Knowledge Distillation (KD)
Final Paper

Dev Patel & Omar Obeid
Department of Computer Science - CSC591
North Carolina State University
dmpatel3@ncsu.edu — ofobeid@ncsu.edu

Abstract

Knowledge distillation (KD) is a model compression technique that focuses on transferring knowledge from a large model to a smaller one as a way to have the compressed model have a similar accuracy to the large model. KD can either be response-based, feature-based or relation-based, and training networks with KD can be done either offline, online or by itself. This paper will mainly focus on exploring response-based KD through offline training by using the VGG16, DenseNet121, and ResNet18 models as teachers and compressed versions of those models as their students, with the compressed versions without KD being used as a baseline to compare against the compressed versions with KD.

Background & Motivation

Neural networks tend to be large and require a lot of memory and computational power, which poses a problem for low-power systems such as mobile or embedded devices [3]. This results in a need to compress down model sizes so that they can fit within the memory and power requirements of these smaller-scale platforms. Other techniques have been developed to address this issue, such as weight pruning, weight quantization, and Huffman coding. However, these techniques run into an issue of being too restrictive in practice as a unique implementation must be written for each model family, making it difficult and expensive to apply the techniques across different families [1]. In addition, different architectures can have unique challenges, such as how models with group normalization have increased pruning complexity, further complicating the effectiveness of applying the techniques across different architectures [1].

An alternative solution for model compression that we pursued in this paper is Knowledge Distillation (KD). Rather than training up a large model and then removing from it, KD instead focuses on using a large model to train a smaller one that can be employed on low-power systems. In addition, KD introduces a way to compress models across different families as it can also be implemented across model families, where a model in the ResNet family could be used to train a DenseNet model for example.

What Is Knowledge Distillation (KD)?

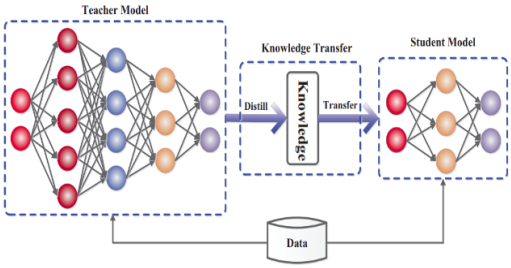

Figure 1: KD Overview Diagram

Knowledge distillation (KD) is a model compression technique that involves transferring learned knowledge from a larger model ("teacher network") to a smaller model ("student network") in order to create a compressed model that can achieve a similar accuracy to the original large model, as depicted in Figure 1 [3]. Unlike other model compression techniques, KD has the benefit of allowing the smaller model to achieve a better accuracy when it is trained with the larger model compared to training itself from scratch.

When knowledge is transferred from the teacher model to the student model, this is mainly done using a soft probability distribution generated in the teacher's output to supervise the training of the student model, which is defined as:

$$\tilde{p}_k^t(x) = \frac{e^{s_k^t(x)/\tau}}{\sum_j e^{s_j^t(x)/\tau}} \quad (1)$$

where $p_k^t(x)$ represents the soft probabilities output from the teacher, and this distribution is controlled by a temperature (T) parameter used with KD that determines

the amount of soft probabilities that the teacher provides to the student [4]. When calculating the training loss for each training iteration with KD, a new knowledge distillation loss from the teacher model is generated and combined with the student's cross-entropy loss to generate the overall training loss per iteration. Because of this, KD also incorporates an alpha (α) parameter to control the impacts of both the cross-entropy loss and distillation loss on the overall training loss, which is calculated as:

$$\mathcal{L} = \alpha \mathcal{L}_{cls} + (1-\alpha)\mathcal{L}_{KD}$$
$$\text{where} \quad \mathcal{L}_{KD} = -\tau^2 \sum_k \vec{p}_k^t(x) \log \vec{p}_k^s(x) \tag{2}$$

where L_{cls} is the cross-entropy loss and L_{KD} is the knowledge distillation loss that is calculated using the soft probability distribution and T value defined in equation 1 [2]. Overall, the T and α parameters used in KD help with determining how much knowledge the teacher transfers to the student and how closely the student mimics the teacher's behavior.

