ECE 1512 Digital Image Processing and Applications 2024 Fall Project A

Linfeng Ye, and Samuel Bernard
Edward S. Rogers Sr. Department of Electrical & Computer Engineering
University of Toronto

Abstract

In this report, we examine dataset distillation, a modern technique designed to reduce the training costs of deep neural networks by condensing large, annotated datasets of natural images into smaller, synthetic datasets. This technique aims to maintain, or even enhance, model performance in terms of test accuracy compared to models trained on the original, full-scale datasets. The distilled dataset contains images generated to mimic essential features of the original data, effectively reducing redundancy and preserving critical information required for accurate model training.

For the first part of the report, we focus on DataDam, a state-of-the-art distillation method that uses spatial attention matching to selectively capture and recreate the most relevant image features. DataDam's approach leverages spatial attention matching to align key visual elements between original and distilled images, leading to efficient dataset distillation without significant information loss. Compared to other methods in the literature, DataDam offers notable advancements in accuracy and training efficiency due to its attention-based matching mechanism, which ensures that the synthetic images retain meaningful spatial characteristics and similar distribution of the original dataset. We also discuss potential applications for DataDam.

For the second part of the report, we introduce two other state-of-the-art distillation methods and compare them with the Attention Matching algorithm (DataDam). As such, one of these methods is the Prioritize Alignment in Dataset Distillation (PAD) and the other is Condense Dataset by Aligning Features (CAFE). PAD introduces filtering mechanisms at both the information extraction and embedding phases, thus addressing the dataset misalignment issue that arises in DataDam during distillation. By sorting samples by difficulty and selecting high-level features in the embedding stage, PAD allows higher-quality synthetic images and datasets. Alternatively, CAFE aims to solve gradient dominance in gradient matching techniques by introducing layer-wise feature alignment and dynamic bi-level optimization. Using CAFE prevents overfitting and ensures a balanced representation of class-specific features. In this analysis, we report the performance results from DataDam, PAD, and CAFE and compare the testing accuracy of different models trained on their synthetic datasets initialized from both real images and Gaussian noise.

The code is available at https://github.com/SamuelBernardDev/ECE1512_2024F_ProjectRepo_SamuelBernard_LinfengYe.

Index Terms

Deep learning, Dataset distillation, Attention matching, bi-level optimization

I. Dataset Distillation with Attention Matching

A. Basic Concepts

1) What is the purpose of using Dataset Distillation in this paper?:

To reduce training costs in deep learning while maintaining model performance in terms of testing accuracy, one approach is to use a more informative, smaller training set instead of the original large dataset for training deep neural networks. To this end, researchers propose dataset distillation, which condenses the dataset into a smaller size while still allowing the model trained on this condensed dataset to generalize well on unseen data.

2) What are the advantages of their methodology over state-of-the-art? Explain your rationale.:

Dataset Distillation with Attention Matching (DataDam) [1] aims to develop a **simple** yet **efficient** end-to-end dataset distillation algorithm for training on any dataset. DataDam matches the spatial attention map in intermediate layers, reducing memory costs and outperforming most existing methods on standard benchmarks across several datasets and various settings.

3) What novelty did they contribute compared to their prior methods?:

DataDam utilizes spatial attention matching and last-layer feature alignment to synthesize data that closely approximates an unbiased representation of the real training data distribution. Unlike previous methods, DataDam neither relies on pre-trained network parameters nor employs bi-level optimization, making it a promising tool for synthetic data generation.

4) Explain in full detail the methodologies of the paper.:

Due to the nature of these images which lie within an unknown manifold of a high dimensional space, which makes the estimation of image distribution amenable. To overcome this challenge, DataDam leverages spatial attention maps which generated by different layers within a family of randomly initialized neural networks which makes the synthesize data contains the low, mid and high-level features of the real data. DataDam contains two components, each one will be elaborated as follow:

• **S**patial **A**ttention **M**atching (SAM) matching: In order to capture the distribution of original training set at different levels of representations between real and synthetic sets. The loss function associated with the SAM matching can be calculated as:

$$E_{\boldsymbol{\theta}\sim P_\theta}\left[\sum_{k=1}^{K}\sum_{l=1}^{L-1}\left\|E_{\mathcal{T}_k}\left[\frac{\mathbf{z}_{\boldsymbol{\theta},l}^{\mathcal{T}_k}}{\|\mathbf{z}_{\boldsymbol{\theta},l}^{\mathcal{T}_k}\|_2}\right]-E_{\mathcal{S}_k}\left[\frac{\mathbf{z}_{\boldsymbol{\theta},l}^{\mathcal{S}_k}}{\|\mathbf{z}_{\boldsymbol{\theta},l}^{\mathcal{S}_k}\|_2}\right]\right\|^2\right], \tag{1}$$

where, $\mathbf{z}_{\boldsymbol{\theta},l}^{\mathcal{T}_k}$ and $\mathbf{z}_{\boldsymbol{\theta},l}^{\mathcal{S}_k}$ are the l^{th} pair of vectorized attention maps along the spatial dimension of the real and distilled dataset.

In order to capture the highest-level abstract semantic information of the input data in the final layer, DataDam leverage **M**aximum **M**ean **D**iscrepancy (MMD) loss, which is calculated within a family of kernels in the reproducing kernel Hilbert space, which can be calculated as follows:

$$E_{\boldsymbol{\theta}\sim P_\theta}\left[\sum_{k=1}^{K}\left\|E_{\mathcal{T}_k}\left[\tilde{\mathbf{f}}_{\boldsymbol{\theta},L}^{\mathcal{T}_k}\right]-E_{\mathcal{S}_k}\left[\tilde{\mathbf{f}}_{\boldsymbol{\theta},L}^{\mathcal{S}_k}\right]\right\|^2\right], \tag{2}$$

where, $\tilde{\mathbf{f}}_{\boldsymbol{\theta},L}^{\mathcal{T}_k}$ and $\tilde{\mathbf{f}}_{\boldsymbol{\theta},L}^{\mathcal{S}_k}$ are vectorized form of the final feature maps of the real and synthetic datasets.

Finally, the synthetic dataset is created by solving the following optimization problem using Stochastic Gradient Descent (SGD) with momentum:

$$\mathcal{S}^* = \arg\min_{\mathcal{S}}\Big(\mathcal{L}_{SAM}+\lambda\mathcal{L}_{MMD}\Big), \tag{3}$$

where λ is the predefined hyper-parameter which balance the $\mathcal{L}_{SAM}$ and $\mathcal{L}_{MMD}$.

5) Explain the usefulness of the methodology in machine learning applications: Dataset distillation can be used in Neural Architecture Search (NAS) and continual learning.

B. Dataset Distillation Learning

In this subsection, we selected two CNN architectures as backbone and conduct dataset distillation using DataDam method on both the MNIST and MHIST datasets.

1) Train the selected model with the original dataset and report the classification accuracy along with floating-point operations per second (FLOPs) for the test set.: We picked ConvNet-3 for the MNIST [2] dataset and ConvNet-7 for the MHIST [3] dataset and report the floating-point operations per second and testing accuracy in Table I.

