

Towards Mitigating Architecture Overfitting on Distilled Datasets

Xuyang Zhong, Chen Liu*

Department of Computer Science, City University of Hong Kong, Hong Kong SAR, China
xuyang.zhong@my.cityu.edu.hk, chen.liu@cityu.edu.hk

Abstract—Dataset distillation methods have demonstrated remarkable performance for neural networks trained with very limited training data. However, a significant challenge arises in the form of *architecture overfitting*: the distilled training dataset synthesized by a specific network architecture (i.e., training network) generates poor performance when trained by other network architectures (i.e., test networks), especially when the test networks have a larger capacity than the training network. This paper introduces a series of approaches to mitigate this issue. Among them, DropPath renders the large model to be an implicit ensemble of its sub-networks, and knowledge distillation ensures each sub-network acts similarly to the small but well-performing teacher network. These methods, characterized by their smoothing effects, significantly mitigate architecture overfitting. We conduct extensive experiments to demonstrate the effectiveness and generality of our methods. Particularly, across various scenarios involving different tasks and different sizes of distilled data, our approaches significantly mitigate architecture overfitting. Furthermore, our approaches achieve comparable or even superior performance when the test network is larger than the training network.

Index Terms—Dataset distillation. Overfitting. Efficient learning. Neural network architecture.

I. INTRODUCTION

Deep learning has achieved tremendous success in various applications [1, 2], but training a powerful deep neural network requires massive training data [3, 4]. To accelerate training, one possible way is to construct a new but smaller training set that preserves most of the information of the original larger set. In this regard, we can use *coreset* [5, 6] to sample a subset of the original training set or *dataset distillation* [7, 8] to synthesize a small training set. Compared with *coreset*, dataset distillation is demonstrated to achieve much better performance when the amount of data is extremely small [6, 9]. Furthermore, dataset distillation is shown to benefit various applications, such as continual learning [8, 9, 10, 11], neural architecture search [8, 11], and privacy preservation [12, 13]. Therefore, in this work, we focus on dataset distillation to compress the training set.

In the dataset distillation framework, the small training set, which is also called the *distilled dataset*, is learned by using a neural network, which we call *training network*, to extract the most important information from the original training set. Existing data distillation methods are based on various techniques, including meta-learning [7, 14, 15, 16, 17, 18]

and data matching [8, 9, 11, 19, 20, 21, 22]. These methods are then evaluated by the test accuracy of another neural network, which we call *test network*, trained on the distilled dataset. In summary, in the context of dataset distillation, the training network serves as the model utilized for constructing the distilled dataset, while the test network is employed to showcase the performance achievable through the distilled dataset.

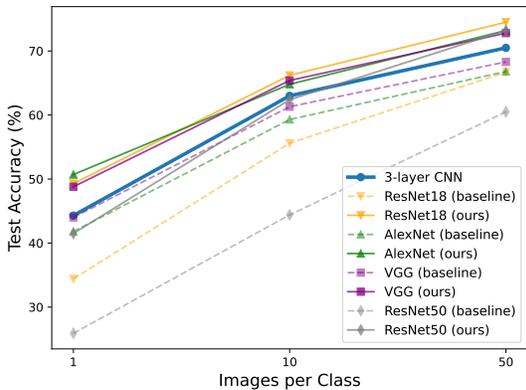
Despite efficiency, dataset distillation methods generally suffer from *architecture overfitting* [9, 11, 17, 18, 20]. That is, the performance of the test network trained on the distilled dataset degrades significantly when it has a different network architecture from the training network. Moreover, the performance deteriorates further when there is a larger difference between the training and test networks in terms of depth and topological structure. Specifically, due to high computational complexity and optimization challenges in dataset distillation, the training networks are usually shallow networks, such as 3-layer convolutional neural networks (CNN) as used in Zhou et al. [18], Cazenavette et al. [20]. However, such shallow networks are rarely employed in practical applications due to their limited representation power. Consequently, we posit that the architecture overfitting seriously undermines the practicality of distilled datasets in real-world scenarios.

Our analysis in this work indicates that the performance gap between different network architectures is larger in the case of training on the distilled dataset than in the case of training on the subset of the original training set. In addition, compared with methods compressing the training set by subset selection, dataset distillation achieves better performance when using the same amount of training instances and is thus more popular in downstream applications [9, 11, 13]. Therefore, we mainly focus on dataset distillation, in which the effectiveness of the proposed method can be better revealed.

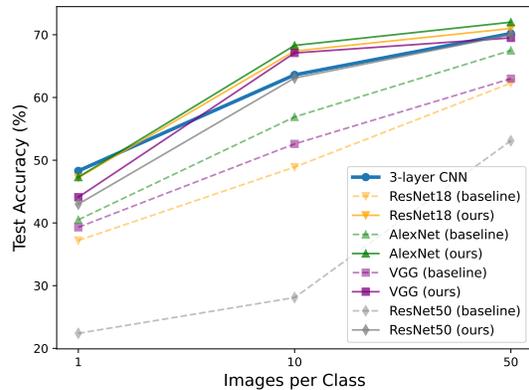
In this work, we demonstrate that the architecture overfitting issue on distilled datasets can be mitigated by a better architecture design and training scheme of test networks on the distilled dataset. Firstly, we combine DropPath with knowledge distillation from a small teacher network. Specifically, DropPath renders the large model to be an implicit ensemble of its sub-networks, and knowledge distillation ensures each sub-network acts similarly to the small but well-performing teacher network. As a result, the large models could outperform small teacher models on distilled datasets. Additionally, we propose a series of approaches, including three-phase DropPath keep rate, improved shortcut connection, periodical learning rates,

* denotes the correspondence author.

Manuscript received February 2024; revised August 2024.



(a) When we use FRePo [18] to construct the distilled dataset.



(b) When we use MTT [20] to construct the distilled dataset.

Fig. 1. Effectiveness of our method on different architectures, different dataset distillation methods, and different images per class (IPCs) on CIFAR10. We use a 3-layer CNN as the training network, so it performs the best among various architectures in baselines (dashed lines). Our methods (solid lines) can significantly narrow down the performance gap between the 3-layer CNN and other architectures.

a better optimizer and a stronger augmentation scheme, to further boost the performance. These methods share a common characteristic of smoothing the optimization problem from different aspects. Notably, our proposed methods are also generic: we conduct comprehensive experiments on different network architectures, different numbers of instances per class (IPC), different dataset distillation methods and different datasets to demonstrate the effectiveness of our methods. Figure 1 above demonstrates the performance of our proposed methods in various scenarios. It is clear that our methods greatly mitigate architecture overfitting and make large networks trained on distilled datasets achieve better performance in most cases. As a result, the utility and transferability of distilled datasets in practice is markedly enhanced even without modifying the dataset distillation algorithm. In addition to dataset distillation, our methods can also improve the performance of training on a small real dataset, including those constructed by coresets. Although some tasks, like synthetic-to-real generalization [23] and few-shot learning [24], are also classical problems, customizing our method for them is out of the scope of this work, because we focus on training large networks on small datasets from scratch. We leave these tasks as future works.

We summarize the contributions of this paper as follows:

- 1) We propose a series of approaches to mitigate architecture overfitting on distilled datasets. Among them, Drop-Path renders the large model to be an implicit ensemble of its sub-networks, and knowledge distillation ensures each sub-network acts similarly to the small teacher network. These methods share a common characteristic of smoothing the optimization problem. Our proposed methods are plug-and-play and applicable to different model architectures and training schemes.
- 2) We conduct extensive experiments to demonstrate that our method significantly mitigates architecture overfitting across different network architectures, different dataset distillation approaches, different numbers of instances per class (IPC), and different datasets.
- 3) Moreover, our method generally improves the perfor-

mance of deep networks trained on limited real data. As a result, large networks outperform small networks on various amounts of training data, even when there are only 100 training samples.

II. RELATED WORKS

Dataset Distillation: The goal of dataset distillation is to learn a smaller set of training samples called *distilled dataset* that preserves essential information of the original large dataset so that models trained on this small dataset have similar performance to those trained on the original large dataset.

Existing dataset distillation approaches are based on either meta-learning or data matching [25]. The former category includes backpropagation through time (BPTT) approach [7, 14, 15] and kernel ridge regression (KRR) approach [16, 17, 18, 26, 27]; the latter category includes gradient matching [11, 19], trajectory matching [20, 21, 28, 29, 30, 31, 32, 33], and distribution matching [9, 22, 34, 35, 36, 37, 38, 39]. In addition, some works [40, 41, 42, 43, 44, 45, 46, 47, 48] leverage better optimization schemes to improve the performance of dataset distillation. However, these methods are shown to suffer from severe *architecture overfitting*: the significant performance degradation when the architecture of the training network and the test network are different. Recently, some factorization methods [49, 50, 51, 52, 53, 54, 55], which learn synthetic datasets by optimizing their factorized features and corresponding decoders, greatly improve the cross-architecture transferability. However, the instance per class (IPC), which indicates the size of the distilled dataset, used in these methods is much larger than that of meta-learning and data matching approaches, which greatly cancels out the advantages of dataset distillation. To better fit the motivation of dataset distillation, we only consider small IPCs (1, 10 and 50) in this work, so the factorization methods are not included for comparison.

Model Ensemble: Model ensemble aims to integrate multiple models to improve the generalization performance. Popular ensemble methods for classification models include bagging

[56], AdaBoost [57], random forest [58], random subspace [59], and gradient boosting [60]. However, these methods require training several models and thus are computationally expensive. By contrast, DropOut [61] trains the model only once but stochastically masks its intermediate feature maps during training. At each training iteration with DropOut, only part of the model parameters are updated, which forms a sub-network of the model. In this regard, DropOut enables implicit model ensembles of different sub-networks to improve the generalization performance. Similar to DropOut, DropPath [62] also implicitly ensembles sub-networks but it blocks a whole layer rather than masking some feature maps. Therefore, it is applicable to network architectures with multiple branches, such as ResNet [63], otherwise, the model output will be zero if a layer of a single branch network is dropped. By contrast, we propose a DropPath variant in this work which is generic, applicable to single-branch networks and effective in mitigating architecture overfitting.

Knowledge Distillation: Knowledge distillation [64] aims to compress a well-trained large model (i.e., teacher model) into a smaller and more efficient model (i.e., student model) with comparable performance. The standard knowledge distillation [64] is also known as offline distillation since the teacher model is fixed when training the student model. Online distillation [65, 66] is proposed to further improve the performance of the student model, especially when a large-capacity high-performance teacher model is not available. In online distillation, both the teacher model and the student model are updated simultaneously. In most cases, knowledge distillation methods use large models as the teachers and small models as the students, which is based on the fact that larger models typically have better performance. However, in the context of dataset distillation, a smaller test network with the same architecture as the training network can achieve a better performance than a larger one on the distilled dataset, so we use the small model as the teacher and the large model as the student in this work. In this way, the performance of large models trained on distilled datasets can be boosted significantly.

We show in the following sections that combining DropPath and knowledge distillation, architecture overfitting on distilled datasets can be almost overcome.

