Divide and Distill (DnD): Efficient Knowledge Distillation with Higher Accuracy and Lower Time Cost

Shiu-Hong Kao, Jierun Chen, Shueng Han Gary Chan

The Hong Kong University of Science and Technology (HKUST)

skao@connect.ust.hk, jierunchen@ust.hk, gchan@cse.ust.hk

November 22, 2022

Abstract

Knowledge distillation, a teacher-student system, has been recognized as one of the most significant approaches for model compression, which aims to generate fast and compact neural networks. However, previous work has shown that the time cost to preserve teacher performance has been extremely expensive. In this work, we aim to propose an approach that can achieve high performance with a low time cost. We first show that stagewise consistency is a critical factor for the convergence speed of the training process. Then we propose a novel technique called *DnD: Divide and Distill* to transfer the knowledge from the teacher in a short period of time. In particular, we divide the student network into different stages and simultaneously transfer the knowledge from the respective stage of the teacher. We conduct two experiments on the datasets Imagewoof and CIFAR100 to empirically show that DnD achieves higher accuracy than previous works, given a small number of training epochs. An extended framework named *Progressive DnD* is also proposed to further improve the performance of DnD.

1 Introduction

Deep learning has gained massive success in computer vision tasks such as image classification. While large and computationally expensive models are mainly used to obtain higher accuracy, these models are not compact with edge devices with smaller memory, such as cellphones and embedded systems, due to the massive memory cost. Hence, several works (Buciluǎ, Caruana, and Niculescu-Mizil (2006); Hinton, Vinyals, Dean, et al. (2015)) about Knowledge Distillation (KD), one of the most significant approaches for model compression, have been proposed to effectively compress huge models into cheaper models while preserving their performance at the same time. In specific, knowledge distillation regards a huge model as a teacher network and a compact model as a student network, then transferring the teacher's knowledge to the student. Previous work Beyer et al. (2022) asserted that a good teacher should be patient during the distillation process, which implies that the time cost to perfectly distill knowledge to the student is extremely expensive. For example, an experiment of

the dataset ImageNet was conducted in the research paper with 9600 epochs, which can take months to complete. In this work, we focus on transferring knowledge in a more efficient way.

There are two factors affecting the long processing time of knowledge distillation, one of which is convergence speed. We define it as the required number of training epochs to achieve accuracy close to the final performance. We empirically show that the critical failure of considering stagewise consistency leads to the slow convergence of the previous work. The other factor is computational complexity, which is determined by two operations: feed-forward pass and back-propagation (LeCun, Touresky, Hinton, and Sejnowski (1988)). Traditional offline knowledge distillation requires the feed-forward operation to pass through the entire student and the entire teacher, and the back-propagation to update the entire student network. In regard to an efficient knowledge distillation framework, this work aims to provide an approach that can achieve high accuracy within a small number of training epochs.

We propose a new knowledge distillation framework called ***DnD: Divide and Distill***, based on our findings of the reasons why previous works suffer from slow convergence speed, to efficiently transfer the knowledge from the teacher to the student. We analyze and assert that DnD has a smaller computational complexity than the state of the art (Chen et al. (2022)), and empirically show that our approach achieves higher accuracy with a small number of training epochs by conducting the experiments distilling ResNet34 to ResNet18 with datasets Imagewoof and CIFAR100 in 60, 100, and 200 epochs. We also show the applications of DnD on data efficiency, hoping to provide a baseline for futural works.

2 Related Works

Several works about knowledge distillation (Gou, Yu, Maybank, and Tao (2021); Tang et al. (2020)) have been proposed in recent years. Beyer et al. (2022) claimed that a good teacher should be patient, which means a huge dataset and a long training process are both mandatory to better transfer the knowledge from teacher to student and preserve the teacher's performance. These two factors raise the distillation cost tremendously.

There are two ways to transfer knowledge from the teacher to the student, one of which includes the knowledge distillation terms directly to the traditional training process (Hinton et al. (2015); Srinivas and Fleuret (2018); Zagoruyko and Komodakis (2016)), while the other method transfer knowledge in a pre-training stage and fine-tune the student network after knowledge distillation (Romero et al. (2014); Yim, Joo, Bae, and Kim (2017)). To effectively improve the distillation efficiency, we suggest that the latter approach increases the time cost, and a fast knowledge distillation framework should follow the first scheme.

In general, the approaches to reduce the distillation cost can be grouped into two categories:

- **Data-free Knowledge Distillation (DFKD)** uses a meta-learning framework to train the compact model only using the synthetic data (Fang et al. (2022); Lopes, Fenu, and Starner (2017)). This method reduces the training time as the required dataset is significantly small, but the time cost to generate synthetic data is remaining expensive.

- **Data Efficient Knowledge Distillation** reduces the training data during the distillation process (Kulkarni, Panchi, Raparthy, and Chiddarwar (2019)). One critical issue is that the number of training iterations of this method is increased, so the total distillation time is not essentially decreased.

As the previous works did not essentially reduce the required time for knowledge distillation, we argue that a novel approach is crucial for future development in this area.

