Are Existing Knowledge Transfer Techniques Effective For Deep Learning with Edge Devices?

Ragini Sharma, Saman Biookaghazadeh, Baoxin Li and Ming Zhao
Arizona State University, Tempe, Arizona
Email: {rsistla,sbiookag,baoxinli,mingzhao}@asu.edu

***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 for deployment on edge devices, but they may lose their capability and not perform well. 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. This paper provides an extensive study on the performance (in both accuracy and convergence speed) of knowledge transfer, considering different student architectures and different techniques for transferring knowledge from teacher to student. The results show that the performance of KT does vary by architectures and transfer techniques. 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.**

***Index Terms*—Deep neural networks, edge computing, cloud computing, knowledge transfer**

I. Introduction

Deep neural networks (DNNs) have achieved tremendous accuracy improvements compared to conventional machine learning techniques for many important tasks, such as image classification and speech recognition. 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 involves 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. At the same time, the available computing and storage resources on modern edge devices are not utilized to help the learning. 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 users directly interact with; 2) *Responsiveness* to changes in 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 the computational and memory intensive models on edge devices such as mobile phones and smart cameras is challenging, since such devices are based on System on chip (SoC) architecture with limited resources designed to fulfill requirements of embedded/mobile applications. To enable learning on such resource-constrained devices, several different approaches were proposed. The knowledge transfer (***KT***) approach is particularly interesting, which trains smaller networks (termed ***students*** hereinafter) under the supervision of the larger networks (termed ***teachers*** hereinafter) to improve accuracy and speed.

There are potentially several advantages of this KT approach compared to the others. *First*, it can help the student network converge faster by utilizing the information coming from the teacher network. Such information can help the optimization phase of the student network by directing student's parameters into the same representations that the teacher network captures. As a result, the student network can possibly approach an accurate representation faster than an independent network trained without the teacher's supervision. *Second*, it can possibly improve the accuracy of the student network. KT allows student models to arrive at better parameter values based on the teacher model's parameter values, which can deliver a classification function with higher accuracy. *Third*, it can help the student network become more general by preventing from getting biased toward a certain set of data.

Prior works on KT [1]–[4] focused only on the performance comparison between student and teacher models, but they ignored the comparison between the student models trained with and without KT. This comparison is important to understanding the effectiveness of the KT techniques. Moreover, in terms of performance, the related works focused only on the accuracy of the student models, but they ignored the convergence time. Faster convergence can bring several benefits, such as better response time and less power consumption. In addition, each of the related works has only limited coverage on student network architectures which are either shallower or thinner compared to teacher models. The assumption that

a certain type of KT technique is applicable to all types of architectures is not necessarily true. In order to generalize the effectiveness of KT techniques, we believe that the behavior of KT techniques on different architectures needs to be evaluated.

Our goal is to provide a thorough analysis of the effectiveness of the existing KT techniques. We broadly classify the existing KT techniques into three categories based on the type of knowledge transferred from teacher networks to student networks: 1) the ***hard logits*** of a network [2], which are defined as the output of the last layer of the teacher network before passing to the softmax activation function; 2) the ***soft logits*** of a network [1], which are obtained by softening hard logits with the help of temperature softmax variable and then passing softened logits to the softmax activation function; and 3) the ***intermediate representations*** of a network (in addition to the soft logits) [3], [4], which are the outputs of the middle layers of the teacher network.

We study these KT techniques on three different types of student-teacher architectures: 1) ***Type I***, where the student network is shallower than teacher network [1], [2], [4]. 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 [5] with the width multiplier set to 0.05, whereas the teacher is the baseline MobileNet model with the width multiplier of 1.0; and 3) ***Type III*** student network [3] is thinner but deeper than teacher network where the student consists of 19 layers and the teacher consists of 5 layers.

We build the above models on TensorFlow [6], and evaluate them using the CIFAR-10 and Caltech 101 datasets on a GPU-based testbed. Overall our study shows some positive results for KT, but existing KT techniques do not behave the same on all architectures. In terms of accuracy, only the intermediate-representations KT technique and Type I student architecture achieve 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-representations KT technique on Type I architecture.