TABLE I
THE FLOATING-POINT OPERATIONS PER SECOND AND TEST ACCURACY ON MNIST AND MHIST DATASET, OF THE CONVNET PROVIDED IN THE NETWORKS.PY FILE.

Dataset	FLOPS	Accuracy
MNIST	25.126 GFLOPS	99.03
MHIST	342.46 GFLOPS	89.76

2) Visualization of condensed images with real initialization:

After applying the Attention Matching algorithm to the MNIST and MHIST datasets, we can visualize the condensed images for each class.

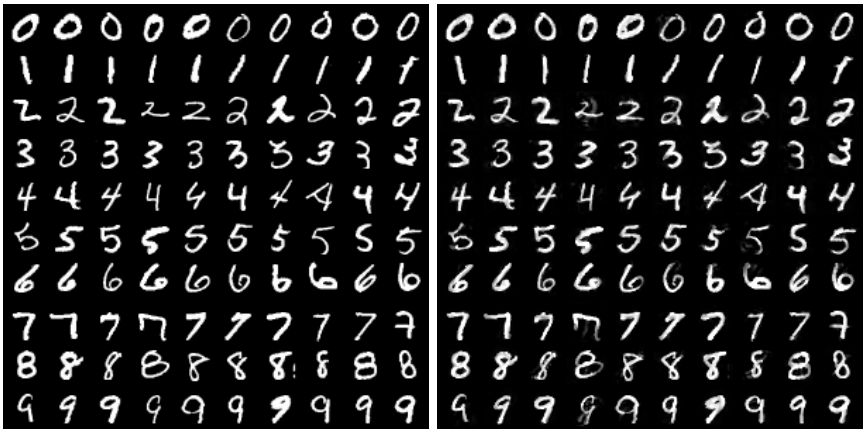

Fig. 1. MNIST: Visualization of condensed images using the Datadam method with real initialization. The left figure shows the initial images, while the right one displays the distilled dataset.

Figure 1 depicts the MNIST distilled dataset using the DataDam method with real image initialization. As such, the left image represents the original synthetic dataset before learning, whereas the right image represents the distilled dataset. These distilled images are quite clear and resemble the original dataset of handwritten digits. Each digit remains recognizable after the distillation process, however, a few added features and a slight blurring effect can be observed, thereby indicating that the distillation process effectively extracted the important features from the dataset.

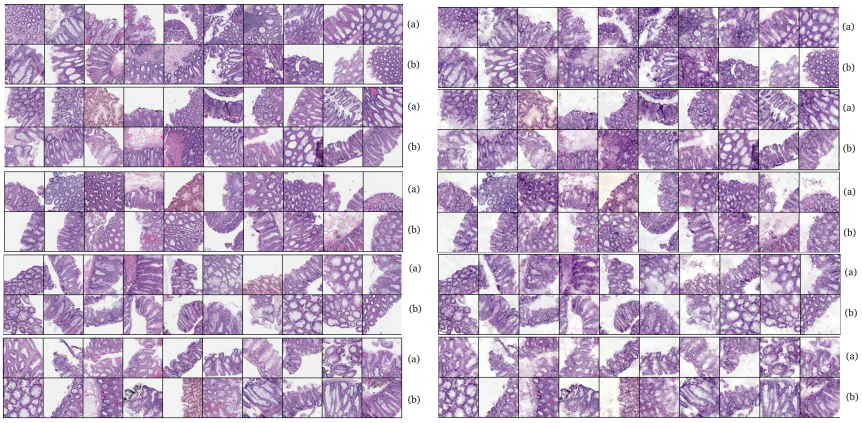

Fig. 2. MHIST: Visualization of condensed images using the Datadam method with real initialization. The left figure shows the initial images, while the right one displays the distilled dataset. Where the row (a) is the distilled images of class plastic polyp, and row (b) is is the distilled images of class sessile serrated adenoma

Figure 2 shows the Datadam results on the MHIST dataset. The images placed at the left of the aforementioned figure represent the original synthetic dataset, while the right images depict the distilled synthetic dataset. The DataDam results remain recognizable after the distillation process, though the distilled images contain some additional blurred details. While some images from the original synthetic dataset seem to have been quite drastically modified (i.e. first image at the top left), others seem to have undergone no transformation between the original and the distilled dataset (i.e. last image right left).

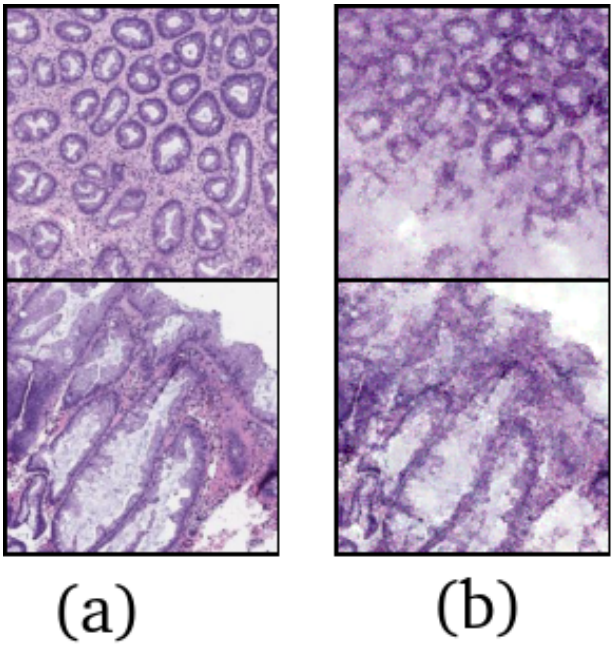

Fig. 3. MHIST: Visualization of original images (column (a)) and condensed images (column (b)) using Datadam.

Additionally, figure 3 better represents the changes DataDam introduced to the synthetic dataset during learning. Although both images shown in column a) and b) are quite recognizable, some of the finer details contained in the images of column a) have been modified in column b). These changes can be attributed to the inherent trade-offs in the dataset distillation process when using DataDam. As such, this process often reduces these finer details to redirect the focus towards essential features,

thus ensuring the creation of an effective representation of the entirety of the original dataset.

3) Visualization of condensed images with Gaussian noise initialization:

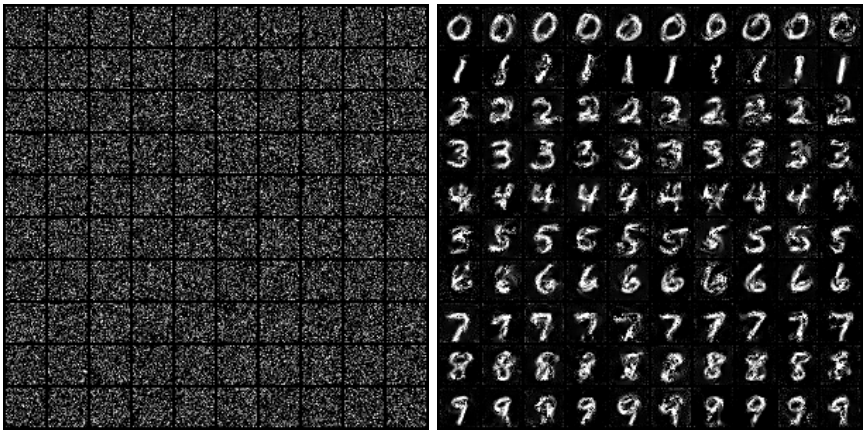

Fig. 4. MNIST: Visualization of condensed images using the Datadam method with random initialization. The left figure shows the initial images, while the right one displays the distilled dataset.

