Supplementary of Dataset Condensation with Distribution Matching

Bo Zhao, Hakan Bilen
School of Informatics, The University of Edinburgh
{bo.zhao, hbilen}@ed.ac.uk

1. Implementation details

1.1. Dataset Condensation

DSA Results. As [12] 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 2 and Table 3, we also run DSA code and set batch normalization in ConvNet.

ResNet with Batch Normalization. We follow the modification of ResNet in [13]. 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.

1.2. Continual Learning

Data Augmentation. Prabhu *et al*. [8] use cutmix [11] augmentation strategy for training models. Different from them, we follow [12] 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 on the new training classes and images only in every learning step. It is not a easy work to take old model and memory into DSA training and achieve better performance. 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.

1.3. 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 includes 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 F1. The top 5% architectures are selected based on the validation accuracies of models trained on each proxy-set. Each point represents 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.

2. Comparison to More Baselines and Related Works

2.1. Comparison to Generative Models

In this subsection, we compare the data-efficiency of samples generated by our dataset condensation method to those generated by traditional generative models, namely VAE and GAN. Specifically, we choose the state-of-the-art DC-VAE [7] and BigGAN [1]. The BigGAN model is trained with the differentiable augmentation [14]. In addition, we also compare to a related generative model GMMN [3] which aims to learn an image generator that can map a uniform distribution to real image distribution. Our method differs from GMMN in many ways significantly. First, GMMN aims to generate real-looking images, while our goal is to condense a training set by synthesizing informative training samples that can be used to efficiently train deep networks through MMD. Second, our method learns pixels directly, while GMMN learns a generator network. Third, our method learns **a few** synthetic samples to approximate the distribution of large real training set in **any** embedding space with **any** augmentation, while GMMN learns to map a uniform distribution to real image distri-

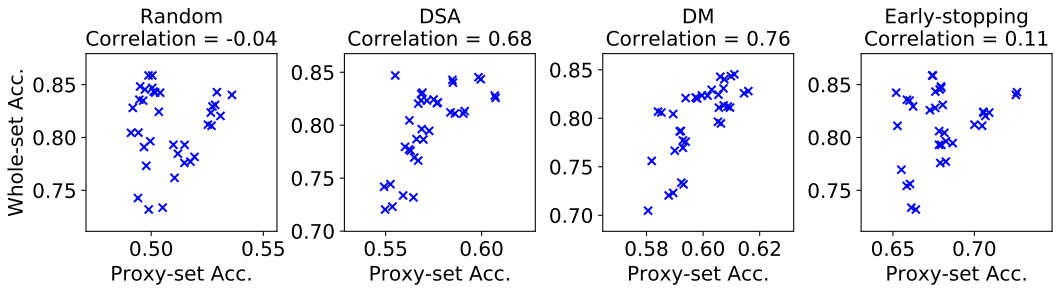

Figure F1. Performance rank correlation between proxy-set and whole-dataset training.

bution which is an easier task.

We train these generative models on CIFAR10 dataset. ConvNets are trained on these synthetic images and then evaluated on real testing images. The results in Table T1 verify that our method outperforms them by large margins, indicating that our synthetic images are more informative for training deep neural networks. The comparison to random baseline indicates that the images generated by traditional generative models are not more informative than randomly selected real images.

Img/Cls	Random	GMMN	VAE	BigGAN	MMD	*DM*
1	14.4±2.0	16.1±2.0	15.7±2.1	15.8±1.2	22.7±0.6	**26.0±0.8**
10	26.0±1.2	32.2±1.3	29.8±1.0	31.0±1.4	34.9±0.3	**48.9±0.6**
50	43.4±1.0	45.3±1.0	44.0±0.8	46.2±0.9	50.9±0.3	**63.0±0.4**

Table T1. Comparison to traditional generative models and MMD baseline. Random means randomly selected real images. The experiments are implemented with ConvNets on CIFAR10 dataset.

2.2. Comparison to MMD Baseline

Another baseline is to learn synthetic images by distribution matching with vanilla MMD in the pixel space. This baseline can also been considered as the ablation study of the embedding function and differentiable augmentation in our method. We try this baseline with linear, polynomial, RBF and Laplacian kernels and with various kernel hyperparameters. We find that only MMD with linear kernel can achieve better synthetic images, *i.e*. better than randomly selected real images. The performance of MMD with linear kernel in the pixel space is presented in Table T1, which outperforms all generative models while is inferior to our method. This result also verifies that the distribution matching mechanism enables learning more informative synthetic samples.

2.3. Comparison to GTN and KIP Methods

We notice the recent works Generative Teaching Networks (GTN) [9] and Kernel Inducing Point (KIP) [5, 6] on dataset condensation. Such *et al.* [9] propose to learn a generative network that outputs condensed training samples by minimizing the meta-loss on real data. They report the performance of 4,096 synthetic images learned on MNIST which is comparable to our 50 images/class synthetic set (*i.e*. 500 images in total) performance.