3 Methodology

In this section, we will explain and compare the methodology of vanilla knowledge distillation and our proposed approach *DnD*. Let b imply the mini-batch size in the experiments, $\mathbf{x} = \{x_1, x_2, ..., x_b\}$ imply the b sample images, and $\mathbf{y} = \{y_1, y_2, ..., y_b\}$ imply the corresponding labels in the mini-batch. Suppose that the teacher network $f^{\mathcal{T}}$ and the student network $f^{\mathcal{S}}$ can be divided into k stages, denoted as $f_1^{\mathcal{T}}, f_2^{\mathcal{T}}, ..., f_n^{\mathcal{T}}$, and $f_1^{\mathcal{S}}, f_2^{\mathcal{S}}, ..., f_n^{\mathcal{S}}$ respectively, and the teacher classifier and student classifier are denoted as $\mathcal{C}^{\mathcal{T}}$ and $\mathcal{C}^{\mathcal{S}}$ respectively. We regard the neural network as a function composed of several subfunctions, i.e. layers. Also, for function f and g, we define $f \circ g(\cdot)$ as the function composition $f(g(\cdot))$.

3.1 Vanilla KD

In Vanilla KD, only logit knowledge is transferred to the student network. A common belief is that soft labels from the teacher regularize the student network by providing "dark knowledge" (Allen-Zhu and Li (2020)). The loss function $\mathcal{L}(\mathbf{x})$ is composed of two terms $\mathcal{L}_{CE}$ and $\mathcal{L}_{KL}$ with the following representation:

$$\begin{aligned} \mathcal{L}_{CE} &= \mathcal{L}_{CE}(f_n^{\mathcal{S}} \circ f_{n-1}^{\mathcal{S}} \circ ... \circ f_1^{\mathcal{S}}(\mathbf{x}), \mathbf{y}), \\ \mathcal{L}_{KL} &= \mathcal{L}_{KL}(f_n^{\mathcal{S}} \circ f_{n-1}^{\mathcal{S}} \circ ... \circ f_1^{\mathcal{S}}(\mathbf{x}), f_n^{\mathcal{T}} \circ f_{n-1}^{\mathcal{T}} \circ ... \circ f_1^{\mathcal{T}}(\mathbf{x})) \end{aligned} \tag{1}$$

, where $\mathcal{L}_{CE}(\cdot, \cdot)$ and $\mathcal{L}_{KL}(\cdot, \cdot)$ imply the cross-entropy loss and Kullback–Leibler divergence between two inputs respectively.

Then the knowledge distillation process is processed by a combination of these two elements with a hyper-parameter weight. In specific,

$$\mathcal{L}(\mathbf{x}) = (1 - \alpha)\mathcal{L}_{CE} + \alpha\mathcal{L}_{KD} \tag{2}$$

, where α is a hyperparameter for the weight of KD loss.

In this view, the feedforward complexity of Vanilla KD is decided by the entire teacher-student architecture, and the backward takes place in the whole student network.

3.2 DnD

Regarding a fast framework for knowledge distillation, there are two major issues to be considered: computational cost and convergence speed. The computational cost of the training scheme

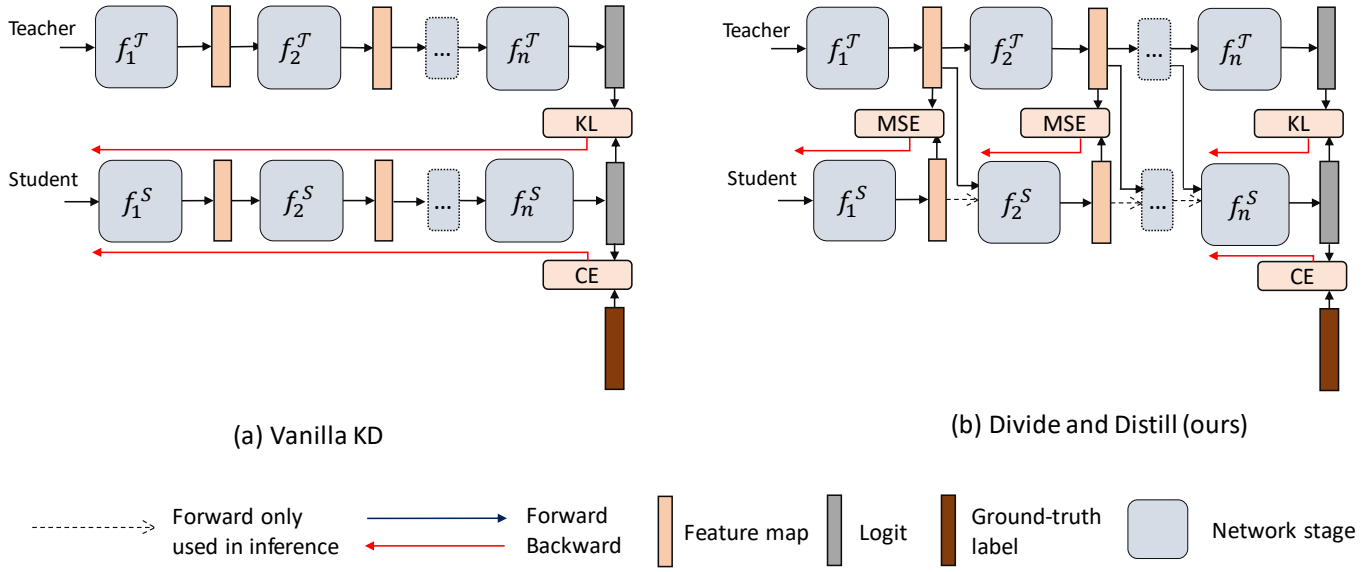


Figure 1: Comparison of different KD techniques: (a) Vanilla KD takes the loss term from the labels and trains the student classifier from scratch. The feedforward and backpropagation are both passed through the entire network. (b) *Divide and Distill (DnD)* divides both the networks into n stages, and each pair of the stages from the teacher and student contains its own loss term. These loss terms are independent of each other and can be combined in practice. The feedforward operations navigate the whole system as it does in Vanilla KD, and the backpropagation is composed of n short sequences.

