DATASET CONDENSATION WITH DISTRIBUTION MATCHING

Anonymous authors
Paper under double-blind review

ABSTRACT

Computational cost to train state-of-the-art deep models in many learning problems is rapidly increasing due to more sophisticated models and larger datasets. A recent promising direction to reduce training time is dataset condensation that aims to replace the original large training set with a significantly smaller learned synthetic set while preserving its information. While training deep models on the small set of condensed images can be extremely fast, their synthesis remains computationally expensive due to the complex bi-level optimization and second-order derivative computation. In this work, we propose a simple yet effective dataset condensation technique that requires significantly lower training cost with comparable performance by matching feature distributions of the synthetic and original training images in sampled embedding spaces. Thanks to its efficiency, we apply our method to more realistic and larger datasets with sophisticated neural architectures and achieve a significant performance boost while using larger synthetic training set. We also show various practical benefits of our method in continual learning and neural architecture search.

1 INTRODUCTION

Computational cost for training a single state-of-the-art model in various fields including computer vision and natural language processing doubles every 3.4 months in the deep learning era due to larger models and datasets. The pace is significantly higher than the Moore's Law that the hardware performance would roughly double every other year Amodei et al. (2018). While training a single model can be expensive, designing new deep learning models or applying them to new tasks certainly require substantially more computations, as they involve to train multiple models on the same dataset for many times to verify the design choices, such as loss functions, architectures and hyper-parameters Bergstra & Bengio (2012); Elsken et al. (2019). For instance, (Ying et al., 2019) spent 100 TPU years of computation time conducting an exhaustive neural architecture search on CIFAR10 dataset Krizhevsky et al. (2009), while training the best-performing architectures take only dozens of TPU minutes. Hence, there is a strong demand for techniques that can reduce the computational cost for training multiple models on the same dataset with minimal performance drop. To this end, this paper focuses on lowering the training cost by reducing the training set size.

The traditional solution to reduce the training set size is coreset selection. Typically, coreset selection methods choose samples that are important for training based on heuristic criteria, for example, minimizing distance between coreset and whole-dataset centers Chen et al. (2010); Rebuffi et al. (2017); Castro et al. (2018); Belouadah & Popescu (2020), maximizing the diversity of selected samples Aljundi et al. (2019), discovering cluster centers Wolf (2011); Sener & Savarese (2018), counting the mis-classification frequency Toneva et al. (2019) and choosing samples with the largest negative implicit gradient Borsos et al. (2020). Although coreset selection methods can be very computationally efficient, they have two major limitations. First most methods incrementally and greedily select samples, which doesn't guarantee the global optimal. Second their efficiency is upper bounded by the information in the most representative samples in the original dataset.

An effective way of tackling the information bottleneck is *synthesizing* informative samples rather than selecting from given samples. A recent approach, dataset condensation (or distillation) Wang et al. (2018); Zhao et al. (2021), aims to learn a small synthetic training set so that a model trained on it can obtain comparable testing accuracy to that trained on the original training set. Wang et al. (2018) pose the problem in a learning-to-learning framework by formulating the network parameters as a function of synthetic data and learning them through the network parameters to minimize

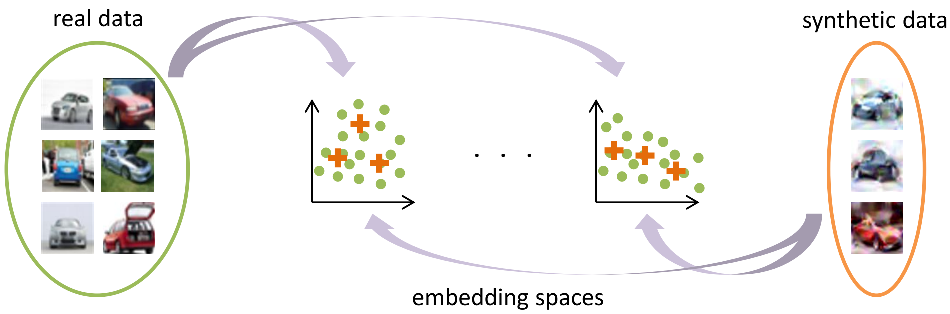

Figure 1: Dataset Condensation with Distribution Matching. We randomly sample real and synthetic data, and then embed them with the randomly sampled deep neural networks. We learn the synthetic data by minimizing the distribution discrepancy between real and synthetic data in these sampled embedding spaces.

the training loss over the original data. An important shortcoming of this method is the expensive optimization procedure that involves optimizing network weights for multiple steps for each outer iteration and unrolling the its recursive computation graph. Zhao et al. (2021) propose a matching method between the gradients w.r.t. the network weights giving real and synthetic training images that successfully avoids the expensive unrolling of the computational graph. (Bohdal et al., 2020; Nguyen et al., 2021) introduced a closed form optimizer by posing the classification task as a ridge regression problem to simplify the inner-loop model optimization. In spite of the recent improvements, the dataset condensation still requires solving an expensive inner loop optimization which jeopardize their goal of reducing train time due to the expensive image synthesis process. For instance, the state-of-the-art Zhao & Bilen (2021) requires 15 hours of GPU time to learn 500 synthetic images from CIFAR10 which equals to 6 times the cost of training a single network on the same dataset. In addition, these methods also require tuning multiple hyperparameters that are different for learning different sizes of synthetic sets.

In this paper, we propose a novel training set synthesis technique that combines the advantages of previous coreset and dataset condensation methods while avoiding their limitations. Like the latter and unlike the former, our method is not limited to individual samples from the original data and can synthesize training images. Like the former and unlike the latter, our method can very efficiently produce a training set and avoid expensive nested-loop optimization. In particular, we pose this task as a distribution matching problem such that the synthetic data are optimized to match the original data distribution in a family of embedding spaces by using the maximum mean discrepancy (MMD) Gretton et al. (2012) measure (see fig. 1). Distance between data distributions are commonly used as the criterion for coreset selection Chen et al. (2010); Wolf (2011); Wang & Ye (2015); Sener & Savarese (2018), however, it has not been used to synthesize training data before. We show that the family of embedding spaces can efficiently be obtained by sampling randomly initialized deep neural networks. Hence, our method is significantly faster (*e.g.* 45× in CIFAR10 for synthesizing 500 images) and involves tuning less hyperparameters than the state-of-the-art Zhao & Bilen (2021) while obtaining comparable or better results. Finally, unlike the dataset condensation methods, our training can be independently run for each class in parallel and its computation load can be distributed. We validate these benefits in two downstream tasks by producing more data-efficient memory for continual learning and generating more representative proxy dataset for accelerating neural architecture search (NAS).

