to train the models faster as the transferred knowledge would help the smaller model to converge soon. It was acting more like a regularizer. The other usage was when the dataset to train the smaller model was not sufficient enough, it would get some knowledge from larger model and would predict the result. Though the smaller model was not trained on some of the specific images and if the larger model was trained on it, the smaller model would still be able to predict as the knowledge was transferred to it. But, in this approach, the focus is to make the smaller model behave like a larger model. Hence, various ways in which the knowledge can be transferred are discussed in the below lines.

In terms of mathematical functions, knowledge transfer techniques can be elaborated as follows. Achieving high accuracy in turn results in minimizing the cross entropy between empirical posterior distribution and predicted posterior distribution. This minimization can be done in several ways. First, by minimizing the combination of two convex functions: L_a: cross entropy between softmax output of deficient model and the softmax output of efficient model (teacher model). L_b: The cross entropy between the softmax output of deficient model (student model) and true labels of the training data. Hence,the total loss function would be a linear combination of L_a and L_b where $L=L_a+(1-\alpha)L_b$

Second, by minimizing the combination of multiple convex functions: L_a: cross entropy between the fourth layer of deficient model and the second layer of efficient model. L_b: The cross entropy between the sixth layer of deficient model and fourth layer of efficient model.L_c: The cross entropy between the softmax output of deficient model and true labels of the data. L_d: The cross entropy between the softmax output of deficient model and the softmax output of efficient model. Hence the total loss function would be a linear combination of L_a, L_b, L_c and L_d where $L= L_a+(L_b) + L_c + L_d$.

Third by minimizing the convex function, L =L_a + noise, where L_a would be the cross entropy between softmax layer of deficient model and softmax layer of efficient model with some noise added to it.

2^{nd} and 3^{rd} approaches can be combined and minimized together with the convex function L = L_a+ L_b + L_c + L_d + noise.

1.3 Replacing Operations and Thinning the Network

Using depthwise convolutions Howard *et al.* (2017) helps reducing storage consumption and computation intensity. This new way of convolution is breaking the whole process in two steps, where each step is much less intensive compared to the single conventional convolution. Combining both can lead into faster convolution with the same functionality. It also introduces less number of parameters to be held. In short, depthwise convolutions can capture almost the same amount of information as traditional convolutions, while not altering the final accuracy. In this work, depthwise convolutions are utilized for the small network on the mobile side.

As it is studied in Howard *et al.* (2017), width multiplier can further reduce network size and computational complexity to some extent. Width multiplier is basically altering the number of inputs and outputs channels, by a constant value between 0 and 1. Choosing higher width multiplier can reduce network width, in exchange with acceptable accuracy.

This work extends the power of Howard *et al.* (2017) by studying the feasibility of combination between Dark Knowledge and depthwise convolution. Using higher degree of width multiplier alone can compromise the final accuracy of the model, but it can be improved significantly using supervision from another big network. This approach will be further discussed.

1.4 Model Cutting

Another approach to enable stand-alone AI on mobile devices, is separating the model into two pieces. Separation could be after any number of layers. Each piece can be deployed on either host or the mobile phone. Since we need embedded inference capability, whole graph can be deployed on the phone. During training phase, only latest layers will be updated. Generalization approach has shown that for tuning a model for a new set of data of the same category, one only need to update the latest layers. For instance, layers at the beginning of the model only capture fine grained features, which works perfectly for any dataset of the same category. As a result, mobile device is only responsible for tuning few final layers, which are small in size and number of parameters. Han *et al.* (2015a) studied the feasibility performance and energy consumption in this approach. This work demonstrates the feasibility of training in the mentioned method. In summary, the whole model can completely be deployed on both cloud and mobile device, while mobile only updates final section of the model for customization purpose.

1.5 Computation Acceleration on Mobile Devices

Convolutional Neural Networks are memory and computation hungry by nature. As a result, even using high-end systems without accelerators are not sufficient for training these networks in a reasonable amount of time. This enforces CNN users to utilize GPUs to accelerate training process. This issue can become more significant in embedded and mobile devices, due to their lack of enough memory and sophisticated processor. Therefore, training models on mobile devices is not considered as an option.

In order to overcome this problem, approach of using available accelerators, such as GPU, in the mobile phone is proposed. Both popular mobile platforms, IOS and

Android, support utilization of the embedded GPU through their toolchains, BNNS and RenderScript. Both libraries are using GPU API at the bottom to parallelize a specific operation. In addition, GPUs on mobile devices are power efficient, which makes utilizing them more reasonable. Extension of well-known deep learning frameworks, such as TensorFlow, to support operation execution over RenderScript or BNNS is important.

1.6 Proposed System Architecture

Combining all the above techniques, the description about the proposed systems architecture is mentioned here. There are two architectures which are being investigated in this study: First, ***Mentee-Mentor network***, where the two models are designed to represent the same model with different sizes, as it is demonstrated in Fig. 1.4. In this approach the smaller network will be deployed in the embedded devices and the bigger network will reside in the cloud. The model on the embedded devices is designed to capture and reflect users unique requirements, while the bigger network on the cloud intends to be a much more sophisticated and generic version of the model.

Utilizing Dark Knowledge technique, the smaller model can continuously be trained, while receiving supervision from the bigger network. The supervision helps smaller network not to deviate from the accurate representation. In detail, supervision comes as a loss value from the mentor network into the mentee network. The knowledge can also be transferred from any intermediate layer of the mentor network along with the last layer, which helps mentee to maintain a mapping into the mentor network.

The smaller model can further be optimized, by using lightweight operations and utilizing the available accelerators. For instance, all the conventional convolutions of the smaller model will be replaced with depthwise convolutions thereby reducing the

width of the network. In addition, all the expensive operations can be deployed on available accelerators, such as GPUs on the mobile phones.

Second, ***Model Cutting and Placement***, in this approach, the model is cut into two parts from one of the intermediate layers. The model will be deployed completely on both cloud and mobile device. The mobile phone is only updating the second part of the model, since they have the major contribution for new dataset categorization. On the other hand, cloud can update the first section of the model repetitively, after receiving large batch of new data from many users. Since second part of the model is both small and low in number of parameters, training on the mobile phone is practical. Similar to previous architecture, the complete model on the graph can replace the operations with their lightweight counterparts, and also all operations can utilize accelerators.

1.7 AI on Edge Devices

Deep neural networks (DNN) have achieved tremendous accuracy improvements compared to conventional machine learning techniques for many important tasks, such as image classification and speech recognition mentioned in Simonyan and Zisserman (2014), He *et al.* (2016) and Chan *et al.* (2016). Edge applications are adopting AI, specifically deep learning, to assist users in a better and more intelligent way. For example, augmented reality, face recognition, and intelligent personal assistants require deep networks for complex classification and decision making. However, state-of-the-art deep learning models typically require storing millions of parameters and need to perform large amounts of operations, which requires hours or even days of training using many CPUs and GPUs on large-scale systems such as the cloud.

Such a cloud-only deep learning approach does not work well when the network is not reliable or when the cloud is not responsive enough to handle sudden load surge.

Also, they rely on pre-trained neural networks, which stops the networks to adapt to user local inputs. At the same time, it can also affect user experience due to network latency. Further, the good computing and storage resources that modern edge devices typically possess are not utilized to help the learning. Combining previous arguments, cloud-based deep learning is not sufficient for edge applications.

Moreover, there are also important benefits from performing learning on edge devices: 1) *Personalization* of the models based on user-specific behaviors and requirements can be more effective and scalable, by learning on the devices that the users directly interact with; 2) *Responsiveness* to user behaviors and environments can be better achieved by adapting the models quickly and dynamically, using the on-device resources; and 3) *Privacy* of user-specific information learned by the models can be better protected on a device owned by the user compared to public resources shared by many.

Deploying these computational and memory intensive models on edge devices, such as mobile phones and smart cameras is challenging, since such devices are based on Silicon on chip (SoC) architecture with limited resources designed to fulfill requirements of embedded applications. To enable learning on such resource-constrained devices, several model compression techniques are proposed: 1) *Parameter sharing* where a single parameter represents multiple parameters with slight differences mentioned in Chandakkar *et al.* (2017); Han *et al.* (2015a); Chen *et al.* (2015) ; 2) *Quantization* which reduces the number of bits required by the weights in the network mentioned in Han *et al.* (2015a); Kadetotad *et al.* (2016); 3) *Pruning* which removes negligible weights from the network referred in Han *et al.* (2015a); Le Cun *et al.* (1990); Srinivas and Babu (2015), and 4) *Knowledge transfer* (***KT***) which trains smaller networks (also termed ***students***) under the supervision of the larger networks (also termed ***teachers***) to improve accuracy and speed.

Among all the aforementioned approaches, the KT approach is particularly interesting, because it allows the student network to receive mentoring from the teacher network while still learning independently. There are potentially several advantages of this approach compared to the others. First, it can help the student network to converge sooner by utilizing the information coming from the teacher in the form of a single or multiple values. This information from the teacher model can help the optimization phase of the student network by directing students' parameters into the same representations as teacher model parameters thereby allowing the student network to approach local minima faster. Second, it can improve the accuracy of the student network. KT allows the student models to arrive at better parameter values based on the teacher model's parameter values, which can deliver almost the most optimized version of the classification function. Third, it helps the student network to become more general by preventing from getting biased toward a certain set of data.

Prior works Hinton *et al.* (2015); Ba and Caruana (2014); Romero *et al.* (2014); Venkatesan and Li (2016) focused only on performance comparison of student and teacher models, but they ignored the performance comparison between the student models under supervision with the student models trained without supervision. This can be of interest when the student with no KT utilization can perform the same as student with KT utilization. The lack of comparison can invalidate the feasibility of KT technique. Further, in terms of performance, the related works focused only on the accuracy of the student models, but they ignored the convergence time. Faster convergence time can bring several benefits, such as less power consumption and shorter training time. In addition, each of the works have limited coverage on architecture types and make the assumption that a certain type of KT technique is applicable on all types of architectures, which is not necessarily true.

The goal of this study is to provide a thorough analysis of the effectiveness of the existing KT techniques. The existing KT techniques are broadly classified into three categories based on the type of knowledge transferred from teacher networks to student networks: First, the *hard logits* of a network mentioned by Ba and Caruana (2014). Hard logits is defined as the output of the last layer of the teacher network before passing to the softmax activation function; Second, the soft logits of a network mentioned in Hinton *et al.* (2015). Soft logits are obtained by softening hard logits with the help of temperature softmax variable and then passing softened logits to the softmax activation function; Third, the *intermediate layer representations* of a network mentioned in both Romero *et al.* (2014); Venkatesan and Li (2016). Intermediate layer representations are the outputs of the middle layers of the teacher network.