Figure 4 shows the MNIST distilled dataset using the DataDam method with Gaussian noise initialization. Although both Gaussian noise initialization and real image initialization resulted in a recognizable distilled dataset, the dataset initialized with random noise resulted in images that are less clear compared to those initialized with real images. While the general shape of the digits is present, most of the digits appear ”noisier”, and can be quite difficult to distinguish. For example, some of the number 8’s would be hard to recognize if isolated.

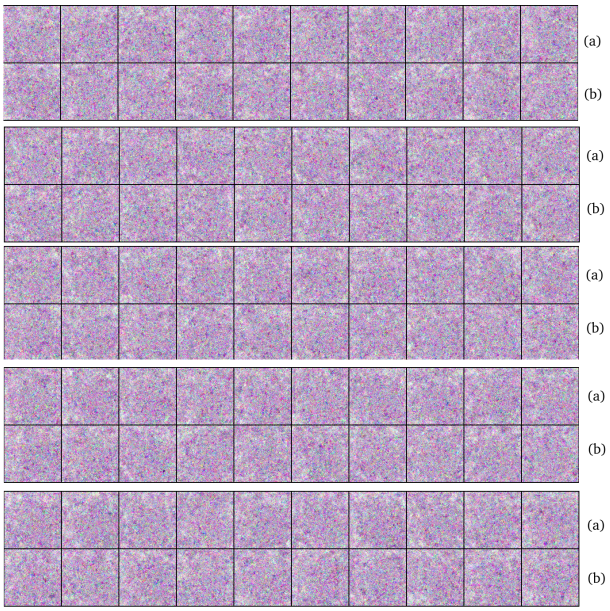

Fig. 5. MHIST: Visualization of condensed images using the Datadam method with Gaussian noise initialization. Where the row (a) is the distilled images of class plastic polyp, and row (b) is the distilled images of class sessile serrated adenoma

In contrast, Figure 5 illustrates the results from Gaussian noise initialization with the MHIST dataset. Compared to the real image initialization, here, the images are challenging to distinguish. There seems to be little improvement to that of the Gaussian noise initialized images in terms of general structure. Although the condensed images retain some color patterns of the original histopathological images from the original MHIST dataset, most of the details are not distinct. The images have no clear structure and identifying the difference between classes is difficult.

The MNIST dataset, with a resolution of (28×28), is significantly lower in resolution than the MHIST dataset, which is $(224\times224\times3)$. This makes the optimization problem more challenging for the MHIST dataset. As such, the increased resolution in the MHIST dataset means that the distillation algorithm needs to capture and extract a large amount of information, thus

making the process more extensive. To generate recognizable images from random noise, a prior of natural image characteristics is needed. However, the Datadam algorithm lacks such a prior, which explains why the distilled MHIST dataset from random noise initialization is not recognizable.

4) Accuracy Performance of the Distilled Dataset:

TABLE II
DATADAM RESULTS ON MHIST DATASET

Method	Random Init	Real Init
Datadam	60.02%	72.06%

TABLE III
DATADAM RESULTS ON MNIST DATASET

Method	Random Init	Real Init
Datadam	93.61%	95.75%

As mentioned previously, the accuracy results from training ConvNet-3 for the MNIST dataset and ConvNet-7 for the MHIST dataset are recorded in Table I. These accuracy results serve as a baseline to compare with the testing accuracies on the distilled MNIST and MHIST dataset which are reported in in Table III and Table II. As observed, the DataDam results are consistently lower than those recorded in Table I from training models on the entirety of the datasets for both MNIST and MHIST. Furthermore, results presented in Tables III and II indicate that initializing the distillation process with real images leads to higher classification accuracy and more recognizable condensed images compared to initializing with Gaussian noise. This effect is more pronounced in the DataDam results on the MHIST dataset, likely due to the the dataset's higher complexity and resolution. By using real images for to initialize the distillation process, additional structural information is provided, which can be crucial to help generate synthetic data that best resembles the original dataset. Thus, although these results are promising for the usage of a synthetic dataset to train complex models, the results reported previously suggest obtaining a higher testing accuracy is better accomplished using the entirety of the dataset or initializing the synthetic dataset with real images.

C. Cross-architecture Generalization.

We evaluate the performance of a distilled dataset across different models, specifically training ResNet-18 [4] on this distilled dataset. As such, Table IV indicates that the distilled dataset generalizes well to the MNIST dataset. However, for the MHIST dataset, only the distilled dataset initialized with real data yields good results. This is understandable, as the distilled dataset derived from Gaussian noise is not recognizable compared to real data.

TABLE IV
CROSS ARCHITECTURE GENERALIZATION PERFORMANCE

Dataset	MNIST		MHIST	
Init	Real	Random	Real	Random
Acc	96.15	96.03	72.05	56.58

D. Application.

We conduct continual learning experiments following the settings outlined in [5]. Continual learning addresses the challenge of catastrophic forgetting when models are trained incrementally. The method discussed involves creating an efficient memory of training samples to enhance performance. We evaluated the distilled synthetic MNIST dataset, dividing 10 classes into 5 learning steps, with 10 images per class for 5 steps and using a default ConvNet architecture.

As shown in Figure 6, DataDam effectively mitigates catastrophic forgetting in continual learning, consistently outperforming the randomly selected subset of the MNIST dataset.

II. PRIORITIZE ALIGNMENT IN DATASET DISTILLATION (PAD)

A. Basic Concepts

1) What knowledge gap did PAD fill?: Dataset distillation [6] consists of two steps: information extraction, where a model is trained to capture meaningful features from the dataset, and information embedding, where synthetic data is generated to embed the meaningful information in the dataset. In [7], researchers found that both steps can introduce misalignment in the final distilled dataset.

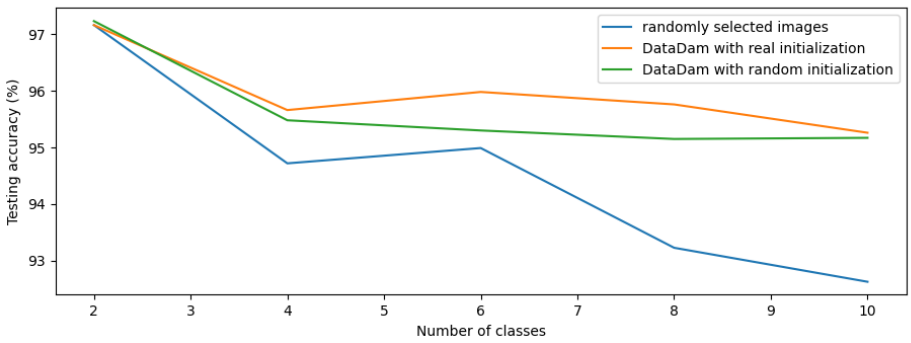

Fig. 6. 5-steps continual learning results on MNIST.