III. METHODS

In this section, we introduce the approaches that are effective in mitigating architecture overfitting on distilled datasets. Our methods are motivated by the intuition that the large model can act as an implicit ensemble of small models [61, 62]. First, we propose a DropPath variant, which implicitly ensemble sub-networks of models and is different from vanilla DropPath [62] in that the proposed DropPath variant is also applicable to single-branch architectures. Correspondingly, we optimize the shortcut connections of ResNet-like architecture to accommodate DropPath better. Second, we use knowledge distillation [64] as a form of regularization to ensure each sub-network induced by DropPath acts similarly to the teacher network. In contrast to traditional knowledge

distillation approaches [64, 65], the teacher model is smaller than the student model in our cases. Finally, we adopt a periodical learning rate scheduler, a gradient symbol-based optimizer [67], and a stronger data augmentation scheme to improve the performance further.

A. DropPath with Three-Phase Keep Rate

Similar to DropOut [61], DropPath [62], a.k.a., stochastic depth, was proposed to improve generalization. While DropOut masks some entries of feature maps, DropPath randomly prunes the entire branch in a multi-branch architecture. To obtain a deterministic model for evaluation, DropPath is deactivated during inference. To ensure the expectation of the feature maps to be consistent for training and inference, we scale the output of feature maps after DropPath during training. Mathematically, DropPath works as follows:

$$\text{DropPath}(\mathbf{x}) = \frac{m}{p} \cdot \mathbf{x}, \quad m = \text{Bernoulli}(p). \quad (1)$$

where $p \in [0, 1]$ denotes the keep rate, $m = \text{Bernoulli}(p) \in \{0, 1\}$ outputs 1 with probability p and 0 with probability $1-p$. The scaling factor $1/p$ is used to ensure the expectation of the feature maps remains unchanged after DropPath. The detailed derivation is in Appendix C. Figure 2 (a) illustrates how DropPath is integrated into networks. It effectively decreases the model complexity during training and can force the model to learn more generalizable representations using fewer layers. Same as DropOut, any network trained with DropPath can be regarded as an ensemble of its subnetworks [68], which has been proven to improve generalization [56, 57, 58, 59, 60]. Note that, DropOut masks part of the feature maps and effectively decreases the network width; by contrast, DropPath removes a branch and thus decreases the effective network depth. In the context of dataset distillation, the test network is deeper than the training network, so we can decrease the effective depth of the test network by DropPath. This approach implicitly bridges the architecture disparity between the training and test networks. Consequently, we anticipate that DropPath will mitigate the problem of architecture overfitting on distilled datasets.

Three-Phase Keep Rate: The keep rate p is the key parameter that controls the effective depth of model architecture when using DropPath. Since the mask $m = \text{Bernoulli}(p)$, the effective depth gets smaller as p decreases. In the early phase of training, the model is underfitting, stochastic architecture brings optimization challenges for training the model, so we turn off DropPath by setting the keep rate $p = 1$ in the first few epochs to ensure that the network learns meaningful representations. We then gradually decrease p to decrease the effective depth and thus to decrease the architecture disparity between the effective test network and the training network until the value of p reaches the predefined minimum value after several epochs. In the final phase of training, we decrease the architecture stochasticity by increasing the value of p to a higher value to ensure good training convergence. In the experiments, we shrink the keep rate every few epochs.

The pseudo-code is demonstrated in Algorithm 1. Unless specified, we set $\gamma = 0.1$, $p_{\min} = 0.5$, $p_{\text{final}} = 0.8$, $T = 500$,

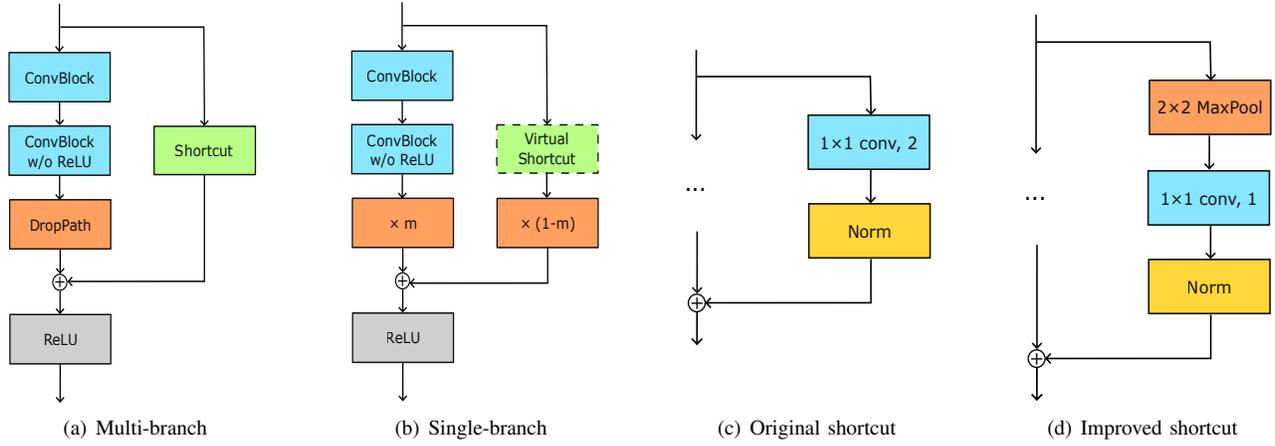


Fig. 2. **(a)** The DropPath used for multi-branch residual blocks during training, it does not block the shortcut path. **(b)** The DropPath used for single-branch networks during training. Here, $m = \text{Bernoulli}(p) \in \{0, 1\}$, $p \in [0, 1]$ denotes the keep rate. Only when the main path is pruned ($m = 0$), the virtual shortcut is activated, and vice versa. DropPath is always deactivated, i.e., $p = 1$, during inference. **(c)** The original architecture of a shortcut connection to downsample feature maps, which consists of a 1×1 convolution layer with the stride of 2 and a normalization layer. **(d)** The improved architecture of a shortcut connection to downsample feature maps, which is a sequence of a 2×2 max pooling layer, a 1×1 convolution layer with the stride of 1 and a normalization layer.

Algorithm 1 DropPath with Three-Phase Keep Rate

- 1: **Input:** the data: \mathbf{x} ; current epoch index: i ; decaying factor: $0 < \gamma < 1$; minimum keep rate: p_{\min} ; final keep rate: p_{final} ; period of decay: T ; warmup period: W ; stabilization epoch: S .
 - 2: **if** $i < W$ **then**
 - 3: $p \leftarrow 1$
 - 4: **else if** $i < S$ **then**
 - 5: $p \leftarrow \max(p_{\min}, 1 - \gamma \cdot \text{ceil}((i - W)/T))$ {ceil function returns the smallest integer bigger than the input}
 - 6: **else**
 - 7: $p \leftarrow p_{\text{final}}$
 - 8: **end if**
 - 9: **if** is training **then**
 - 10: $m \leftarrow \text{Bernoulli}(p)$ {Bernoulli distribution}
 - 11: $\mathbf{y} \leftarrow \frac{m}{p} \cdot \mathbf{x}$
 - 12: **else**
 - 13: $\mathbf{y} \leftarrow \mathbf{x}$
 - 14: **end if**
 - 15: **Output:** \mathbf{y}
-

$W = 500$, $S = 3000$ in the experiments. The corresponding curve of the dynamic keep rate is shown in Figure ?? of Appendix B.

Generalize to Single-Branch Networks: DropPath prunes the entire branch, so it is not applicable to single-branch networks, such as VGG [69]. This is because we need to ensure the input and the output of the network are always connected, otherwise, we will obtain a trivial constant model. By contrast, in the case of multi-branch networks such as ResNet, we prune the main path of a residual block stochastically, while the shortcut connections are always kept.

To improve the performance of single-branch networks, we propose a variant of DropPath. As illustrated in Figure 2(b), we add a virtual shortcut connection between two layers, such

as two consecutive convolutional layers in VGG, to form a “pseudo-residual” block. This structure is similar to a real residual block, however, since we are training a single-branch architecture instead of a real ResNet, the virtual shortcut connection is only used when the main path is pruned by DropPath during training. That is to say when the main path is not pruned, the virtual shortcut connection is removed so that we are still training a single-branch network. Correspondingly, the virtual shortcut connection is discarded during inference. It should be noted that the feature is not scaled in virtual shortcut connection. The detailed derivation is also deferred to Appendix C.

Improved Shortcut Connection: In the original ResNet [63], if one residual block’s input shape is the same as its output shape, the shortcut connection is just an identity function, otherwise a 1×1 convolution layer of a stride larger than one, which may be followed by a normalization layer as shown in Figure 2(c), is adopted in the shortcut connection to transform the input’s shape to match the output’s. In the latter case, the resolution of the feature maps is divided by the stride. For example, if the stride is 2, the top left entry in each 2×2 area of the input feature map is sampled, whereas the rest 3 entities of the same area are directly dropped.

This naive subsampling strategy will cause dramatic information loss when we use DropPath. Specifically, if DropPath prunes the main path as in Figure 2 (a), the shortcut connection will dominate the output of the residual block. In this regard, the naive subsampling strategy may corrupt or degrade the quality of the features, since it always picks a fixed entry of a grid. To tackle this issue, we replace the original shortcut connect with a 2×2 max pooling followed by a 1×1 convolutional layer with the stride of 1. This improved structure will preserve the most important information after pooling instead of the one from a fixed entry. Figure 2 (c) and (d) show the comparison between the original and improved shortcut connections when the shapes of input and output are different.

B. Knowledge Distillation from Small Teacher Model

Given sufficient training data, large models usually perform better than small models due to their larger representation capability. Knowledge distillation aims to compress a well-trained large model (i.e., teacher model) into a smaller model (i.e., student model) without compromising too much performance. The basic idea behind knowledge distillation is to distill the knowledge from a teacher model into a student model by forcing the student’s predictions (or internal activations) to match those of the teacher [70]. Specifically, we can use Kullback-Leibler (KL) divergence \mathcal{L}_{KL} with temperature [64] to match the predictions of student and teacher models. Then, we can combine the KL divergence as the regularization term in addition to the classification loss. Mathematically, the overall loss with knowledge distillation is:

$$\mathcal{L}(\mathbf{y}_s, \mathbf{y}_t, y) = \alpha \cdot \tau^2 \cdot \mathcal{L}_{KL}(\mathbf{y}_s, \mathbf{y}_t) + (1 - \alpha) \cdot \mathcal{L}_{CE}(\mathbf{y}_s, y) \quad (2)$$

where τ denotes the temperature factor, and $\alpha \in (0, 1)$ denotes the weight factor to balance the KL divergence \mathcal{L}_{KL} and the cross-entropy loss \mathcal{L}_{CE} . The output logits of the student model and teacher model are denoted by \mathbf{y}_s and \mathbf{y}_t , respectively. y denotes the target.

When training on distalled dataset, small models perform better than large ones, since small models are employed as the training network to construct distilled dataset. As a result, we adopt the small training network as the teacher model \mathbf{y}_t and the large test network as the student model \mathbf{y}_s . The computational overhead in knowledge distillation mainly arises from calculating \mathbf{y}_t , i.e., the output of the teacher model. In this case, the computational overhead is negligible because evaluating on the small teacher model is much more efficient than on the large student model.