The above KT techniques are studied on three different types of student-teacher architectures: 1) *Type I* is based on the networks used in Hinton *et al.* (2015); Venkatesan and Li (2016); Ba and Caruana (2014), where the student network is shorter than teacher network. For example, the student consists of six layers whereas the teacher consists of 16 layers. 2) *Type II* utilizes a student network thinner than the teacher network. For example, the student is MobileNet used in Howard *et al.* (2017) with the width multiplier set to 0.05 whereas the teacher is the baseline MobileNet model with the width multiplier of 1.0. 3) *Type III* student network mentioned by Romero *et al.* (2014) is deeper and thinner than teacher network where the student consists of 19 layers and the teacher consists of 5 layers.

The above models are built using TensorFlow introduced by Abadi *et al.* (2015), and evaluated using CIFAR-10 and Caltech 101 datasets on a Tesla K40 testbed. Following are the observations based on the experimental results: (1) Existing KT techniques do not behave the same on all architectures ; Surprisingly, hard logits and

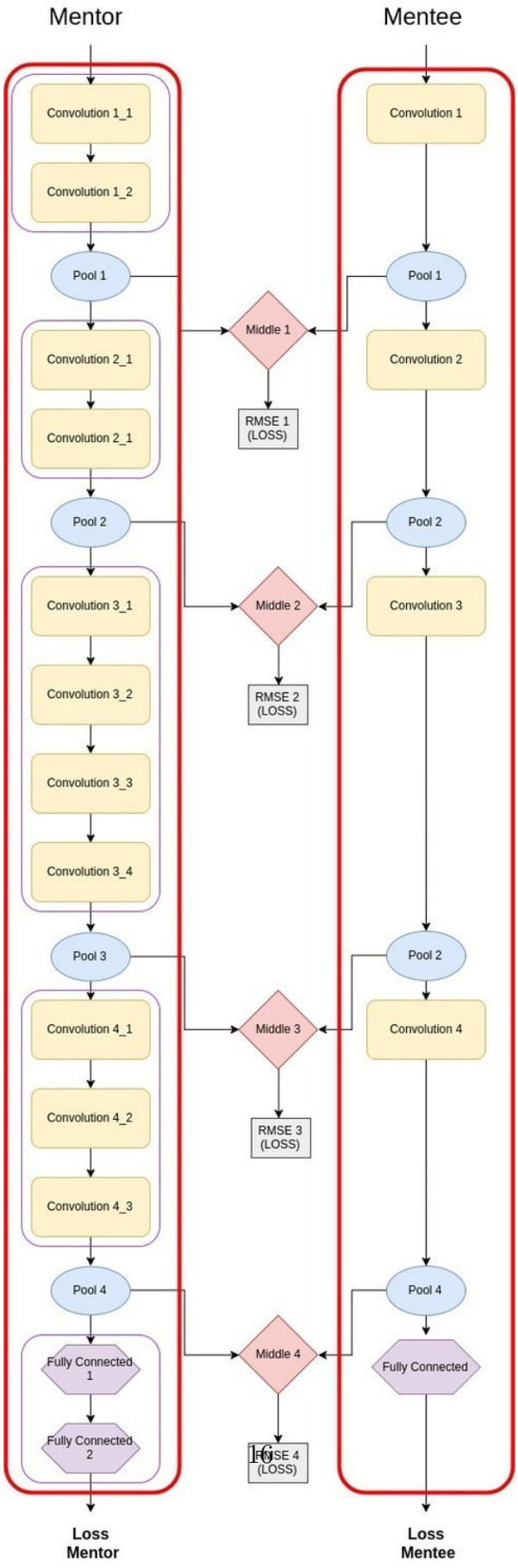
soft logits techniques Hinton *et al.* (2015); Ba and Caruana (2014) perform negatively when applied to student networks of the Type II architecture; accuracy drops by 9.48% on Caltech 101 and by 17% on CIFAR-10; (2) No significant improvement is achieved in the student networks' accuracy trained with soft logits and hard logits techniques over the ones trained without applying KT techniques. (3) Intermediate representations (single layer) technique improves the accuracy of Type I student network by 7.36% while trained on Caltech 101. However, the same KT technique when applied to Type III student network drops the accuracy by 5.23% on Caltech 101. (4) With respect to convergence time, student network of Type I when trained with intermediate representations (single layer) based KT on Caltech 101 converges 16 times faster than the student trained without any supervision. (5) At the same time, student model of Type III converges as twice as independent student when trained with soft logits, although improvement in the accuracy is only 1.85%. (6) Finally, among all the KT techniques, intermediate representations KT technique seems to be the best. Also, all the KT techniques trained the student models at a faster rate ranging from 9% to 94% compared to student model trained without any KT techniques.

In short, in terms of accuracy, only the intermediate-representation KT technique and Type I student architecture achieves significant improvement (7.36%) for the dependent student (trained with KT) over the independent student (trained without KT). The other KT techniques do not perform well on this architecture, and this particular technique also does not perform well on the other architectures. In many cases, the use of KT in fact reduces the accuracy of the dependent student compared to an independent student and the drop can also be significant (up to 60.43%). In terms of convergence time, KT can achieve some level of speedup on all the architectures, where the best result (16X) is still from using the intermediate-representation KT technique on Type I architecture.

There is lack of investigation in the granular level of KT. KT does have potential to improve both accuracy and speed of a small network, but it is sensitive to how the knowledge is transfered from the teacher and the architecture of the student. In particular, transferring knowledge through the intermediate layers (in addition to the last layer) is the most promising KT technique. Therefore, a more focused study on the intermediate-representation KT technique should be performed in order to understand its full potential. Thus, extensive study in how the knowledge is transferred from the larger model to the smaller model is needed, to gain significant improvement in the accuracy of the smaller model.

The rest of my work is organized as follows: Section 2 introduces the background; Section 2.3 examines the related works; Sections 3 and 3.2 details the existing KT techniques and their advantages; Sections 5 and 6 describe the evaluation methodology and results; finally, Section 7 concludes my work.

Figure 1.4: Mentor-Mentee



Chapter 2

BACKGROUND LITERATURE

2.1 Neural Networks

Neural networks typically consist of convolution, pooling, activation and fully connected layers. Each layer is made up of neurons with learnable weights and biases. These neurons receive inputs, take weigthed sum over them, pass it through an activation function and respond with an output. Main purpose of the convolution layer is to extract features from the input image. This is achieved through a filter which slides over the input image by sharing the weights leading to lesser number of parameters. The output of the convolution layer is the dot product between the filter and chunks of the input image.

Activation layer, applied after every convolution layer, consists of an activation function which decides whether a particular neuron should be considered or discarded by the other connections in the network. Pooling layer, applied after few convolution layers, reduces the spatial dimensions of the input resulting in reduction of number of parameters and computations of the network. Fully connected layer, connects every neuron with every other neuron in the network, without sharing the weights, leading to larger number of parameters.

Weights and biases of the neural network are termed as parameters. In other words, every connection in the neural network is treated as a parameter. Teacher (mentor) is a sophisticated neural network with larger number of parameters compared to the student (mentee) network.

Traditional knowledge transfer technique mentioned in Hinton *et al.* (2015) com-

bines two loss functions into one by adding them up. First, Root Mean Squared Error Loss Function defined as the loss between the outputs of the last layers of the mentor and mentee network as mentioned in Eq 2.1.

$$RMSE = \sqrt{\sum_{i=1}^{n}(y_{Lm} - y_{LM})^2} \tag{2.1}$$

L_m is the output of mentee's last layer, L_M is the output of mentor's last layer.

Second, Cross Entropy Loss Function is defined as the loss between the true labels of the dataset and the output of the last layer of the mentee network expressed as follows.

$$CE = \sum_{i=1}^{n}[y_{Lm} log \hat{y_{Lm}} + (1 - y_{Lm}) log(1 - \hat{y}_{Lm})] \tag{2.2}$$

The summation of the above two equations is the overall loss function as mentioned in Eq 2.3.

$$\begin{aligned} J = &\sum_{i=1}^{n}[y_{Lm} log \hat{y_{Lm}} + (1 - y_{Lm}) log(1 - \hat{y}_{Lm})] \\ &+ \sqrt{\sum_{i=1}^{n}(y_{Lm} - y_{LM})^2} \end{aligned} \tag{2.3}$$

J is minimized by a single optimizer during the training of the mentee network Hinton *et al.* (2015). Two major algorithms, forward-propagation and backward-propagation, are used during the training.

In Forward-propagation, the output of the last layers of the mentor and the mentee is evaluated by taking the weighted sum of the input dataset.

During Backward-propagation, J is minimized by updating the weights of each layer of the mentee network with the gradients. The gradient is the partial derivatives of J w.r.t weights.

Other publications Venkatesan and Li (2016), Romero *et al.* (2014) discuss a different type of knowledge transfer, which includes an extra loss from the middle

layers of mentee and mentor. In this study, for better understanding, only 2nd and 3rd layer losses are shown and the overall loss function is mentioned below,

$$
\begin{aligned}
J = & \sqrt{\sum_{i=1}^{n}(y_{Lm0} - y_{LM0})^2} \\
& + \sqrt{\sum_{i=1}^{n}(y_{Lm1} - y_{LM1})^2} \\
& + \sqrt{\sum_{i=1}^{n}(y_{Lm2} - y_{LM2})^2} \\
& + \sum_{i=1}^{n}[y_m log \hat{y}_{Lm2} + log(1 - \hat{y}_{Lm2})(1 - y_m)]
\end{aligned}
\tag{2.4}
$$

L_{m0}, L_{M0} are the outputs of 2nd layer of mentee and mentor. L_{m1}, L_{M1} are the outputs of 3rd layer of mentee and mentor. L_{m2}, L_{M2} are the outputs of last layer of mentee and mentor as mentioned in Venkatesan and Li (2016), Romero *et al.* (2014).

2.2 Deployment of DNN's on Edge Devices

DNNs require large volumes of input data to train the models, while the training also requires large amounts of computational resources in order to reach a good accuracy within a reasonable time. Therefore, DNNs have been traditionally hosted on large-scale systems such as cloud datacenters. The learning-based applications (e.g., Siri, Google Now, Cortana, Alexa) running on the edge devices have to send their requests (e.g., image classification, voice recognition) to the cloud where DNNs are used to perform inference and return the results to the applications across the network. A significant drawback of this cloud-based learning approach is that it relies solely on the cloud resources for the learning and cannot perform well when the cloud is overloaded or the network is unreliable.

Although in some cases, it is possible to download a trained DNN from the cloud

to an edge device and use the model to perform inference for prediction, classification, etc. on the device, it still requires substantial space (to store the model) and time to perform the inference.

Deployment of neural networks on edge devices has attractive prospects as studied in Han *et al.* (2016, 2017), and related works have studied how to reduce the computational complexity and storage requirement of DNNs in order to fit the networks to the edge devices. One of the early works applies singular value decomposition (SVD) to a pre-trained model to achieve weight compression as mentioned in Denton *et al.* (2014). Magnitude-based weight pruning was introduced in Han *et al.* (2015b,a) based on the observation that many weights have small values that produce negligible output response. Making these weights zero could remove connections between neurons which saves memory. Adaptive quantization and weight sharing can also be applied to reduce the number of bits needed per weight as mentioned in Han *et al.* (2015a). Huffman coding has also been explored to quantize the weights in Han *et al.* (2015a).