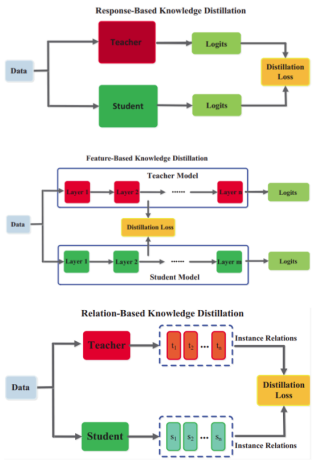

Figure 2: KD Types Diagram

KD has 3 different types of distillation that can be used, which are response-based, feature-based and relation-based. The main idea with each type of KD is that they focus on trying to get the student to learn and mimic a certain part of the teacher, and then calculating and minimizing the distillation loss between the teacher and student at that part in both models, as shown in Figure 2 [5]. With response-based distillation, the part used is the predictions generated in the final output layers of both models, where the prediction loss between the models represents the distillation loss [5]. With feature-based distillation, the part used is the feature activations stored in the intermediate layers of both models, where the feature activation loss between the models represents the distillation loss [5]. Finally, with relation-based distillation, the part used is the feature maps contained in the intermediate layers of both models, where the feature map correlation loss between the models represents the distillation loss [5].

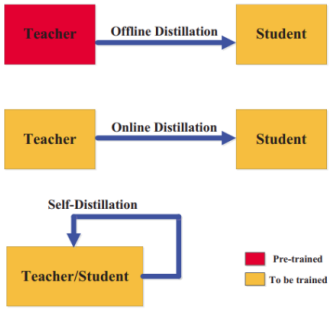

Figure 3: KD Modes Diagram

Furthermore, KD also has 3 different modes for transferring knowledge between models during training, which are offline, online and self distillation. Offline and online distillation are similar in the fact that they both involve using 2 separate models as the teacher and student models and transferring knowledge from the teacher to the student as seen in Figure 3, but offline distillation uses a pre-trained teacher while online distillation trains both the student and teacher at the same time [5]. Self distillation, on the other hand, has 1 model acting as both the teacher and student as seen in Figure 3, and knowledge is transferred from the early stages of the teacher to the later stages of the student [5].

Implementation Setup

For our implementation we aimed to test across a variety of model architectures to observe if the model used impacted its efficacy. The teacher models were VGG16, ResNet18, and DenseNet121 and they were all tested on the CIFAR-10 dataset. These models were chosen because we felt they provided a good enough range of simple to complex networks to properly display the effects of KD. The children nets we chose were the same as the teacher nets, except a 0.5 width multiplier was applied in every convolutional/FC layers, and these nets were called VGG16_half, Resnet18_half, and DenseNet121_half. This was done because we did not want to risk complicating our data by using too many different architectures.

Due to the variety of models implemented and our plan to teach each student model on each teacher model, we went with offline training and response-based KD. Offline training was chosen to reduce the total amount of training time so we would not have to retrain the teacher model each time we wanted to train a student model, and

response-based KD was chosen because of the variety of models implemented. Due to many differences in the internal layout and layering between the different architectures, this was deemed the best approach to take when training the students.

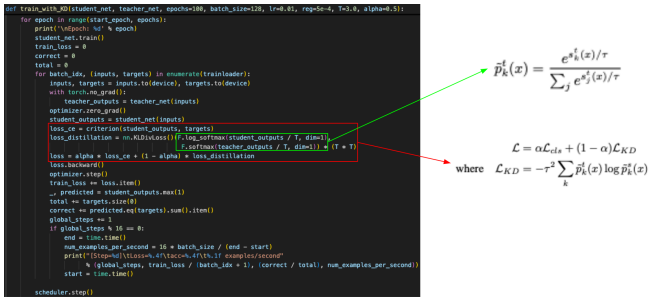

Figure 4: KD Implementation Code Snippet

With our KD implementation, the training that occurs with KD is pretty identical to the normal training implementation that is used for pre-training the teacher models with offline distillation, but the main difference is how the overall training loss with each training iteration is calculated. Normal training only accounts for the cross-entropy loss when finding the overall training loss, but KD training also adds a knowledge distillation loss that's combined with the cross-entropy loss to generate the overall training loss. As depicted in Figure 4, the code in the green box represents the soft probability distribution from the teacher's output as seen in equation 1, and the code in the red box represents the calculation of the overall training loss with KD as seen in equation 2.

All training runs used the same hyperparameters: Epochs = 100, Batch Size = 128, Learning Rate = 0.01, and Regularization = 5e-4. The only differences were with the learning rate scheduler type and momentum values. The VGG16 models used the MultiStepLR scheduler with a momentum of 0.9, while the ResNet18 and DenseNet121 models used the CosineAnnealingLR scheduler with a momentum of 0.95. The KD runs also used the same hyperparameters, with T = 3 and $\alpha = 0.7$. This was done in order to make the impact of KD more apparent rather than being concerned over whether hyperparameters were optimized perfectly.

A total of fifteen training sessions were performed: training the teachers models, training the student models without KD, and training each student model on each teacher model. The student models without KD served as a benchmark with which to compare the KD runs to. Additionally, all these training sessions were performed twice, each on different hardware. The two hardwares used were the NCSU VCL platform using Ubuntu22 with a CUDA GPU, and Google Colab while using a T4 GPU.