2) What novelty did they contribute compared to their prior methods?: To address the misalignment problem discussed previously, researchers apply data selection methods during information extraction. During the information embedding step, only the weights from the deep layers of the model are used to create a high-quality distilled dataset.

3) Explain in full detail the methodologies of PAD.:

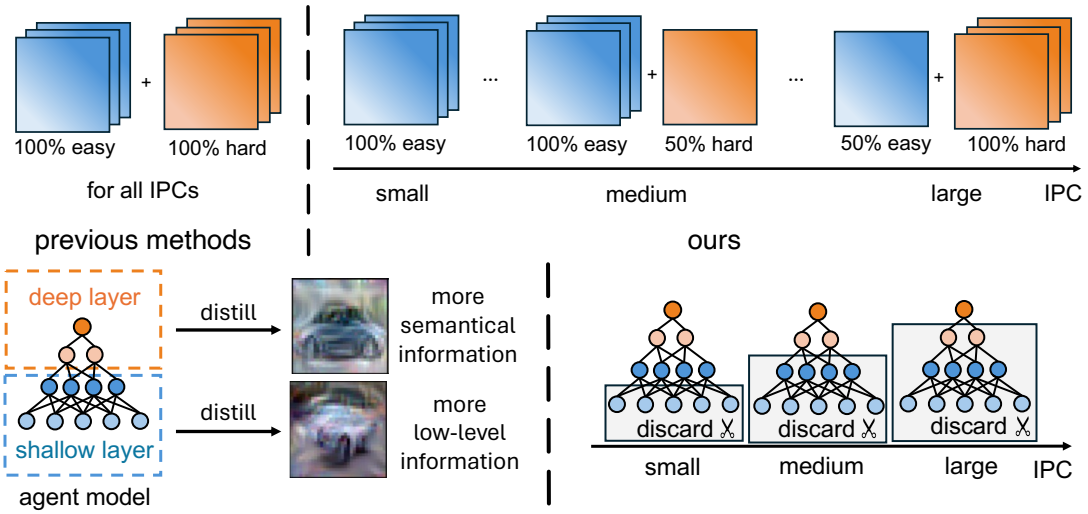

Fig. 7. We reference the figure from the introduction of the PAD [6] paper here. The upper section illustrates the data selection strategy for information extraction, while the lower section shows the weights used during the feature embedding step.

Specifically, in Figure 7, which is adapted from their paper [6]. Here are the main methods of PAD:

Filtering Information Extraction. Trajectory-matching-based methods favor easier samples when the images per class (IPC) are low, while higher IPC values prioritize harder samples. The PAD method sorts the samples based on their EL2N value which measures sample difficulty and are calculated as follows:

$$\mathcal{X}_t(x, y) = \mathbb{E}\|p(w_t, x) - y\|_2, \tag{4}$$

where $p(w_t, x)$ is the output of the model parameterized by the weight at time t. At the beginning of training, the hardest samples are eliminated from the training set to obtain the trajectory. These harder samples are then gradually reintroduced back into the training set. Once all data points have been added, the easier samples are gradually removed from the training set.

Filtering Information Embedding. Parameter selection is introduced in the information embedding step, where only a subset of the feature extractor's parameters are used. Specifically, the model's lower layers capture low-level visual features, while higher layers capture high-level visual features. For smaller IPCs, which focus on basic information, parameters from the lower layers are included in the feature embedding. As IPCs increase, more of these lower layers are excluded from the embedding process.

4) Advantages and disadvantages of your selected methods.: PAD builds on the DATM method [8] by incorporating filtering in both the information extraction and embedding stages. As a general approach, it can be applied to any trajectory-matching or distribution-matching methods. However, extensive fine-tuning may be needed for each method to adjust the scheduling of information extraction filtering and the rate of information embedding filtering.

B. Dataset Distillation Learning

1) Experiment:

Condensed synthetic data was produced using the Prioritize Alignment method from real image initialization as well as Gaussian noise initialization.

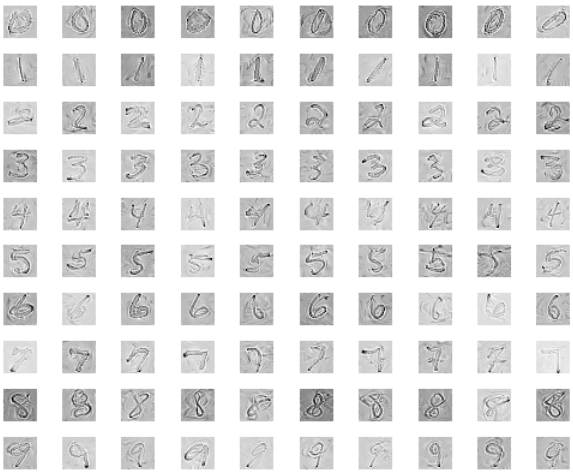

Fig. 8. Pad: visualization of condensed images using the Pad method with real initialization.

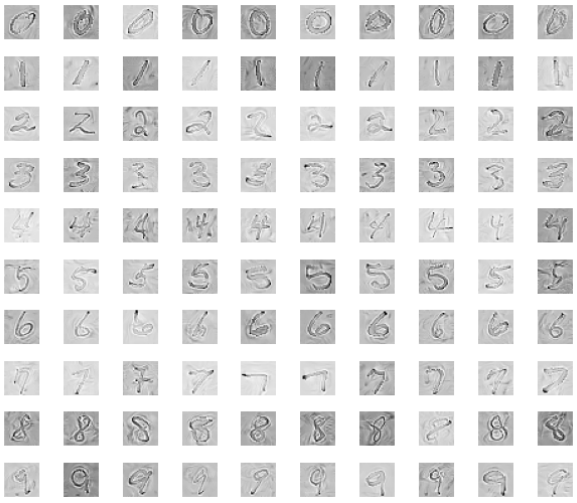

Fig. 9. Pad: visualization of condensed images using the Pad method with random initialization.

These learned condensed synthetic datasets are depicted in figures 8 and 9. As such, Figure 8 represents the condensed dataset from the distillation process initialized with real images, whereas Figure 9 contains the condensed dataset initialized with Gaussian noise.

A ConvNet-3 model was trained on both synthetic datasets, and the testing accuracy results are reported in Table VI.

TABLE V
PAD RESULTS ON MNIST DATASET

Method	Random Init	Real Init
PAD	97.56%	98.28%

2) Discussion on PAD Results:

Similarly to the condensed synthetic dataset generated by DataDam using the MNIST dataset, both condensed images produced by the PAD method with real and Gaussian noise initialization yielded distinguishable images. Although the color patterns differ, with the PAD results appearing as negative images of the original dataset, these digits are still very recognizable as the general structure of the images remains. Another detail that can be observed within both Figures 8 and 9 is how fine the lines defining the digits are, whereas the handwritten digits in the condensed synthetic dataset from DataDam appear slightly more spread out. Additionally, while greater noise can still be observed in the DataDam synthetic dataset initialized with Gaussian noise compared to the dataset initialized with real images, this does not seem to be the case for PAD. In fact, both figures appear to contain a similar level of noise.