Modifying the original architecture of large DNNs was explored in Iandola *et al.* (2016), based on certain guidelines, such as replacing most 3×3 filters with those of 1×1, thereby saving $8\times$ parameters. Delayed downsampling was employed to produce large activation maps early on the network that helps maximize accuracy with a given number of parameters. Other line of research includes developing specialized hardware accelerators mentioned in Han *et al.* (2016, 2017) and guiding shallow CNNs using the activations of deeper ones as mentioned in Venkatesan and Li (2016).

Binarized neural networks (BNN) that have binary weights and activations (1 or -1) are proposed in Courbariaux *et al.* (2016). Only real-valued quantities in these networks are the gradients that are obtained through standard DNN optimization algorithms such as stochastic gradient descent or Adam. BNN reduces time complexity

by almost 60%.

Operation replacement was considered to replace convolutions with less computational intensive ones, without much loss of accuracy mentioned in Howard *et al.* (2017); Chollet (2016). Le Cun *et al.* (1990) is showing that certain number of parameters are duplicating the knowledge representation, and by eliminating we can reduce the size of model while preserving the accuracy. This is the only work, which reduces number of parameters during the training phase. Knowledge distillation was explored to transfer the knowledge from a large model to a small model, that is more suitable for deployment as described in Romero *et al.* (2014); Hinton *et al.* (2015); Sau and Balasubramanian (2016). Han *et al.* (2015a) is demonstrating pruning and compression techniques on trained model. Their results are showing feasibility of model size reduction, while preserving the same accuracy. Finally, related work Kang *et al.* (2017) also studied the approach of partitioning a large model between the cloud and edge device.

While these existing solutions are able to reduce a trained DNN and deploy it on a device for inference, none of them can support the training (and re-training) of a network on the device with the same level of accuracy as a DNN trained on the cloud. However, there are important reasons on why the capabilities of edge devices to support deep learning should be exploited:

- **Personalization**: For many applications, custom DNNs tailored to individual users' behaviors and/or requirements are important to deliver accurate results to the users. While it is possible to train and run all the personalized models on the cloud, it can be slow, costly, and difficult to scale. Although existing works allow a generic model to be downloaded to edge devices and use the local resources to perform inference, they do not allow such a model to be personalized on the devices to meet the user's specific needs. In comparison, it

is advantageous if the personalized models can be trained *in situ* on the devices, while the user-specific training data is collected by the local sensors and user interfaces.

- **Responsiveness**: Using edge devices to support deep learning can provide better responsiveness than relying only on the cloud resources. On one hand, a locally stored model on an edge device can be readily used to perform training and inference and respond to user requests using local resources, regardless of the network connectivity and the load on the cloud system. On the other hand, by using the local data and resources to continuously train the model on the device, it can also quickly respond to the dynamic changes in the user's behaviors, situations, and requirements.

- **Privacy**: For certain DNN applications (e.g., biometric authentication), the privacy of the data and/or model needs to be protected. Such privacy concerns can be more effectively addressed if a user's personal data and model are stored and used only on the user's own device, while cloud resources can still be involved to train a non-private, generic model to assist the learning on the devices.

Nonetheless, it is also important to recognize the limitations of edge devices (limited computing and storage capacity, limited battery life, limited access to data, etc.), and it is inappropriate and infeasible to train and run large models entirely on the device. Therefore, in this study, distributed, collaborative deep learning is advocated, a new paradigm that utilizes the complementary strengths of edge and cloud resources to substantially improve the speed and accuracy of learning for diverse applications.

2.3 Related Work

As DNNs require large memory and huge computational power, different techniques to reduce their size without affecting the performance of the model have gained a lot of attention. Related works proposed several model compression techniques, which can be broadly classifed into three categories as mentioned below.

- **Weight Sharing**: This technique reduces the memory occupied by the model by sharing weights. K-means clustering technique was used to find out weights that can be shared by the network mentioned in Han *et al.* (2015a). HashedNets model was proposed in Chen *et al.* (2015). They shared weights by using a hash function which groups weights into hash buckets mentioned in Chen *et al.* (2015).

- **Quantization**: Memory occupied by the model is reduced by shrinking the number of bits needed by the weights. This technique was adopted in Han *et al.* (2015a) and showed that the number of bits reduced from 35 to 5 for every connection in the network. Kadetotad *et al.* (2016) applied blockwise structured sparsity and quantized the weights and activations after training was complete resulting in reduction of 5-6 bit. Number of effective weights to be stored can be limited by having multiple connections sharing same weights.

- **Pruning Techniques**: Memory occupied by the model can be reduced extensively using pruning techniques. Magnitude based pruning removes weights or connections which produce negligible response. This technique is adpoted in Han *et al.* (2015a). They remove all the weights which are below a particular threshold value, resulting in reduction of number of parameters by 9x and 13x for AlexNet and VGG-16 model. The Optimal brain damage method reduces

the number of weights based on the Hessian loss function mentioned in LeCun *et al.* (1990). Identifying redundant neurons and removal of such neurons is explored in Srinivas and Babu (2015). They used data-driven pruning technique to remove redundant neurons.

Above mentioned categories focused on reducing memory, thereby the computational cost of the student model while keeping the accuracy same. However, none of the categories focused on improving the accuracy of the student model. In other words, the above mentioned techniques all focus on reducing the size of a model so that it can be deployed in a resource-constrained environment.

KT is one of the model compression techniques, which aims for improving the accuracy of the student models. It achieves this with the help of a teacher model which provides hints (knowledge) in order to perform and generalize better. None of the related works have performed a thorough investigation on the effectiveness of these existing knowledge transfer techniques.

Although there are several related works on KT, none of them provides a thorough study on the effectiveness of such techniques, and still several key questions do not have good answers: 1) Do all the KT techniques bring significant improvement to the accuracy of the student network? 2) Can we apply a single KT technique on any student architecture with any training dataset and yet see consistent results? 3) Do all the KT techniques improve the convergence speed of the student model? Therefore, the goal of this study is to provide a good understanding of KT by finding answers to these questions through a comprehensive analysis of accuracy and speed of different KT techniques on different model architectures.

In this study, following details are discussed: (1) Various forms of KT techniques, (2) Effectiveness, (3) Limitations, and (4) Feasibility of applying them to train student models.

Chapter 3

KNOWLEDGE TRANSFER

Transfer of knowledge from a larger model to a smaller model is termed as knowledge transfer.

3.1 Different Techniques of Knowledge Transfer

As mentioned in Section 2, large DNNs require substantial space and time, while they perform better and are more accurate compared to compact models. Prior works adopted KT techniques for training a fast and compact model as it approximates functions learned by a larger and more accurate model (DNN). Difference between KT and other model compression techniques is the existence of a teacher model which provides supervision. In this method, large amounts of training data are passed through the teacher model to collect the output, to be used as target labels for training student models. In this technique, unlabeled data is passed through the large and accurate model to collect the output (logits). These logits are used as target labels for training compact models. The compact models are not trained on original labels. By doing so, student models have the potential to learn the representation that is being learned by the teacher model and perform close to the teacher despite being small. In this section, the existing KT techniques are broadly classified into three categories, and the basic approach is explained. Further, the potential strengths of each of the techniques is discussed.

3.1.1 Transfering Hard Logits

Hard logits KT technique was introduced by Ba *et al.* Ba and Caruana (2014). They first trained a deep teacher model to achieve a good accuracy. Then they trained a shallow student model on TIMIT and CIFAR-10 datasets to mimic the behavior of the deep teacher model, by formulating a regression problem which minimizes squared difference (RMSE) between logits (output of the last layer) of the deep teacher model and softmax output of shallow student model, as shown in Fig. 3.1. They used the logits of the teacher model directly as opposed to probabilities produced by passing logits to softmax activation function, in order to learn the valuable similarity structure over the data. Since these logits are not softened and used directly to train student network, this KT approach is named transferring the hard logits.

In this approach, the student network is trained only on the teacher network's logits, unlike the other approaches mentioned in this section where original labels of the dataset are also used. Loss function of this approach is formulated in Eq. 3.1.

$$\sqrt{\sum_{i=1}^{n}(\hat{y}_s - z_t)^2} \tag{3.1}$$

Where:

- $\hat{y}_s$: predicted softmax output of student
- z_t: predicted hard logits of teacher

This technique can be helpful for two major reasons. First, if true labels have errors, the teacher model may eliminate few of them making the student model learn easily. Second, the original labels may depend on the features that are not available as inputs to the student network. Thus, teacher model eliminates those that are

dependent on unavailable data and gives the labels which are dependent only on the input features.

This was the first model compression technique which trained a shallow network using the outputs of a deeper network and could learn complex functions previously learnt by the deeper model and empirically showed that shallow neural networks can be trained to achieve performances comparable to that of deep neural networks.

3.1.2 Transferring Soft Logits

Hinton et al. proposed a knowledge distillation approach Hinton *et al.* (2015) which helps compress the knowledge of ensemble models into a student model. They achieved this by introducing a temperature softmax variable (T) as mentioned in Eq. 3.2.

$$q_i = exp(z_i/T)/\sum_j exp(z_j/T) \tag{3.2}$$

where,

- z_i: is the output of i^{th} neuron of teacher's layer termed as hard logits;
- T: is the temperature softmax variable - parameter to control the relative importance of the soft targets provided by the larger model. Higher the value of T softer are the targets;
- q_i: is the output of i^{th} neuron of teacher's softmax layer termed as soft logit; and
- j: is the number of neurons at the teacher's softmax layer.

Hard logits of teacher and student models are divided by temperature softmax variable (T) and passed through softmax activation function to obtain softer probabilities (*soft logits*). The student model minimizes the sum of two objective functions: (1) mean squared difference (RMSE) between the soft logits, and (2) cross entropy loss between the softmax output and correct labels of the dataset as shown in Fig. 3.2.

I name this KT approach as transferring soft logits because soft logits of the teacher network are used to train the student network. They evaluated their approach on MNIST dataset and showed that soft targets can effectively transfer information including the knowledge (about the data which smaller model has never seen before) from complex model to distilled smaller model.