contains two parts: forward-pass and backward-pass. We argue that the backward-pass over the entire student required by vanilla knowledge distillation degrades the efficiency due to the long sequence. In Vanilla KD, the gradient of the parameters in shallow layers waits and can only be computed after the system finishes computing those in deep layers. By dividing the backward-pass into several stages and distilling each stage simultaneously, *DnD* aims to reduce the waiting time during back-propagation. On the other hand, convergence speed defines a characteristic of the training scheme. Our criteria to optimize the convergence speed can be described as the number of epochs the distillation process needs for good accuracy. Considering the results we found in section **??**, stage-wise consistency is a critical factor for the convergence speed of knowledge distillation. In specific, informative parameters in the shallower and deeper neighbor layers accelerate the convergence speed of those parameters in a specific layer. In this regard, we are convinced that it is effective to improve the convergence speed by transferring knowledge between each sub-network in the teacher and the student.

We propose an approach called *Divide and Distill (DnD)*, as presented in 1, by first dividing the networks into multiple stages and simultaneously training these stages. The loss function of *DnD* contains $n+1$ terms $\mathcal{L}_1, \mathcal{L}_2, ..., \mathcal{L}_n$, and $\mathcal{L}_{KL}$ as follows:

$$\mathcal{L}_k = \begin{cases} \mathcal{L}_{mse}(f_1^{\mathcal{T}}(\mathbf{x}), f_1^{\mathcal{S}}(\mathbf{x})) & , \quad \text{if} \quad k=1 \\ \mathcal{L}_{mse}(f_k^{\mathcal{T}} \circ f_{k-1}^{\mathcal{T}} \circ ... \circ f_1^{\mathcal{T}}(\mathbf{x}), f_k^{\mathcal{S}} \circ f_{k-1}^{\mathcal{T}} \circ ... \circ f_1^{\mathcal{T}}(\mathbf{x})) & , \quad \text{if} \quad k=2,3,...,n \end{cases} \tag{3}$$

$$\mathcal{L}_{KL} = \mathcal{L}_{KL}(f_k^{\mathcal{T}} \circ f_{k-1}^{\mathcal{T}} \circ ... \circ f_1^{\mathcal{S}}(\mathbf{x}), \mathbf{y}) \tag{4}$$

, where $\mathcal{L}_{mse}(\cdot,\cdot)$ and $\mathcal{L}_{KL}(\cdot,\cdot)$ implies the mean squared error loss and the Kullback–Leibler divergence between two inputs respectively.

In particular, we train the stage k of the student to mimic the stage k of the teacher with the loss term $\mathcal{L}_k$. Then these loss terms can be combined due to the independence, the generalized loss function is represented as:

$$\mathcal{L}(\mathbf{x}) = \mathcal{L}_{KL} + \sum_{k=1}^{n} \mathcal{L}_k \tag{5}$$

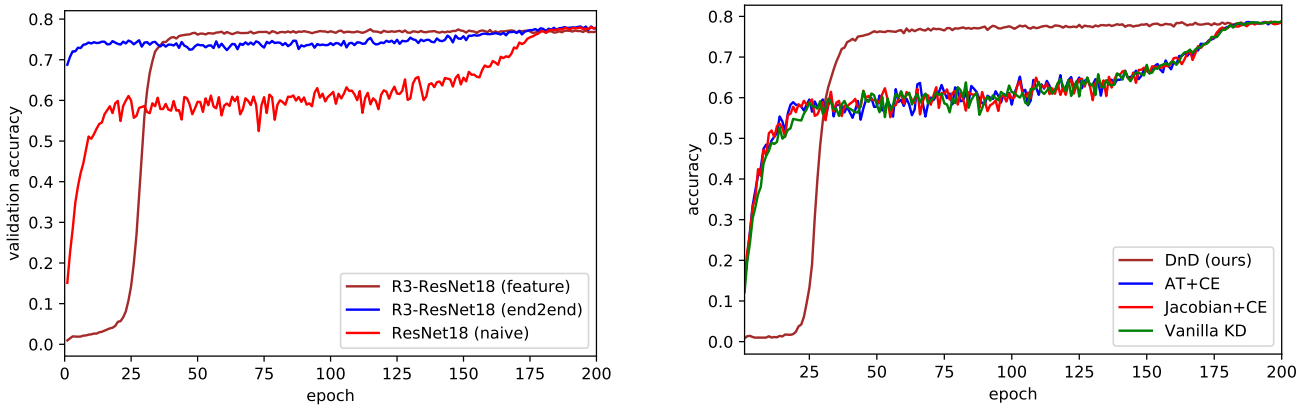
As shown in Figure 1, the computational cost of the feedforward pass of *DnD* is equivalent to Vanilla KD, determined by the entire student network and the entire teacher network. However, instead of back-propagating the whole student model in Vanilla KD, the backpropagation of *DnD* is divided into several short blocks and updated simultaneously.