C. Training and Data Augmentation

Besides aforementioned methods, we use the following methods to further improve the performance.

Periodical Learning Rate: Because of the three-phase stepwise scheduler for the keep rate p , we expect the network to jump out of the current local minima, and tries to search for a better one when p changes. Inspired by [71], we use a cosine annealing curve with warmup to adjust the learning rate, and we periodically reset it when p changes. Formally, the learning rate lr_i in the i -th epoch is calculated as follows:

$$\text{lr}_i = \begin{cases} \lambda_i \cdot \frac{\text{mod}(i, t)}{T_{\text{warm}}} \cdot \text{lr}_{\text{max}}, & \text{if } \text{mod}(i, t) \leq T_{\text{warm}}, \\ 0.5\lambda_i(1 + \cos(\pi \frac{\text{mod}(i, t) - T_{\text{warm}}}{T_{\text{max}} - T_{\text{warm}}})) \cdot \text{lr}_{\text{max}}, & \text{otherwise.} \end{cases} \quad (3)$$

where T is the decay period of the keep rate p of DropPath, S is the stabilization epoch. $t = T$ when $i < S$, otherwise $t = S$. $\lambda_i = \lambda^{\lfloor \min(i, S) / T \rfloor}$ where λ is a base decaying factor, and $\lfloor \cdot \rfloor$ denotes the floor function. lr_{max} denotes the maximum learning rate, $\text{mod}(x, y)$ denotes the remainder of x/y . The maximum iterations of the cosine annealing function and the number of warmup epochs are denoted by T_{max} and T_{warm} , respectively. Figure ?? of Appendix B shows an example of how the learning rate changes.

Better Optimizer: Lion [67] is a gradient symbol-based optimizer. It has faster convergence speed and is capable of finding better local minima for ResNets. Thus, we use Lion as the default optimizer in our experiments.

Stronger Augmentation: The data augmentation strategy used in MTT [20] samples a single augmentation operation from a pool to augment the input image. However, we observe that sampling more operations will better diversify the model’s inputs and thus improve the performance, especially when IPC is small. For convenience, when sampling k operations, we call this strategy k -fold augmentation. Empirically, we use 2-fold augmentation when IPC is 10 or 50 and 4-fold augmentation when IPC is 1.

In summary, our proposed methods share a common characteristic of smoothing the optimization problem that can improve generalization: **(a)** in terms of architecture, DropPath smooths the predictions by forming an implicit ensemble of sub-networks; **(b)** knowledge distillation smooths the objective function by introducing the predictions of teacher models as soft labels; **(c)** better optimizer is capable of finding flatter local minima; **(d)** stronger data augmentation smooth the loss landscape in the sample space [72, 73].

IV. EXPERIMENTS

In this section, we evaluate our method on different dataset distillation algorithms, different numbers of instances per class (IPC), different datasets and different network architectures. Our methods are shown effective in mitigating architecture overfitting in these settings and generic to improve the performance on limited real data. In addition, we plot the Hessian eigenvalues and visualize the landscape of different models to corroborate the smoothing effect of the proposed methods. Ultimately, we conduct extensive ablation studies for analysis. Implementation details are deferred to Appendix A.

A. Mitigate Architecture Overfitting in Datasets Distillation

TABLE I
EXPERIMENTAL SETTINGS. DP DENOTES DROPPath WITH THREE-PHASE KEEP RATE, KD DENOTES KNOWLEDGE DISTILLATION. BESIDES, THE MISCELLANEOUS (MISC.) INCLUDES THE METHODS IN SECTION III-C.

Method	DP	KD	Misc.
Baseline	✗	✗	✗
w/o DP&KD	✗	✗	✓
w/o DP	✗	✓	✓
w/o KD	✓	✗	✓
Full	✓	✓	✓

We first evaluate our method on three representative dataset distillation (DD) algorithms, i.e., neural Feature Regression with Pooling (FRePo) [18], Matching Training Trajectories (MTT) [20] and Difficulty-Aligned Trajectory Matching (DATM) [29]. Furthermore, we test several ablations of our methods, the names and the settings of each ablation are elaborated in Table I.

We comprehensively evaluate the performance of these methods under various settings, including different numbers

TABLE II

TEST ACCURACIES OF MODELS TRAINED ON THE DISTILLED DATA OF **CIFAR10** AND **CIFAR100** [74] WITH DIFFERENT IPCs. 3-LAYER CNN IS THE ARCHITECTURE USED FOR DATA DISTILLATION AND IS THE TEACHER MODEL OF KNOWLEDGE DISTILLATION. THE RESULTS IN THE BRACKET INDICATE THE GAPS FROM THE BASELINE PERFORMANCE OF 3-LAYER CNN. NOTE THAT FOR IPC=100/500, THE TEACHER MODEL OF RESNET50 IS RESNET18 W/O DP&KD. THE RESULTS IN BOLD ARE THE BEST RESULTS AMONG DIFFERENT SETTINGS. NOTE THAT DP AND KD ARE NOT APPLICABLE FOR 3-LAYER CNN, SO WE DO NOT HAVE THE TEST ACCURACY OF 3-LAYER CNN IN THESE SETTINGS.