Based on our results, we conclude that KT does have potential to improve both accuracy and speed of a small network, but it is sensitive to how the knowledge is transferred 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, in our future work, we will conduct a more focused study on the intermediate-representations KT technique in order to understand its full potential.

The rest of the paper is organized as follows: Section II introduces the background and related works; Sections III and IV describe the existing KT techniques and possible student network architectures; Sections V and VI present the evaluation methodology and results; and finally, Section VII concludes the paper.

II. Background and Related Works

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 learning and cannot perform well when the cloud is overloaded or the network is unreliable. However, there are important reasons that we should exploit the capabilities of edge devices for deep learning:

- **Personalization**: For many applications, custom models 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 solely on 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 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 in training a generic model to assist the learning on the devices.

Related works proposed several model compression techniques in order to reduce a deep model to one that can be trained on an edge device. These techniques can be broadly classified into three categories as mentioned below.

- **Weight Sharing**: This technique reduces the memory occupied by the model by grouping connection weights and replacing them with a single value, which leads to storage of fewer parameters. K-means clustering technique was used to group connections weights, assign an average weight to each group, and replace all those weights with the average weight value [7]. HashedNets model shares weights by using a hash function to group weights into hash buckets and then using a single value to replace all of them [8].
- **Quantization**: The size of the model can be reduced by shrinking the number of bits needed by the weights. For example, the number of bits is reduced from 35 to 5 for every connection in the network in one related work [7]. Blockwise structured sparsity technique can be used to quantize the weights and activations, resulting in a reduction of 5-6 bit from each single weight variable [9]. By reducing the model's size, quantization helps reduce both the model's time and space costs.
- **Pruning Techniques**: The complexity of a model can be reduced extensively using pruning techniques. Magnitude-based pruning removes weights or connections which produces a negligible response. All the weights that are below a particular threshold value can be removed, resulting in the reduction of parameters by 9X and 13X for AlexNet and VGG-16 models, respectively [7]. The Optimal Brain Damage work reduces the number of weights based on the Hessian loss function [10]. The data-driven pruning technique removes redundant neurons which learn the same representations [11].

The above-mentioned techniques all focus on reducing the size of a model so that it can be deployed in a resource-constrained environment such as an edge device. The key difference between KT and these techniques is the existence of a teacher model which provides supervision to the model trained on the device and helps it achieve potentially better accuracy and speed.

In the KT approach, input data are passed through both the teacher, which is already trained and the student, which is trained from scratch. At every iteration, the inference outputs from the teacher are collected, which can come from the last layer and possibly some other layers, and provided to the student as a knowledge to help train the student. By doing so, student model has the potential to learn the representation that is already learned by the teacher model, despite being small.

Although there are several related works on KT, none of them provides a thorough study on the effectiveness of such techniques, and we still do not have good answers to several key questions: 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 paper is to provide a good understanding of KT by finding answers to the above questions through a comprehensive analysis of accuracy and speed of different KT techniques on different architectures. The rest of the paper details our methodology and results.

III. Knowledge Transfer Techniques

In this section, we broadly classify the existing KT techniques into three categories, and explain the basic approach and discuss the potential strengths of each of them.

A. Transferring Hard Logits

Hard-logits-based KT technique was introduced by Ba *et al.* [2]. 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 the logits (output of the last layer) of the deep teacher model and the softmax output of the shallow student model, as shown in Fig. 1(a). They used the logits of the teacher model directly as opposed to probabilities produced by passing logits to the 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, we name this KT approach ***transferring 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 the original labels of the dataset are also used. Loss function of this approach is formulated as follows:

$$\sqrt{\sum_{i=1}^{n}(\hat{y}_i^s - z_i^t)^2} \tag{1}$$

where:

$\hat{y}^s$: predicted softmax output of student
z^t: predicted hard logits of teacher
i: the ith feature map of teacher/student
n: the number of feature maps of teacher/student

This technique can be helpful for two major reasons. First, if true labels have errors, the teacher model helps eliminate some of them and helps the student learn the correct model. Second, the original labels may depend on the features that are not available as inputs to the student network. Thus, the teacher model helps eliminate those labels that are dependent on unavailable data and provide the labels which are dependent only on the input features.