2 METHODOLOGY

2.1 DATASET CONDENSATION PROBLEM

The goal of dataset condensation is to condense the large-scale training set $\mathcal{T} = \{(\boldsymbol{x}_1, y_1), \dots, (\boldsymbol{x}_{|\mathcal{T}|}, y_{|\mathcal{T}|})\}$ with $|\mathcal{T}|$ image and label pairs into a small synthetic set with $|\mathcal{S}|$ synthetic image and label pairs $\mathcal{S} = \{(\boldsymbol{s}_1, y_1), \dots, (\boldsymbol{s}_{|\mathcal{S}|}, y_{|\mathcal{S}|})\}$ so that it is expected to replace $\mathcal{T}$ for training deep neural networks and obtain comparable generalization performance on unseen testing data:

$$\mathbb{E}_{\boldsymbol{x}\sim P_{\mathcal{D}}}[\ell(\phi_{\boldsymbol{\theta}^{\mathcal{T}}}(\boldsymbol{x}), y)] \simeq \mathbb{E}_{\boldsymbol{x}\sim P_{\mathcal{D}}}[\ell(\phi_{\boldsymbol{\theta}^{\mathcal{S}}}(\boldsymbol{x}), y)], \tag{1}$$

where $P_{\mathcal{D}}$ is the real data distribution, ℓ is the loss function (*i.e.* cross-entropy loss), ϕ is a deep neural network parameterized by $\boldsymbol{\theta}$, and $\phi_{\boldsymbol{\theta}^{\mathcal{T}}}$ and $\phi_{\boldsymbol{\theta}^{\mathcal{S}}}$ are the networks that are trained on $\mathcal{T}$ and $\mathcal{S}$ respectively.

Existing solutions. Previous works Wang et al. (2018); Sucholutsky & Schonlau (2019); Such et al. (2020); Bohdal et al. (2020); Nguyen et al. (2021) formulate the dataset condensation as a learning-to-learn problem, pose the network parameters $\theta^{\mathcal{S}}$ as a function of synthetic data $\mathcal{S}$ and aim to obtain the optimum $\mathcal{S}$ that minimizes the training loss over the original data $\mathcal{T}$:

$$\mathcal{S}^* = \arg\min_{\mathcal{S}} \mathcal{L}^{\mathcal{T}}(\boldsymbol{\theta}^{\mathcal{S}}(\mathcal{S})) \quad \text{subject to} \quad \boldsymbol{\theta}^{\mathcal{S}}(\mathcal{S}) = \arg\min_{\boldsymbol{\theta}} \mathcal{L}^{\mathcal{S}}(\boldsymbol{\theta}). \tag{2}$$

Recently the authors of Zhao et al. (2021); Zhao & Bilen (2021) show that a similar goal can be achieved by matching gradients of a network w.r.t. to the synthetic training data and real training data respectively while optimizing the network parameter $\boldsymbol{\theta}$ with the synthetic data $\mathcal{S}$ alternatively:

$$\begin{aligned} \mathcal{S}^* = \arg\min_{\mathcal{S}} \mathrm{E}_{\boldsymbol{\theta}_0 \sim P_{\boldsymbol{\theta}_0}} [\sum_{t=0}^{T-1} D(\nabla_{\boldsymbol{\theta}} \mathcal{L}^{\mathcal{S}}(\boldsymbol{\theta}_t), \nabla_{\boldsymbol{\theta}} \mathcal{L}^{\mathcal{T}}(\boldsymbol{\theta}_t))] \\ \text{subject to} \quad \boldsymbol{\theta}_{t+1} \leftarrow \texttt{opt-alg}_{\boldsymbol{\theta}}(\mathcal{L}^{\mathcal{S}}(\boldsymbol{\theta}_t), \varsigma_{\boldsymbol{\theta}}, \eta_{\boldsymbol{\theta}}), \end{aligned} \tag{3}$$

where $P_{\boldsymbol{\theta}_0}$ is the distribution of parameter initialization, T is the outer-loop iteration for updating synthetic data, $\varsigma_{\boldsymbol{\theta}}$ is the inner-loop iteration for updating network parameters and $\eta_{\boldsymbol{\theta}}$ is the parameter learning rate.

Dilemma. Solving the problems in eq. (2) and eq. (3) involve solving an expensive bi-level optimization: first optimizing the model $\boldsymbol{\theta}^{\mathcal{S}}$ or $\boldsymbol{\theta}_t$ at the inner loop, then optimizing the synthetic data $\mathcal{S}$, which also includes second-order derivative computation, at the outer loop. For example, training 50 images/class synthetic set $\mathcal{S}$ by using the method in Zhao et al. (2021) requires 500K epochs of updating network parameters $\boldsymbol{\theta}_t$ on $\mathcal{S}$, in addition to the 50K updating of $\mathcal{S}$. Furthermore, Zhao et al. (2021) need to tune the hyper-parameters of the outer and inner loop optimization (*i.e.* how many steps to update $\mathcal{S}$ and $\boldsymbol{\theta}_t$) for different learning settings, which probably means more training time for learning larger synthetic sets.

2.2 Dataset Condensation with Distribution Matching

Our goal is to synthesize data that can accurately approximate the data distribution of real training data in a similar spirit to coreset techniques (*e.g.* Welling (2009); Sener & Savarese (2018)). However, to this end, we do not limit our method to select the informative samples but to synthesize them as in Wang et al. (2018); Zhao et al. (2021). As the training images are typically very high dimensional, estimating the real data distribution $P_{\mathcal{D}}$ can be expensive and inaccurate. Instead, we assume that each training image $\boldsymbol{x} \in \Re^d$ can be embedded into a lower dimensional space by using a family of parametric functions $\psi_{\boldsymbol{\vartheta}} : \Re^d \rightarrow \Re^{d'}$ where $d' \ll d$ and $\boldsymbol{\vartheta}$ are the parameters. In other words, each embedding function ψ can be seen as a partial interpretation of data, while their combination provides a complete one.