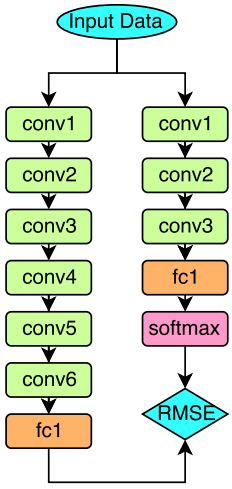
The loss function of this technique is mentioned in Eq. 3.3. Interpretation of the loss function is as follows: the first term indicates the student's cross entropy loss; the second term indicates the MSE loss between the soft logits of teacher and student. The student model minimzes the MSE loss while minimizing the overall loss. By doing so, parameters of the student's model (weights and biases) tend to move towards parameters of the teacher's model, which results in learning same representations as that of teacher's. The soft logits approach is similar to the hard logits approach but with a slight difference. Soft logits approach uses the loss function mentioned in Eq. 3.3.

$$\sum_{i=1}^{n}[y_s log\hat{y}_s + (1-\hat{y}_s)log(1-y_s)] + \sum_{i=1}^{n}(q_s - q_t)^2 \tag{3.3}$$

Where:

- y_s: true labels of the datasets
- $\hat{y_s}$: same as q_s but at a temperature softmax variable (T) of 1

Figure 3.1: Hard-Logits



- q_s: predicted softened softmax output of student (soft logits)
- q_t: predicted softened softmax output of teacher (soft logits)

It is claimed that by using soft targets instead of hard targets (logits), more useful information can be carried which is not possibly be encoded with hard targets Hinton *et al.* (2015). The other advantage of this approach is that the student network can be trained with much less data than before, since soft targets with high entropy provide more information compared to hard targets and much less variance in the gradient between training cases.

3.1.3 Transferring Intermediate Layer Representations

Both above approaches use only the hard or soft logits of the teacher model as knowledge to the student model. In addition to these logits, knowledge from the teacher's intermediate layers is also used to train the student model. Therefore this type of approach is named as transferring intermediate layer representations.

Romero et al. proposed to use the output of the middle layer of the teacher model as hint to improve the performance of the deep and thin student model Romero *et al.*

Figure 3.2: Soft-Logits

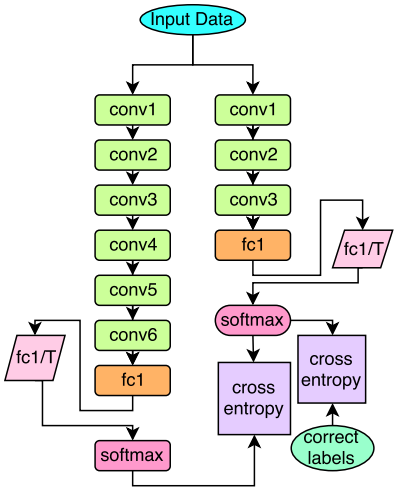
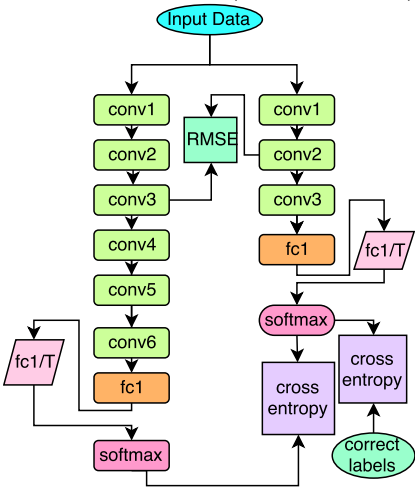


Figure 3.3: Intermediate-Representations (Single Layer)

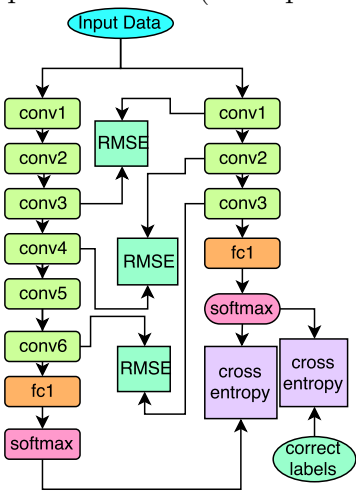


(2014).

Unlike shallow models used in the previous two approaches, this approach assumes that the compact model is thin but deep. Thinner model reduces the computational burden whereas deeper model takes advantage of depth to reuse the features and are exponentially more expressive than the shallow ones. They termed these intermediate layers representations (knowledge) as hints.

By using the hints derived from the intermediate layer representations, on the CIFAR-10 dataset, they claim that a student network, which is thin and deep and

Figure 3.4: Intermediate-Representations (Multiple Layer)



contains ten times fewer parameters, outperforms a larger teacher network.

Knowledge transfer is achieved by training the student model in two stages. In the first stage, the student model is trained up to the guided layer (the 11^{th} layer of the student network) with the output of teacher's hint layer (the 2^{nd} layer of the teacher model) as target labels as shown in Fig. 3.3. During the training, student network updates the weights of all the layers up to the guided layer by minimizing the loss between the teacher's hint layer and the student's guided layer. The reason for training up to the guided layer is to obtain a good starting point in the parameter space, which helps training teacher model easily. Reason behind carrying out the first stage is this kind of curriculum learning would initialize the student network in a way that it would receive exampls of increasing and appropriate difficulty w.r.t already learned concepts.

In the second stage, the student model continues training on the pre-trained parameters obtained in the first stage. Further, it updates the weights of entire network by minimizing the Knowledge distillation (KD) loss. KD loss is defined as the sum of two cross entropy loss functions: (1) cross entropy loss between the outputs of student network and true labels of the dataset and (2) cross entropy loss between the

softened outputs of student and teacher network. The overall loss function of this approach is formulated in Eq. 3.4.

The loss function is interpreted as follows: the First term is the MSE loss between the outputs of teacher-student intermediate layer pairs; the Second term is the cross-entropy loss of the student model; and the third term is the cross entropy loss between the soft logits of the teacher and student models.

Student model minimizes the MSE loss while minimizing the overall loss function. By doing so, parameters of student's model (weights and biases) tend to move towards parameters of teacher's model. Thus, student's intermediate layers outputs are approximated to teacher's intermediate layers outputs. As a result, student network generalizes well and perform in comparison to teacher network. They term this approach as hint based training.

$$\begin{aligned} J = & \sqrt{\sum_{i=1}^{n}(y_{sji} - y_{tji})^2} \\ & + \sum_{i=1}^{n}[y_s log \hat{y}_s + (1 - \hat{y}_s) log(1 - y_s)] + \sum_{i=1}^{n}(q_s - q_t)^2 \end{aligned} \tag{3.4}$$

Where:

- y_{sji}: output of student's j^{th} layer
- y_{tji}: output of teacher's j^{th} layer
- $\hat{y}_s$: predicted softmax output of student
- q_s: predicted softened softmax output of student (soft logits)
- q_t: predicted softened softmax output of teacher (soft logits)

The technique of transferring of intermediate representations is extended further by Venkatesan *et al.* Venkatesan and Li (2016). Here, instead of considering only the

middle layer pairs of student-teacher they experimented with multiple intermediate layer pairs, including softmax layers, as shown in Fig. 3.4. First, they trained a teacher model on Dataset D1. Then they used the representations from the middle layers including last layers of the pre-trained teacher model to train a student model on Dataset D2. D2 is much smaller and less general compared to D1. Overall loss function of this approach is formulated in Eq. 3.5

$$\begin{aligned} J = & \sum_{j=1}^{m} \sqrt{\sum_{i=1}^{n} (y_{sji} - y_{tji})^2} \\ & + \sum_{i=1}^{n} [y_s log \hat{y}_s + (1 - \hat{y}_s) log(1 - y_s)] + \sum_{i=1}^{n} (\hat{y}_s - \hat{y}_t)^2 \end{aligned} \tag{3.5}$$

Where:

- m: indicates number of teacher-student layer pairs
- y_{sji}: output of student's j^{th} layer
- y_{tji}: output of teacher's j^{th} layer
- $\hat{y}_s$: predicted softmax output of student
- $\hat{y}_t$: predicted softmax output of teacher

Romoreo et al. claims that a student network with 10 times fewer parameters than a teacher network could outperform the teacher network with the help of knowledge transferred through intermediate layer representations. Venkatesan et al. claimed that multi-layer KT provides better generalization accuracy compared to the popular regularization techniques, such as L2 , L1 and dropout.

3.2 Architectures

In this section, three different types of student-teacher architectures are discussed. In each of the architectures, student model is constructed in a way that it has fewer parameters compared to teacher model. These architectures are used to evaluate the effectiveness of KT techniques. Each KT technique in Section 3 is applied on all three architectures to examine whether a particular KT technique is equally effective on three architectures. Further, to drive general conclusions on the effectiveness of KT techniques, we analyze the behavior of KT techniques on different architectures. Details of the architectures are mentioned below.

- Type I: Teacher model is VGG16 described in Simonyan and Zisserman (2014). Student model is mentee network in Venkatesan and Li (2016). Here, student network is shorter than the teacher network and consists of 3.2M parameters. On the other hand, number of parameters in teacher model are 8.5M (2.5 times more than the student model). Fig. 3.2 demonstrates Type I architecture.

- Type II: Teacher and student models use the MobileNet architecture of Howard *et al.* (2017) with different widths. The width is changed by tuning the width multiplier parameter present in mobilenet architecture. Width multiplier of baseline model is 1.0. In all the experiments the width multiplier of teacher is set to 1.0 and that of student to 0.1 as mentioned in Howard *et al.* (2017). Unlike Type I, here, student network is thinner and of same depth as that of teacher. In terms of number of parameters, student model consists of 1.3M whereas teacher model consists of 4.2M. This architecture is shown in Fig. 3.2.

- Type III: Teacher model is Maxout model of Goodfellow *et al.* (2014). Student model is FitNet4 in Romero *et al.* (2014). In this case, student network is thin-

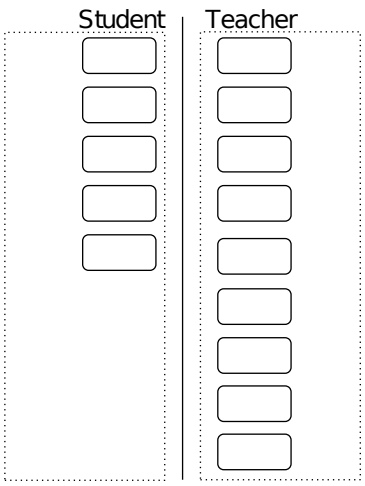


Figure 3.5: Type I Architecture:: Student Model is Shorter than the Teacher Model

ner and deeper compared to teacher network.. Number of parameters in student model equals 2.5M which is approximately 4 times lesser than the number of parameters in teacher model (9M). Fig. 3.2 shows architecture of Type III.

All the related works have a limited coverage on the architecture types. Hard-logits, soft-logits, and intermediate-representations (multiple layer) based KT techniques were evaluated only on shallow models similar to Type I used by Hinton *et al.* (2015); Venkatesan and Li (2016); Ba and Caruana (2014). Intermediate-representations (single layer) based KT technique was applied only on Type III. None of these works justified why they evaluated their KT techniques only on specific architectures. In order to drive general conclusions on the effectiveness of KT techniques, the behavior of KT techniques on different architectures is analyzed.