(a) CIFAR10								(b) CIFAR100							
DD	IPC	Methods	3-layer CNN	ResNet18	AlexNet	VGG11	ResNet50	DD	IPC	Methods	3-layer CNN	ResNet18	AlexNet	VGG11	ResNet50
FRePo [18]	1	Baseline	44.3	34.4 (-9.9)	41.8 (-2.5)	44.0 (-0.3)	25.9 (-18.4)	1	1	Baseline	26.2	18.7 (-7.5)	22.9 (-3.3)	22.6 (-3.6)	13.5 (-12.7)
		w/o DP&KD	44.8 (+0.5)	35.6 (-8.7)	47.4 (+3.1)	41.5 (-2.8)	30.3 (-14.0)			w/o DP&KD	26.1 (-0.1)	16.0 (-10.2)	22.3 (-3.9)	18.4 (-7.8)	14.5 (-11.7)
		w/o DP	-	47.2 (+2.9)	49.7 (+5.4)	48.7 (+4.4)	39.3 (-5.0)			w/o DP	-	21.3 (-4.9)	23.9 (-2.3)	21.8 (-4.4)	18.2 (-8.0)
		w/o KD	-	37.0 (-7.3)	46.0 (+1.7)	41.1 (-3.2)	32.5 (-11.8)			w/o KD	-	17.1 (-9.1)	22.1 (-4.1)	17.9 (-8.3)	14.3 (-11.9)
	Full	-	49.3 (+5.0)	50.7 (+6.4)	48.8 (+4.5)	41.5 (-2.8)	Full	-	24.4 (-1.8)	25.3 (-0.9)	24.0 (-2.2)	23.7 (-2.5)			
	10	Baseline	63.0	55.6 (-7.4)	59.3 (-3.6)	61.3 (-1.7)	44.4 (-18.6)	10	10	Baseline	34.4	32.1 (-2.3)	33.1 (-1.3)	34.1 (-0.3)	28.1 (-6.3)
		w/o DP&KD	64.7 (+1.7)	61.0 (-2.0)	62.3 (-0.7)	62.4 (-0.6)	54.7 (-8.3)			w/o DP&KD	40.2 (+5.8)	35.3 (+0.9)	37.9 (+3.5)	37.2 (+2.8)	33.7 (-0.7)
		w/o DP	-	64.0 (+1.0)	63.3 (+0.3)	63.6 (+0.6)	57.7 (-5.3)			w/o DP	-	39.4 (+5.0)	39.2 (+4.8)	38.9 (+4.5)	38.5 (+4.1)
		w/o KD	-	63.9 (+0.9)	63.8 (+0.8)	62.2 (-0.8)	54.0 (-9.0)			w/o KD	-	34.8 (+0.4)	38.5 (+4.1)	36.6 (+2.2)	35.0 (+0.6)
	Full	-	66.6 (+3.6)	64.8 (+1.8)	65.4 (+2.4)	62.4 (-0.6)	Full	-	40.6 (+6.2)	39.9 (+5.5)	39.4 (+5.0)	40.1 (+5.7)			
	50	Baseline	70.5	66.7 (-3.8)	66.8 (-3.7)	68.3 (-2.2)	60.5 (-10.0)	50	50	Baseline	42.1	46.7 (+4.6)	45.5 (+3.4)	45.5 (+3.4)	45.8 (+3.7)
		w/o DP&KD	72.4 (+1.9)	73.0 (+2.5)	71.0 (+0.5)	70.9 (+0.4)	71.2 (+0.7)			w/o DP&KD	46.2 (+4.1)	46.8 (+4.7)	46.1 (+4.0)	45.5 (+3.4)	46.9 (+4.8)
w/o DP		-	73.9 (+3.4)	72.1 (+1.6)	72.0 (+1.5)	72.9 (+2.4)	w/o DP			-	48.3 (+6.2)	44.6 (+2.5)	45.8 (+3.7)	48.7 (+6.6)	
w/o KD		-	74.5 (+4.0)	71.5 (+1.0)	70.1 (-0.4)	70.6 (+0.1)	w/o KD			-	47.2 (+5.1)	47.0 (+4.9)	45.0 (+2.9)	46.1 (+4.0)	
Full	-	74.5 (+4.0)	73.2 (+2.7)	72.8 (+2.3)	73.2 (+2.7)	Full	-	48.5 (+6.4)	46.6 (+4.5)	46.7 (+4.6)	49.1 (+7.0)				
MTT [20]	1	Baseline	48.3	37.2 (-11.1)	40.5 (-7.8)	39.3 (-9.0)	22.4 (-25.9)	1	1	Baseline	24.4	14.3 (-10.1)	17.0 (-7.4)	15.6 (-8.8)	4.6 (-19.8)
		w/o DP&KD	46.8 (-1.5)	36.9 (-11.4)	43.2 (-5.1)	36.7 (-11.6)	24.7 (-23.6)			w/o DP&KD	25.0 (+0.6)	12.5 (-11.9)	20.6 (-3.8)	8.2 (-16.2)	6.0 (-18.4)
		w/o DP	-	41.6 (-6.7)	46.7 (-1.6)	38.6 (-9.7)	32.4 (-15.9)			w/o DP	-	13.3 (-11.1)	24.4 (+0.0)	10.2 (-14.2)	8.5 (-15.9)
		w/o KD	-	35.5 (-12.8)	41.1 (-7.2)	34.4 (-13.9)	28.5 (-19.8)			w/o KD	-	13.6 (-10.8)	19.7 (-4.7)	12.4 (-12.0)	9.3 (-15.1)
	Full	-	47.2 (-1.1)	47.3 (-1.0)	44.1 (-4.2)	43.0 (-5.3)	Full	-	24.9 (+0.5)	25.8 (+1.4)	22.1 (-2.3)	24.6 (+0.2)			
	10	Baseline	63.6	48.9 (-14.7)	56.9 (-6.7)	52.6 (-11.0)	28.1 (-35.5)	10	10	Baseline	38.4	32.9 (-5.5)	33.7 (-4.7)	28.8 (-9.6)	22.5 (-15.9)
		w/o DP&KD	65.0 (+1.4)	51.3 (-12.3)	60.7 (-2.9)	56.0 (-7.6)	39.8 (-23.8)			w/o DP&KD	38.5(+0.1)	32.7 (-5.7)	36.0 (-2.4)	33.9 (-4.5)	30.6 (-7.8)
		w/o DP	-	61.4 (-2.2)	52.7 (-10.9)	48.8 (-14.8)	49.9 (-13.7)			w/o DP	-	35.0 (-3.4)	38.2 (-0.2)	35.5 (-2.9)	34.2 (-4.2)
		w/o KD	-	60.7 (-2.9)	59.2 (-4.4)	57.6 (-6.0)	47.5 (-16.1)			w/o KD	-	34.6 (-3.8)	34.9 (-3.5)	33.2 (-5.2)	32.9 (-5.5)
	Full	-	67.4 (+3.8)	68.3 (+4.7)	67.1 (+3.5)	63.8 (+0.2)	Full	-	38.4 (+0.0)	39.9 (+1.5)	36.0 (-2.0)	38.5 (+0.1)			
	50	Baseline	70.2	62.3 (-7.9)	67.5 (-2.7)	63.0 (-7.2)	53.1 (-17.1)	50	50	Baseline	44.5	43.1 (-1.4)	41.4 (-3.1)	39.3 (-5.2)	38.7 (-5.8)
		w/o DP&KD	70.5 (+0.3)	68.1 (-2.1)	69.5 (-0.7)	67.6 (-2.6)	66.5 (-3.7)			w/o DP&KD	46.0 (+1.5)	46.2 (+1.7)	46.1 (+1.6)	44.5 (+0.0)	45.5 (+1.0)
w/o DP		-	66.9 (-3.3)	63.8 (-6.4)	61.2 (-9.0)	66.8 (-3.4)	w/o DP			-	47.2 (+2.7)	47.1 (+2.6)	45.1 (+0.6)	47.2 (+2.7)	
w/o KD		-	69.8 (-0.4)	67.2 (-3.0)	69.0 (-1.2)	65.0 (-5.2)	w/o KD			-	46.9 (+2.4)	45.7 (+1.2)	43.4 (-1.1)	46.8 (+2.3)	
Full	-	71.0 (+0.8)	72.0 (+1.8)	69.5 (-1.2)	70.0 (-0.2)	Full	-	48.9 (+4.4)	47.6 (+3.1)	45.1 (+0.6)	49.4 (+4.9)				
DATM [29]	10	Baseline	58.2	50.4 (-7.8)	58.4 (+0.2)	53.1 (-5.1)	28.8 (-29.4)	10	10	Baseline	29.6	21.9 (-7.7)	26.8 (-2.8)	21.2 (-8.4)	8.7 (-20.9)
		w/o DP&KD	66.5 (+8.3)	51.0 (-7.2)	60.3 (+5.1)	57.4 (-0.8)	39.6 (-18.6)			w/o DP&KD	32.4 (+2.8)	25.5 (-4.1)	32.4 (+2.8)	26.4 (-3.2)	17.9 (-11.7)
		w/o DP	-	54.9 (-3.3)	65.8 (+7.6)	61.9 (+3.7)	43.2 (-15.0)			w/o DP	-	29.7 (+0.1)	34.6 (+5.0)	31.1 (+1.5)	24.2 (-5.4)
		w/o KD	-	59.6 (+1.4)	63.5 (+5.3)	60.5 (+2.3)	53.5 (-4.7)			w/o KD	-	29.5 (-0.1)	32.3 (+2.7)	30.6 (+1.0)	28.2 (-1.4)
	Full	-	64.3 (+6.1)	67.5 (+9.3)	63.4 (+5.2)	59.8 (+1.6)	Full	-	33.9 (+4.3)	35.3 (+5.7)	33.1 (+3.5)	35.2 (+5.6)			
	50	Baseline	70.0	69.2 (-0.8)	71.5 (+1.5)	66.9 (-3.1)	54.4 (-15.6)	50	50	Baseline	46.8	44.0 (-2.8)	44.7 (-2.1)	41.7 (-5.1)	39.1 (-7.7)
		w/o DP&KD	74.5 (+4.5)	72.1 (+2.1)	73.7 (+3.7)	71.8 (+1.8)	70.0 (+0.0)			w/o DP&KD	47.4 (+0.6)	47.6 (+0.8)	48.6 (+1.8)	46.5 (-0.3)	46.6 (-0.2)
		w/o DP	-	73.0 (+3.0)	75.3 (+5.3)	71.9 (+1.9)	72.5 (+2.5)			w/o DP	-	51.3 (+4.5)	50.6 (+3.8)	48.8 (+2.0)	50.5 (+3.7)
		w/o KD	-	75.4 (+5.4)	73.8 (+3.8)	73.1 (+3.1)	73.3 (+3.3)			w/o KD	-	49.9 (+3.4)	48.0 (+1.2)	48.2 (+1.4)	50.7 (+3.9)
	Full	-	75.7 (+5.7)	77.2 (+7.2)	74.7 (+4.7)	75.7 (+5.7)	Full	-	52.0 (+5.2)	50.6 (+3.8)	50.3 (+3.5)	54.0 (+7.2)			
	500	Baseline	76.5	82.5 (+6.0)	80.1 (+3.6)	77.0 (+0.5)	79.8 (+3.3)	500	100	Baseline	52.5	55.0 (+2.5)	53.1 (+0.6)	51.0 (-1.5)	52.1 (-0.4)
		w/o DP&KD	83.3 (+6.8)	86.0 (+9.5)	83.5 (+7.0)	82.3 (+5.8)	85.8 (+9.3)			w/o DP&KD	53.6 (+1.1)	58.4 (+5.9)	55.6 (+3.1)	55.1 (+2.6)	58.9 (+6.4)
w/o DP		-	85.9 (+9.4)	84.4 (+7.9)	83.6 (+7.1)	86.5 (+10.0)	w/o DP			-	59.3 (+6.8)	56.7 (+4.2)	56.3 (+3.8)	59.7 (+7.2)	
w/o KD		-	86.7 (+10.2)	84.3 (+7.8)	84.2 (+7.7)	87.2 (+10.7)	w/o KD			-	60.5 (+8.0)	55.8 (+3.3)	57.2 (+4.7)	60.6 (+8.1)	
Full	-	86.8 (+10.3)	85.1 (+8.6)	85.3 (+8.8)	87.9 (+11.4)	Full	-	60.5 (+8.0)	56.5 (+4.0)	58.0 (+5.5)	60.9 (+8.4)				

of instances per class (IPC), different datasets and different architectures of the test networks. Table II(a) demonstrate the results on CIFAR10, and the results on CIFAR100 and Tiny-ImageNet are reported in Table II(b) and Table VIII, respectively. Note that, DropPath and knowledge distillation are not applicable when we use the same architecture for training and test networks, i.e., 3-layer CNN, because 1) it is too shallow for DropPath; 2) we will converge to the teacher model if we use the same model architecture for the teacher and the student models. We can observe from these results that architecture overfitting is more severe in the case of small IPCs and large architecture discrepancy between the training networks and the test networks, but both DropPath and knowledge distillation is capable of mitigating it. In addition, combining them can

further improve the performance and overcome architecture overfitting in many cases. For instance, when evaluating our method on distilled images of MTT (CIFAR10, IPC=10), it contributes performance gains of 18.5% and 35.7% for ResNet18 and ResNet50, respectively. We are also interested in how much performance gap between training and test networks we can close. Surprisingly, when IPC=10 and 50, the test accuracies of most network architectures surpass that of the architecture identical to the training network. Along with it, the gaps between different test networks, such as ResNet18 and ResNet50, are also narrowed down in most cases. Additionally, when the IPC reaches 500, our method can still contribute to performance gain.

DropPath enables an implicit ensemble of the shallow

subnetworks and thus mitigates architecture overfitting. However, each of these sub-networks may have sub-optimal performance. Knowledge distillation can address this issue by encouraging similar outputs between the teacher model and the sub-networks and thus further improves the performance. By contrast, the contribution of knowledge distillation could be marginal without DropPath due to the big difference in architecture [75]. Empirically, combining DropPath with knowledge distillation not only achieves the best performance, but also greatly decreases the performance difference among different test network architectures.

To better validate the effectiveness of our method, we report the standard deviations of test accuracies of CIFAR10 (FRePo) in Table III. We calculate these standard deviations by running the experiments three times with different random seeds. It can be observed that the standard deviation generally increases as IPC decreases. The reason could be that when IPC gets smaller, there are more solutions that make the training error zero, so the performance of training becomes more sensitive to initialization. Despite this, we can still see significant improvement introduced by our methods.

TABLE III

THE AVERAGE TEST ACCURACIES OF MODELS TRAINED ON THE DISTILLED DATA OF CIFAR10 [74] WITH DIFFERENT IPCS. THE NUMBER AFTER \pm DENOTES THE STANDARD DEVIATION. THESE RESULTS ARE OBTAINED THROUGH THREE REPETITIVE EXPERIMENTS WITH DIFFERENT RANDOM SEEDS. 3-LAYER CNN IS THE ARCHITECTURE USED IN DISTILLATION AND IS THE TEACHER MODEL OF KNOWLEDGE DISTILLATION.

DD	IPC	Methods	ResNet18	AlexNet	VGG11	ResNet50
FRePo	1	w/o DP&KD	35.6 \pm 2.5	47.4 \pm 0.9	41.5 \pm 1.1	30.3 \pm 1.9
		w/o DP	47.2 \pm 0.5	49.7 \pm 0.7	48.7 \pm 0.6	39.3 \pm 1.4
		w/o KD	37.0 \pm 1.0	46.0 \pm 0.6	41.1 \pm 1.3	32.5 \pm 1.4
		Full	49.3 \pm 0.6	50.7 \pm 0.1	48.8 \pm 0.4	41.5 \pm 1.0
	10	Full	66.2 \pm 0.5	64.8 \pm 0.9	65.4 \pm 0.2	62.4 \pm 0.9
50	Full	74.5 \pm 0.1	73.2 \pm 0.3	72.8 \pm 0.0	73.2 \pm 0.2	

B. Comparison with Other Baselines

TABLE IV

COMPARISON WITH BASELINES. P IN DROPPATH AND DROPOUT DENOTES THE KEEP RATE AND α IN MIXUP AND CUTMIX IS THE PARAMETERS α AND β IN BETA DISTRIBUTION, WHERE α AND β ARE THE SAME.

Method	Baseline	DropPath (p=0.5)	DropOut (p=0.5)	MixUp (alpha=0.5)	CutMix (alpha=0.5)	Ours
Test Acc.	55.6	46.2	44.3	57.6	56.9	66.2

We further compare our method with some regularization methods on architectures, including DropOut [61] and DropPath [62], and data augmentation methods, including MixUp [76] and CutMix [77]. We use 3-layer CNN as the training network and ResNet18 as the test network. We use FRePo to generate distilled dataset and set IPC to be 10. Our results are reported in Table IV. We observe that DropPath and DropOut with a constant keep rate deteriorate the performance, compared with the DropPath variant proposed by us. In addition, MixUp and CutMix only contribute marginal performance improvement, compared with 2-fold augmentation in this work. These results further demonstrate the effectiveness of our methods to train distilled datasets.