Now we can estimate the distance between the real and synthetic data distribution with commonly used maximum mean discrepancy (MMD) Gretton et al. (2012):

$$\sup_{\|\psi_{\boldsymbol{\vartheta}}\|_{\mathcal{H}} \leq 1} (\mathbb{E}[\psi_{\boldsymbol{\vartheta}}(\mathcal{T})] - \mathbb{E}[\psi_{\boldsymbol{\vartheta}}(\mathcal{S})]) \tag{4}$$

where $\mathcal{H}$ is reproducing kernel Hilbert space. As we do not have access to ground-truth data distributions, we use the empirical estimate of the MMD:

$$\mathbb{E}_{\boldsymbol{\vartheta} \sim P_{\boldsymbol{\vartheta}}} \| \frac{1}{|\mathcal{T}|} \sum_{i=1}^{|\mathcal{T}|} \psi_{\boldsymbol{\vartheta}}(\boldsymbol{x}_i) - \frac{1}{|\mathcal{S}|} \sum_{j=1}^{|\mathcal{S}|} \psi_{\boldsymbol{\vartheta}}(\boldsymbol{s}_j) \|^2 \tag{5}$$

where $P_{\boldsymbol{\vartheta}}$ is the distribution of network parameters.

Following Zhao & Bilen (2021), we also apply differentiable Siamese augmentation $\mathcal{A}(\cdot, \omega)$ to real and synthetic data, where $\omega \sim \Omega$ is the augmentation parameter such as the rotation degree. Thus, the learned synthetic data can benefit from semantic-preserved transforms and learn prior knowledge for training deep neural networks with data augmentation. Finally, we solve the following optimization problem:

$$\min_{\mathcal{S}} \mathbb{E}_{\boldsymbol{\vartheta}\sim P_{\boldsymbol{\vartheta}},\omega\sim\Omega} \| \frac{1}{|\mathcal{T}|}\sum_{i=1}^{|\mathcal{T}|}\psi_{\boldsymbol{\vartheta}}(\mathcal{A}(\boldsymbol{x}_i,\omega)) - \frac{1}{|\mathcal{S}|}\sum_{j=1}^{|\mathcal{S}|}\psi_{\boldsymbol{\vartheta}}(\mathcal{A}(\boldsymbol{s}_j,\omega))\|^2. \tag{6}$$

We learn the synthetic data $\mathcal{S}$ by minimizing the discrepancy between two distributions in various embedding spaces by sampling $\boldsymbol{\vartheta}$. In contrast to the existing formulations (see eq. (2) and eq. (3)) that involve optimizing network parameters $\boldsymbol{\theta}$ and the synthetic data $\mathcal{S}$, our method requires only optimizing $\mathcal{S}$ and avoids expensive nested loop optimization.

Note that, as we target image classification problems, we minimizes the discrepancy between the real and synthetic samples of the same class only. We assume that each real training sample is labelled and assign a label to each synthetic sample.

Discussion. The family of embedding functions $\psi_{\boldsymbol{\vartheta}}$ can be designed in different ways. Here we use a deep neural network with different random initializations rather than sampling its parameters from a set of pre-trained networks which is computationally expensive to obtain. We also experimentally validate that our random initialization strategy produces better or comparable results with the more expensive strategy of using pretrained networks. However, one may still question why randomly initialized networks provide meaningful embeddings for the real data distribution. Here we list two reasons based on the observations from previous work. First randomly initialized networks are reported to produce powerful representations for multiple computer vision tasks Saxe et al. (2011). Second, the embeddings extracted from such networks are showed to perform a distance-preserving embedding of the data, *i.e.* smaller distances between samples of same class and larger distances across samples of different classes, in Giryes et al. (2016).

2.3 Training Algorithm

We depict the mini-batch based training algorithm in Alg. 1. We train the synthetic data for K iterations. In each iteration, we randomly sample the model $\psi_{\boldsymbol{\vartheta}}$ with parameter $\boldsymbol{\vartheta} \sim P_{\boldsymbol{\vartheta}}$. Then, we sample a pair of real and synthetic data batches ($B_c^{\mathcal{T}} \sim \mathcal{T}$ and $B_c^{\mathcal{S}} \sim \mathcal{S}$) and augmentation parameter $\omega_c \sim \Omega$ for every class c. The mean discrepancy between the augmented real and synthetic batches of every class is calculated and then summed as loss $\mathcal{L}$. The synthetic data $\mathcal{S}$ is updated by minimizing $\mathcal{L}$ with gradient descent and learning rate η.

Algorithm 1: Dataset condensation with distribution matching

Input: Training set $\mathcal{T}$

Required: Randomly initialized set of synthetic samples $\mathcal{S}$ for C classes, probability distribution over parameters $P_{\boldsymbol{\vartheta}}$, deep neural network $\psi_{\boldsymbol{\vartheta}}$, differentiable augmentation $\mathcal{A}_\omega$ parameterized with ω, augmentation parameter distribution Ω, training iterations K, learning rate η.

for $k = 0, \cdots, K-1$ **do**

- Sample $\boldsymbol{\vartheta} \sim P_{\boldsymbol{\vartheta}}$
- Sample minibatch pairs $B_c^{\mathcal{T}} \sim \mathcal{T}$ and $B_c^{\mathcal{S}} \sim \mathcal{S}$ and $\omega_c \sim \Omega$ for every class c
- Compute $\mathcal{L} = \sum_{c=0}^{C-1} \| \frac{1}{|B_c^{\mathcal{T}}|}\sum_{(\boldsymbol{x},y)\in B_c^{\mathcal{T}}}\psi_{\boldsymbol{\vartheta}}(\mathcal{A}_{\omega_c}(\boldsymbol{x})) - \frac{1}{|B_c^{\mathcal{S}}|}\sum_{(\boldsymbol{s},y)\in B_c^{\mathcal{S}}}\psi_{\boldsymbol{\vartheta}}(\mathcal{A}_{\omega_c}(\boldsymbol{s}))\|^2$
- Update $\mathcal{S} \leftarrow \mathcal{S} - \eta\nabla_{\mathcal{S}}\mathcal{L}$

Output: $\mathcal{S}$

3 Experiments

3.1 Experimental Settings

Datasets. First we evaluate the classification performance of deep networks that are trained on the synthetic images generated by our method. We conduct experiments on four datasets including MNIST LeCun et al. (1990), CIFAR10, CIFAR100 Krizhevsky et al. (2009) and TinyImageNet Le & Yang (2015). MNIST consists of 60,000 training and 10,000 testing 28×28 gray-scale images