B. Transferring Soft Logits

Hinton *et al.* proposed a knowledge distillation approach [1] to compress the knowledge of ensemble models into a student model. They achieved this by introducing a temperature softmax variable (T) as follows:

$$q_i = exp(z_i/T)/\sum_j exp(z_j/T) \tag{2}$$

where,

z_i: the output of i^{th} neuron of teacher's first fully connected layer (i.e., hard logits)

T: the temperature softmax variable—parameter to control the relative importance of the soft targets provided by the larger model. The higher the value of T, the softer are the targets

q_i: the output of i^{th} neuron of teacher's softmax layer (soft logits)

j: 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*) q_i as mentioned in Eq. 2. The student model minimizes the sum of two objective functions: (1) cross entropy loss between the soft logits, and (2) cross entropy loss between the softmax output and correct labels of the dataset as shown in Fig. 1(b). We name this KT approach ***transferring soft logits*** because the soft logits of the teacher network are used to train the student network.

The loss function of this technique is mentioned in Eq. 3. Interpretation of the loss function is as follow: the first term indicates the student's cross-entropy loss, and the second term indicates the cross-entropy loss between the soft logits of teacher and student. The student model minimizes the cross entropy loss between the soft logits while minimizing the overall loss as well. By doing so, the parameters of the student's model (weights and biases) tend to move towards the parameters of the teacher's model, which results in learning the same representations as that of teacher's.

$$J = \sum_{i=1}^{n}[y_i^s log\hat{y}_i^s + (1-y_i^s)log(1-\hat{y}_i^s)] + \sum_{i=1}^{n}[q_i^t log q_i^s + (1-q_i^t)log(1-q_i^s)] \quad (3)$$

where:

y^s: true labels of the datasets

$\hat{y^s}$: equal to q_s with a temperature softmax (T) value of 1

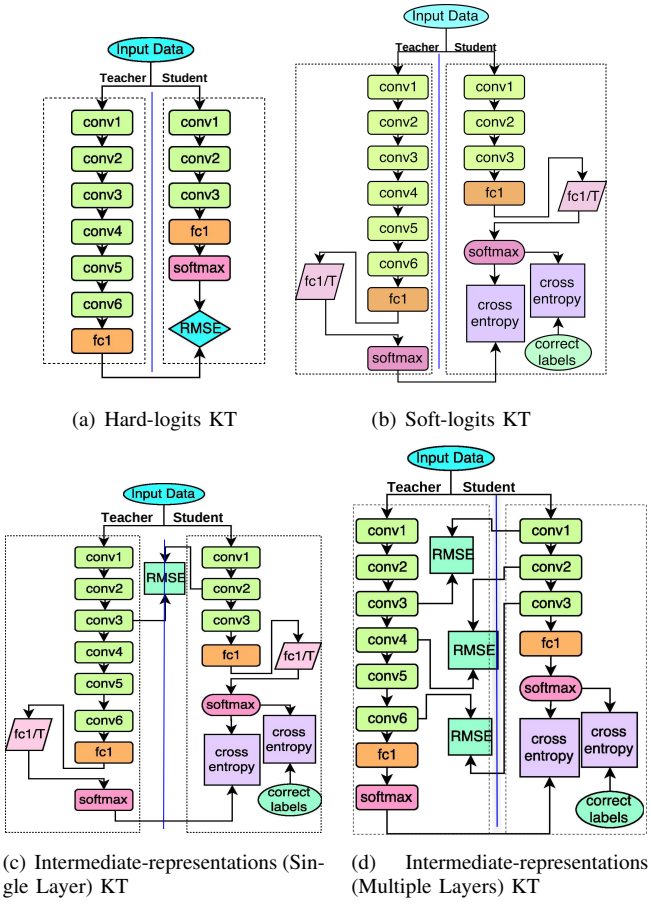
q^s: predicted softened softmax output of student (soft logits)

q^t: predicted softened softmax output of teacher (soft logits)

It is claimed that by using soft targets instead of hard targets, more useful information can be carried which is not possible if encoded with hard targets [1]. 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 stages.