C. Improve the Performance of Training on Limited Real Data

In this section, we discuss the performance of our methods when training on a limited amount of real data and compare it with the case of the distilled dataset. Our methods have shown effective on the distilled dataset, we expect them to improve the performance on limited real training data as well. In this case, smaller models also tend to perform better than larger models because both can fit the training set perfectly but the latter suffers more from overfitting.

As illustrated in Figure 3, we train models on different fractions of CIFAR10 training set which are randomly sampled. The 3-layer CNN still serves as the teacher model when we use knowledge distillation. Since ResNet18 and ResNet50 exhibit the largest performance differences from the 3-layer CNN in the previous experiments, we only show the results of ResNet18 and ResNet50 here. ResNet18 and ResNet50 significantly outperform 3-layer CNN with enough training data, but they show worse generalization performance than CNN when the fraction is lower than 0.02, i.e., 1000 training instances. Under our methods, the performances of both ResNet18 and ResNet50 surpass that of 3-layer CNN even when the fraction is as small as 0.002, i.e., 100 training instances. However, the performance gain saturates or even declines when the fraction of training data exceeds 0.05. This can be attributed to the suboptimal performance of the teacher model (blue line). Nevertheless, Figure 3 (d) shows that when the current teacher does not contribute to performance gain anymore, a stronger teacher can further improve the performance.

Furthermore, we observe that the performance gap of training on limited real data is much smaller than that of training on distilled images. For instance, when the fraction of training data is 0.002, which is equivalent to IPC=10, the performance gap between 3-layer CNN and ResNet50 is 4.9% when they are trained on real images. However, when we train them on distilled images of FRePo, the performance gap increases to 18.6%. As for the distilled images generated by MTT, the gap is even larger, which reaches 35.5%. Meanwhile, training on a distilled dataset results in much better performance than training on real data of the same size, which makes it popular in downstream applications. Therefore, we focus on applying our method in the context of dataset distillation, in which the effectiveness of our method can be better revealed.

D. Smoothing Effect Induced by Proposed Methods

To corroborate the smoothing effect induced by our proposed methods, we analyze the Hessian spectrum of models trained with different ablations. It is known that the curvature in the neighborhood of model parameters is dominated by the top eigenvalues of the Hessian matrix $\nabla^2 \mathcal{L}_{CE}(\theta)$, where $\mathcal{L}_{CE}(\theta)$ denotes the cross-entropy loss w.r.t model parameters θ . In the implementation, we use the power iteration as in [78, 79] to iteratively estimate the top 20 eigenvalues and the corresponding eigenvectors of the Hessian matrix.

As shown in Figure 4 (a), the eigenvalues of the Hessian matrix for the ResNet18 trained with full setting and Lion optimizer are the lowest among the evaluated settings, which quantitatively indicates that the neighborhood of the minima

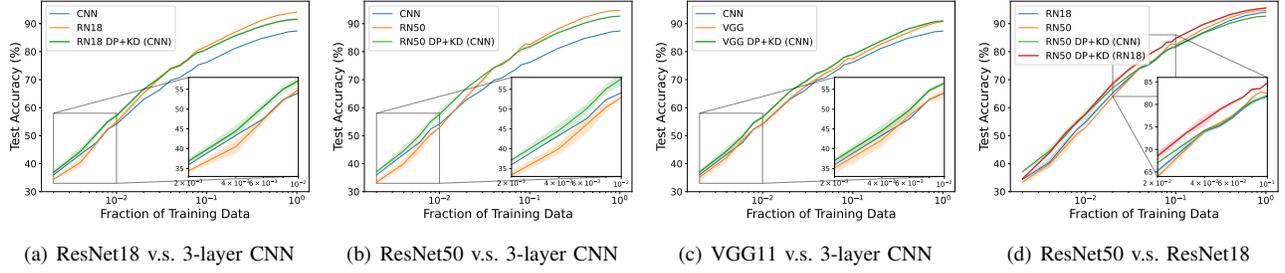


Fig. 3. Test accuracies obtained from training on different fractions of CIFAR10, the shadow indicates the standard deviation. We compare the test accuracies (a) between ResNet18 (RN18) and 3-layer CNN (CNN), (b) between ResNet50 (RN50) and CNN, (c) between VGG11 and 3-layer CNN (CNN), and (d) between ResNet50 (RN50) and ResNet18, respectively. The x-axis denotes the fraction of training data, *DP+KD* denotes that the network is trained with DropPath and knowledge distillation. The model enclosed in the brackets after KD represents the teacher model used. Note that we run the experiments three times with different random seeds.

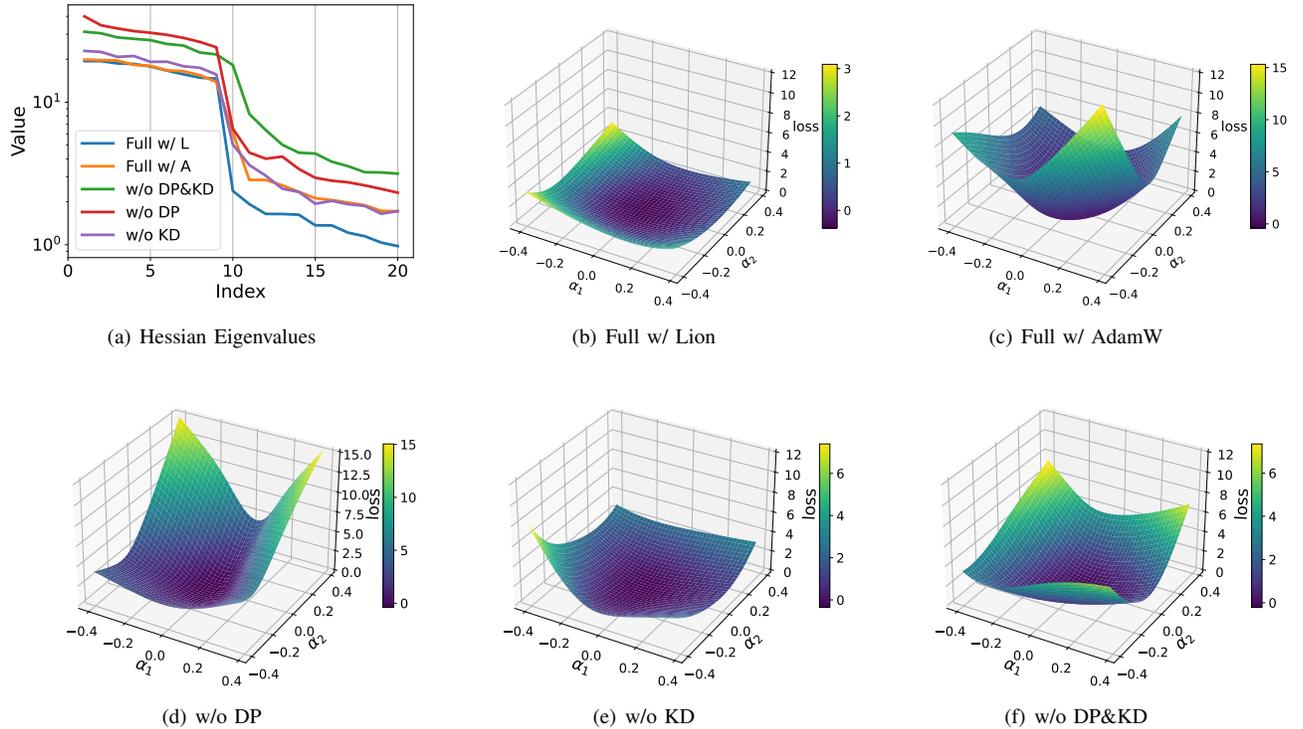


Fig. 4. Visualization of the smoothing effect induced by proposed methods. (a) Top 20 eigenvalues of Hessian matrix for ResNet18 trained with different settings, including full setting with Lion optimizer (Full w/ L), full setting with AdamW (Full w/ A), w/o DP, w/o KD and w/o DP&KD. For w/o DP, w/o KD and w/o DP&KD, Lion is adopted by default. (b)-(f) Loss landscape $\mathcal{L}_{CE}(\theta + \alpha_1 \mathbf{v}_1 + \alpha_2 \mathbf{v}_2)$ of ResNet18 around the minima found by models with different settings, where \mathbf{v}_1 and \mathbf{v}_2 are the eigenvectors corresponding to the top two eigenvalues of Hessian matrices, respectively. Note that the training data is 100 (IPC=10) distilled images of CIFAR10 by FRePo. ResNet18 is trained with DropPath and knowledge distillation. 3-layer CNN serves as the teacher model where knowledge distillation is adopted.

found by our method has smaller curvature. Furthermore, Figure 4 (b)-(f) qualitatively shows that our method induces a smoother loss landscape. Notably, DropPath, forming an implicit ensemble of sub-networks, contributes the most to loss landscape smoothing. By contrast, knowledge distillation only has a marginal effect on smoothing.

E. Ablation Studies

We conduct extensive ablation studies here to validate the effectiveness of each component in our methods. In this subsection, we focus on the case of using 3-layer CNN as the training network, ResNet18 as the test network, setting

IPC to 10 and generating the distilled dataset by FRePo. Note that the baseline performance of 3-layer CNN trained on the distilled data is 63.0%, its performance improves to 64.7% with better optimization and data augmentation.

DropPath: We first try different minimum keep rates in the three-phase scheduler introduced in Section III-A. As illustrated in Figure 5 (a) and (c), a lower minimum keep rate and a longer period of decay induce better performance, but both of them make the training longer. To balance performance and efficiency, we set the minimum keep rate and period of decay to 0.5 and 500, respectively. Figure 5 (b) shows that different final keep rates do not significantly affect the

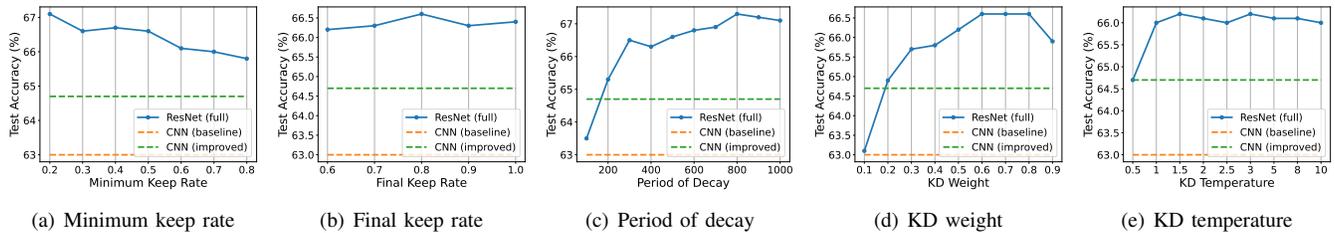


Fig. 5. Ablation studies on minimum keep rate, final keep rate, period of decay, weight and temperature of knowledge distillation (KD). (a) Test accuracies of different minimum keep rates. (b) Test accuracies of different keep rates at the final phase. (c) Test accuracies of different periods of decay. (d) Test accuracies of different KD weights. (e) Test accuracies of different KD temperatures. Regardless of the variation of hyperparameters, ResNet18 trained with our approach generally outperforms 3-layer CNN trained with baseline (orange dashed line) and that trained with better optimization and data augmentation (green dashed line).