	Img/Cls	Ratio %	Coreset Selection			Training Set Synthesis					Whole Dataset
			Random	Herding	Forgetting	DD†	LD†	DC	DSA	*DM*	
MNIST	1	0.017	64.9±3.5	89.2±1.6	35.5±5.6		60.9±3.2	**91.7±0.5**	88.7±0.6	89.7±0.6	99.6±0.0
	10	0.17	95.1±0.9	93.7±0.3	68.1±3.3	79.5±8.1	87.3±0.7	97.4±0.2	**97.8±0.1**	97.5±0.1	
	50	0.83	97.9±0.2	94.8±0.2	88.2±1.2	-	93.3±0.3	98.8±0.2	**99.2±0.1**	98.6±0.1	
CIFAR10	1	0.02	14.4±2.0	21.5±1.2	13.5±1.2	-	25.7±0.7	**28.3±0.5**	**28.8±0.7**	26.0±0.8	84.8±0.1
	10	0.2	26.0±1.2	31.6±0.7	23.3±1.0	36.8±1.2	38.3±0.4	44.9±0.5	**52.1±0.5**	48.9±0.6	
	50	1	43.4±1.0	40.4±0.6	23.3±1.1	-	42.5±0.4	53.9±0.5	60.6±0.5	**63.0±0.4**	
CIFAR100	1	0.2	4.2±0.3	8.4±0.3	4.5±0.2	-	11.5±0.4	12.8±0.3	**13.9±0.3**	11.4±0.3	56.2±0.3
	10	2	14.6±0.5	17.3±0.3	15.1±0.3	-	-	25.2±0.3	**32.3±0.3**	29.7±0.3	
	50	10	30.0±0.4	33.7±0.5	30.5±0.3	-	-	-	42.8±0.4	**43.6±0.4**	
TinyImageNet	1	0.2	1.4±0.1	2.8±0.2	1.6±0.1	-	-	-	-	**3.9±0.2**	37.6±0.4
	10	2	5.0±0.2	6.3±0.2	5.1±0.2	-	-	-	-	**12.9±0.4**	
	50	10	15.0±0.4	16.7±0.3	15.0±0.3	-	-	-	-	**24.1±0.3**	

Table 1: Comparing to coreset selection and training set synthesis methods. We first learn the synthetic data and then evaluate them by training neural networks from scratch and testing on real testing data. The testing accuracies (%) are reported. Img/Cls: image(s) per class. Ratio (%): the ratio of condensed set size to the whole training set size. Note: DD† and LD† use different architectures *i.e.* LeNet for MNIST and AlexNet for CIFAR10. The rest methods all use ConvNet.

of 10 classes. CIFAR10 and CIFAR100 contain 50,000/10,000 32×32 training/testing images from 10 and 100 object categories respectively. We also evaluate our method on TinyImageNet that contains 100,000 training and 10,000 testing images from 200 categories with a higher resolution 64×64. This dataset has not been previously studied by the prior dataset condensation methods and is significantly more challenging than the MNIST, CIFAR10/100 datasets due to its large number of classes and larger image resolution.

Experimental Settings. We first learn 1/10/50 image(s) per class synthetic sets for all datasets by using the same ConvNet architecture in Zhao et al. (2021). Then, we use the learned synthetic sets to train randomly initialized ConvNets from scratch and evaluate them on real test data. The default ConvNet includes three repeated convolutional blocks, and each block involves a 128-kernel convolution layer, instance normalization layer Ulyanov et al. (2016), ReLU activation function Nair & Hinton (2010) and average pooling. In each experiment, we learn one synthetic set and use it to test 20 randomly initialized networks. We repeat each experiment for 5 times and report the mean testing accuracy of the 100 trained networks. We also do cross-architecture experiments in section 3.3 where we learn the synthetic set on one network architecture and use them to train networks with different architectures. We will release the code and learned synthetic data before the final version.

Hyper-parameters. We use a fixed learning rate 1 for optimizing synthetic images for all 1/10/50 images/class learning on all datasets. When learning larger synthetic sets such as 100/200/500 images per class, we can use larger learning rate (*e.g.* 10) due to the smaller distribution matching loss. We train synthetic images for 20,000 iterations. The mini-batch size for sampling real data is 256. All synthetic images of a class are used to compute the class mean. We use the same augmentation strategy as Zhao & Bilen (2021). Note that our method involves fewer hyper-parameters than the prior work Wang et al. (2018); Zhao et al. (2021); Zhao & Bilen (2021) thanks to its simple optimization.

3.2 Comparison to the State-of-the-art

Competitors. We compare our method to three standard coreset selection methods, namely, Random Selection, Herding Chen et al. (2010); Rebuffi et al. (2017); Castro et al. (2018); Belouadah & Popescu (2020) and Forgetting Toneva et al. (2019). Random selection means randomly selecting real images as the coreset. Herding method greedily adds samples into the coreset so that the mean vector is approaching the whole dataset mean. Toneva et al. (2019) count how many times a training sample is learned and then forgotten during network training. The samples that are less forgetful can be dropped. We also compare our method to four state-of-the-art training set synthesis methods, namely, DD Wang et al. (2018), LD Bohdal et al. (2020), DC Zhao et al. (2021) and DSA Zhao & Bilen (2021).

Small-scale Dataset Condensation. Here we evaluate our method in three small scale datasets including MNIST, CIFAR10 and CIFAR100 and report the results in Table 1. Among the coreset

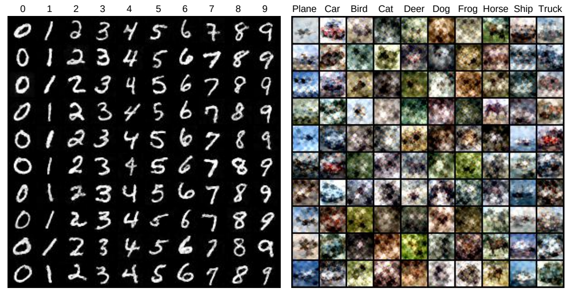

Figure 2: Visualization of generated 10 images per class synthetic sets of MNIST and CIFAR10 datasets.

selection methods, Herding performances the best in most settings. Especially, when small synthetic sets are learned, Herding method performs significantly better. For example, Herding achieves 8.4% testing accuracy when learning 1 image/class synthetic set on CIFAR100, while Random and Forgetting obtains only 4.2% and 4.5% testing accuracies respectively.

Training set synthesis methods have clear superiority over coreset selection methods, as the synthetic training data is not limited to a set of real images. Best results are obtained either by DSA or our method. While DSA produces more data-efficient samples with a small number of synthetic samples (1/10 image(s) per class), our method outperforms DSA at 50 images/class setting in CIFAR10 and CIFAR100. The possible reason is that the inner-loop model optimization in DSA with limited number of steps is more effective to fit the network parameters on smaller synthetic data (see eq. (3)). In case of bigger learned synthetic data, the solution obtained in the inner-loop becomes less accurate as it can use only limited number of steps to keep the algorithm scalable. In contrast, our method is robust to increasing synthetic data size, can be efficiently optimized significantly faster than DSA.