4 Experiments

In this section, we first conduct an experiment related to stagewise consistency in 4.1 to illustrate its importance for the convergence speed in knowledge distillation. Then we demonstrate the significantly fast convergence speed of *DnD* in 4.2 by conducting experiments on Imagewoof and CIFAR100 with the same number of training epochs, yet we observe that training with the ordinary *DnD* in the distillation process can overfit on the dataset and converges to slightly worse accuracy. We, therefore, extend our concept and propose an adjusted scheme called *Progressive DnD*. We also reduce the training epochs to present the effect of *Progressive DnD* and show that our proposed approach significantly outperforms the previous works when a small number of epochs is given; in specific, DnD and Progressive DnD have demonstrated their success in the experiments of 60 and 100 epochs. Finally, we present the high data efficiency of DnD in 4.3, which is the research focus on Kulkarni et al. (2019)'s study and has a direct impact on the training time cost.

4.1 Stagewise Consistency

We argue one factor degrading the efficiency of vanilla knowledge distillation is stagewise consistency. According to Beyer et al. (2022)'s study, the teacher should be consistent during the training process, which highlights the importance of the consistency between the student input and the teacher input. Regarding the stages as the subnetworks of the entire model, we suggest that vanilla KD is consistent only in the first subnetwork of the model. Consider the shallow subnetworks as the encoding layers of the samples. The deeper subnetworks of the student and the teacher in vanilla KD receive the same samples with different encodings as the inputs, while all the stages are consistent in the DnD framework since we always pass the shallow layers of the teacher to be the encoding layers. In other words, during the training, not well-trained shallow layers tend to pass less informative features to their deeper layers. Similarly, we argue that the loss term conditioned on poor deeper layers tends to backward-pass less informative gradient information to the shallower layers. We call this effect as ***stagewise consistency***.

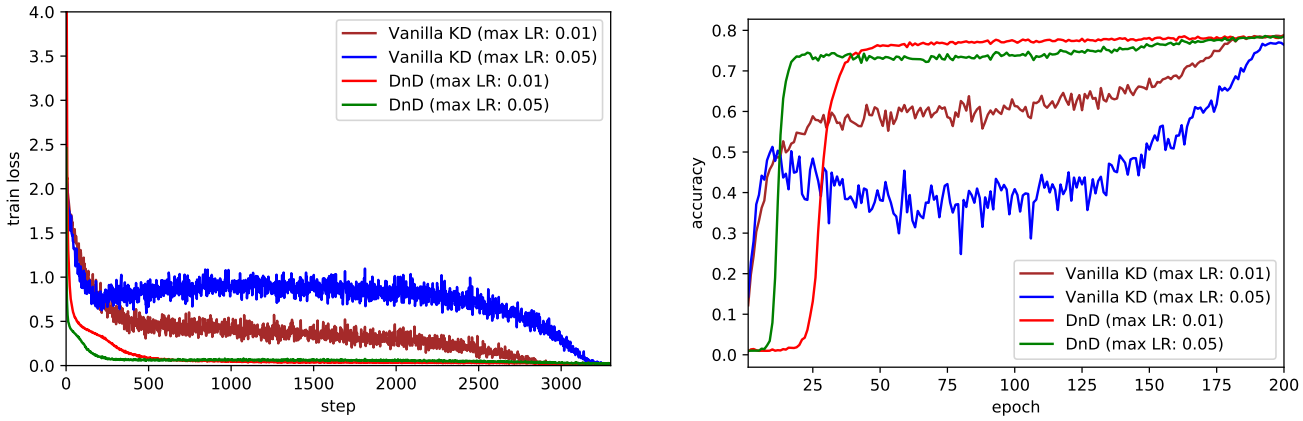


(a) Stagewise consistency

(b) Convergence speed

Figure 2: (a)Stagewise consistency: *ResNet18 (naive)* indicates the naive training scheme for ResNet18, while training *R3-ResNet18 (feature)* involves inputs with pre-trained shallow layers and excludes the impact of deep layers, and training *R3-ResNet18 (end2end)* contains inputs with pre-trained shallow layers and loss term conditioned on pre-trained deep layers. This experiment shows that pre-trained neighbor layers tend to send more informative input or gradient to the trained layer, while poor neighbors degrade the efficiency of the training process. (b)Convergence speed: Four experiments with different knowledge distillation methods, including DnD, are conducted to demonstrate the fast convergence of our proposed framework. The experiments are all done with the dataset CIFAR100 and use ResNet18 as the student and pre-trained ResNet34 as the teacher. Compared to the previous works, DnD converges significantly faster and achieves comparable accuracy at the end of the training process.