C. Transferring Intermediate Representations

Both above approaches use only the hard or soft logits of the teacher model as knowledge to a student model. In addition to these logits, knowledge from the teacher's intermediate layers can also be used in the training phase. Therefore, we name this type of approach as ***transferring intermediate representations***.



(a) Hard-logits KT (b) Soft-logits KT

(c) Intermediate-representations (Single Layer) KT (d) Intermediate-representations (Multiple Layers) KT

Fig. 1: Different types of KT techniques

Romero *et al.* proposed to use the output of the middle layer of the teacher model as a hint to improve the performance of the deep and thin student model [3]. Unlike shallow models used in the previous two approaches, this approach assumes that the student 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. By using the hints derived from the intermediate representations, on the CIFAR-10 dataset, the student network, which is thinner but deeper than the teacher network and contains ten times fewer parameters, can outperform the teacher.

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 the teacher's hint layer (the 2^{nd} layer of the teacher network) as target labels, as illustrated in Fig. 1(c). 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 for training of the student model.

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 the entire network by minimizing the knowledge distillation (KD) loss. KD loss is defined as the sum of two cross entropy loss functions as mentioned in Eq 3.

$$\begin{aligned} J = & \sqrt{\sum_{i=1}^{n}(y_{ji}^{s} - y_{ki}^{t})^2} \\ & + \sum_{i=1}^{n}[y_i^s log\hat{y}_i^s + (1-y_i^s)log(1-\hat{y}_i^s)] \\ & + \sum_{i=1}^{n}[q_i^t logq_i^s + (1-q_i^t)log(1-q_i^s)] \end{aligned} \tag{4}$$

where:

y_{ji}^s: output of student's j^{th} layer ($j = 11$)
y_{ki}^t: output of teacher's k^{th} layer ($k = 2$)
$\hat{y}^s$: 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 loss function is interpreted as follows: the first term is the RMSE 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 RMSE loss while minimizing the overall loss function. By doing so, parameters of the student model (weights and biases) tend to move towards the parameters of the teacher's model. Thus, the student's intermediate layers' outputs approximate the teacher's intermediate layers' outputs. As a result, the student network has the potential to generalize and perform as well as the teacher network. Romoreo *et al.* claimed that a student network with 10 times fewer parameters than the teacher network could outperform the teacher network with the help of knowledge transferred through middle layer representations. As the intermediate layer considered for KT in this approach is only the middle layer of the teacher model, we name this specific KT technique as ***transferring intermediate representations (single layer)***.

The technique of transferring intermediate representations is extended further by Venkatesan *et al.* [4]. Here, instead of considering only the middle layer pairs of student-teacher models, they experimented with multiple intermediate layer pairs, including softmax layers, as shown in Fig. 1(d). First, they trained a teacher model on the dataset Caltech 256. Then they used the representations from the middle layers including the last layer of the pre-trained teacher model to train a student model on Caltech 101. Overall loss function of this approach is formulated as follows:

$$\begin{aligned} J = & \sum_{(j,k)\in M} \sqrt{\sum_{i=1}^{n}(y_{ji}^{s} - y_{ki}^{t})^2} \\ & + \sum_{i=1}^{n}[y_i^s log\hat{y}_i^s + (1-y_i^s)log(1-\hat{y}_i^s)] \\ & + \sum_{i=1}^{n}[\hat{y}_i^t log\hat{y}_i^s + (1-\hat{y}_i^t)log(1-\hat{y}_i^s)] \end{aligned} \tag{5}$$

where:

M: the set of teacher-student layer pairs (jth layer of student, kth layer of teacher)
y_{ji}^s: output of student's j^{th} layer
y_{ki}^t: output of teacher's k^{th} layer
$\hat{y}^s$: predicted softmax output of student
$\hat{y}^t$: predicted softmax output of teacher

This approach can reduce to the previous one when knowledge from only one of the layers is utilized as opposed to multiple layers [4]. Venkatesan *et al.* claimed that student model when trained with multi-layer KT technique improved the accuracy by 10% when compared to the student model trained without any KT techniques. As this approach transfers the representations from multiple layers of the teacher model, we name this specific KT technique as ***transferring intermediate representations (multiple layers)***.