We also compare to Kernel Inducing Point (KIP) Nguyen et al. (2021), a recent dataset distillation method. Nguyen et al. (2021) learn synthetic images on kernels and evaluate them on one-layer fully connected network. According to their reported results, the 50 images/class synthetic sets learned by our method (98.6% and 63.0%) overwhelm their 500 images/class synthetic sets (98.0% and 50.1%) on MNIST and CIFAR10 datasets respectively.

TinyImageNet Condensation. We also evaluate our method on a bigger and more challenging dataset, TinyImageNet, due to higher image resolution and more (and diverse) classes. The prior training set synthesis techniques have not been evaluated in this dataset. Unfortunately these methods do not scale to TinyImageNet in terms of training time and memory usage due to the complex nested loop optimization. So we train only our model and report its performance for learning three condensed sets (1/10/50 images/class synthetic sets) in this dataset which in total takes around 27 hours on one Tesla V100 GPU. Different from other datasets, we use ConvNet with 4 blocks for TinyImageNet to adjust to the larger input size. Our method achieves 3.9%, 12.9% and 24.1% testing accuracies when learning 1, 10 and 50 images/class synthetic sets for TinyImageNet and recovers 60% classification performance of the baseline that is trained on the whole original training set with only 10% of data. Our method significantly outperforms the best coreset selection method - Herding, which obtains 2.8%, 6.3% and 16.7% testing accuracies.

Visualization. The learned synthetic images of MNIST and CIFAR10 are visualized in Figure 2. We find that the synthetic MNIST images are clear and noise free, while the number images synthesized by previous methods contain obvious noise and some unnatural strokes. The synthetic images for CIFAR10 dataset are also visually recognizable and diverse. It is easy to distinguish the background and foreground object. More visualization results are provided in the appendix.

Figure 3 depict the feature distribution of the (50 images/class) synthetic sets learned by DC, DSA and our method (DM). We use a network trained on the whole training set to extract features and visualize the features with T-SNE Van der Maaten & Hinton (2008). We find that the synthetic images learned by DC and DSA cannot cover the real image distribution. In contrast, our synthetic images successfully cover the real image distribution. Furthermore, less outlier synthetic samples are produced by in our method.

Learning with Batch Normalization. Zhao et al. (2021) showed that instance normalization (Ulyanov et al., 2016) works better than batch normalization (BN) Ioffe & Szegedy (2015) when

	InstanceNorm		BatchNorm	
	DSA	DM	DSA	DM
CIFAR10	60.6±0.5	**63.0±0.4**	59.9±0.8	**65.2±0.4**
CIFAR100	42.8±0.4	**43.6±0.4**	44.6±0.5	**48.0±0.4**
TinyImageNet	-	**24.1±0.3**	-	**28.2±0.5**

Table 2: 50 images/class learning with Batch Normalization.

	C\T	ConvNet	AlexNet	VGG	ResNet
DSA	ConvNet	59.9±0.8	53.3±0.7	51.0±1.1	47.3±1.0
DM	ConvNet	65.2±0.4	61.3±0.6	59.9±0.8	57.0±0.9
	AlexNet	60.5±0.4	59.8±0.6	58.9±0.4	54.6±0.7
	VGG	54.2±0.6	52.6±1.0	52.8±1.1	49.1±1.0
	ResNet	52.2±1.0	50.9±1.4	49.6±0.9	52.2±0.4

Table 3: Cross-architecture testing performance (%). The 50 img/cls synthetic set is learned on one architecture (C), and then tested on another architecture (T).

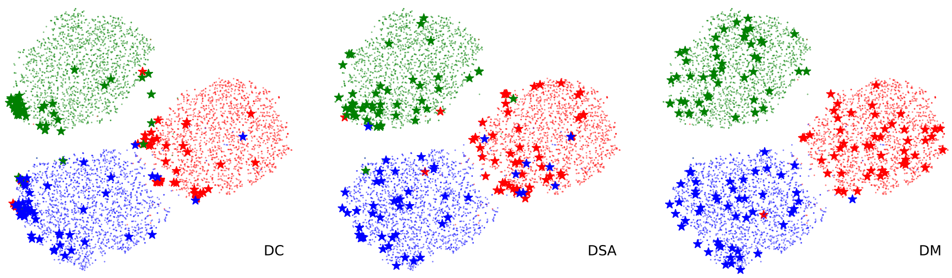

Figure 3: Distributions of synthetic images learned by DC, DSA and DM. The red, green and blue points are the real images of first three classes in CIFAR10. The stars are corresponding learned synthetic images.

learning small synthetic sets because the synthetic data number is too small to calculate stable running mean and standard derivation (std). When learning with batch normalization, they first pre-set the BN mean and std using many real training data and then freeze them for synthetic data. Thus, the inaccurate mean and std will make optimization difficult. In contrast, we estimate running mean and std by inputting augmented synthetic data from all classes. Hence, our method benefits from the true mean and std of synthetic data. Table 2 show that using ConvNet with BN can further improve our performance. Specifically, our method with BN achieves 65.2%, 48.0% and 28.2% testing accuracies when learning 50 images/class synthetic sets on CIFAR10, CIFAR100 and TinyImageNet respectively, which means 2.2%, 4.4% and 4.1% improvements over our method with the default instance normalization, and also outperforms DSA with BN by 5.3% and 3.4% on the CIFAR10 and CIFAR100 datasets respectively.

Training Cost Comparison Our method is significantly more efficient than those bi-level optimization based methods. Without loss of generality, we compare the training time of our method and DSA in the setting of learning 50 images/class synthetic data on CIFAR10 dataset. Figure 4 shows that our method needs less than 20 minutes to reach the performance of DSA trained for 15 hours, which means less than 2.2% training cost. Note that we run the two methods in the same computation environment with one GTX 1080 GPU.

Learning Larger Synthetic Sets We show that our method can also be used to learn larger synthetic sets, while the bi-level optimization based methods typically requires more training time and elaborate hyper-parameter tuning for larger settings. Figure 5 compares our method to random selection baseline in CIFAR10 in terms of absolute and relative performance w.r.t. whole dataset training performance. Clearly our method outperforms random baseline at all operating points which means that our synthetic set is more data-efficient. We see that the performance gap between the two methods narrows when we learn larger synthetic data. This is somehow expected, as randomly selecting more samples will approach the whole dataset training which can be considered as the upper-bound.

3.3 Cross-architecture Generalization

Zhao et al. (2021); Zhao & Bilen (2021) verified the cross-architecture generalization ability of synthetic data in an easy setting - learning 1 image/class for MNIST dataset. In this paper, we implement a more challenging cross-architecture experiment - learning 50 images/class for CIFAR10 dataset. In Table 3, the synthetic data are learned with one architecture (denoted as C) and then be evaluated on another architecture (denoted as T) by training a model from scratch and testing on real testing data. We test several sophisticated neural architectures namely ConvNet, AlexNet Krizhevsky et al. (2012), VGG-11 Simonyan & Zisserman (2014) and ResNet-18 He et al. (2016). Batch Normalization is used in all architectures.