With regards to the testing accuracy results recorded in Table VI, it can be observed that the PAD results are only slightly lower than using the entirety of the MNIST dataset for training. However, the PAD results with the MNIST dataset are higher than that of the results from DataDam. These better results could be explained by the lower amount of noise and blurriness contained within the PAD-created synthetic dataset. The ability of the PAD method to produce clear condensed images indicates strong recognition performance, even when initialized with Gaussian noise. This could be an indication of the effectiveness and robustness of PAD's method, which can both maintain clear images and enhance model generalization. Its approach of selecting model parameters from deeper layers as IPCs increase makes it able to perform more effectively than the Attention Matching algorithm by focusing on embedding features important for both recognition and generalization. Furthermore, given that similar amounts of noise between both figures can be observed visually, this could explain the similarity between testing accuracy results from initializing the distillation process with real images or Gaussian noise.

III. CAFE: LEARNING TO CONDENSE DATASET BY ALIGNING FEATURES

A. Basic Concepts

1) What knowledge gap did CAFE fill?: CAFE, short for "Condense dataset by Aligning Features," is a dataset distillation method proposed in [9]. The authors found that (i) gradient-matching methods can lead to significant overfitting because some samples have much larger gradients than others, causing the averaged gradient to be dominated by these large-gradient samples. They discovered that using an early stopping method effectively prevents overfitting during the bi-level optimization process.

2) What novelty did they contribute compared to their prior methods?: The authors observed that the optimization process in gradient-based dataset distillation methods is often dominated by samples with large gradients, which prevents accurate representation of the original dataset's distribution. To address this issue, [9] proposed layer-wise feature alignment, which is further back-upped by a novel dynamic bi-level optimization algorithm.

3) Explain in full detail the methodologies of CAFE.: In this section, we explain the details of aforementioned methods:
Layer-wise features alignment which is achieved by minimizing the following objective function

$$\mathcal{L}_f = \sum_{k=1}^{K} \sum_{l=1}^{L} \left| \bar{f}_{k,l}^{\mathcal{S}} - \bar{f}_{k,l}^{\mathcal{T}} \right|, \tag{5}$$

where $\bar{f}_{k,l}^{\mathcal{S}}$ and $\bar{f}_{k,l}^{\mathcal{T}}$ are the averaged feature over N samples over distilled and original dataset, respectively, and K is number of classes in a dataset.

Discrimination Loss The discrimination loss is applied to capture class information in the dataset, calculated only at the model's final layer. The prediction probability p_i is obtained by applying softmax to the inner product between the centers of real data and distilled data. The cross-entropy loss is then calculated as follows:

$$\mathcal{L}_d = -\frac{1}{N} \sum_{i=1}^{N'} \log p_i \tag{6}$$

The overall training loss can be written as

$$\mathcal{L}_{total} = \mathcal{L}_f + \beta \mathcal{L}_d, \tag{7}$$

here β is a hyperparameter which balance the effect of $\mathcal{L}_f$ and $\mathcal{L}_d$.

Dynamic Bi-level Optimization The authors introduce a new optimization method called dynamic bi-level optimization. Instead of using a fixed number of inner-loop optimization steps, which can lead to underfitting or overfitting of the synthetic data, this approach allows researchers to terminate the loop when the model converges on a query set composed of real data samples.

4) Advantages and disadvantages of your selected methods.:

B. Dataset Distillation Learning

1) Experiment:

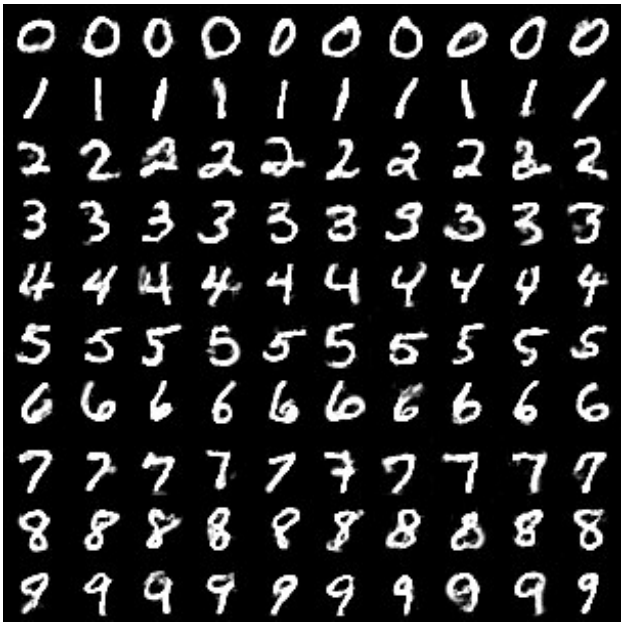

Fig. 10. CAFE: visualization of condensed images using the CAFE method with real initialization.

The condensed synthetic dataset resulting from the CAFE method using real image initialization is depicted in Figure 10 and using Gaussian noise initialization in Figure 11.

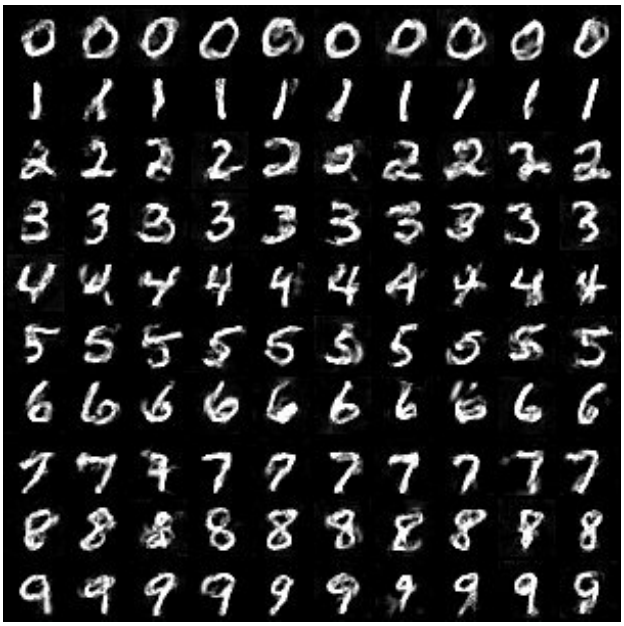

Fig. 11. CAFE: visualization of condensed images using the CAFE method with random initialization.

TABLE VI
CAFE RESULTS ON MNIST DATASET

Method	Random Init	Real Init
Datadam	90.41%	96.32%

The results from training Convnet-3 on the MNIST condensed synthetic dataset initialized by real images and Gaussian noise are recorded in Table VI.

2) Discussion of CAFE Results:

Figures 10 and 11 contain similar results to the synthetic datasets created using Datadam. The digits from both figures are recognizable as they retain their general shapes and color patterns. However, Figure 10, initialized with real images, appears much less noisier than Figure 11. In fact, most of the digits from this real-initialized dataset are much brighter and contain fewer artifacts. This results in a clearer and brighter dataset compared to DataDam's results with real image initialization. However, for Gaussian noise initialization, CAFE generates a lower-quality dataset than DataDam, with some digits barely recognizable. Although both synthetic datasets created by the CAFE method have retained the original color pattern, the datasets created by PAD appear to be more detailed and seem to have a better preservation of the images' general structure.