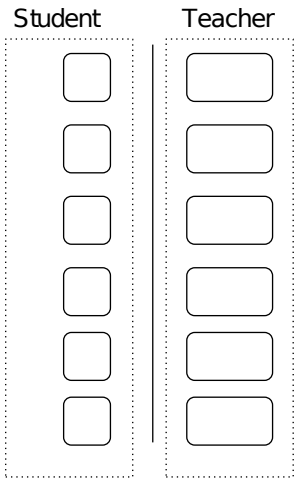


Figure 3.6: Type II Architecture:: Student Model is 1/20 Times Thinner than the Teacher Model but Both have Same Height

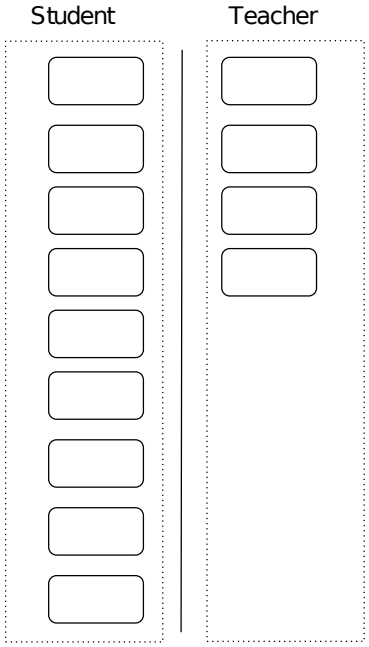


Figure 3.7: Type III Architecture:: Student Model is Thinner and Deeper Than the Teacher Model

Chapter 4

MOTIVATION & PROPOSED WORK

4.1 Knowledge Transfer from Multi Task Learning Perspective

Multi task learning (MTL) as mentioned in Ruder (2017) has been used in various areas of machine learning such as natural language processing and computer vision described in Collobert and Weston (2008) and Ren *et al.* (2015). MTL is a machine learning technique which learns multiple tasks at the same time. In neural networks perspective, the learning technique becomes MTL if the neural network is trained to classify more than one task. For instance, a neural network while classifying an image as a cat or a dog can also predict the breed of it. In this case, classifying the image as an animal is one task and predicting it's breed is another task.

MTL in neural networks is achieved by having multiple loss functions; each assigned to a task. Neural network learns each task by optimizing the assigned loss function. Traditional knowledge transfer in student-teacher neural networks can effectively be viewed as MTL, where the student network learns two tasks simultaneously by optimizing two loss functions: (1) cross entropy loss between the true labels of the dataset and the predicted labels of the student. By optimizing the cross entropy loss, student learns the representations from the dataset; (2) RMSE loss between the guided layers of the student and the hint layers of the teacher. By minimizing the RMSE loss, the student network learns the representations of the teacher's hint layers. Here, learning the representations from the teacher is one task and learning from the dataset is another. In this way, the student also learns from the teacher's hint layers while learning from the dataset, leading to better performance. So far,

student-teacher knowledge transfer was not seen from the perspective of MTL.

The proposed knowledge transfer technique in the student-teacher network is also based on the idea of multi task learning with the following tasks: (1) to learn the representations of the teacher's layers by minimizing the RMSE loss between the outputs (representations) of the x^{th} layers of the teacher and the student; (2) to learn the representations of the dataset. However, as opposed to traditional knowledge transfer technique, the tasks here are independent of each other. For example, one of the tasks is, student's x^{th} layer learns the representations from the x^{th} layer of the teacher; the other task is, student's y^{th} layer learns the representations of the teacher's y^{th} layer. In this case, while the student's x^{th} layer learns the representations from the teacher's x^{th} layer, student's y^{th} layer learns only from the y^{th} layer of the teacher as opposed to traditional knowledge transfer technique where the student's y^{th} layer not only learns the representations from the teacher's y^{th} layer but also from the teacher's x^{th}layer.

The proposed knowledge transfer technique brings an improvement of 10% over the traditional knowledge transfer techniques. Reason for this improvement is, since the tasks are independent, in every task, student is determined to learn only appropriate representations rather than learning all the representations of the teacher. For a better understanding, assume the first layer of the teacher and student network extract edges of the input image. Second layer extracts objects like circles. Hence, learning the representations of the first layer of the teacher for the second layer of the student is not beneficial.

Now let's see, how to map the layers of student-teacher? the mapping of x and y layers of student-teacher is analysed and found that the performance varies based on the combination of x and y. Random mapping of the teacher-student layers can sometimes be lucky but not always. I propose to use cosine similarity metric to

Table 4.1: Mapping of Teacher→Student Layers of Same Widths

Architecture	**Same Width Mapping**
Type I	$2^{nd}\rightarrow 1^{st}$, $4^{rd}\rightarrow 2^{nd}$,$5^{th}\rightarrow 3^{rd}$, $10^{th}\rightarrow 4^{rd}$, $12^{th}\rightarrow 5^{rd}$

Table 4.2: Mapping of Teacher→Student Layers of Different Widths

Architecture	**Different Width Mapping**
Type I	$3^{nd}\rightarrow 1^{st}$, $5^{rd}\rightarrow 2^{nd}$, $8^{th}\rightarrow 3^{rd}$, $10^{th}\rightarrow 4^{th}$, $12^{th}\rightarrow 5^{th}$

find the mapping between student-teacher layers. In cosine similarity metric, 1) the outputs of the student-teacher layer pairs are normalized; 2) dot product of the normalized outputs is calculated; (3) layer pair with highest dot product value is chosen.

The proposed technique is implemented on Type I architecture. Table 4.1 and 4.2 show the layers mapped from the teacher model to the student model. In Table 4.1, the mapping is obtained by (1) calculating the cosine similarity metric between the student-teacher layer pairs having the same width, where width is defined as the number of feature maps of the layer; (2) choosing the layer pair with the highest cosine similarity value. On the other hand, in Table 4.2, mapping between the student-teacher layer pairs is obtained by calculating cosine similarity metric between all the student-teacher layer pairs including the ones with different widths, and the layer pair with highest cosine similarity value is chosen.

Mapping between the student-teacher layers of the same width is straightforward. Fig 4.2 demonstrates student-teacher mapping layers of same width. Here, the overall loss function is the sum of RMSE losses evaluated between the student-teacher mapping layers. RMSE loss between the student-teacher mapping layers is evaluated by summing up the RMSE values between its feature maps as shown in Fig. 4.5. In Fig. 4.5, a,b indicate the feature maps of the student-mapping layer, and c, d indicate

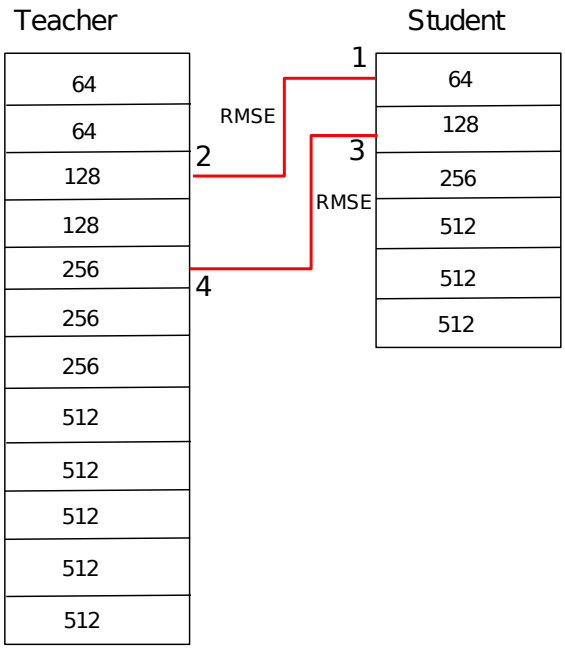


Figure 4.1: Demonstrates Mapping between Teacher and Student Layers of Different Widths. Teacher's 3rd Layer Consisting of 128 Neurons is Mapped to Student's 1st Layer Consisting of 64 Neurons. RMSE loss is Evaluated Between 1 and 2 Mapping Layers. Similarly, RMSE loss is Evaluated Between 3 and 4 Mapping Layers.

the feature maps of the teacher-mapping layer. Thus, the RMSE loss of the student-teacher mapping layer is the addition of the RMSE evaluated between a and c feature maps and the RMSE evaluated between b and d. Further, RMSE between the feature maps is calculated element wise. On the other hand, Fig 4.1, shows the mapping of the student and teacher layers of different widths. Here, the RMSE between the student-teacher mapping layers cannot be evaluated as the number of feature maps of the student and teacher mapping layers are not equal as shown in Fig. 4.6. In Fig . 4.6, a, b are two feature maps of the student layer and c is the only feature map of the teacher layer. Thus, RMSE of the c feature map cannot be evaluated.

For the student-teacher mapping layers of different widths, an extra layer, which is called as embed, and consists of 64 neurons is added to both the teacher and student mapping layers to produce equal number of feature maps. This is achieved

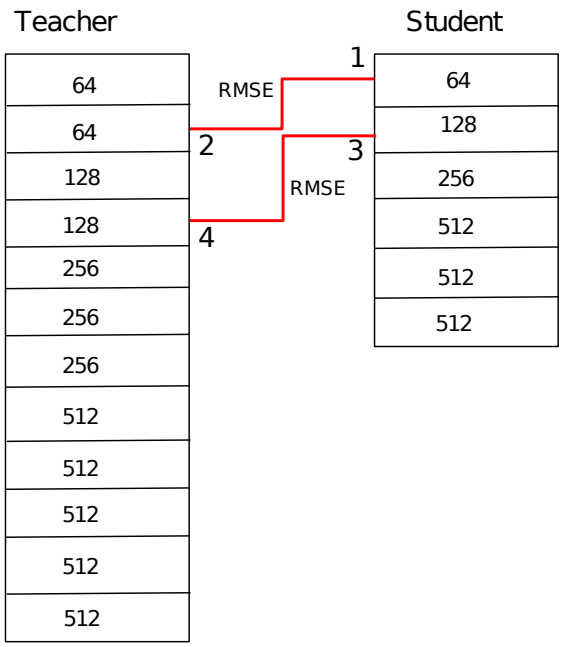


Figure 4.2: Demonstrates Mapping Between Teacher and Student Layers of Same Widths. 3rd Layer of Teacher Consisting of 128 Neurons is Mapped to 1st Layer of Student Consisting of 128 Neurons. RMSE loss is Evaluated Between 1 and 2 Mapping Layers. Similarly, RMSE loss is Evaluated Between 3 and 4 Mapping Layers.

by connecting an embed layer in the form of fully connected or convolution layer to the teacher-mapping layer; similarly, connecting another embed layer to the student-mapping layer as shown in Fig 4.3 and Fig 4.4. As a result, RMSE loss between the teacher-student mapping layers can be evaluated by calculating the RMSE between the feature maps of the student-teacher embed layers.