TABLE V

ABLATION STUDIES ON THE HIGH KEEP RATE IN THE FINAL PHASE OF TRAINING AND IMPROVED SHORTCUT CONNECTION (SC).

Final phase	Improved SC	Test Acc.
✗	✗	65.2
✓	✗	65.6
✗	✓	65.9
✓	✓	66.6

TABLE VI

ABLATION STUDIES ABOUT OPTIMIZATION AND DATA AUGMENTATION. IF PERIODICAL LEARNING RATE (LR), LION OPTIMIZER AND STRONGER AUGMENTATION (AUG.) ARE NOT ADOPTED, WE REPLACE THEM WITH COSINE ANNEALING LEARNING RATE, ADAMW AND 1-FOLD AUGMENTATION, RESPECTIVELY.

Periodical LR	Lion	stronger Aug.	Test Acc.
✗	✗	✗	61.6
✓	✗	✗	61.9
✓	✓	✗	64.8
✓	✓	✓	66.6

performance. Moreover, we verify the effectiveness of the high keep rate in the final phase of training, and the improved shortcut connection (SC) introduced in Section III-A. The results shown in Table V indicate that both of them contribute to the performance.

Knowledge Distillation: We also test different hyperparameters of knowledge distillation (KD). As illustrated in Figure 5 (d) and (e), when weight α and temperature τ are in the range of [0.5, 0.8] and [1, 10], respectively, the performance does not vary significantly. It indicates that our method is quite robust to different hyperparameter choices.

Optimization and Data Augmentation: In Table VI, we replace each of the optimization and data augmentation approaches with a baseline. The results indicate that each of these approaches improves performance. Among them, Lion optimizer contributes a performance improvement of 2.9%.

Impact of Augmentation when IPC=1: It should be noted that the results of IPC=1 in Table II(a) are obtained with 4-fold augmentation. For comparison, we also get the results with 2-fold augmentation (see in Table VII). Compared with Table II(a), the test accuracies of *w/o DP&KD* and *w/o & KD* in Table VII are higher, but those of *w/o DP* and *Full* are lower. Especially for ResNet50, the performance of *Full*

increases by 7.7% with 4-fold augmentation. This indicates that stronger augmentation is necessary when using knowledge distillation when there are extremely limited data, and when the architecture difference between the training and test networks is big. Moreover, we observe that the contribution of 4-fold augmentation is marginal under a larger IPC, so we adopt 4-fold augmentation only when IPC=1.

TABLE VII

TEST ACCURACIES OF MODELS TRAINED ON THE DISTILLED DATA OF CIFAR10 (FREPo, IPC=1). HOWEVER, 2-FOLD AUGMENTATION IS ADOPTED HERE. EXCEPT THAT, THE OTHER SETTINGS ARE THE SAME AS TABLE II(A).

IPC	Methods	3-layer CNN	ResNet18	AlexNet	VGG11	ResNet50
1	Baseline	44.3	34.4 (-9.9)	41.8 (-2.5)	44.0 (-0.3)	25.9 (-18.4)
	w/o DP&KD	44.8 (+0.5)	41.2 (-3.1)	45.4 (+1.1)	45.9 (+1.6)	32.8 (-11.5)
	w/o DP	-	41.0 (-3.3)	44.5 (+0.2)	47.0 (+2.7)	30.0 (-14.3)
	w/o KD	-	39.4 (-4.9)	47.1 (+2.8)	38.9 (-5.4)	31.0 (-13.3)
	Full	-	45.5 (+1.2)	47.8 (+3.5)	46.7 (+2.4)	33.8 (-10.5)

V. CONCLUSION

This paper studies architecture overfitting when we train models on distilled datasets. To mitigate this issue, we propose a series of approaches based on the intuition that the large model can act as an implicit ensemble of small models. These methods also exhibit a smoothing effect from different aspects. Our methods are efficient and generic, and can improve the performance when training on a small real dataset directly. We believe that our work can help extend the utility of distilled datasets in more real-world scenarios. Recognizing that this work only mitigates architecture overfitting in the evaluation stage, our future work will focus on developing a more generalizable dataset distillation algorithm to address this issue in essence.

ACKNOWLEDGMENTS

This work is supported by the internal funds of City University of Hong Kong (No. 9610614 and No. 9229130). It is also supported by the NSFC project (No. 62306250). We also thank Shuqi Liu for her support in experiments.

REFERENCES

[1] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer, "High-resolution image synthesis with latent diffusion models," 2022.

- [2] J. Jumper, R. Evans, A. Pritzel *et al.*, “Highly accurate protein structure prediction with alphafold,” *Nature*, vol. 596, pp. 583–589, 2021. [Online]. Available: <https://doi.org/10.1038/s41586-021-03819-2>
- [3] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly *et al.*, “An image is worth 16x16 words: Transformers for image recognition at scale,” *arXiv preprint:2010.11929*, 2020.
- [4] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, “Language models are few-shot learners,” *Advances in neural information processing systems*, vol. 33, pp. 1877–1901, 2020.
- [5] C. Coleman, C. Yeh, S. Musmann, B. Mirzasoleiman, P. Bailis, P. Liang, J. Leskovec, and M. Zaharia, “Selection via proxy: Efficient data selection for deep learning,” *arXiv preprint:1906.11829*, 2019.
- [6] M. Hwang, Y. Jeong, and W. Sung, “Data distribution search to select core-set for machine learning,” in *The 9th International Conference on Smart Media and Applications*, 2020, pp. 172–176.
- [7] T. Wang, J.-Y. Zhu, A. Torralba, and A. A. Efros, “Dataset distillation,” *arXiv preprint:1811.10959*, 2018.
- [8] B. Zhao, K. R. Mopuri, and H. Bilen, “Dataset condensation with gradient matching,” *arXiv preprint:2006.05929*, 2020.
- [9] B. Zhao and H. Bilen, “Dataset condensation with distribution matching,” in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, 2023, pp. 6514–6523.
- [10] A. Rosasco, A. Carta, A. Cossu, V. Lomonaco, and D. Bacciu, “Distilled replay: Overcoming forgetting through synthetic samples,” in *Continual Semi-Supervised Learning: First International Workshop, CSSL 2021, Virtual Event, August 19–20, 2021, Revised Selected Papers*. Springer, 2022, pp. 104–117.
- [11] B. Zhao and H. Bilen, “Dataset condensation with differentiable siamese augmentation,” in *International Conference on Machine Learning*. PMLR, 2021, pp. 12 674–12 685.
- [12] G. Li, R. Togo, T. Ogawa, and M. Haseyama, “Soft-label anonymous gastric x-ray image distillation,” in *2020 IEEE International Conference on Image Processing (ICIP)*. IEEE, 2020, pp. 305–309.
- [13] J. Goetz and A. Tewari, “Federated learning via synthetic data,” *arXiv preprint:2008.04489*, 2020.
- [14] O. Bohdal, Y. Yang, and T. Hospedales, “Flexible dataset distillation: Learn labels instead of images,” *arXiv preprint:2006.08572*, 2020.
- [15] I. Sucholutsky and M. Schonlau, “Soft-label dataset distillation and text dataset distillation,” in *2021 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2021, pp. 1–8.
- [16] T. Nguyen, Z. Chen, and J. Lee, “Dataset meta-learning from kernel ridge-regression,” in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021.
- [17] T. Nguyen, R. Novak, L. Xiao, and J. Lee, “Dataset distillation with infinitely wide convolutional networks,” *Advances in Neural Information Processing Systems*, vol. 34, pp. 5186–5198, 2021.
- [18] Y. Zhou, E. Nezhadarya, and J. Ba, “Dataset distillation using neural feature regression,” *arXiv preprint:2206.00719*, 2022.
- [19] S. Lee, S. Chun, S. Jung, S. Yun, and S. Yoon, “Dataset condensation with contrastive signals,” in *International Conference on Machine Learning*. PMLR, 2022, pp. 12 352–12 364.
- [20] G. Cazenavette, T. Wang, A. Torralba, A. A. Efros, and J.-Y. Zhu, “Dataset distillation by matching training trajectories,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 4750–4759.
- [21] J. Cui, R. Wang, S. Si, and C.-J. Hsieh, “Scaling up dataset distillation to imagenet-1k with constant memory,” *arXiv preprint:2211.10586*, 2022.
- [22] K. Wang, B. Zhao, X. Peng, Z. Zhu, S. Yang, S. Wang, G. Huang, H. Bilen, X. Wang, and Y. You, “Cafe: Learning to condense dataset by aligning features,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 12 196–12 205.
- [23] W. Chen, Z. Yu, S. De Mello, S. Liu, J. M. Alvarez, Z. Wang, and A. Anandkumar, “Contrastive syn-to-real generalization,” *arXiv preprint:2104.02290*, 2021.
- [24] A. Parnami and M. Lee, “Learning from few examples: A summary of approaches to few-shot learning,” *arXiv preprint:2203.04291*, 2022.
- [25] S. Lei and D. Tao, “A comprehensive survey to dataset distillation,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 46, no. 1, pp. 17–32, 2023.
- [26] N. Loo, R. Hasani, A. Amini, and D. Rus, “Efficient dataset distillation using random feature approximation,” in *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS)*, 2022.
- [27] N. Loo, R. Hasani, M. Lechner, and D. Rus, “Dataset distillation with convexified implicit gradients,” in *Proceedings of the International Conference on Machine Learning (ICML)*, 2023, pp. 22 649–22 674.
- [28] J. Du, Y. Jiang, V. T. F. Tan, J. T. Zhou, and H. Li, “Minimizing the accumulated trajectory error to improve dataset distillation,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023, pp. 3749–3758.
- [29] Z. Guo, K. Wang, G. Cazenavette, H. Li, K. Zhang, and Y. You, “Towards lossless dataset distillation via difficulty-aligned trajectory matching,” in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2024.
- [30] J. Du, Q. Shi, and J. T. Zhou, “Sequential subset matching for dataset distillation,” in *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS)*, 2023.
- [31] Y. Lee and H. W. Chung, “SelMatch: Effectively scaling up dataset distillation via selection-based initialization and partial updates by trajectory matching,” in *Pro-*