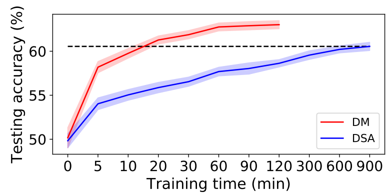

Figure 4: Training time comparison to DSA when learning 50 img/cls synthetic sets on CIFAR10.

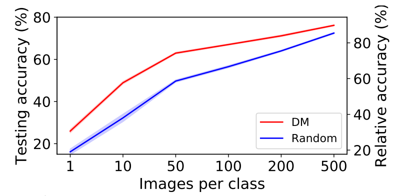

Figure 5: Performance of learning larger synthetic sets on CIFAR10.

	Random	10-20	20-30	30-40	40-50	50-60	60-70	≥70	All
1	26.0±0.8	26.2±0.7	25.9±0.7	26.1±0.8	26.7±0.5	26.8±0.6	27.3±0.7	26.5±0.9	26.4±0.7
10	48.9±0.6	48.7±0.6	48.1±0.7	50.7±0.5	51.1±0.6	49.9±0.5	48.6±0.7	48.2±0.8	50.7±0.6
50	63.0±0.4	62.7±0.4	62.1±0.5	62.8±0.4	63.0±0.4	61.9±0.5	60.6±0.5	60.0±0.5	62.5±0.4

Table 4: The performance of synthetic data learned on CIFAR10 dataset with different network distributions. These networks are trained on the whole training set and grouped based on the validation accuracy (%).

Table 3 shows that learning and evaluating synthetic set on ConvNet achieves the best performance 65.2%. Comparing with DSA, the synthetic data learned by our method with ConvNet have better generalization performance than that learned by DSA with the ConvNet. Specifically, our method outperforms DSA by 8.0%, 8.9% and 9.7% when testing with AlexNet, VGG and ResNet respectively. The learning of synthetic set can be worse with more sophisticated architecture such as ResNet. It is reasonable that the synthetic data fitted on sophisticated architecture will contain some bias that doesn't exist in other architectures, therefore cause worse cross-architecture generalization performance. We also find that the evaluation of the same synthetic set on more sophisticated architectures will be worse. The reason may be that sophisticated architectures are under-fitted using small synthetic set.

3.4 Ablation Study on Network Distribution

Here we study the effect of using different network distributions while learning 1/10/50 image(s)/class synthetic sets on CIFAR10 with ConvNet architecture. Besides sampling randomly initialized network parameters, we also generate a set of network initializations that are sampled from networks pre-trained on the original training set. In particular, we train 1,000 ConvNets with different random initializations on the whole original training set and also store their intermediate states. We roughly divide these networks into nine groups according to their validation accuracies, sample from each group while minimizing the objective in eq. (6) to learn the synthetic data and use them to train randomly initialized neural networks. Interestingly we see in Table 4 that our method works well with all nine network distributions and the performance variance is small. We also visualize the synthetic sets learned with different network distributions in the appendix.

3.5 Continual Learning

We also use our method to store more efficient training samples in the memory for relieving the catastrophic forgetting problem in continual (incremental) learning Rebuffi et al. (2017). We set up the baseline based on GDumb Prabhu et al. (2020) which stores training samples in memory greedily and keeps class-balance. The model is trained from scratch on the latest memory only. Hence, the continual learning performance completely depends on the quality of the memory construction. We compare our memory construction method *i.e.* training set condensation to the *random* selection that is used in Prabhu et al. (2020), *herding* Chen et al. (2010); Rebuffi et al. (2017); Castro et al. (2018); Belouadah & Popescu (2020) and *DSA* Zhao & Bilen (2021). We implement class-incremental learning on CIFAR100 dataset with an increasing memory budget of 20 images/class. We implement 5 and 10 step learning, in which we randomly and evenly split the 100 classes into 5 and 10 learning steps *i.e.* 20 and 10 classes per step respectively. The default ConvNet is used in this experiment.

As depicted in Figure 6 and Figure 7, we find that our method GDumb + DM outperforms others in both two settings, which means that our method can produce the best condensed set as the memory. The final performances of ours, DSA, herding and random are 34.4%, 31.7%, 28.2% and 24.8% in 5-step learning and 34.6%, 30.5%, 27.4% and 24.8% in 10-step learning. We find that ours and random selection performances are not influenced by how the classes are split namely how many new

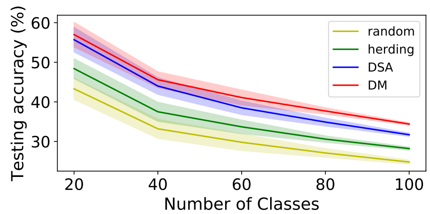

Figure 6: 5-step class-incremental learning.

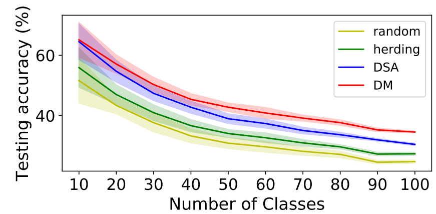

Figure 7: 10-step class-incremental learning.

	Random	DSA	DM	Early-stopping	Whole Dataset
Performance (%)	84.0	82.6	82.8	**84.3**	85.9
Correlation	-0.04	0.68	**0.76**	0.11	1.00
Time cost (min)	142.6	142.6	142.6	142.6	3580.2
Storage (imgs)	**500**	**500**	**500**	5×10^4	5×10^4

Table 5: We implement neural architecture search on CIFAR10 dataset with the search space of 720 ConvNets.

training classes and images occur in each learning step, because both two methods learn/generate the sets independently for each class. However, DSA and herding methods perform worse when the training class and image numbers become smaller in every step. The reason is that DSA and herding needs to learn/generate sets based on the model(s) trained on the current training data, which is influenced by the data split. More implementation details can be found in appendix.