This qualitative analysis is further supported by the quantitative results reported in Table VI. Although, like DataDam and PAD, the testing accuracy results are lower than simply training with the entirety of the MNIST training dataset, the CAFE method yielded a higher testing accuracy (96.32%) than DataDam (95.75%) with real image initialization. This supports the observation made previously indicating that the condensed images from CAFE's real image initialization were clearer and more distinguishable. Additionally, the testing accuracy from CAFE initialized with Gaussian noise (90.41%) is lower than that of DataDam (93.61%), which also supports the qualitative analysis. Compared to PAD, however, CAFE generated lower testing accuracies for both initialization methods, thus confirming PAD's ability to create more detailed and recognizable digits.

The generalization and recognition abilities of datasets distilled with CAFE are due in part because of its unique feature alignment method. As such, layer-wise feature alignment allows the synthetic dataset to capture low-level and high-level feature representations, allowing models trained on the CAFE-created dataset to generalize more effectively to unseen data. This is because the distilled dataset can extract key class-specific features without overfitting. Additionally, CAFE's feature distribution consistency improves its recognition abilities by making the synthetic dataset resemble the real dataset more closely. Compared to the Attention Matching algorithm, although a higher testing accuracy was obtained by the DataDam on Gaussian noise initialization, the dynamic bi-level optimization from CAFE makes it more robust by allowing optimization to terminate based on the model's convergence.

DECLARATION OF NO PLAGIARISM

I hereby declare that this report is an original work. I have independently implemented all algorithms and drafted all content myself. Any sources referenced in this report are properly cited and credited.

I acknowledge that the inclusion of source code result in a higher match rate; however, this does not detract from the originality of my analysis and findings.

REFERENCES

[1] A. Sajedi, S. Khaki, E. Amjadian, L. Z. Liu, Y. A. Lawryshyn, and K. N. Plataniotis, " DataDAM: Efficient Dataset Distillation with Attention Matching ," in *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*. Los Alamitos, CA, USA: IEEE Computer Society, Oct. 2023, pp. 17 051–17 061. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/ICCV51070.2023.01568

[2] Y. B. Y. LeCun, L. Bottou and P. Haffner, "Gradient-based learning applied to document recognition," 1998.

[3] J. Wei, A. Suriawinata, B. Ren, X. Liu, M. Lisovsky, L. Vaickus, C. Brown, M. Baker, N. Tomita, L. Torresani *et al.*, "A petri dish for histopathology image analysis," in *Artificial Intelligence in Medicine: 19th International Conference on Artificial Intelligence in Medicine, AIME 2021, Virtual Event, June 15–18, 2021, Proceedings*. Springer, 2021, pp. 11–24.

[4] 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.

[5] 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.

[6] T. Wang, J.-Y. Zhu, A. Torralba, and A. A. Efros, "Dataset distillation," *arXiv preprint arXiv:1811.10959*, 2018.

[7] Z. Li, Z. Guo, W. Zhao, T. Zhang, Z.-Q. Cheng, S. Khaki, K. Zhang, A. Sajed, K. N. Plataniotis, K. Wang *et al.*, "Prioritize alignment in dataset distillation," *arXiv preprint arXiv:2408.03360*, 2024.

[8] Z. Guo, K. Wang, G. Cazenavette, H. Li, K. Zhang, and Y. You, "Towards lossless dataset distillation via difficulty-aligned trajectory matching," *arXiv preprint arXiv:2310.05773*, 2023.

[9] 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.

IV. Appendix

A. Code of Datadam

The following code implementation is derived from the DataDam [1] GitHub repository.

1) MHIST Dataloader: Code of MHIST Dataloader:

```
class MHISTDataset(Dataset):
    def __init__(self, csv_file, root_dir, transform=None, train=True):
        self.annotations = pd.read_csv(csv_file)
        self.root_dir = root_dir
        self.transform = transform
        self.train = train
        # breakpoint()
        if train:
            self.fileLst = [" " for _ in range(2175)]
        else:
            self.fileLst = [" " for _ in range(977)]
        idx = 0
        if self.train:
            Partation = 'train'
        else:
            Partation = 'test'
        for i in range(len(self.annotations)):

            if self.annotations.iloc[i]['Partition']==Partation:
                self.fileLst[idx] = self.annotations.iloc[i]['Image Name']
                idx = idx+1
        # breakpoint()
    def __len__(self):
        if self.train:
            return 2175
        else:
            return 977
    def __getitem__(self, idx):
        if torch.is_tensor(idx):
            idx = idx.tolist()
        # print(idx, flush=True)
        img_name = os.path.join(self.root_dir, self.fileLst[idx])
        image = Image.open(img_name).convert("RGB")
        if self.annotations.iloc[idx]['Majority Vote Label'] =='HP':
            label = 0
        else:
            label = 1
        if self.transform:
            image = self.transform(image)
        return image, label
```

2) Estimate Mean and STD of MHIST: Code for estimating the Mean and STD of MHIST MHIST Dataloader

```
import os
import pandas as pd
from PIL import Image
from torch.utils.data import Dataset, DataLoader
from torchvision import transforms
import torch
transform = transforms.Compose([
    transforms.Resize((128, 128)),  # Resize images to a standard size (optional)
    transforms.ToTensor()])
dataset = MHISTDataset(csv_file='./annotations.csv',
                       root_dir='./images/images',
```

```
                    transform=, train=True)

# Create the DataLoader
dataloader = DataLoader(dataset, batch_size=1, shuffle=True, num_workers=4)
# Usage example
Imgs = torch.zeros(2175, 3, 128, 128)
for idx, (images, labels) in enumerate(dataloader):
    Imgs[idx] = images
print(torch.mean(Imgs, (0,2, 3)), torch.std(Imgs, (0,2, 3)))
```