Results and Analysis

All runs are evaluated on three different metrics: model size (measured by number of parameters), training time, and accuracy. Model size is compared between the student and teacher models to denote the amount of compression. Training time and accuracy are primarily compared between the student models with and without KD. This is because the aim is to observe the impact that KD has on the student model, however some comparison will be done to the teacher model as well. Ideally, the KD runs will be able to achieve a high compression rate without taking much longer to run than without KD while providing a meaningful accuracy increase.

			Acc (%)		Time	
Model		**Params**	VCL	Colab	VCL	Colab
VGG	**T**	15239872	92.72	93.4	20m 19s	56m 44s
	S	3811680	91.52	92.12	17m 4s	51m 41s
ResNet	**T**	11164352	94.49	94.36	30m 4s	79m 19s
	S	2792800	93.31	93.16	19m 16s	50m 29s
DenseNet	**T**	6880448	88.36	88.30	51m 48s	82m 32s
	S	1725024	86.46	86.48	50m 49s	81m 27s

Table 1: Training Stats without KD. Top shows Teacher values, bottom shows Student values. Left values are from the NCSU VCL runs, right values are from the Google Colab runs

Table 1 highlights the training results of the teacher nets and the student nets without taking KD into account. This is to establish baseline values with which later comparisons will be made. As can be immediately observed, even without KD the student nets' accuracies are already similar to those of the teachers while being significantly smaller and taking much less time to train than the teacher nets.

		Teacher Models		
Student Models		**VGG**	**ResNet**	**DenseNet**
VGG	**Acc**	91.98	91.86	91.55
	Time	17m 57s	19m 45s	25m 29s
	Acc Gain	+0.46	+0.34	+0.03
	Comp	.25	.34	.55
ResNet	**Acc**	93.48	93.61	93.14
	Time	23m 15s	25m 44s	30m 30s
	Acc Gain	+0.17	+0.30	-0.17
	Comp	.18	.25	.4
DenseNet	**Acc**	87.05	86.86	86.79
	Time	53m 49s	53m 44s	60m 55s
	Acc Gain	+0.59	+0.40	+0.33
	Comp	.11	.15	.25

Table 2: Training Stats with KD for the NCSU VCL training runs. From top to bottom, the values in each cell is the Test Accuracy, Training Time, Accuracy Gain, and Compression Rate

		Teacher Models		
Student Models		**VGG**	**ResNet**	**DenseNet**
VGG	**Acc**	91.63	91.87	91.68
	Time	50m 22s	52m 35s	61m 41s
	Acc Gain	-0.49	-0.25	-0.44
	Comp	.25	.34	.55
ResNet	**Acc**	93.60	93.61	93.45
	Time	59m 24s	65m 18s	69m 22s
	Acc Gain	+0.41	+0.42	+0.26
	Comp	.18	.25	.4
DenseNet	**Acc**	86.56	86.34	86.31
	Time	86m 28s	89m 56s	104m 3s
	Acc Gain	+0.08	-0.14	-0.17
	Comp	.11	.15	.25

Table 3: Training Stats with KD for the Google Colab training runs. From top to bottom, the values in each cell is the Test Accuracy, Training Time, Accuracy Gain, and Compression Rate

Tables 2 & 3 showcase the results of training the students nets using the teacher nets through KD. As indicated in Table 2, the accuracy gain with KD through the NCSU VCL hardware ranged from -0.17% to 0.59%, with the average accuracy gain being 0.27%. However, as shown in Table 3, the accuracy gain with KD through the Google Colab hardware ranged from -0.49% to 0.42%, with the average accuracy gain being -0.04%. To add to this, the training times with/without KD were significantly lower when using the NCSU VCL hardware compared to the Google Colab hardware. In general, beneficial performance was more apparent on the NCSU VCL hardware whereas the Google Colab hardware showed more detrimental performance when using KD, possibly indicating that hardware has an impact on how effective KD can be with improving performance.

Despite the differences with the results between the 2 hardwares used, there were some patterns that seemed to be apparent across both hardwares. For example, the training times across all the KD runs were higher than the times training the student models from scratch, with some of them even being higher than when training the teacher models from scratch. Following this, the training times using KD increased in length as the teacher model being used became more complex, such as when training a student on the ResNet18 model when compared to the DenseNet121 teacher model. The longer training times are likely due to the fact that KD has to access both the teacher and student networks separately, meaning extra time is spent having to access another network. This does not even take into account the time spent pre-training the teacher networks to use with KD, meaning that the total KD training time is a lot higher than just training the networks without KD. Additionally, using the VGG16 network as the teacher model tended to produce the best accuracy gain when using KD compared to the other teacher networks. The accuracy gain when using KD also seemed to increase as the compression rate decreased (the size between the student and teacher models increased).