An experiment about stagewise consistency is conducted to examine this hypothesis. To begin with, we naively train a ResNet18 with the Cifar-100 dataset for 200 epochs by dividing the model into five stages. (The dividing and training strategies are provided in the Appendix **??**.) Then we froze its parameters except those in its third stage and randomized the parameters in the third stage again. Denote this adjusted network as *R3-ResNet18*. In particular, *R3-ResNet18* is a network with well-trained layers in each stage except the third, which is composed of random parameters. In the next step, we conduct two experiments on *R3-ResNet18* corresponding to two different loss terms. The first one is the end-to-end cross-entropy loss between *R3-ResNet18*'s logits and ground-truth labels, denoted as *R3-ResNet18 (end2end)*, and the other is the mean squared error loss between output features of the third stages of *R3-ResNet18* and ResNet18 respectively, denoted as *R3-ResNet18 (feature)*. The training results are shown in Figure 2(a), where *ResNet18 (naive)* implies the naive training scheme for ResNet18. We observe that *R3-ResNet18* shows a significant improvement in convergence speed compared to ResNet18, which supports our hypothesis about the stagewise consistency. In detail, this experiment shows the maturity of shallower stages and deeper stages matters in terms of training a particular stage.



(a) Training loss

(b) Validation accuracy

Figure 3: Learning Rate Tuning: Experiments with CIFAR100 were conducted to compare the learning rate preferences of DnD and Vanilla KD. Due to the distinction between mean squared error and cross-entropy loss, DnD generally has a smaller value of training loss. This helps the model adapt to a high learning rate and speed up the training process.

4.2 Fast Knowledge Distillation

Convergence Speed We explore the convergence speed of DnD by comparing it with vanilla KD and two other KD schemes, Attention Transfer (AT) (Zagoruyko and Komodakis (2016)) and Jacobian matching (Czarnecki, Osindero, Jaderberg, Swirszcz, and Pascanu (2017); Srinivas and Fleuret (2018)). We use ResNet18 as the student and ResNet34 as the teacher and then train each framework in 200 epochs with the same strategy (OneCycle with a maximal learning rate of 0.1). In Fig 2(b), we observe that DnD converges significantly faster than the previous works. However, these experiments cannot essentially reduce the training time and achieve fast knowledge distillation, as we maintain the number of epochs for different KD approaches. Instead, it inspires us to reduce the number of epochs and to compare the degraded performances of these approaches.

Learning Rate Tuning Another factor related to training efficiency is the learning rate, which is usually regarded as a hyperparameter. While a large learning rate tends to speed up the training process, a potential problem of loss oscillation is likely to happen and leads to a failure of convergence. Therefore, the learning rate has a critical effect on training efficiency. We here compare DnD and Vanilla KD with two learning rate strategies; in specific, the experiments involve the OneCycle learning rate scheduler (Smith and Topin (2019)) with different values of maximal learning rate. The result indicates that DnD generally adapts to a high learning rate better than Vanilla KD, as the former can converge to high accuracy rapidly without serious oscillation and help us obtain a comparable model while abridging the training duration. We argue that the distinction comes from the selection of the loss function since most parameters in DnD are trained with a mean square error loss, which has a smaller altitude than the cross-entropy loss. Hence, it's reasonable for us to choose a larger learning rate for DnD in the experiments in the following sections.

Cross-stage Knowledge One potential drawback of DnD is the ignorance of the cross-stage knowledge in DnD. Instead of dividing the network into k stages and fixing the architecture, we provide an adjusted version of DnD, named ***Progressive DnD***, to dynamically reduce the number

KD approach	60 epochs		100 epochs		200 epochs	
	acc. (%)	time (mins)	acc. (%)	time (mins)	acc. (%)	time (mins)
Vanilla KD	79.03 ± 0.38	15	80.45 ± 0.33	25	$\mathbf{81.65 \pm 0.74}$	49
AT+CE	78.83 ± 0.04	15	80.29 ± 0.46	25	81.23 ± 0.43	50
Jacobian+CE	78.49 ± 0.39	44	80.46 ± 0.38	73	81.48 ± 0.26	146
DnD (ours)	81.45 ± 0.40	**14**	$\mathbf{81.22 \pm 0.48}$	**24**	81.29 ± 0.30	**47**
Progressive DnD (ours)	$\mathbf{82.74 \pm 0.19}$	15	81.04 ± 1.14	**24**	81.09 ± 0.26	49

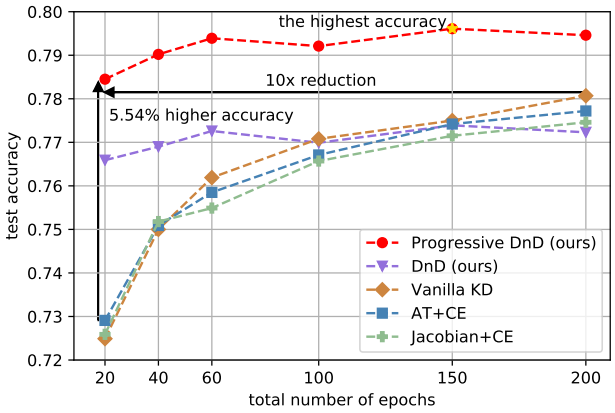
Table 1: Imagewoof experiments (Teacher: ResNet34 with an accuracy of 82.66%; Student: ResNet18)