3.6 Neural Architecture Search

The synthetic sets can also be used as a proxy set to accelerate model evaluation in Neural Architecture Search (NAS) Elsken et al. (2019). Following Zhao et al. (2021), we implement NAS on CIFAR10 with the search space of 720 ConvNets varying in network depth, width, activation, normalization and pooling layers. Please refer to Zhao et al. (2021) for more details. We train all architectures on the learned 50 images/class synthetic set, *i.e*. 1% size of the whole dataset, from scratch and then rank them based on the accuracy on a small validation set. We compare to *random*, *DSA* and *early-stopping* methods. The same size of real images are selected as the proxy set in *random*. *DSA* means that we use the synthetic set learned by DSA in the same setting. In *early-stopping*, we use the whole training set to train the model but with the same training iterations like training on the proxy datasets. Therefore, all these methods have the same training time. We train models on the proxy sets for 200 epochs and whole dataset for 100 epochs. Then, the best model is selected based on validation accuracies obtained by different methods. The Spearman's rank correlation between performances of proxy-set and whole-dataset training is computed for the top 5% architectures selected by the proxy-set.

The NAS results are provided in Table 5. Although the architecture selected by early-stopping achieves the best performance (84.3%), its performance rank correlation (0.11) is remarkably lower than DSA (0.68) and DM (0.76). In addition, early-stopping needs to use the whole training set, while other proxy-set methods need only 500 training samples. The performance rank correlation of Random (-0.04) is too low to provide a reliable ranking for the architectures. Our method (DM) achieves the highest performance rank correlation (0.76), which means that our method can produce reliable ranking for those candidate architectures while using only around $\frac{1}{25}$ training time of whole dataset training. More implementation details and analysis can be found in appendix.

4 Conclusion

In this paper, we propose an efficient training set synthesis method based on distribution matching. The synthetic data of different classes can be learned independently and in parallel. Thanks to its efficiency, we can apply our method to more challenging dataset - TinyImageNet, and learn larger and higher resolution synthetic sets. Our method is 45 times faster than the state-of-the-art for learning 50 images/class synthetic set on CIFAR10. We also empirically prove that our method can produce more informative memory for continual learning and better proxy set for speeding up model evaluation in neural architecture search.

Ethics Statement

This work is for general purpose of dataset condensation. The goal of our research is to reduce the training cost while achieving comparable testing accuracies. Our research doesn't involve anything about discrimination, bias, fairness, privacy, etc. We don't violet any ethics code.

Reproducibility Statement

We have specified the key implementation details, such as activation, normalization, learning rate, training iteration and batch size, in Section 3.1. We also supplement more implementation details about specific experiments, such as continual learning and neural architecture search, in Appendix A. We will release the code and learned synthetic data before the final version.

References

Rahaf Aljundi, Min Lin, Baptiste Goujaud, and Yoshua Bengio. Gradient based sample selection for online continual learning. In *Advances in Neural Information Processing Systems*, pp. 11816–11825, 2019.

Dario Amodei, Danny Hernandez, Girish Sastry, Jack Clark, Greg Brockman, and Ilya Sutskever. Ai and compute. In *OpenAI Blog*, 2018. URL `openai.com/blog/ai-and-compute/`.

Eden Belouadah and Adrian Popescu. Scail: Classifier weights scaling for class incremental learning. In *The IEEE Winter Conference on Applications of Computer Vision*, 2020.

James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of machine learning research*, 13(Feb):281–305, 2012.

Ondrej Bohdal, Yongxin Yang, and Timothy Hospedales. Flexible dataset distillation: Learn labels instead of images. *Neural Information Processing Systems Workshop*, 2020.

Zalán Borsos, Mojmír Mutnỳ, and Andreas Krause. Coresets via bilevel optimization for continual learning and streaming. *NeurIPS*, 2020.

Francisco M Castro, Manuel J Marín-Jiménez, Nicolás Guil, Cordelia Schmid, and Karteek Alahari. End-to-end incremental learning. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 233–248, 2018.

Yutian Chen, Max Welling, and Alex Smola. Super-samples from kernel herding. *The Twenty-Sixth Conference Annual Conference on Uncertainty in Artificial Intelligence*, 2010.

Thomas Elsken, Jan Hendrik Metzen, Frank Hutter, et al. Neural architecture search: A survey. *J. Mach. Learn. Res.*, 20(55):1–21, 2019.

Raja Giryes, Guillermo Sapiro, and Alex M Bronstein. Deep neural networks with random gaussian weights: A universal classification strategy? *IEEE Transactions on Signal Processing*, 64(13): 3444–3457, 2016.

Arthur Gretton, Karsten M Borgwardt, Malte J Rasch, Bernhard Schölkopf, and Alexander Smola. A kernel two-sample test. *The Journal of Machine Learning Research*, 13(1):723–773, 2012.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *ArXiv*, abs/1502.03167, 2015.

Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pp. 1097–1105, 2012.

Ya Le and Xuan Yang. Tiny imagenet visual recognition challenge. *CS 231N*, 7(7):3, 2015.

Yann LeCun, John S Denker, and Sara A Solla. Optimal brain damage. In *Advances in neural information processing systems*, pp. 598–605, 1990.

Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pp. 807–814, 2010.

Timothy Nguyen, Zhourong Chen, and Jaehoon Lee. Dataset meta-learning from kernel-ridge regression. In *International Conference on Learning Representations*, 2021. URL `https://openreview.net/forum?id=l-PrrQrK0QR`.

Ameya Prabhu, Philip HS Torr, and Puneet K Dokania. Gdumb: A simple approach that questions our progress in continual learning. In *European Conference on Computer Vision*, pp. 524–540. Springer, 2020.

Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H Lampert. icarl: Incremental classifier and representation learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2001–2010, 2017.

Andrew M Saxe, Pang Wei Koh, Zhenghao Chen, Maneesh Bhand, Bipin Suresh, and Andrew Y Ng. On random weights and unsupervised feature learning. In *Icml*, 2011.

Ozan Sener and Silvio Savarese. Active learning for convolutional neural networks: A core-set approach. *ICLR*, 2018.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Felipe Petroski Such, Aditya Rawal, Joel Lehman, Kenneth O Stanley, and Jeff Clune. Generative teaching networks: Accelerating neural architecture search by learning to generate synthetic training data. *International Conference on Machine Learning (ICML)*, 2020.

Ilia Sucholutsky and Matthias Schonlau. Soft-label dataset distillation and text dataset distillation. *arXiv preprint arXiv:1910.02551*, 2019.

Mariya Toneva, Alessandro Sordoni, Remi Tachet des Combes, Adam Trischler, Yoshua Bengio, and Geoffrey J Gordon. An empirical study of example forgetting during deep neural network learning. *ICLR*, 2019.

Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Instance normalization: The missing ingredient for fast stylization. *arXiv preprint arXiv:1607.08022*, 2016.

Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(11), 2008.

Tongzhou Wang, Jun-Yan Zhu, Antonio Torralba, and Alexei A Efros. Dataset distillation. *arXiv preprint arXiv:1811.10959*, 2018.