IV. Architectures

In this study, we consider three different types of student-teacher architectures proposed by the related works. In each of the architectures, the student model is constructed in a way that requires much fewer parameters than the teacher model.

- ***Type I***: The teacher model is VGG16 [12] and the student model is a network that is shorter than the teacher and consists of much fewer parameters (3.2M vs. 8.5M) [4]. Venkatesan *et al.* [4] originally proposed this architecture and applied intermediate-representations (multiple layers) based KT on it.
- ***Type II***: Howard *et al.* [5] originally proposed a lightweight architecture named MobileNet for mobile and embedded applications. We use this architecture and apply KT techniques on it. Here, teacher and student models are MobileNets of different widths. We change the width by tuning the width multiplier parameter present in the MobileNet architecture. In our experiments, we set the width multiplier of the teacher to 1.0 and that of the student to 0.1 [5]. Unlike Type I, the student network is thinner and of the same depth as that of the teacher. In terms of the number of parameters, the student model consists of 1.3M whereas the teacher model consists of 4.2M.
- ***Type III***: The teacher model is Maxout model [13], and the student model is FitNet4 [3]. In this case, the student network is thinner and deeper compared to the teacher network. The number of parameters in the student model is 2.5M, which is about a quarter of the teacher model's

(9M). Romero *et al.* [3] initially proposed this architecture and applied intermediate-representations (single layer) based KT technique on it.

All the related works have only 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 [1], [2], [4]. 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, we analyze the behavior of KT techniques on different architectures.

V. Methodology

In this section, we discuss in detail our methodology for evaluating the different KT techniques described in Section III. All the models were built on TensorFlow [6] version r1.3, and run on a Nvidia 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, the relative performance of KT w.r.t. the baselines should hold on edge devices.

A. Benchmark Datasets

- **CIFAR-10** dataset consists of 60,000 (32X32) RGB natural images from 10 different object classes with 6000 images per class [14]. 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 the total dataset), the testing set consists of 1829 images (20%), and the validation set consists of 1463 images (16%) [15].

CIFAR-10 and Caltech 101 data sets are augmented with random left and right flipping during training. We also normalize both datasets with zero mean and unit standard deviation before feeding the data into the network. This preprocessing adds synthetic data, which exposes the model to additional variations without the cost of collecting more data and thereby improves the models' ability to generalize [3].

B. Training Methodology

In order to apply KT, we passed the same batch of input data (Caltech 101/CIFAR-10) to the teacher and student models. The teacher model did the inference on the input data and predicted the output. The student model was trained by minimizing the loss function formulated with the teacher's output, as described in Section III.

While applying intermediate-representations (single layer) KT technique, we chose layer pairs as mentioned in Table I. Similarly, Table II shows the mapping of layer pairs for intermediate-representations (multiple layers) based KT technique.

TABLE I: 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 II: 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	$1^{st} \rightarrow 1^{st}$, $2^{nd} \rightarrow 2^{nd}$, $3^{rd} \rightarrow 3^{rd}$

While training Type III student network with KT techniques on Caltech 101, we observed out-of-memory issue even for a batch size of 1. Thus, for this setting only, we used a different server equipped with four GPUs (one Nvidia TITAN Xp and three Nvidia TITAN X) to train this student network.

We compared the performance of the student model that was trained with the help of the teacher (named ***dependent student model***), with the following two baselines:

- ***Teacher model*** was used as a baseline to see how much the student represents the state-of-the-art accuracy. The teacher model provides supervision for the student model, so it was trained ahead of the student model.
- ***Independent student model*** was trained independently from scratch without applying any form of knowledge transfer, and was used as a baseline to see how much improvement the KT technique brought to the dependent student.

Both the teacher and independent student models were trained to minimize the cross-entropy loss formulated as follows:

$$\sum_{i=1}^{n} [y_i log \hat{y_i} + (1 - \hat{y_i}) log(1 - y_i)] \tag{6}$$

where:

y: True labels of the dataset
$\hat{y}$: Predicted softmax output of model

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

VI. Evaluation Results

We consider two important performance metrics:

- *Classification accuracy* is the proportion of correctly predicted labels among all the predictions obtained by the network. In Top-1 accuracy, predicted label is counted as correct when the label with the highest probability equals the target label.