3) Code for : cross archtecture

```
# from utils import evaluate_synset, get_dataset
from utils import get_loops, get_dataset, get_network, get_eval_pool, evaluate_synset, get_dapa
from networks import ResNet18
import argparse
import torch
import torch.distributed as dist
import torch.cuda.comm
def main():
    parser = argparse.ArgumentParser(description='Parameter Processing')
    parser.add_argument('--dataset', type=str, default='MHIST', help='dataset')
    parser.add_argument('--model', type=str, default='ConvNetD7', help='model')
    parser.add_argument('--ipc', type=int, default=50, help='image(s) per class')
    parser.add_argument('--eval_mode', type=str, default='SS', help='eval_mode')
    parser.add_argument('--num_exp', type=int, default=1, help='the number of experiments')
    parser.add_argument('--num_eval', type=int, default=200, help='the number of evaluating ra
    parser.add_argument('--epoch_eval_train', type=int, default=1000, help='epochs to train a
    parser.add_argument('--Iteration', type=int, default=10, help='training iterations')
    parser.add_argument('--lr_img', type=float, default=0.1, help='learning rate for updating
    parser.add_argument('--lr_net', type=float, default=0.01, help='learning rate for updating
    parser.add_argument('--batch_real', type=int, default=128, help='batch size for real data'
    parser.add_argument('--batch_train', type=int, default=128, help='batch size for training
    parser.add_argument('--init', type=str, default='real', help='noise/real/smart: initialize
    parser.add_argument('--dsa_strategy', type=str, default='color_crop_cutout_flip_scale_rotat
    parser.add_argument('--data_path', type=str, default='C:/Users/samsa/Downloads/ECE1512_20
    parser.add_argument('--zca', type=bool, default=False, help='Zca Whitening')
    parser.add_argument('--save_path', type=str, default='./data_h', help='path to save results
    parser.add_argument('--dd', type=str, default='./data_h', help='dataset distillation path'
    parser.add_argument('--task_balance', type=float, default=0.01, help='balance attention wi
    args = parser.parse_args()
    args.method = 'DataDAM'
    args.device = 'cuda' if torch.cuda.is_available() else 'cpu'
    args.dsa_param = ParamDiffAug()
    args.dsa = False if args.dsa_strategy in ['none', 'None'] else True

    channel, im_size, num_classes, class_names, mean, std, dst_train, dst_test, testloader, zca
    net = ResNet18(channel=channel, num_classes=num_classes)
    tmp = torch.load(args.dd)
    image_syn_eval = tmp['data'][-1][0]
    label_syn_eval = tmp['data'][-1][1]
    for i in range(3):
        mini_net, acc_train, acc_test = evaluate_synset(i, net, image_syn_eval,
                                                        label_syn_eval, testloader, args)

        print(mini_net, acc_train, acc_test)
if __name__ == '__main__':
    main()
```

4) Code for : continual learning

The following code implementation is derived from the [5] GitHub repository.

```
import os
import numpy as np
import torch
import argparse
from utils import get_dataset, get_network, get_eval_pool, evaluate_synset, ParamDiffAug, Tens
import copy
import gc
import torch.nn as nn
def main():
    parser = argparse.ArgumentParser(description='Parameter Processing')
    parser.add_argument('--method', type=str, default='random', help='random/herding/DSA/DM')
    parser.add_argument('--datadam', type=int, default=0, help='Using datadam')
    parser.add_argument('--dataset', type=str, default='CIFAR100', help='dataset')
    parser.add_argument('--model', type=str, default='ConvNet', help='model')
    parser.add_argument('--ipc', type=int, default=20, help='image(s) per class')
    parser.add_argument('--steps', type=int, default=5, help='5/10-step learning')
    parser.add_argument('--num_eval', type=int, default=3, help='evaluation number')
    parser.add_argument('--epoch_eval_train', type=int, default=1000, help='epochs to train a
    parser.add_argument('--lr_net', type=float, default=0.01, help='learning rate for updating
    parser.add_argument('--batch_train', type=int, default=256, help='batch size for training
    parser.add_argument('--data_path', type=str, default='./../dd', help='dataset path')
    parser.add_argument('--dd', type=str, default='./../data', help='dataset path')
    args = parser.parse_args()
    args.device = 'cuda' if torch.cuda.is_available() else 'cpu'
    args.dsa_param = ParamDiffAug()
    args.dsa = True # augment images for all methods
    args.dsa_strategy = 'color_crop_cutout_flip_scale_rotate' # for CIFAR10/100
    if not os.path.exists(args.data_path):
        os.mkdir(args.data_path)
    channel, im_size, num_classes, class_names, mean, std, dst_train, dst_test, testloader = g
    images_all = []
    labels_all = []
    indices_class = [[] for c in range(num_classes)]
    images_all = [torch.unsqueeze(dst_train[i][0], dim=0) for i in range(len(dst_train))]
    labels_all = [dst_train[i][1] for i in range(len(dst_train))]
    for i, lab in enumerate(labels_all):
        indices_class[lab].append(i)
    images_all = torch.cat(images_all, dim=0).to(args.device)
    labels_all = torch.tensor(labels_all, dtype=torch.long, device=args.device)
    tmp = torch.load(args.dd)
    image_syn_eval = tmp['data'][-1][0]
    label_syn_eval = tmp['data'][-1][1]
    images_train = image_syn_eval.to(args.device)
    labels_train = label_syn_eval.to(args.device)
    def get_images(c, n):  # get random n images from class c
        idx_shuffle = np.random.permutation(indices_class[c])[:n]
        return images_all[idx_shuffle]
    print('method: ', args.method)
    results = np.zeros((args.steps, 5*args.num_eval))
    for seed_cl in range(5):
        num_classes_step = num_classes // args.steps
        np.random.seed(seed_cl)
        class_order = np.random.permutation(num_classes).tolist()
        print('=========================================')
        print('seed: ', seed_cl)
```

```
print('class_order: ', class_order)
print('augmentation strategy: \n', args.dsa_strategy)
print('augmentation parameters: \n', args.dsa_param.__dict__)
if args.method == 'random':
    images_train_all = []
    labels_train_all = []
    for step in range(args.steps):
        classes_current = class_order[step * num_classes_step: (step + 1) * num_classe
        images_train_all += [torch.cat([get_images(c, args.ipc) for c in classes_curre
        labels_train_all += [torch.tensor([c for c in classes_current for i in range(a
elif args.method == 'herding':
    fname = os.path.join(args.data_path, 'metasets', 'cl_data', 'cl_herding_CIFAR100_C
    data = torch.load(fname, map_location='cpu')['data']
    images_train_all = [data[step][0] for step in range(args.steps)]
    labels_train_all = [data[step][1] for step in range(args.steps)]
    print('use data: ', fname)
elif args.method == 'DSA':
    fname = os.path.join(args.data_path, 'metasets', 'cl_data', 'cl_res_DSA_CIFAR100_C
    data = torch.load(fname, map_location='cpu')['data']
    images_train_all = [data[step][0] for step in range(args.steps)]
    labels_train_all = [data[step][1] for step in range(args.steps)]
    print('use data: ', fname)

elif args.method == 'DM':
    fname = os.path.join(args.data_path, 'metasets', 'cl_data', 'cl_DM_CIFAR100_ConvN
    data = torch.load(fname, map_location='cpu')['data']
    images_train_all = [data[step][0] for step in range(args.steps)]
    labels_train_all = [data[step][1] for step in range(args.steps)]
    print('use data: ', fname)
else:
    exit('unknown method: %s'%args.method)
for step in range(args.steps):
    print('\n----------------------------\nmethod %s seed %d step %d ' % (args.method
    classes_seen = class_order[: (step+1)*num_classes_step]
    print('classes_seen: ', classes_seen)
    images_test = []
    labels_test = []
    for i in range(len(dst_test)):
        lab = int(dst_test[i][1])
        if lab in classes_seen:
            images_test.append(torch.unsqueeze(dst_test[i][0], dim=0))
            labels_test.append(dst_test[i][1])
    images_test = torch.cat(images_test, dim=0).to(args.device)
    labels_test = torch.tensor(labels_test, dtype=torch.long, device=args.device)
    dst_test_current = TensorDataset(images_test, labels_test)
    testloader = torch.utils.data.DataLoader(dst_test_current, batch_size=256, shuffle=
    print('test set size: ', images_test.shape)
    accs = []
    for ep_eval in range(args.num_eval):
        net_eval = get_network(args.model, channel, num_classes, im_size)
        net_eval = net_eval.to(args.device)
        img_syn_eval = copy.deepcopy(images_train.detach())
        lab_syn_eval = copy.deepcopy(labels_train.detach())

        _, acc_train, acc_test = evaluate_synset(ep_eval, net_eval, img_syn_eval, Due
        del net_eval, img_syn_eval, lab_syn_eval
        gc.collect()  # to reduce memory cost
        accs.append(acc_test)
```

```
                results[step, seed_cl*args.num_eval + ep_eval] = acc_test
            print('Evaluate %d random %s, mean = %.4f std = %.4f' % (len(accs), args.model, np
    results_str = ''
    for step in range(args.steps):
        results_str += '& %.1f$\pm$%.1f  ' % (np.mean(results[step]) * 100, np.std(results[ste
    print('%d step learning %s perforamnce:'%(args.steps, args.method))
    print(results_str)
    print('Done')Due today
if __name__ == '__main__':
    main()
```