In order to have equal number of feature maps for the teacher-student mapping layers, an embed layer is utilized. However, the embed layer connects the mapping layers having features maps of same size. Now, let's discuss how to connect layers having feature maps of different sizes? As shown in Fig 4.5, if the feature maps have the same size the RMSE value is calculated element wise. However, as shown in Fig 4.6 the RMSE value cannot be calculated on the feature maps of different sizes such as 2X2 of student and 2X3 of teacher. Thus, all the feature maps are reduced

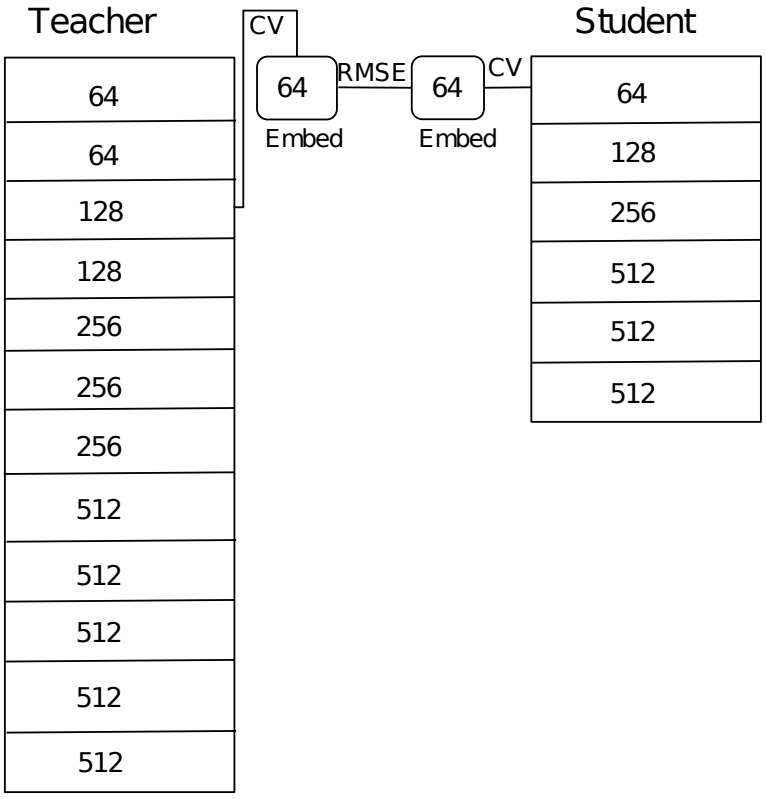


Figure 4.3: CV Indicates Convolution Layer; Embed Layers Consists of 64 Feature Maps and are Connected to Teacher and Student Layers as Convolution Layers. RMSE Loss Between the Embed Layers of Teacher and Student is Calculated

to equal size of 1x1 by taking the mean of all the elements of each feature map as shown in Fig 4.9.

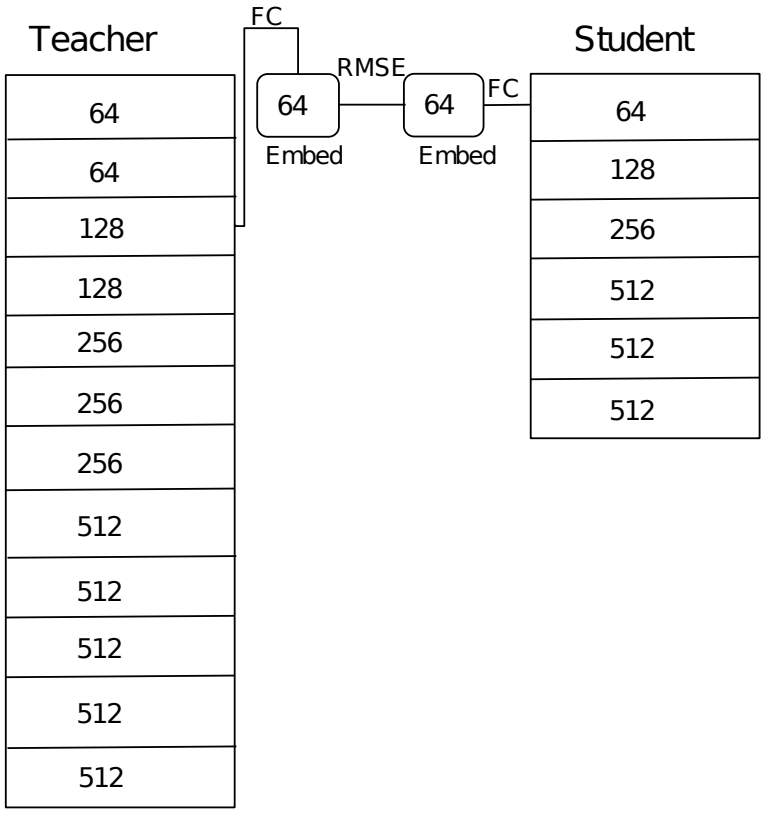


Figure 4.4: FC Indicates Fully Connected Layer; Embed Layers Consists of 64 Feature Maps and are Connected to Teacher and Student Layers as Fully Connected Layers.

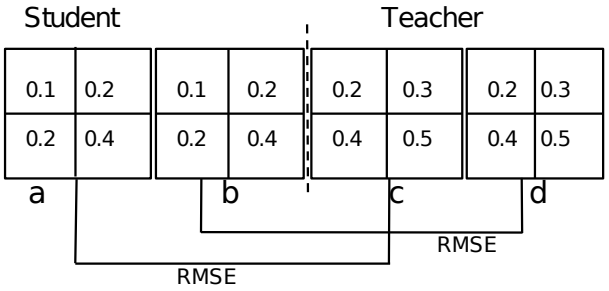


Figure 4.5: a, b are the Feature Maps of the Student Layer. c,d are the Feature Maps of the Teacher Layer. Both Student and Teacher Layers have Equal Number of Feature Maps.

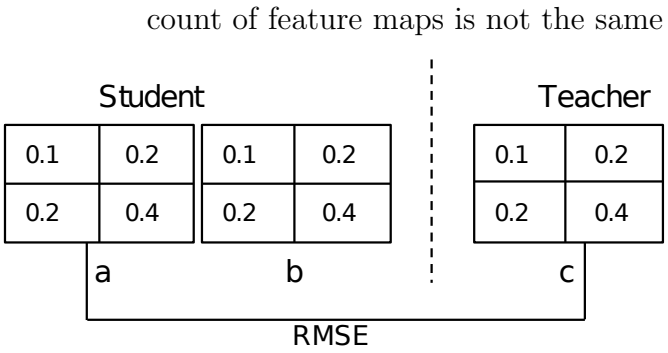


Figure 4.6: *a*, *b* are the Feature Maps of the Student Layer. *c* is the Feature Map of the Teacher Layer. Student and Teacher Layers have Unequal Number of Feature Maps.

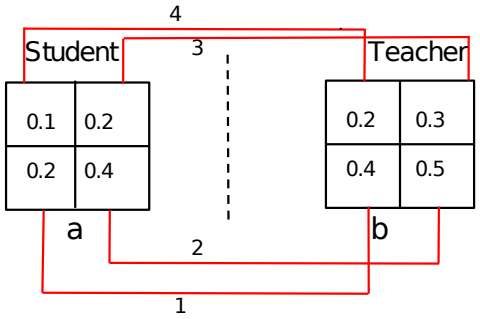


Figure 4.7: *a* is the Feature Map of the Student Layer. *b* is the Feature Map of the Teacher Layer. *a*, *b* Feature Maps have Same Shape of 2X2.

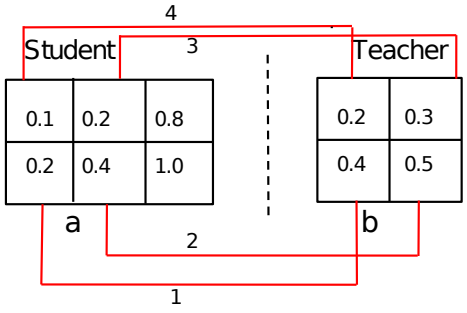


Figure 4.8: *a* is the Feature Map of the Student Layer. *b* is the Feature Map of the Teacher Layer. *a*, *b* Feature Maps have Different Shape. *a* is of Shape 2X3; Whereas, *b* is of Shape 2X2.

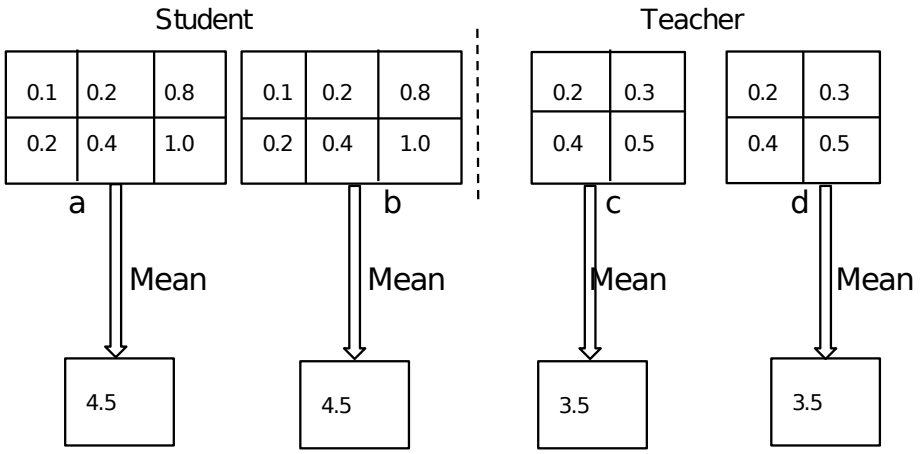


Figure 4.9: Mean of all the Elements of Each Feature Map a, feature map b is Calculated. Similarly, Mean of all the Elements of the Feature Map c and d is Calculated. The Output Feature Maps are of Size 1X1.

Chapter 5

METHODOLOGY

In this section, details of the methodology are discussed to evaluate the performance in terms of accuracy and convergence rate of student models trained by applying different KT techniques described in Section 3.

In addition, (1) Characteristics of datasets, (2) Training methodology and (3) Hyperparameters tuned while training networks are also discussed. Tensorflow introduced by Abadi *et al.* (2015) is used to build the models, and are run on a Tesla K40 GPU, hosted on a server equipped with dual Intel Xeon E5-2630 processors and 64GB of main memory (unless otherwise noted). Although the experiments were run on a server, all the student models can also run on a typical edge device such as a smart phone (Google Nexus 5). The relative performance of KT w.r.t. the baselines should hold on edge devices.

5.1 Benchmark Datasets

- **CIFAR-10** dataset consists of 60,000 (32X32) RGB natural images from 10 different object classes with 6000 images per class Krizhevsky and Hinton (2009). There are 50,000 training images and 10,000 test images.

- **Caltech 101** dataset consists of 9145 RGB images (224X224), belonging to 101 classes. Each class has 40 to 800 images. We divided the dataset into three parts: the training set consists of 5853 images (64% of total dataset), the testing set consists of 1829 images (20%), and the validation set consists of 1463 images (16%) Fei-Fei *et al.* (2007).

CIFAR-10 and Caltech 101 data sets are augmented with random left and right flipping during training. Both the datasets are normalized with zero mean and unit standard deviation before feeding into network. This preprocessing is being done to better generalize the trained model.