TABLE III: Type I architecture

KT Techniques	Caltech 101	CIFAR-10
Baseline (Teacher)	74.12%	77.71%
Baseline (Independent Student)	61.24%	73.31%
Hard Logits	61.27%	**75.19%**
Soft Logits	63.73%	75.01%
Intermediate Rep. (single layer)	**68.60%**	74.98%
Intermediate Rep. (multi-layer)	68.22%	74.47%

TABLE IV: Type II architecture

KT Techniques	Caltech 101	CIFAR-10
Baseline (Teacher)	70.74%	75.64%
Baseline (Independent Student)	60.11%	47.72%
Hard Logits	29.89%	41.09%
Soft Logits	**50.63%**	29.08%
Intermediate Rep. (single layer)	29.68%	**49.12%**
Intermediate Rep. (multi-layer)	10.25%	39.09%

TABLE V: Type III architecture

KT Techniques	CIFAR-10	Caltech 101
Baseline (Teacher)	68.08%	64.04%
Baseline (Independent Student)	73.47%	70.67%
Hard Logits	14.54%	10.24%
Soft Logits	**75.32%**	**74.32%**
Intermediate Rep. (single layer)	68.24%	68.35%
Intermediate Rep. (multi-layer)	72.56%	73.34%

TABLE VI: Numbers of iterations required by the independent and dependent student models (trained with the respective best KT techniques) to reach 90% of their best accuracy

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

- *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. We evaluate convergence time of the network as the total number of iterations required to reach 90% of the Top-1 accuracy.

A. Accuracy

Table III shows the accuracy of the dependent student and the baseline models using the Type I architecture. For Caltech 101, the best result from the dependent student is from intermediate-representations (single layer) based KT, which is 7.36% better than the independent student and only 5.52% less than the teacher. The result from transferring multiple intermediate layers' representations is not the best one but is still 6.98% better than the independent student. We believe that the large improvement in the accuracy is because, in addition to the knowledge from the last layer, the intermediate layer(s) also contributes thereby giving adequate knowledge to the dependent student.

For CIFAR-10, the best dependent student, from using hard-logits-based KT, performs only 1.88% better than the independent student. Since the student model is trained on the CIFAR-10 dataset which has 40,000 images in the training set as opposed to 5853 images in Caltech 101, we believe that it gets sufficient supervision from the dataset, which makes extra supervision from the teacher model ineffective. This observation is also confirmed by the small difference among the different KT techniques.

Table IV shows the accuracy of the dependent student and baseline models using Type II architecture. For Caltech 101, none of the KT techniques improves the accuracy of the dependent student over the independent student. For CIFAR-10, only the dependent student trained using the intermediate-representations (single layer) based KT performs better than the independent student, but only 1.4% better.

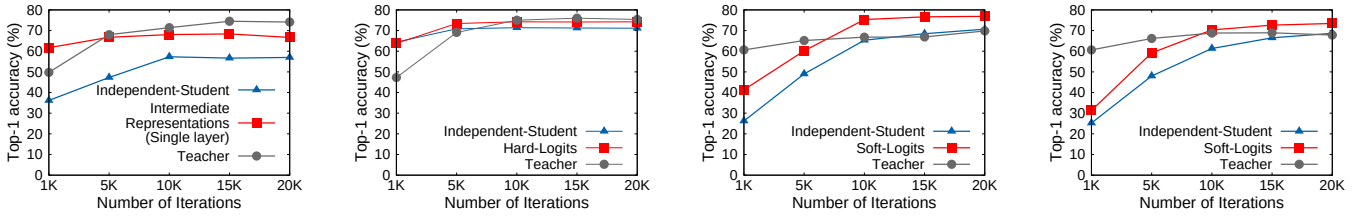
Table V shows the accuracy of the dependent student and the baseline models using Type III architecture. As claimed in Romero *et al.*'s work [3], the student model trained using their proposed intermediate-representations (single layer) based KT technique outperforms the teacher model on both Caltech 101 and CIFAR-10. However, we also noticed that 1) this specific KT technique is not as good as the independent student (5% lower for CIFAR-10, and 2% lower for Caltech 101); 2) it is also not the best KT technique—soft-logits-based KT outperforms it by 6%; and 3) most surprisingly, even the independent student outperforms the teacher. Therefore, we believe that this improvement achieved by the dependent student is due to its deeper network than the teacher, but not because of the use of KT technique. Non-linearity generally increases with the growing depth of a network, which results in learning more complex representations and achieving higher classification accuracy.