- ceedings of the International Conference on Machine Learning (ICML)*, 2024.
- [32] D. Liu, J. Gu, H. Cao, C. Trinitis, and M. Schulz, “Dataset distillation by automatic training trajectories,” in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2024.
- [33] S. Yang, S. Cheng, M. Hong, H. Fan, X. Wei, and S. Liu, “Neural spectral decomposition for dataset distillation,” in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2024.
- [34] A. Sajedi, S. Khaki, E. Amjadian, L. Z. Liu, Y. A. Lawryshyn, and K. N. Plataniotis, “Datadam: Efficient dataset distillation with attention matching,” in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2023, pp. 17 097–17 107.
- [35] G. Zhao, G. Li, Y. Qin, and Y. Yu, “Improved distribution matching for dataset condensation,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023, pp. 7856–7865.
- [36] H. Zhang, S. Li, P. Wang, and S. Zeng, Dan Ge, “M3D: Dataset condensation by minimizing maximum mean discrepancy,” in *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 2024.
- [37] W. Deng, W. Li, T. Ding, L. Wang, H. Zhang, K. Huang, J. Huo, and Y. Gao, “Exploiting inter-sample and inter-feature relations in dataset distillation,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024, pp. 17 057–17 066.
- [38] H. Zhang, S. Li, F. Lin, W. Wang, Z. Qian, and S. Ge, “DANCE: Dual-view distribution alignment for dataset condensation,” in *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2024.
- [39] H. Li, Y. Zhou, X. Gu, B. Li, and W. Wang, “Diversified semantic distribution matching for dataset distillation,” in *Proceedings of the ACM International Conference on Multimedia (MM)*, 2024.
- [40] L. Zhang, J. Zhang, B. Lei, S. Mukherjee, X. Pan, B. Zhao, C. Ding, Y. Li, and X. Dongkuan, “Accelerating dataset distillation via model augmentation,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023, pp. 11 950–11 959.
- [41] Y. Liu, J. Gu, K. Wang, Z. Zhu, W. Jiang, and Y. You, “DREAM: Efficient dataset distillation by representative matching,” in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2023, pp. 17 314–17 324.
- [42] Y. He, L. Xiao, and T. J. Zhou, “You Only Condense Once: Two rules for pruning condensed datasets,” in *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS)*, 2023.
- [43] Y. Shang, Z. Yuan, and Y. Yan, “MIM4DD: Mutual information maximization for dataset distillation,” in *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS)*, 2023.
- [44] X. Chen, Y. Yang, Z. Wang, and B. Mirzasoleiman, “Data distillation can be like vodka: Distilling more times for better quality,” in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2024.
- [45] F. Yunzhen, V. Ramakrishna, and K. Julia, “Embarrassingly simple dataset distillation,” in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2024.
- [46] Y. He, L. Xiao, J. Tianyi Zhou, and I. Tsang, “Multisize dataset condensation,” in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2024.
- [47] N. Loo, A. Maalouf, R. Hasani, M. Lechner, A. Amini, and D. Rus, “Large scale dataset distillation with domain shift,” in *Proceedings of the International Conference on Machine Learning (ICML)*, 2024.
- [48] Y. Xu, Y.-L. Li, K. Cui, Z. Wang, C. Lu, Y.-W. Tai, and C.-K. Tang, “Distill gold from massive ores: Bi-level data pruning towards efficient dataset distillation,” in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2024.
- [49] J.-H. Kim, J. Kim, S. J. Oh, S. Yun, H. Song, J. Jeong, J.-W. Ha, and H. O. Song, “Dataset condensation via efficient synthetic-data parameterization,” in *International Conference on Machine Learning*. PMLR, 2022, pp. 11 102–11 118.
- [50] Z. Deng and O. Russakovsky, “Remember the past: Distilling datasets into addressable memories for neural networks,” *arXiv preprint:2206.02916*, 2022.
- [51] S. Liu, K. Wang, X. Yang, J. Ye, and X. Wang, “Dataset distillation via factorization,” *arXiv preprint:2210.16774*, 2022.
- [52] H. B. Lee, D. B. Lee, and S. J. Hwang, “Dataset condensation with latent space knowledge factorization and sharing,” *arXiv preprint:2208.10494*, 2022.
- [53] X. Wei, A. Cao, F. Yang, and Z. Ma, “Sparse parameterization for epitomic dataset distillation,” in *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS)*, 2023.
- [54] D. Shin, S. Shin, and I.-c. Moon, “Frequency domain-based dataset distillation,” in *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS)*, 2023.
- [55] H. Zheng, J. Sun, S. Wu, B. Kailkhura, Z. Mao, C. Xiao, and A. Prakash, “Leveraging hierarchical feature sharing for efficient dataset condensation,” in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2024.
- [56] L. Breiman, “Bagging predictors,” *Machine learning*, vol. 24, pp. 123–140, 1996.
- [57] T. Hastie, S. Rosset, J. Zhu, and H. Zou, “Multi-class adaboost,” *Statistics and its Interface*, vol. 2, no. 3, pp. 349–360, 2009.
- [58] L. Breiman, “Random forests,” *Machine learning*, vol. 45, pp. 5–32, 2001.
- [59] T. K. Ho, “Random decision forests,” in *Proceedings of 3rd international conference on document analysis and recognition*, vol. 1. IEEE, 1995, pp. 278–282.
- [60] J. H. Friedman, “Stochastic gradient boosting,” *Computational statistics & data analysis*, vol. 38, no. 4, pp. 367–378, 2002.

- [61] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.
- [62] G. Larsson, M. Maire, and G. Shakhnarovich, "Fractalnet: Ultra-deep neural networks without residuals," *arXiv preprint:1605.07648*, 2016.
- [63] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.
- [64] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *arXiv preprint:1503.02531*, 2015.
- [65] Y. Zhang, T. Xiang, T. M. Hospedales, and H. Lu, "Deep mutual learning," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 4320–4328.
- [66] D. Chen, J.-P. Mei, C. Wang, Y. Feng, and C. Chen, "Online knowledge distillation with diverse peers," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 04, 2020, pp. 3430–3437.
- [67] X. Chen, C. Liang, D. Huang, E. Real, K. Wang, Y. Liu, H. Pham, X. Dong, T. Luong, C.-J. Hsieh *et al.*, "Symbolic discovery of optimization algorithms," *arXiv preprint:2302.06675*, 2023.
- [68] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, F. Pereira, C. Burges, L. Bottou, and K. Weinberger, Eds., vol. 25. Curran Associates, Inc., 2012. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf
- [69] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint:1409.1556*, 2014.
- [70] L. Beyer, X. Zhai, A. Royer, L. Markeeva, R. Anil, and A. Kolesnikov, "Knowledge distillation: A good teacher is patient and consistent," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 10925–10934.
- [71] G. Huang, Y. Li, G. Pleiss, Z. Liu, J. E. Hopcroft, and K. Q. Weinberger, "Snapshot ensembles: Train 1, get m for free," *arXiv preprint:1704.00109*, 2017.
- [72] C. Shorten and T. M. Khoshgoftaar, "A survey on image data augmentation for deep learning," *Journal of big data*, vol. 6, no. 1, pp. 1–48, 2019.
- [73] S.-A. Rebuffi, S. Gowal, D. A. Calian, F. Stimberg, O. Wiles, and T. A. Mann, "Data augmentation can improve robustness," *Advances in Neural Information Processing Systems*, vol. 34, pp. 29935–29948, 2021.
- [74] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," 2009.
- [75] S. I. Mirzadeh, M. Farajtabar, A. Li, N. Levine, A. Matsukawa, and H. Ghasemzadeh, "Improved knowledge distillation via teacher assistant," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 34, no. 04, 2020, pp. 5191–5198.
- [76] H. Zhang, M. Cisse, Y. N. Dauphin, and D. Lopez-Paz, "mixup: Beyond empirical risk minimization," *arXiv preprint:1710.09412*, 2017.
- [77] S. Yun, D. Han, S. J. Oh, S. Chun, J. Choe, and Y. Yoo, "Cutmix: Regularization strategy to train strong classifiers with localizable features," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2019, pp. 6023–6032.
- [78] Z. Yao, A. Gholami, Q. Lei, K. Keutzer, and M. W. Mahoney, "Hessian-based analysis of large batch training and robustness to adversaries," *Advances in Neural Information Processing Systems*, vol. 31, 2018.
- [79] C. Liu, M. Salzmann, T. Lin, R. Tomioka, and S. Sússtrunk, "On the loss landscape of adversarial training: Identifying challenges and how to overcome them," *Advances in Neural Information Processing Systems*, vol. 33, pp. 21476–21487, 2020.
- [80] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 2009, pp. 248–255.



Xuyang Zhong is currently a Ph.D. student at the department of computer science, City University of Hong Kong and is supervised by Chen Liu. He obtained his Master's degree from Technical University of Munich in 2022 and his Bachelor's degree from Beijing Institute of Technology in 2020. His research interest mainly focuses on the optimization and robustness in deep learning.



Chen Liu is currently an assistant professor at the department of computer science, City University of Hong Kong. He obtained his Ph.D. degree from École Polytechnique Fédérale de Lausanne (EPFL) in August 2022 and was supervised by Prof. Sabine Sússtrunk and Dr. Mathieu Salzmann. Previously, he obtained his Master degree from École Polytechnique Fédérale de Lausanne (EPFL) in 2017 and Bachelor degree from Tsinghua University in 2015, both in Computer Science. Chen's research interest focuses on machine learning, optimization, robustness and privacy. He was supported by Microsoft Research Ph.D Scholarship programme between 2017 and 2019.

APPENDIX

A. Implementation Details

Datasets: The training sets in the experiments are the distilled datasets of CIFAR10, CIFAR100 [74] and Tiny-ImageNet [80], but the test sets are their respective original test sets. To better validate the effectiveness of our method, we use the distilled images synthesized by different dataset distillation algorithms, e.g., neural Feature Regression with Pooling (FRePo) [18] and Matching Training Trajectories (MTT) [20]. In addition, we evaluate the performance of our method in different numbers of instances per class (IPC), e.g., 1, 10 and 50. Note that MTT does not provide the final trainable learning rates in the released checkpoints, we adopt the reported initial learning rates in our baselines.

Models: The networks used to synthesize the distilled images (training networks) in FRePo and MTT are 3-layer CNN. Consistent with the hyperparameters reported in the paper, the output channels of hidden layers of the network used in FRePo are 128, 256 and 512, respectively. However, in MTT, all the output channels of hidden layers are set to 128. ResNet18, ResNet50 [63], AlexNet [68] and VGG11 [69] are adopted in the evaluation, they are thereby called test networks. The hyperparameters of networks are the same as those set in [20]. Note that when training networks on distilled images of FRePo and MTT, batch and instance normalization layers are adopted in networks following the settings of [18, 20], respectively.

DropPath: As shown in Algorithm 1, the decaying factor of keep rate $\gamma = 0.1$, minimum keep rate $p_{min} = 0.5$, final keep rate $p_{final} = 0.8$, period of decay $T = 500$, warmup period $W = 50$, stabilization epoch $S = (1 + p_{min}/\gamma) \times T = 3000$. The total epochs N is set to $S + 2 \times T = 4000$. In the improved shortcut, the pooling area depends on the stride of 1×1 convolutional layer in the original one. e.g., if the stride of 1×1 convolutional layer in the original shortcut is 2, we use a 2×2 max pooling.