Nguyen *et al.* [5, 6] propose to replace the neural network optimization in the bi-level optimization [10] with kernel ridge regression which has a closed-form solution. Zero Component Analysis (ZCA) [2] is applied for preprocessing images. Although Nguyen *et al.* [6] report the results on 1024-width neural networks while we train and test 128-width neural networks, our results still outperform theirs in some settings, for example $98.6 \pm 0.1\%$ v.s. $98.3 \pm 0.1\%$ when learning 50 images/class on MNIST and $29.7 \pm 0.3\%$ v.s. $28.3 \pm 0.1\%$ when learning 10 images/class on CIFAR100. Note that they achieve those results by leveraging distributed computation environment and training for thousands of GPU hours. In contrast, our method can learn synthetic sets with one GTX 1080 GPU in dozens of minutes, which is significantly more efficient.

3. Extended Visualization and Analysis

We visualize the 10 images/class synthetic sets learned on CIFAR10 dataset with different network parameter distributions in Figure F2. 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 semantics. It means that our method can produce synthetic images with similar network optimization effects while significantly different visual effects. Our method may have promising applications in protecting data privacy and federated learning [4].

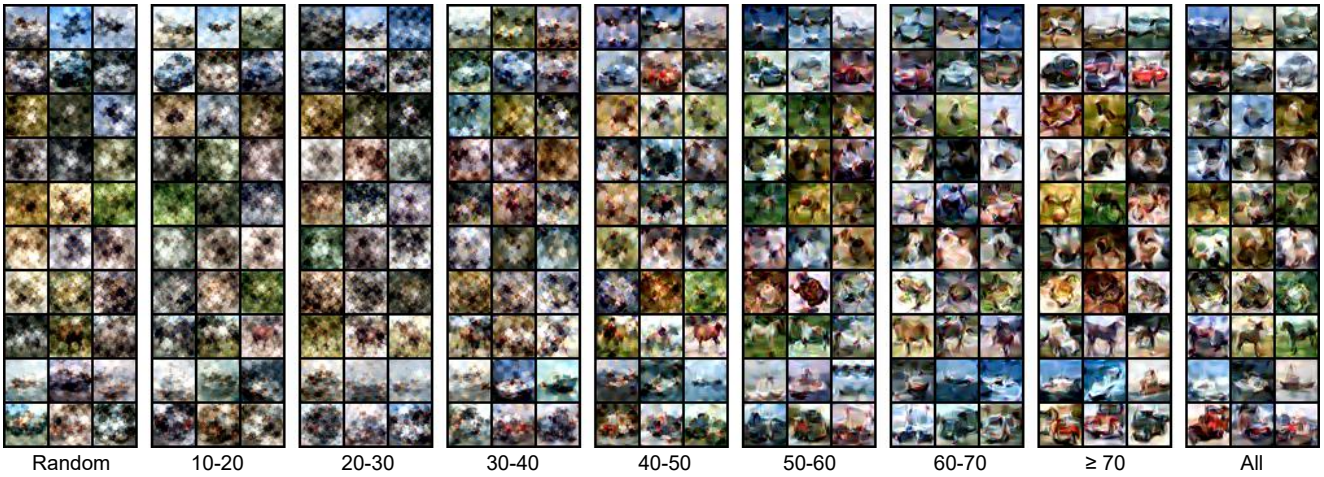

Figure F2. Synthetic images of CIFAR10 dataset learned with different network parameter distributions, *i.e*. networks with different validation accuracies (%). Each row represents a class.

4. Connection to Gradient Matching

In this section, we show the connection between gradient matching [13] and our method. Both [13] and our training algorithm sample real and synthetic image batches from one class in each iteration, which is denoted as class y. We embed each training sample $(\boldsymbol{x}_i, y)$ and obtain the feature $\boldsymbol{e}_i$ using a neural network $\psi_{\boldsymbol{\vartheta}}$ followed a linear classifier $\mathbf{W} = [\boldsymbol{w}_0, ..., \boldsymbol{w}_{C-1}]$, where $\boldsymbol{w}_j$ is the weight vector connected to the j^{th} output neuron and C is the number of all classes. Note that the weight and its gradient vector are organized in the same way in [13]. We focus on the weight and gradient of the linear classification layer (*i.e*. the last layer) of a network in this paper. The classification loss J_i of each sample is denoted as

$$J_i = -\log \frac{\exp\left(\boldsymbol{w}_y^T \cdot \boldsymbol{e}_i\right)}{\Sigma_k \exp\left(\boldsymbol{w}_k^T \cdot \boldsymbol{e}_i\right)}. \tag{1}$$