Finally, comparing the dependent student model's results across all the three architectures, we observe that 1) the highest improvement is from using intermediate-representations KT on Type I architecture; 2) Type II does not support KT well as the accuracy drops for all the KT techniques; and 3) in many cases, improper KT techniques in fact make the dependent student's accuracy worse than the independent student.

B. Convergence Time

Since the previous results show that KT techniques bring positive improvement only on Type I and Type III architectures, here we study the convergence time of only these two architectures. For each architecture and dataset setting, we consider only the best KT technique (as shown in the previous results) and the baselines.

Table VI shows the number of iterations required by the independent/dependent student (trained with the respective best KT technique) to reach 90% of its best accuracy. Fig. 2 shows how their Top-1 accuracies evolve over time. Note that each iteration of the dependent student does slightly more



(a) Type I & Caltech 101 (b) Type I & CIFAR-10 (c) Type III & Caltech 101 (d) Type III & CIFAR-10

Fig. 2: Accuracy of dependent student and baseline models after every 5K iterations

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 device). 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-representations KT.

VII. Conclusions and Future Work

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. We considered four different KT techniques and three different model architectures and evaluated their performance in terms of both accuracy and convergence time. Our results show that only intermediate-representations KT technique and Type I model achieve significant accuracy improvement (up to 7.36%) for the dependent student model compared to the independent student. The intermediate-representations KT technique is the most promising one 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 the independent student model. The intermediate-representations KT technique also achieves the best speedup in convergence time compared to the other KT techniques.

Based on these results, in our future work, we will study more effective edge-based deep learning along the following possible directions. First, we will further investigate the use of intermediate layers to enable more fine-grained transfer of knowledge. Second, we will consider using the available accelerators on edge devices to improve the speed of learning on devices [16].

Acknowledgment

The authors thank the anonymous reviewers for their helpful comments. This research is sponsored by U.S. National Science Foundation CAREER award CNS-1619653 and award CNS-1562837, CNS-1629888, IIS-1633381, and CMMI-1610282.

References

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

[2] J. Ba and R. Caruana, "Do deep nets really need to be deep?" in *Proceedings of Advances in Neural Information Processing Systems*, 2014, pp. 2654–2662.

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

[4] R. Venkatesan and B. Li, "Diving deeper into mentee networks," *arXiv preprint arXiv:1604.08220*, 2016.

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

[6] M. Abadi, 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," 2015. [Online]. Available: https://www.tensorflow.org/

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

[8] W. Chen, J. Wilson, S. Tyree, K. Weinberger, and Y. Chen, "Compressing neural networks with the hashing trick," in *Proceedings of International Conference on Machine Learning*, 2015, pp. 2285–2294.

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

[10] Y. LeCun, J. S. Denker, and S. A. Solla, "Optimal brain damage," in *Proceedings of Advances in Neural Information Processing Systems*, 1990, pp. 598–605.

[11] S. Srinivas and R. V. Babu, "Data-free parameter pruning for deep neural networks," *arXiv preprint arXiv:1507.06149*, 2015.

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

[13] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Proceedings of Advances in Neural Information Processing Systems*, 2014, pp. 2672–2680.

[14] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," *Master's thesis, Department of Computer Science, University of Toronto*, 2009.

[15] L. Fei-Fei, 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*, vol. 106, no. 1, pp. 59–70, 2007.

[16] S. Biookaghazadeh, M. Zhao, and F. Ren, "Are fpgas suitable for edge computing?" in *Proceedings of USENIX Workshop on Hot Topics in Edge Computing (HotEdge 18)*, 2018.