KD approach	60 epochs		100 epochs		200 epochs	
	acc. (%)	time (mins)	acc. (%)	time (mins)	acc. (%)	time (mins)
Vanilla KD	76.19 ± 0.28	37	77.08 ± 0.14	61	78.07 ± 0.12	122
AT+CE	75.85 ± 0.46	38	76.71 ± 0.45	63	77.72 ± 0.23	126
Jacobian+CE	75.49 ± 0.15	120	76.57 ± 0.22	198	77.46 ± 0.26	395
DnD (ours)	77.26 ± 0.14	**33**	76.99 ± 0.20	**55**	77.23 ± 0.10	**113**
Progressive DnD (ours)	$\mathbf{79.39 \pm 0.16}$	35	$\mathbf{79.21 \pm 0.34}$	58	$\mathbf{79.46 \pm 0.09}$	116

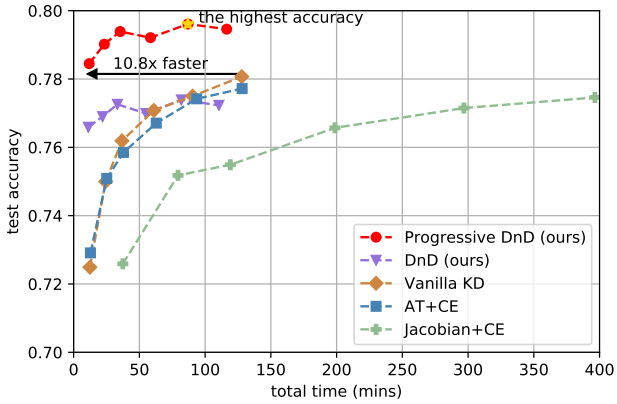
Table 2: CIFAR100 experiments (Teacher: ResNet34 with an accuracy of 78.39%; Student: ResNet18)

of stages in the architecture by combining the neighboring stages together as a new stage. For example, DnD in Table 1 and 2 indicates the five-stage architecture of the teacher-student system, while *Progressive DnD* starts the distillation with 5 stages, switching to 3 stages after training with one-third of the number of epochs, and contains only 2 stages during the last one-third of the training epochs. (A detailed dividing strategy can be found in the Appendix A.) One of the interesting issues is that we end the training with a two-stage architecture instead of a one-stage architecture because we notice the model overfits after merging the convolutional layers and the fully-connected layers together. Hence, it is redundant to merge the architecture into one stage, which increases the time cost of the process as well. Also, we observe that it is important to have a warm-up learning rate strategy every time we change the architecture; otherwise, a sudden and severe accuracy drop may occur.

Imagewoof Experiments with the dataset Imagewoof_160 (Howard (n.d.)) were conducted to compare our proposed frameworks with different KD approaches, including Vanilla KD (Hinton et al. (2015)), Attention Transfer (Zagoruyko and Komodakis (2016)), and Jacobian Matching (Srinivas and Fleuret (2018)). We used the OneCycleLR learning rate scheduler (Smith and Topin (2019)) with a maximal learning rate of 0.1 in these traditional methods. In DnD, OneCycle was also adopted but the maximal learning rate was set to 0.5 due to its special learning rate preference. In Progressive DnD, we introduced the CyclicLR learning rate scheduler (Smith (2017)) with the mode of *triangular2* and the maximal learning rate of 0.5. In fact, we also conducted the experiments of Vanilla KD, Attention Transfer, and Jacobian Matching with a learning rate of 0.5 but obtained worse performances than those with 0.1.



(a) Test accuracy-Total number of epochs



(b) Test accuracy-Total time cost

Figure 4: CIFAR100 test accuracy: the experiments are conducted in 20, 40, 60, 100, 150, and 200 epochs. While testing the accuracy, we use the best model with the highest validation accuracy for each experiment. Progressive DnD achieves comparable accuracy with Vanilla KD with only 10% of the number of epochs, and it speeds up the KD process by 10.8 times.

In addition to the fast convergence of DnD, the actual efficiency of knowledge distillation depends on the performance of experiments with less training time. To illustrate this effect, the experiments were implemented in 60, 100, and 200 epochs respectively. Each experiment was conducted three times with different random seeds, and the mean and standard deviation are shown in Table 1. According to the result, ResNet18 trained with DnD and Progressive DnD in 60 epochs outperforms those trained with Vanilla KD, Attention Transfer, or Jacobian Matching by more than 2 and 3 percent respectively. Also, due to the fast convergence, there is no apparent improvement for DnD and Progressive DnD when the number of training epochs is set larger than 60 epochs. In addition, the experiments of 200 epochs are done to compare the effectiveness of a long training period, where we observe that Vanilla KD, Attention Transfer, and Jacobian Matching need such a large training cost because of their slower convergence speed. We claim that our approaches can achieve significantly high performance with only a small amount of training time cost.

CIFAR100 We also conducted other experiments on the dataset CIFAR100 (Krizhevsky, Hinton, et al. (2009)) to demonstrate the effect of Progressive DnD. Similar to the experiments of Imagewoof, each experiment here was conducted three times with different random seeds. As shown in Table 2, DnD and Progressive DnD also improve the accuracy of the previous methods by at least 1 and 3 percent respectively. These experiments demonstrate the effectiveness of DnD and Progressive DnD and provide a clearer view of how the fast convergence speed can be applied to the reduction of training costs. Figure 4 gives out a more comprehensive picture of this phenomenon. By further reducing the number of epochs to 40 and even 20, DnD and Progressive DnD show a significantly high accuracy when the given time cost is low. In particular, Progressive DnD with 20 epochs outperforms previous works with 200 epochs. As the processing time for one epoch of Progressive DnD is less than that of Vanilla KD, the total time cost can be reduced by at least 90%.