Zheng Wang and Jieping Ye. Querying discriminative and representative samples for batch mode active learning. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 9(3):1–23, 2015.

Max Welling. Herding dynamical weights to learn. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pp. 1121–1128. ACM, 2009.

G W Wolf. Facility location: concepts, models, algorithms and case studies. 2011.

Chris Ying, Aaron Klein, Eric Christiansen, Esteban Real, Kevin Murphy, and Frank Hutter. Nas-bench-101: Towards reproducible neural architecture search. In *International Conference on Machine Learning*, pp. 7105–7114. PMLR, 2019.

Sangdoo Yun, Dongyoon Han, Seong Joon Oh, Sanghyuk Chun, Junsuk Choe, and Youngjoon Yoo. Cutmix: Regularization strategy to train strong classifiers with localizable features. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 6023–6032, 2019.

Bo Zhao and Hakan Bilen. Dataset condensation with differentiable siamese augmentation. In *International Conference on Machine Learning*, 2021.

Bo Zhao, Konda Reddy Mopuri, and Hakan Bilen. Dataset condensation with gradient matching. In *International Conference on Learning Representations*, 2021. URL `https://openreview.net/forum?id=mSAKhLYLSsl`.

A Implementation details

A.1 Dataset Condensation

DSA Results. As Zhao & Bilen (2021) didn't report 50 images/class learning performance on CIFAR100, we obtain the result in Table 1 by running their released code and coarsely searching the hyper-parameters (outer and inner loop steps). Then, we set both outer and inner loop to be 10 steps. The rest hyper-parameters are the default ones in their released code. To obtain the DSA results with batch normalization in Table 3, we also run DSA code and set batch normalization in ConvNet.

ResNet with Batch Normalization. We follow the modification in Zhao et al. (2021) that they replace the $stride = 2$ convolution layer with $stride = 1$ convolution layer followed by an average pooling layer in the ResNet architecture that is used to learn the synthetic data. This modification enables smooth error back-propagation to the input images. We directly use their released ResNet architecture.

A.2 Continual Learning

Data Augmentation. Prabhu et al. (2020) use cutmix Yun et al. (2019) augmentation strategy for training models. Different from them, we follow Zhao & Bilen (2021) and use the default DSA augmentation strategy in order to be consistent with other experiments in this paper.

DSA and Herding Training. Without loss of generality, we run DSA training algorithm with only the new training classes and images in every learning step. It is not easy to take old model and memory into DSA training. The synthetic data learned with old model can also be biased to it, and thus perform worse. Similarly, we train the embedding function (ConvNet) for herding method on the new training classes and images only.

A.3 Neural Architecture Search.

We randomly select 10% training samples in CIFAR10 dataset as the validation set. The rest are the training set. The batch size is 250, then one training epoch on the small (50 images/class) proxy sets needs 2 batches. The DSA augmentation strategy is applied to all proxy-set methods and early-stopping. We train each model 5 times and report the mean accuracies. We do NAS experiment on one Tesla v100 GPU.

We visualize the performance rank correlation between proxy-set and whole-dataset training in Figure F8. The top 5% architectures are selected based on the proxy-set trained models' validation accuracy. Each point represent a selected architecture. The horizontal and vertical axes are the testing accuracies of models trained on the proxy-set and the whole dataset respectively. The figure shows that our method can produce better proxy set to obtain more reliable performance ranking of candidate architectures.

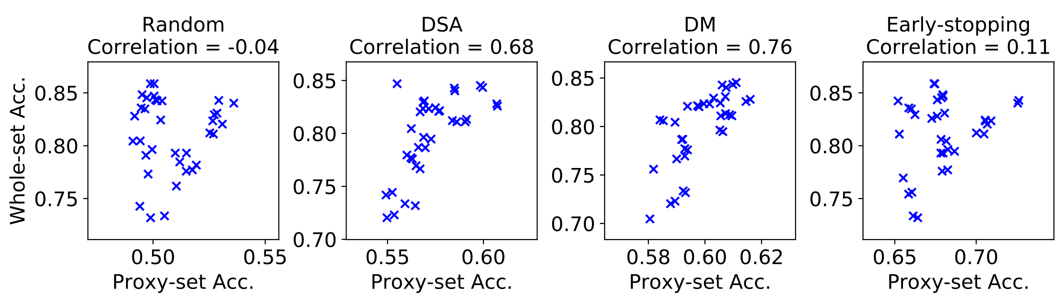

Figure F8: Performance rank correlation between proxy-set and whole-dataset training.

B Extended Visualization and Analysis

We visualize the 10 images/class synthetic sets learned on CIFAR10 dataset with different network parameter distributions in Figure F9. It is interesting that images learned with "poor" networks that have lower validation accuracies look blur. We can find obvious checkerboard patterns in them. In contrast, images learned with "good" networks that have higher validation accuracies look colorful. Some twisty patterns can be found in these images. Although synthetic images learned with different network parameter distributions look quite different, they have similar generalization performance. We think that these images are mainly different in terms of their background patterns but similar in foreground objects *i.e.* semantics.

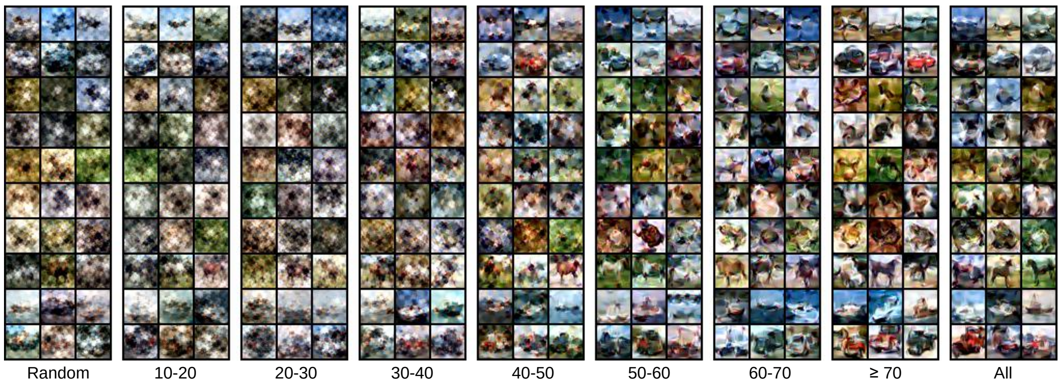

Figure F9: Synthetic images of CIFAR10 dataset learned with different network parameter distributions, *i.e.* networks with different validation accuracies (%). Each row represents a class.