5.2 Training Methodology

Two baselines are considered:

- **Teacher:** Teacher model can be used for comparison with student model, in order to see how much the student represents the state-of-the-art accuracy. Teacher model provides supervision for the student model. Thus, the teacher model should be trained ahead of the student model. All three types of teacher models mentioned in Section 3.2 are trained from scratch on target datasets Caltech 101 and CIFAR-10 in the usual way using the cross-entropy loss formulated in Eq. 5.1.

$$\sum_{i=1}^{n}[y_t log\hat{y}_t + (1-\hat{y}_t)log(1-y_t)] \tag{5.1}$$

  Where:

  - y_t: True labels of the datasets Cifar10/Caltech101
  - $\hat{y}_t$: Predicted softmax output of teacher

- **Independent Student:** As a baseline, an independent student model is also considered which is trained independently from scratch without any form of knowledge transfer techniques. Input data to the student model is the image and target data is the true labels of the datasets Caltech 101/CIFAR-10. Student model is trained to minimize the cross-entropy loss formulated in Eq. 5.2.

In order to measure the effectiveness of the KT techniques, the performance of the student models trained under supervision is compared with the baseline independent student model.

$$\sum_{i=1}^{n}[y_s log\hat{y}_s + (1-\hat{y}_s)log(1-y_s)] \tag{5.2}$$

Where:

- y_s: True labels of the datasets CIFAR-10/Caltech 101
- $\hat{y}_s$: Predicted softmax output of student

- **Dependent Student:** In this approach, the student model is trained under the supervision of the teacher model using KT techniques mentioned in Section 3.

  In order to apply each of these KT techniques same batch of input data (Caltech 101/CIFAR-10) is passed to the teacher and student models. The teacher model does the inference on the input data and predicts the output. Student model is trained by minimizing the loss function formulated with teacher's output, as mentioned in Section 3.

  While applying intermediate-layer-representations (single layer) KT technique, 7^{th} layer of the teacher and 3^{rd} layer of the student are chosen as intermediate layers in Type I architecture. In Type II, 7^{th} layer of the teacher and 7^{th} layer of the student are chosen. Finally, in Type III, 2^{nd} layer of the teacher and 11^{th} layer of the student are chosen as demonstrated in Fig. 5.1.

  Similarly, while applying intermediate-layer-representations (multiple layers) KT technique, 2^{nd}, 3^{nd} and 5^{th} layers of the teacher and 1^{st}, 2^{nd} and 3^{rd} layers of the student are chosen as intermediate layers in Type I architecture. In Type

Table 5.1: Mapping of Teacher→Student Single Layer Pairs

Single Layer	Mapping
Type I	$7^{th} \rightarrow 3^{rd}$
Type II	$7^{th} \rightarrow 7^{th}$
Type III	$2^{nd} \rightarrow 11^{th}$

Table 5.2: Mapping of Teacher→Student Multiple Layer Pairs

Multiple Layers	Mapping
Type I	$2^{nd} \rightarrow 1^{st}$, $3^{rd} \rightarrow 2^{nd}$, $5^{th} \rightarrow 3^{rd}$
Type II & Type III	$1^{st} \rightarrow 1^{st}$, $2^{nd} \rightarrow 2^{nd}$, $3^{rd} \rightarrow 3^{rd}$

II and Type III, 1^{st}, 2^{nd}, and 3^{rd} are chosen as intermediate layers of the teacher and student models as shown in Fig. 5.2.

5.3 Hyperparameters

Batch size of 128 is used to train networks of all three types on CIFAR-10 dataset and batch size of 25 on Caltech 101 dataset. Initial learning rates for all three types are set to $10e^{-2}$. They are decayed exponentially each epoch with a factor of 0.98. All the networks are trained for 100K iterations and validation and test accuracy are calculated at each epoch. Final accuracy of the model is determined as the test accuracy attained at the epoch with highest validation accuracy.

Chapter 6

DISCUSSION AND RESULTS

In this section, the performance in terms of accuracy and convergence time of the dependent student that belongs to Type I, Type II, and Type III architectures with the corresponding independent student models are compared, in order to measure the effectiveness of KT techniques.

- *Convergence Time*, which is the total training time required by the network to reach the smallest possible validation loss. After convergence, loss value will not decrease, but only fluctuates around a specific value. In this study, convergence time of the network is evaluated as the total number of iterations required to reach 90% of the Top-1 accuracy.

- *Classification Accuracy*, is termed as the proportion of correct predicted labels among all the predictions obtained by the network. Following are the observations and analysis of the convergence time and Top-1 accuracy.

6.1 Results & Discussion on Convergence Time

Fig. 6.1 and Fig. 6.2 represent top-1 accuracy after every thousand iterations of the dependent student and baseline models. Dependent student and baseline models in Fig. 6.1 are of Type I architecture and are trained on Caltech 101 dataset whereas the ones in Fig. 6.2 follow Type III architecture while trained on CIFAR-10 dataset.

In Fig. 6.1, dependent student outperforms independent student after every iteration. Further, the number of iterations required by the independent student to

reach 90% of its top-1 accuracy (55.11%) is 17K whereas the dependent student model trained using intermediate layer representations (single layer) requires only 0.95K iterations to reach 90% of its best accuracy (61.74%). This shows dependent student converges approximately 17 times faster than the independent student.

Similarly in Fig. 6.2, dependent student outperforms independent student after every iteration. Moreover, number of iterations required by the independent student to reach 90% of its top-1 accuracy (66.12%) is 15K whereas the dependent student model trained using soft logits requires only 7K iterations to reach 90% of its best accuracy (67.78%). This indicates KT improves the convergence time of the student model drastically, allowing the dependent student to converge in less than half of the number of iterations required by the independent student.

In addition, difference in the accuracy gap between the dependent and independent student in Fig. 6.1 is higher compared to Fig. 6.2. Reason could be the teacher model lacks the capability to capture more complicated features compared to Type I teacher model. Thus, the knowledge transferred from the teacher model is not as effective as Type I. Moreover, CIFAR-10 dataset consists of 40,000 images in the training set. Since the student model is trained on 40,000 images, student model gets sufficient supervision from the dataset itself. As a result, the extra supervision from the teacher model does not improve the accuracy of the student model drastically.

In short, the results show that some level of speedup is achieved on all the architectures, whereas the best improvement (16X) still comes from Type 1 architecture with the use of intermediate-representation KT.

6.2 Results & Discussion on Top-1 Accuracy

Table. 6.1 and Table. 6.2 shows Top-1 accuracy of the dependent student and baseline models that belong to Type I architecture, while trained on Caltech 101 and CIFAR-10 datasets. Here, dependent student, trained using Intermediate layer representations (single layer) KT technique on Caltech 101, performed 7.36% better than the independent student. We believe, the huge improvement in the accuracy of the dependent student is because, in Type I, the teacher model is deeper than the student model. Thus, the teacher model can capture more complicated features, which results in more effective knowledge transfer.

However, dependent student trained using hard logits technique on CIFAR-10 performed only 1.88% better than the independent student. Since, student model is trained on CIFAR-10 dataset which has 40,000 images in the training set as opposed to 5853 images in Caltech 101, student model gets sufficient supervision from the dataset itself which makes extra supervision from the teacher model ineffective.

Results in Table. 6.1 are similar to the results shown in Hinton *et al.* (2015); Ba and Caruana (2014). Table. 6.3 and Table. 6.4 indicates Top-1 accuracy of dependent and baseline models of Type II architecture while trained on Caltech 101 and CIFAR-10. None of the KT techniques improved the accuracy of the dependent student trained on Caltech 101 over the independent student.

However, dependent student trained using Intermediate layer representations (single layer) technique on CIFAR-10, performed 1.4% better than the independent student.

Table. 6.5 demonstrates that the independent student outperformed the teacher

model. Here, all models belong to Type III architecture and are trained on CIFAR-10 dataset. Although Romero *et al.* (2014) claims that the student model trained using intermediate layer representations (single layer) KT technique outperformed the teacher model, I believe the major improvement is due to the depth of the student model, but not because of KT technique. To elaborate on this, non-linearity increases with the increase in depth of the student model. This results in learning more complex representations, thereby higher classification accuracy.

Similar to Romero *et al.* (2014), I also observe that student model with intermediate layer representations (single layer) KT technique gained an accuracy of 68.24% compared to the baseline teacher model accuracy, which is 68.08%. However, the accuracy of student trained with intermediate layer representations (single layer) KT technique is much lower than the independent student, making the usage of KT technique ineffective.

In addition, the improvement in the accuracy of the student model combined with soft logits KT technique is only 1.85%. I believe the knowledge transfer through teacher is not much effective in this case for two reasons: (1) Since the teacher model is not deeper than the student model as opposed to Type I architecture, it cannot capture more complicated features. Thus, the knowledge transferred from the teacher model is not as effective as Type I. (2) Same as Table. 6.2 student model gets sufficient knowledge from the dataset, allowing the supervision from teacher model inadequate.

Further, intermediate layer resentations KT technique decreased the accuracy of the dependent network over the independent network opposed to the behavior shown in Table. 6.1 and Table. 6.2. This shows, a particular KT technique

Table 6.1: Type I architecture & Caltech 101 Dataset

KT techniques	**Top-1 accuracy (%)**
Baseline (Teacher)	74.12
Baseline (Independent Student)	61.24
Hard Logits	61.27
Soft Logits	63.73
Intermediate Representations (Single Layer)	**68.60**
Intermediate Representations (Multiple Layers)	68.22

behaves differently on different architectures.

Table 6.13 shows number of iterations required by the independent and dependent student (trained with respective best KT techniques) to reach 90% of its best accuracy. Fig. 6.1 - Fig. 6.4 shows how their Top-1 accuracies evolve over time. Note that each iteration of the dependent student does slightly more work than the independent student, because the former requires the transfer of the teacher's output values. However, the time spent on this transfer is insignificant compared to the student's training time (especially when the student runs on a resource-constrained edge devcie). Therefore, here we use only the number of iterations to measure the convergence speed.

The results show that some level of speedup is achieved on all the architectures, whereas the best improvement (16X) still comes from Type 1 architecture with the use of intermediate-representation KT.