CIFAR100 subset size	20% data	40% data	60% data	80% data	100% data
KD approach	Accuracy (%)				
Vanilla KD	55.65	67.64	72.88	75.91	78.12
AT+CE	56.53	67.22	71.99	75.31	77.99
Jacobian+CE	55.47	67.54	71.87	75.26	77.75
DnD (ours)	**76.26**	77.23	77.02	77.35	77.26
Progressive DnD (ours)	76.10	**78.50**	**79.18**	**79.63**	**79.41**

Table 3: Data efficiency (Teacher: ResNet34 with an accuracy of 78.39%; Student: ResNet18)

4.3 Data Efficiency

Traditionally, knowledge distillation relies on large training datasets. However, the cost to store and process large datasets is extremely expensive. Various methods to reduce required training data were published, for example, dataset condensation (Nguyen, Novak, Xiao, and Lee (2021); Wang, Zhu, Torralba, and Efros (2018); Zhao and Bilen (2021); Zhao, Mopuri, and Bilen (2020)) and corset selection (Agarwal, Har-Peled, and Varadarajan (2004); Aljundi, Lin, Goujaud, and Bengio (2019); Feldman, Schmidt, and Sohler (2020)). These methods focused on generating or choosing appropriate samples from the dataset. On the other hand, Pena, Nilsson, Björkegren, and Tegnér (2007) and Kulkarni et al. (2019) focus on designing a training approach with high data efficiency to adapt to the few-data environments. We follow this idea, aiming to transfer knowledge from the teacher using only a small amount of data, where the dataset is a random subset of the original one.

Considering a scenario with a much smaller dataset, we conduct an experiment to examine the data efficiency of DnD. The same teacher-student setting as it is in section 4 was introduced, and the dataset was randomly sampled from the entire CIFAR100. All the experiments were trained in 200 epochs. Table 3 shows the promising result of our proposed KD approaches, where Progressive DnD achieves high data efficiency by outperforming Vanilla KD with only 40% of the training data. It also shows that DnD and Progressive DnD generally obtain high accuracy with fewer data. We argue that the reason is similar to the work proposed by Kulkarni et al. (2019), as there are fewer parameters to be trained for the back-propagation of each loss term, but our approach is more computationally cheap and less time-consuming.

5 Conclusion

DnD has gained great success in improving the efficiency of knowledge distillation. We hope this study can be beneficial for futural knowledge distillation approaches. Also, we here discuss a way to improve the DnD training scheme in practice, which is related to memory efficiency. In specific, this method aims to use less memory allocation to achieve the same performance. Since we divide the encoders into stages, the DnD framework is composed of shallower networks to update for back-propagation. We expect to save the hidden feature maps from the teacher in one feedforward pass and update each of the student stages separately. This framework doesn't increase the computational

complexity of the complete training process but is expected to consume less memory. By increasing memory efficiency, we are able to use a larger batch size to speed up the training process.

References

Agarwal, P. K., Har-Peled, S., & Varadarajan, K. R. (2004). Approximating extent measures of points. *Journal of the ACM (JACM)*, *51*(4), 606–635.

Aljundi, R., Lin, M., Goujaud, B., & Bengio, Y. (2019). Gradient based sample selection for online continual learning. *Advances in neural information processing systems*, *32*.

Allen-Zhu, Z., & Li, Y. (2020). Towards understanding ensemble, knowledge distillation and self-distillation in deep learning. *arXiv preprint arXiv:2012.09816*.

Beyer, L., Zhai, X., Royer, A., Markeeva, L., Anil, R., & Kolesnikov, A. (2022). Knowledge distillation: A good teacher is patient and consistent. In *Proceedings of the ieee/cvf conference on computer vision and pattern recognition* (pp. 10925–10934).

Buciluǎ, C., Caruana, R., & Niculescu-Mizil, A. (2006). Model compression. In *Proceedings of the 12th acm sigkdd international conference on knowledge discovery and data mining* (pp. 535–541).

Chen, D., Mei, J.-P., Zhang, H., Wang, C., Feng, Y., & Chen, C. (2022). Knowledge distillation with the reused teacher classifier. In *Proceedings of the ieee/cvf conference on computer vision and pattern recognition* (pp. 11933–11942).

Czarnecki, W. M., Osindero, S., Jaderberg, M., Swirszcz, G., & Pascanu, R. (2017). Sobolev training for neural networks. *Advances in Neural Information Processing Systems*, *30*.

Fang, G., Mo, K., Wang, X., Song, J., Bei, S., Zhang, H., & Song, M. (2022). Up to 100x faster data-free knowledge distillation. In *Proceedings of the aaai conference on artificial intelligence* (Vol. 36, pp. 6597–6604).

Feldman, D., Schmidt, M., & Sohler, C. (2020). Turning big data into tiny data: Constant-size coresets for k-means, pca, and projective clustering. *SIAM Journal on Computing*, *49*(3), 601–657.