Knowledge distillation: As shown in Eq. 2, the temperature factor $\tau = 1.5$, and the weight factor $\alpha = 0.5$. If not specifically indicated, the default teacher model is the 3-layer CNN. Note that the teacher model is trained on the same data set as the student model.

Periodical learning rate: In Eq. 5, the maximum learning rate $lr_{max} = 5 \times 10^{-5}$, the base decaying factor for learning rate $\lambda = 0.8$. The period of the cosine function T_{max} and the number of warmup epochs T_{warm} are 1000 and 50, respectively.

Optimizer: Lion [67] is adopted in our method, where weight decay $\lambda_{wd} = 5 \times 10^{-3}$, coefficient $\beta_1 = 0.95$, and $\beta_2 = 0.98$.

Augmentation: There are color jittering, cropping, cutout, flipping, scaling, and rotating in the augmentation pool, we sample more operations instead of just one as in [20].

Training: For CIFAR10, the batch sizes for different IPCs are 10 (IPC=1), 100 (IPC=10) and 128 (IPC=50), respectively. For CIFAR100, the batch sizes are 100 (IPC=1), 256 (IPC=10) and 256 (IPC=50), respectively. Cross-entropy is adopted as the loss function in our experiments. Since the labels of images are learnable in FRePo, we divide them with a temperature

factor $t = 0.3$ for CIFAR10, 0.04 for CIFAR100, and 0.02 for Tiny-ImageNet, respectively.

B. Supplementary Figures of Methods

The corresponding curve of the dynamic keep rate is shown in Figure 6 (a). Mathematically, the dynamic keep rate p is formulated as

$$p = \begin{cases} \max(p_{min}, 1 - \gamma \cdot \text{ceil}((i - W)/T)), & \text{if } i < S, \\ p_{final}, & \text{otherwise.} \end{cases} \quad (4)$$

where $\gamma \in [0, 1]$ is a decaying factor. i , W , T and S denote the current epoch, warmup period, decay period and stabilization epoch, respectively. Unless specified, we set $\gamma = 0.1$, $p_{min} = 0.5$, $p_{final} = 0.8$, $T = 500$, $W = 500$, $S = 3000$ in the experiments.

Figure 6 (b) shows how the periodical learning rate changes. The learning rate lr at epoch i is defined as

$$lr_i = \begin{cases} \lambda_i \cdot \frac{\text{mod}(i,t)}{T_{warm}} \cdot lr_{max}, & \text{if } \text{mod}(i,t) \leq T_{warm}, \\ 0.5\lambda_i(1 + \cos(\pi \frac{\text{mod}(i,t) - T_{warm}}{T_{max} - T_{warm}})) \cdot lr_{max}, & \text{otherwise.} \end{cases} \quad (5)$$

where T is the decay period of the keep rate p of DropPath, S is the stabilization epoch. $t = T$ when $i < S$, otherwise $t = S$. $\lambda_i = \lambda^{\lfloor \min(i,S)/T \rfloor}$ where λ is a base decaying factor, and $\lfloor \cdot \rfloor$ denotes the floor function. lr_{max} denotes the maximum learning rate, $\text{mod}(x,y)$ denotes the remainder of x/y . The maximum iterations of the cosine annealing function and the number of warmup epochs are denoted by T_{max} and T_{warm} , respectively. In implementation, the maximum learning rate $lr_{max} = 5 \times 10^{-5}$, the base decaying factor for learning rate $\lambda = 0.8$. The period of the cosine function T_{max} and the number of warmup epochs T_{warm} are 1000 and 50, respectively.

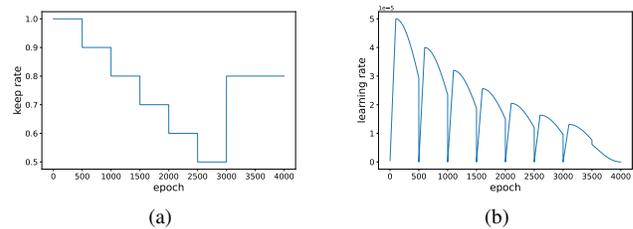


Fig. 6. Supplementary figures. (a) Scheduler of three-phase keep rate. (b) Curve of periodical learning rate.

C. Effect of scaling factor $1/p$ in DropPath

For multi-branch networks, Eq. 1 shows that $\text{DropPath}(\mathbf{x}) = \frac{m}{p} \cdot \mathbf{x}$, $m = \text{Bernoulli}(p)$, where $p \in [0, 1]$ denotes the keep rate, $m = \text{Bernoulli}(p) \in \{0, 1\}$ outputs 1 with probability p and 0 with probability $1 - p$. We consider the module output $\mathbf{y} = \text{DropPath}(\mathbf{x})$ in the training phase, then the expectation of \mathbf{y} given \mathbf{x} is $\mathbb{E}(\mathbf{y}) = p \cdot \frac{1}{p} \cdot \mathbf{x} + (1 - p) \cdot \frac{0}{p} \cdot \mathbf{x} = \mathbf{x}$. In the test phase, DropPath is disabled, so the module output is simply \mathbf{x} and consistent with the expectation in the training phase. If there is no scaling factor $1/p$ in Eq. 1 and $p < 1$, the expectation

TABLE VIII

TEST ACCURACIES OF MODELS TRAINED ON THE DISTILLED DATA OF **TINY-IMAGENET** [80] WITH DIFFERENT IPCS. 3-LAYER CNN IS THE ARCHITECTURE USED FOR DATA DISTILLATION AND IS THE TEACHER MODEL OF KD. NOTE THAT FOR DATM (IPC=50), THE TEACHER MODEL OF RESNET50 IS RESNET18 W/O DP&KD.

DD	IPC	Methods	CNN	ResNet18	AlexNet	VGG11	ResNet50
FRePo [18]	1	Baseline	16.6	15.6 (-1.0)	16.5 (-0.1)	16.6 (+0.0)	13.4 (-3.2)
		w/o DP&KD	17.7 (+1.1)	12.3 (-4.3)	13.7 (-2.9)	14.1 (-2.5)	12.8 (-3.8)
		w/o DP	-	15.8 (-0.8)	16.6 (+0.0)	16.4 (-0.2)	16.6 (+0.0)
		w/o KD	-	12.5 (-4.1)	14.9 (-1.7)	13.6 (-3.0)	11.9 (-4.7)
		Full	-	18.9 (+2.3)	18.5 (+1.9)	18.3 (1.7)	19.1 (+2.5)
	10	Baseline	24.9	24.2 (-0.7)	24.8 (-0.1)	25.2 (+0.3)	24.9 (+0.0)
		w/o DP&KD	23.0 (-1.9)	21.7 (-3.2)	23.8 (-1.1)	24.2 (-0.7)	23.1 (-1.8)
		w/o DP	-	25.4 (+0.5)	25.2 (+0.3)	26.4 (+1.5)	26.9 (+2.0)
w/o KD		-	21.5 (-3.4)	22.4 (-2.5)	24.0 (-0.9)	21.6 (-3.3)	
	Full	-	26.8 (+1.9)	24.9 (+0.0)	26.6 (+1.7)	27.3 (+2.4)	
MTT [20]	1	Baseline	8.8	6.2 (-2.6)	6.7 (-2.1)	7.3 (-1.5)	2.7 (-6.1)
		w/o DP&KD	9.6 (+0.8)	6.1 (-2.7)	8.4 (-0.4)	7.2 (-1.6)	3.2 (-5.6)
		w/o DP	-	6.5 (-2.3)	9.1 (+0.3)	7.9 (-0.9)	3.6 (-5.2)
		w/o KD	-	6.7 (-2.1)	8.1 (-0.7)	6.8 (-2.0)	4.0 (-4.8)
		Full	-	8.1 (-0.7)	9.2 (+0.4)	8.2 (-0.6)	8.2 (-0.6)
	10	Baseline	19.3	17.2 (-2.1)	14.3 (-5.0)	15.1 (-4.2)	11.2 (-8.1)
		w/o DP&KD	20.1(+0.8)	16.6 (-2.7)	18.7 (-0.6)	16.2 (-3.1)	15.2 (-4.1)
		w/o DP	-	17.3 (-2.0)	21.2 (+1.9)	19.9 (+0.6)	18.7 (-0.6)
w/o KD		-	19.0 (-0.3)	17.7 (-1.6)	15.2 (-4.1)	17.7 (-1.6)	
	Full	-	22.6 (+3.3)	21.6 (+2.3)	20.5 (+1.2)	21.8 (+2.5)	
DATM [29]	10	Baseline	16.0	13.8 (-2.2)	16.0 (+0.0)	14.7 (-1.3)	10.6 (-5.4)
		w/o DP&KD	18.4 (+2.4)	13.7 (-2.3)	17.2 (+1.2)	16.4 (+0.4)	13.7 (-2.3)
		w/o DP	-	13.8 (-2.2)	17.8 (+1.8)	18.2 (+2.2)	15.3 (-0.7)
		w/o KD	-	16.3 (+0.3)	15.9 (-0.1)	17.6 (+1.6)	12.7 (-3.3)
		Full	-	17.3 (+1.3)	17.6 (+1.6)	18.2 (+2.2)	15.9 (-0.1)
	50	Baseline	23.5	27.7 (+4.2)	28.4 (+4.9)	24.6 (+1.1)	21.6 (-1.9)
		w/o DP&KD	24.8 (+1.3)	29.2 (+5.7)	29.6 (+6.1)	25.3 (+1.8)	22.8 (-0.7)
		w/o DP	-	28.2 (+4.7)	29.8 (+6.3)	27.7 (+4.2)	26.1 (+2.6)
w/o KD		-	28.5 (+5.0)	28.2 (+4.7)	26.5 (+3.0)	23.2 (-0.2)	
	Full	-	29.9 (+6.4)	30.1 (+6.6)	28.3 (+4.8)	26.9 (+3.4)	

of the module outputs in the training and test phases will be different, which leads to performance degradation.

However, virtual shortcut connection is adopted in single-branch networks, so the implementation of DropPath is different from that in multi-branch networks. In single-branch networks, the DropPath affects both main and shortcut paths. Therefore, given an input \mathbf{x} and the output of the main path \mathbf{x}' , if scaling factor is not considered here, the formulation of DropPath can be rewritten as $\text{DropPath}(\mathbf{x}) = m \cdot \mathbf{x}' + (1-m) \cdot \mathbf{x}$, $m = \text{Bernoulli}(p)$. Assume that the expectations of \mathbf{x}' and \mathbf{x} are the same, the expectation of module output \mathbf{y} given \mathbf{x} is $\mathbb{E}(\mathbf{y}) = m \cdot \mathbb{E}(\mathbf{x}') + (1-m) \cdot \mathbf{x} = m \cdot \mathbf{x} + (1-m) \cdot \mathbf{x} = \mathbf{x}$, which is not changed by m . As a result, the scaling factor is not necessary here.

D. Results on Tiny-ImageNet

The results on Tiny-ImageNet are reported in VIII, respectively. The observations on CIFAR10 and CIFAR100 are analogous to those on and Tiny-ImageNet, which indicates that our method is effective on different datasets.