Then, we compute the partial derivative w.r.t. each weight vector,

$$\boldsymbol{g}_{i,j} = \frac{\partial J_i}{\partial \boldsymbol{w}_j} = \begin{cases} -\boldsymbol{e}_i + \dfrac{\exp\left(\boldsymbol{w}_y^T \cdot \boldsymbol{e}_i\right)}{\Sigma_k \exp\left(\boldsymbol{w}_k^T \cdot \boldsymbol{e}_i\right)} \cdot \boldsymbol{e}_i, & j = y \\ \dfrac{\exp\left(\boldsymbol{w}_j^T \cdot \boldsymbol{e}_i\right)}{\Sigma_k \exp\left(\boldsymbol{w}_k^T \cdot \boldsymbol{e}_i\right)} \cdot \boldsymbol{e}_i, & j \neq y \end{cases} \tag{2}$$

This equation can be simplified using the predicted probability $p_{i,j} = \frac{\exp\left(\boldsymbol{w}_j^T \cdot \boldsymbol{e}_i\right)}{\Sigma_k \exp\left(\boldsymbol{w}_k^T \cdot \boldsymbol{e}_i\right)}$ that classifies sample $\boldsymbol{x}_i$ into category j:

$$\boldsymbol{g}_{i,j} = \begin{cases} (p_{i,y} - 1) \cdot \boldsymbol{e}_i, & j = y \\ (p_{i,j} - 0) \cdot \boldsymbol{e}_i, & j \neq y \end{cases} \tag{3}$$

Eq. 3 shows that **the last-layer gradient vector $\boldsymbol{g}_{i,j}$ is equivalent to a weighted feature vector $\boldsymbol{e}_i$ and vice versa**. The weight is a function of classification probability. Generally speaking, the weight is large when the difference between predicted probability $p_{i,j}$ and ground-truth one-hot label (1 or 0) is large.

As the real and synthetic samples in each training iteration are from the same class y, we can obtain the mean gradient over a data batch by averaging the corresponding gradient components:

$$\frac{1}{N}\Sigma_i^N \boldsymbol{g}_{i,j} = \begin{cases} \dfrac{1}{N}\Sigma_i^N (p_{i,y} - 1) \cdot \boldsymbol{e}_i, & j = y \\ \dfrac{1}{N}\Sigma_i^N (p_{i,j} - 0) \cdot \boldsymbol{e}_i, & j \neq y \end{cases} \tag{4}$$

N is the batch size. **Thus, last-layer mean gradient is equivalent to the weighted mean feature, and the mean gradient matching is equivalent to the matching of weighted mean feature.**

Our method can learn synthetic images with randomly initialized networks. Given networks with random parameters, we assume that the predicted probability is uniform over all categories, *i.e*. $p_{i,j} = \frac{1}{C}$. Then, the mean gradient is

$$\frac{1}{N}\Sigma_i^N \boldsymbol{g}_{i,j} = \begin{cases} \dfrac{1-C}{C} \cdot \dfrac{1}{N}\Sigma_i^N \boldsymbol{e}_i, & j = y \\ \dfrac{1}{C} \cdot \dfrac{1}{N}\Sigma_i^N \boldsymbol{e}_i, & j \neq y \end{cases} \tag{5}$$

which is equivalent to the mean feature with a constant weight. **Thus, with randomly initialized networks, the last-layer mean gradient matching is equivalent to mean feature matching multiplied by a constant.**

References

[1] Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale gan training for high fidelity natural image synthesis. *ICLR*, 2019.

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

[3] Yujia Li, Kevin Swersky, and Rich Zemel. Generative moment matching networks. In *International conference on machine learning*, pages 1718–1727. PMLR, 2015.

[4] Lingjuan Lyu, Han Yu, and Qiang Yang. Threats to federated learning: A survey. *FL-IJCAI*, 2020.

[5] Timothy Nguyen, Zhourong Chen, and Jaehoon Lee. Dataset meta-learning from kernel-ridge regression. In *International Conference on Learning Representations*, 2021.

[6] Timothy Nguyen, Roman Novak, Lechao Xiao, and Jaehoon Lee. Dataset distillation with infinitely wide convolutional networks. *arXiv preprint arXiv:2107.13034*, 2021.

[7] Gaurav Parmar, Dacheng Li, Kwonjoon Lee, and Zhuowen Tu. Dual contradistinctive generative autoencoder. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 823–832, 2021.

[8] 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*, pages 524–540. Springer, 2020.

[9] 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.

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

[11] 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*, pages 6023–6032, 2019.

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

[13] Bo Zhao, Konda Reddy Mopuri, and Hakan Bilen. Dataset condensation with gradient matching. In *International Conference on Learning Representations*, 2021.

[14] Shengyu Zhao, Zhijian Liu, Ji Lin, Jun-Yan Zhu, and Song Han. Differentiable augmentation for data-efficient gan training. *Neural Information Processing Systems*, 2020.