Gou, J., Yu, B., Maybank, S. J., & Tao, D. (2021). Knowledge distillation: A survey. *International Journal of Computer Vision*, *129*(6), 1789–1819.

Hinton, G., Vinyals, O., Dean, J., et al. (2015). Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, *2*(7).

Howard, J. (n.d.). *Imagewang.* Retrieved from https://github.com/fastai/imagenette/

Krizhevsky, A., Hinton, G., et al. (2009). Learning multiple layers of features from tiny images.

Kulkarni, A., Panchi, N., Raparthy, S. C., & Chiddarwar, S. (2019). Data efficient stagewise knowledge distillation. *arXiv preprint arXiv:1911.06786*.

LeCun, Y., Touresky, D., Hinton, G., & Sejnowski, T. (1988). A theoretical framework for back-propagation. In *Proceedings of the 1988 connectionist models summer school* (Vol. 1, pp. 21–28).

Lopes, R. G., Fenu, S., & Starner, T. (2017). Data-free knowledge distillation for deep neural networks. *arXiv preprint arXiv:1710.07535*.

Nguyen, T., Novak, R., Xiao, L., & Lee, J. (2021). Dataset distillation with infinitely wide convolutional networks. *Advances in Neural Information Processing Systems*, *34*, 5186–5198.

Pena, J. M., Nilsson, R., Björkegren, J., & Tegnér, J. (2007). Towards scalable and data efficient learning of markov boundaries. *International Journal of Approximate Reasoning*, *45*(2), 211–232.

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

Smith, L. N. (2017). Cyclical learning rates for training neural networks. In *2017 ieee winter conference on applications of computer vision (wacv)* (pp. 464–472).

Smith, L. N., & Topin, N. (2019). Super-convergence: Very fast training of neural networks using large learning rates. In *Artificial intelligence and machine learning for multi-domain operations applications* (Vol. 11006, pp. 369–386).

Srinivas, S., & Fleuret, F. (2018). Knowledge transfer with jacobian matching. In *International conference on machine learning* (pp. 4723–4731).

Tang, J., Shivanna, R., Zhao, Z., Lin, D., Singh, A., Chi, E. H., & Jain, S. (2020). Understanding and improving knowledge distillation. *arXiv preprint arXiv:2002.03532*.

Wang, T., Zhu, J.-Y., Torralba, A., & Efros, A. A. (2018). Dataset distillation. *arXiv preprint arXiv:1811.10959*.

Yim, J., Joo, D., Bae, J., & Kim, J. (2017). A gift from knowledge distillation: Fast optimization, network minimization and transfer learning. In *Proceedings of the ieee conference on computer vision and pattern recognition* (pp. 4133–4141).

Zagoruyko, S., & Komodakis, N. (2016). Paying more attention to attention: Improving the performance of convolutional neural networks via attention transfer. *arXiv preprint arXiv:1612.03928*.

Zhao, B., & Bilen, H. (2021). Dataset condensation with differentiable siamese augmentation. In *International conference on machine learning* (pp. 12674–12685).

Zhao, B., Mopuri, K. R., & Bilen, H. (2020). Dataset condensation with gradient matching. *arXiv preprint arXiv:2006.05929*.

A Appendix: Division Method

In DnD, the teacher and student networks are divided into some stages to speed up the distillation process. The division method of the experiments in this paper is presented in Table 4. One thing to notice is that conv1 is replaced with a convolutional layer with kernel size 3, stride 1, and padding 1, and the max pooling layer is replaced with the identity layer in the CIFAR100 experiment.

As an extended framework of DnD, Progressive DnD dynamically reduces the number of stages to maintain the cross-stage knowledge in the network. In this paper, we divide the training period into 3 phases, each of which contains one-third of the number of epochs. In the first phase, the networks

are divided into five stages as presented in Table 4. This is equivalent to the DnD architecture. In the second phase, stage1 and stage2 are combined together as a new stage, and so are stage3 and stage4. Finally, in the last one-third of the training epochs, stage1, stage2, stage3, and stage4 are combined as a whole stage.

stage division	layer name	ResNet18	ResNet34
stage1	conv1	7×7, 64, stride 2	
	layer1	3×3 max pool, stride 2	
		$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 3$
stage2	layer2	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 4$
stage3	layer3	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 6$
stage4	layer4	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 3$
stage5		average pool, fully-connected layer, softmax	

Table 4: Division method for DnD

B Appendix: Training Details

This appendix specifies the training details of the Imagewoof and CIFAR100 experiments in 4.2. The details can be found in 5, where n implies the number of training epochs; b implies the batch size, and D implies the size of the training dataset. Also, b was set to 64 in the experiments of Imagewoof and 128 in the experiments of CIFAR100. Unless specified in 5, other hyper-parameters followed the default setting in PyTorch.

KD method		DnD	Progressive DnD	Other approaches
optimizer	type	SGD		
	momentum	0.9		
	weight_decay	$5e-4$		
	lr	0.05		0.01
scheduler	type	OneCycleLR	CyclicLR	OneCycleLR
	epochs	n		
	steps_per_epoch	D/b	-	D/b
	max_lr	10×lr		
	base_lr	-	$3e-3$×lr	-
	step_size_up	-	$D \times n/6$	-
	mode	-	triangular2	-

Table 5: Training details