While KD does show potential with improving the performance of compressed models compared to just training them from scratch, the inconsistent mix of accuracy gains/losses across the KD runs indicates that KD might struggle with consistently improving the performance of compressed models. Additionally, the accuracy gains themselves weren't very significant since they were only small increases in accuracy, and it usually resulted in significantly higher training times compared to when KD wasn't used. Overall, KD can help improve the performance of compressed models even if it is inconsistent and often results in longer training times as a result, especially when using response-based KD and offline distillation.

Limitations & Future Work

Despite our KD implementation showing slight improvements in the test accuracy for the student models compared to when they were trained from scratch, the use of response-based KD and offline distillation did likely

limit how much the student models could have improved with KD. For instance, response-based KD only relies on the final output layer and doesn't take into account the intermediate layers when training the student from the teacher, so the student can't likely gain as much important info from the teacher as necessary to improve itself [5].

To go with this, offline distillation can be ineffective if the teacher is too complex for the student to learn and mimic its behavior, even if the teacher and student have similar architectures, which may explain the fairly low accuracy gains with our KD implementation [6]. The issue of having a big size/complexity gap between the teacher and student models is a general issue with implementing KD, which possibly suggests a tradeoff between how much the student model can be compressed and the student model's performance. In addition, our implementation was overall pretty expensive in terms of memory consumption and execution time likely due to using offline distillation, as indicated with the longer times of the student model runs with KD compared to the student model runs without KD.

Finally, our results also seemed to indicate that hardware has an impact on the effectiveness of KD as the NCSU VCL hardware tended to show better improvements when training models with KD compared to the Google Colab hardware, which showed more detrimental performance when training models with KD than beneficial.

Even though KD does have its limitations on how effective it is at improving the compressed model's performance, there are some strategies that could be used or are being explored to help improve KD. For instance, using feature-based KD or relation-based KD instead would allow the student to learn more parts of the teacher, which would likely help the student better mimic and learn the teacher's behavior and performance. Additionally, using online or self distillation would allow both the teacher and student models to be trained at the same time or only require one model to be used for KD training, which would likely help significantly reduce the execution time and/or memory footprint when using KD.

To address the hardware limitations with KD, future work could possibly explore combining KD with other hardware-friendly model compression techniques like quantization and pruning to help make KD more efficient across different hardware. To possibly help address the issue of having a big size/complexity difference between the teacher and student models, combining different KD types together like response-based KD and relation-based KD could also be explored in future work.

Strategies that are currently being explored to improve KD include early-stopped KD (ESKD), which involves stopping KD early in the training process once it reaches a point where the learning plateaus, and focusing only on the cross-entropy loss for the rest of the training [2], and teacher assistant KD (TAKD). TAKD involves the addition of a teacher assistant model with a size between the teacher and student model sizes and using KD to transfer knowledge from the teacher model to the teacher assistant model before using KD again to transfer knowledge from the teacher assistant model to the student model [6]. Overall, future work could focus on trying to optimize the tradeoff between the size and performance of student models based on the teacher model being used along with finding methods for mitigating the hardware limitations that come with using KD.

References

[1] Beyer, Lucas, et al. "Knowledge Distillation: A Good Teacher Is Patient and Consistent." *Arxiv.Org*, Cornell University, 21 June 2022, arxiv.org/abs/2106.05237.

[2] Cho, Jang Hyun, and Bharath Hariharan. "On the Efficacy of Knowledge Distillation." *CVF Open Access*, Computer Vision Foundation, 2019, openaccess.thecvf.com/content_ICCV_2019/html/Cho_On_the_Efficacy_of_Knowledge_Distillation_ICCV_2019_paper.html.

[3] Gou, Jianping, et al. *Knowledge Distillation: A Survey*, 20 May 2021, arxiv.org/pdf/2006.05525.

[4] Hinton, Geoffrey, et al. "Distilling the Knowledge in a Neural Network." *Arxiv.Org*, Cornell University, 9 Mar. 2015, arxiv.org/abs/1503.02531.

[5] Lendave, Vijaysinh. "What Is Knowledge Distillation in Deep Learning." *AIM*, 1 Aug. 2024, analyticsindiamag.com/topics/what-is-knowledge-distillation-in-deep-learning/.

[6] Mirzadeh, Seyed Iman, et al. "Improved Knowledge Distillation via Teacher Assistant." *Google Research*, Washington State University, 3 Apr. 2020, storage.googleapis.com/gweb-research2023-media/pubtools/6185.pdf.