Table 6.2: Type I Architecture & CIFAR-10 Dataset

KT techniques	**Top-1 accuracy (%)**
Baseline (Teacher)	77.71
Baseline (Independent Student)	73.31
Hard Logits	**75.19**
Soft Logits	75.01
Intermediate Representations (Single Layer)	74.98
Intermediate Representations (Multiple Layers)	74.47

Table 6.3: Type II Architecture & Caltech 101 Dataset

KT techniques	**Top-1 accuracy (%)**
Baseline (Teacher)	70.74
Baseline (Independent Student)	60.11
Hard Logits	29.89
Soft Logits	**50.63**
Intermediate Representations (Single Layer)	29.68
Intermediate Representations (Multiple Layers)	10.25

Table 6.4: Type II Architecture & CIFAR-10 Dataset

KT techniques	**Top-1 accuracy (%)**
Baseline (Teacher)	75.64
Baseline (Independent Student)	47.72
Hard Logits	41.09
Soft Logits	29.08
Intermediate Representations (Single Layer)	**49.12**
Intermediate Representations (Multiple Layers)	39.09

Table 6.5: Type III Architecture & CIFAR-10 Dataset

KT techniques	**Top-1 accuracy (%)**
Baseline (Teacher)	68.08
Baseline (Independent Student)	73.47
Hard Logits	14.54
Soft Logits	**75.32**
Intermediate Representations (Single Layer)	68.24
Intermediate Representations (Multiple Layers)	72.56

Table 6.6: Type III Architecture & Caltech 101 Dataset

KT techniques	**Top-1 accuracy (%)**
Baseline (Teacher)	64.04
Baseline (Independent Student)	70.67
Hard Logits	10.24
Soft Logits	**74.32**
Intermediate Representations (Single Layer)	68.35
Intermediate Representations (Multiple Layers)	73.34

Table 6.7: Proposed KT on Type I Architecture & CIFAR-10 Dataset

KT techniques	**Top-1 accuracy (%)**
Baseline (Teacher)	77.71
Baseline (Independent Student)	73.31
Student with Best KT	75.19
Student with proposed KT	77.80

Table 6.8: Proposed KT on Type I Architecture & Caltech 101 Dataset

KT techniques	**Top-1 accuracy (%)**
Baseline (Teacher)	74.12
Baseline (Independent Student)	61.24
Student with Best KT	68.60
Student with proposed KT	**78.79**

Table 6.9: Proposed KT on Type II Architecture & CIFAR-10 Dataset

KT techniques	**Top-1 accuracy (%)**
Baseline (Teacher)	75.64
Baseline (Independent Student)	47.72
Student with Best KT	49.12
Student with proposed KT	44.57

Table 6.10: Proposed KT on Type II Architecture & Caltech 101 Dataset

KT techniques	**Top-1 accuracy (%)**
Baseline (Teacher)	70.74
Baseline (Independent Student)	60.11
Student with Best KT	50.63
Student with proposed KT	47.90

Table 6.11: Proposed KT on Type III Architecture & CIFAR-10 Dataset

KT techniques	**Top-1 accuracy (%)**
Baseline (Teacher)	68.08
Baseline (Independent Student)	73.47
Student with Best KT	75.32
Student with proposed KT	76.47

Table 6.12: Proposed KT on Type III Architecture & Caltech 101 Dataset

KT techniques	**Top-1 accuracy (%)**
Baseline (Teacher)	64.04
Baseline (Independent Student)	70.67
Student with Best KT	74.32
Student with proposed KT	75.87

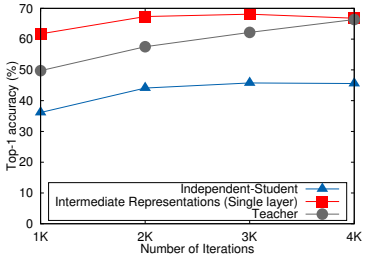


Figure 6.1: Top-1 Accuracy of Dependent Student and Baseline Models of Type I on Caltech 101

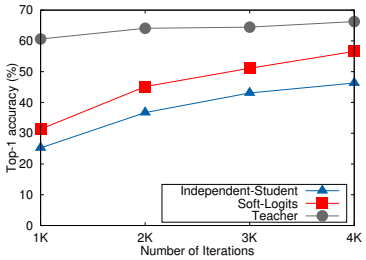


Figure 6.2: Top-1 Accuracy of Dependent Student and Baseline Models of Type III on CIFAR-10

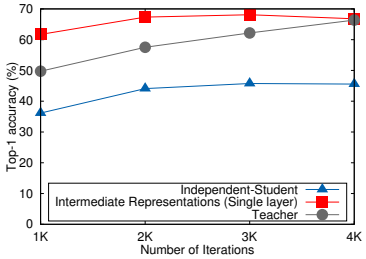


Figure 6.3: Top-1 Accuracy of Dependent Student and Baseline Models of Type I on CIFAR-10

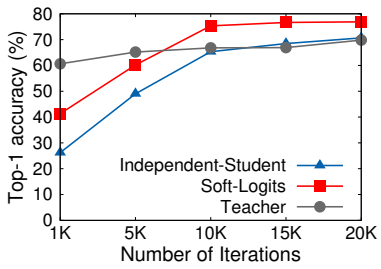


Figure 6.4: Top-1 Accuracy of Dependent Student and Baseline Models of Type III on Caltech 101

Table 6.13: Number of iterations required by the independent and dependent student models (trained with respective best KT techniques) to reach 90% of its best accuracy

Student Model	Caltech 101		CIFAR-10	
	Type 1	Type 3	Type 1	Type 3
Independent	17K	17K	1.1K	15K
Dependent	1K	6K	1K	7K
SpeedUp	1600%	183.33%	10%	114%

Chapter 7

CONCLUSIONS AND FUTURE DIRECTIONS

This paper provides a comprehensive study of existing KT techniques, which is important to understand the effectiveness of the knowledge transfer approach for enabling deep learning on resource-constrained edge devices. Four different KT techniques and three different model architectures are considered to evaluate their performance in terms of both accuracy and convergence time. Results show that only intermediate-representation KT technique and Type I model achieve significant accuracy improvement (up to 7.36%) for the dependent student model compared to the independent student.

Empirically, the effectiveness of KT techniques depend on type of the dataset and architecture used to train the student model. Among the three architectures only Type I architecture showed a significant improvement in the accuracy and convergence time. Although KT did not bring much improvement in the accuracy of Type III the model to achieve best performance much faster than the independent student. Thus student models of Type I can take the advantage of KT to get trained faster and achieve better accuracy.

The intermediate-representation KT technique is the most promising as it allows knowledge to be transferred from the intermediate layers in addition to the last layer. With respect to convergence time, all KT techniques help the dependent student model converge faster with a speedup ranging from 10% to 1600% compared to independent student model. The intermediate-representation KT technique also achieves the best speedup in convergence time compared to the other KT techniques.

REFERENCES

Abadi, M., A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems", URL `https://www.tensorflow.org/`, software available from tensorflow.org (2015).

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

Bengio, Y. *et al.*, "Learning deep architectures for ai", Foundations and trends® in Machine Learning **2**, 1, 1–127 (2009).

Chan, W., N. Jaitly, Q. Le and O. Vinyals, "Listen, attend and spell: A neural network for large vocabulary conversational speech recognition", in "Acoustics, Speech and Signal Processing (ICASSP), 2016 IEEE International Conference on", pp. 4960–4964 (IEEE, 2016).

Chandakkar, P. S., Y. Li, P. L. K. Ding and B. Li, "Strategies for re-training a pruned neural network in an edge computing paradigm", in "IEEE International Conference on Edge Computing", (2017).

Chen, W., J. Wilson, S. Tyree, K. Weinberger and Y. Chen, "Compressing neural networks with the hashing trick", in "International Conference on Machine Learning", pp. 2285–2294 (2015).

Chollet, F., "Xception: Deep learning with depthwise separable convolutions", arXiv preprint arXiv:1610.02357 (2016).

Collobert, R. and J. Weston, "A unified architecture for natural language processing: Deep neural networks with multitask learning", in "Proceedings of the 25th international conference on Machine learning", pp. 160–167 (ACM, 2008).

Courbariaux, M., I. Hubara, D. Soudry, R. El-Yaniv and Y. Bengio, "Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1", arXiv preprint arXiv:1602.02830 (2016).

Denton, E. L., 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", pp. 1269–1277 (2014).

Fei-Fei, L., R. Fergus and P. Perona, "Learning generative visual models from few training examples: An incremental bayesian approach tested on 101 object categories", Computer vision and Image understanding **106**, 1, 59–70 (2007).

Goodfellow, I., J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville and Y. Bengio, "Generative adversarial nets", in "Advances in neural information processing systems", pp. 2672–2680 (2014).

Han, S., J. Kang, H. Mao, Y. Hu, X. Li, Y. Li, D. Xie, H. Luo, S. Yao, Y. Wang *et al.*, "Ese: Efficient speech recognition engine with sparse lstm on fpga", in "ISFPGA", (2017).

Han, S., X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz and W. J. Dally, "EIE: efficient inference engine on compressed deep neural network", in "ISCA", (2016).

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

Han, S., J. Pool, J. Tran and W. Dally, "Learning both weights and connections for efficient neural network", in "NIPS'15", (2015b).

He, K., X. Zhang, S. Ren and J. Sun, "Deep residual learning for image recognition", in "Proceedings of the IEEE conference on computer vision and pattern recognition", pp. 770–778 (2016).

Hinton, G., O. Vinyals and J. Dean, "Distilling the knowledge in a neural network", arXiv preprint arXiv:1503.02531 (2015).

Howard, A. G., 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).

Huang, J.-T., J. Li, D. Yu, L. Deng and Y. Gong, "Cross-language knowledge transfer using multilingual deep neural network with shared hidden layers", in "Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on", pp. 7304–7308 (IEEE, 2013).

Iandola, F. N., S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <1mb model size", arXiv preprint arXiv:1602.07360 (2016).

Kadetotad, D., S. Arunachalam, C. Chakrabarti and J.-s. Seo, "Efficient memory compression in deep neural networks using coarse-grain sparsification for speech applications", in "Computer-Aided Design (ICCAD), 2016 IEEE/ACM International Conference on", pp. 1–8 (IEEE, 2016).

Kang, Y., J. Hauswald, C. Gao, A. Rovinski, T. Mudge, J. Mars and L. Tang, "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge", in "Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems", pp. 615–629 (ACM, 2017).

Krizhevsky, A. and G. Hinton, "Learning multiple layers of features from tiny images", Citeseer (2009).

Le Cun, Y., J. S. Denker and A. Sara, "Solla, optimal brain damage in advances in neural information processing systems 2, edited by david s", (1990).

LeCun, Y., J. S. Denker and S. A. Solla, "Optimal brain damage", in "Advances in neural information processing systems", pp. 598–605 (1990).

Ren, S., K. He, R. Girshick and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks", in "Advances in neural information processing systems", pp. 91–99 (2015).

Romero, A., N. Ballas, S. E. Kahou, A. Chassang, C. Gatta and Y. Bengio, "Fitnets: Hints for thin deep nets", arXiv preprint arXiv:1412.6550 (2014).

Ruder, S., "An overview of multi-task learning in deep neural networks", arXiv preprint arXiv:1706.05098 (2017).

Sau, B. B. and V. N. Balasubramanian, "Deep model compression: Distilling knowledge from noisy teachers", arXiv preprint arXiv:1610.09650 (2016).

Simonyan, K. and A. Zisserman, "Very deep convolutional networks for large-scale image recognition", arXiv preprint arXiv:1409.1556 (2014).

Srinivas, S. and R. V. Babu, "Data-free parameter pruning for deep neural networks", arXiv preprint arXiv:1507.06149 (2015).

Venkatesan, R. and B. Li, "Diving deeper into mentee networks", arXiv preprint arXiv:1604.08220 (2016).