B. Code of Cafe

The following code implementation is derived from the DataDam [9] GitHub repository.

```
import os
import time
import copy
import argparse
import numpy as np
import torch
import torch.nn as nn
from torchvision.utils import save_image
from utils import get_loops, get_dataset, get_network, get_eval_pool, evaluate_synset, get_dapa
    match_loss, get_time, TensorDataset, epoch, DiffAugment, ParamDiffAug
import os
import logging
import random
import torch.nn as nn
def build_logger(work_dir, cfgname):
    assert cfgname is not None
    log_file = cfgname + '.log'
    log_path = os.path.join(work_dir, log_file)
    logger = logging.getLogger(cfgname)
    logger.setLevel(logging.INFO)
    # formatter = logging.Formatter('%(asctime)s %(levelname)s %(message)s')
    formatter = logging.Formatter('%(asctime)s: %(message)s', datefmt='%Y-%m-%d %H:%M:%S')
    handler1 = logging.FileHandler(log_path)
    handler1.setFormatter(formatter)
    logger.addHandler(handler1)
    handler2 = logging.StreamHandler()
    handler2.setFormatter(formatter)
    logger.addHandler(handler2)
    logger.propagate = False
    return logger
def adjust_learning_rate(optimizer, epoch, init_lr):
    """Decay the learning rate based on schedule"""
    lr = init_lr
    for milestone in [1200, 1600, 1800]:
        lr *= 0.5 if epoch >= milestone else 1.
    for param_group in optimizer.param_groups:
        param_group['lr'] = lr
def criterion_middle(real_feature, syn_feature, args):
    MSE_Loss = nn.MSELoss(reduction='sum')
    shape_real = real_feature.shape
    if args.dataset =="MHIST":
        real_feature = torch.mean(real_feature.view(2, shape_real[0] // 2, *shape_real[1:]), di
        shape_syn = syn_feature.shape
```

```
        syn_feature = torch.mean(syn_feature.view(2, shape_syn[0] // 2, *shape_syn[1:]), dim=1
    else:
        real_feature = torch.mean(real_feature.view(10, shape_real[0] // 10, *shape_real[1:]),

        shape_syn = syn_feature.shape
        syn_feature = torch.mean(syn_feature.view(10, shape_syn[0] // 10, *shape_syn[1:]), dim
    return MSE_Loss(real_feature, syn_feature)
def main():
    parser = argparse.ArgumentParser(description='Parameter Processing')
    parser.add_argument('--method', type=str, default='DC', help='DC/DSA')
    parser.add_argument('--dataset', type=str, default='CIFAR10', help='dataset')
    parser.add_argument('--model', type=str, default='ConvNet', help='model')
    parser.add_argument('--ipc', type=int, default=1, help='image(s) per class')
    parser.add_argument('--eval_mode', type=str, default='S',
                        help='eval_mode')  # S: the same to training model, M: multi architectu
W: net width, D: net depth, A: activation function, P: pooling layer, N: normalization layer,
    parser.add_argument('--num_exp', type=int, default=1, help='the number of experiments')
    parser.add_argument('--num_eval', type=int, default=1, help='the number of evaluating rando
    parser.add_argument('--epoch_eval_train', type=int, default=100, help='epochs to train a m
    parser.add_argument('--Iteration', type=int, default=2000, help='training iterations')
    parser.add_argument('--lr_img', type=float, default=0.1, help='learning rate for updating
    parser.add_argument('--lr_net', type=float, default=0.01, help='learning rate for updating
    parser.add_argument('--batch_real', type=int, default=256, help='batch size for real data'
    parser.add_argument('--batch_train', type=int, default=256, help='batch size for training
    parser.add_argument('--init', type=str, default='noise',
                        help='noise/real: initialize synthetic images from random noise or ran
    parser.add_argument('--dsa_strategy', type=str, default='None', help='differentiable Siame
    parser.add_argument('--data_path', type=str, default='data', help='dataset path')
    parser.add_argument('--save_path', type=str, default='oi_cifar10_ipc10_watcher_5_v3', help
    parser.add_argument('--dis_metric', type=str, default='ours', help='distance metric')
    parser.add_argument('--fourth_weight', type=float, default=0.1, help='batch size for traini
    parser.add_argument('--third_weight', type=float, default=0.1, help='batch size for trainin
    parser.add_argument('--second_weight', type=float, default=1.0, help='batch size for traini
    parser.add_argument('--first_weight', type=float, default=1.0, help='batch size for trainin
    parser.add_argument('--inner_weight', type=float, default=0.01, help='batch size for traini
    parser.add_argument('--lambda_1', type=float, default=0.04, help='break outlooper')
    parser.add_argument('--lambda_2', type=float, default=0.03, help='break innerlooper')
    parser.add_argument('--gpu_id', type=str, default='0', help='dataset path')
    args = parser.parse_args()
    logger = build_logger('.', cfgname=str(args.lambda_1) + "_" + str(args.lambda_2) + "_" + s
        args.inner_weight) + '_' + str(args.fourth_weight) + '_' + str(args.third_weight) + '_
        args.second_weight) + '_' + str(args.first_weight) + 'oi_cifar10_dsa_ipc50_watcher_5_v3
    os.environ["CUDA_VISIBLE_DEVICES"] = args.gpu_id
    args.outer_loop, args.inner_loop = get_loops(args.ipc)
    # import pdb; pdb.set_trace()
    args.save_path = str(args.lambda_1) + "_" + str(args.lambda_2) + "_" + 'oi_cifar10_ipc10_wa
    args.device = 'cuda' if torch.cuda.is_available() else 'cpu'
    args.dsa_param = ParamDiffAug()
    args.dsa = True if args.method == 'DSA' else False
    if not os.path.exists(args.data_path):
        os.mkdir(args.data_path)
    if not os.path.exists(args.save_path):
        os.mkdir(args.save_path)
    eval_it_pool = np.arange(0, args.Iteration + 1, 100).tolist() if args.eval_mode == 'S' else
        args.Iteration]  # The list of iterations when we evaluate models and record results.
    channel, im_size, num_classes, class_names, mean, std, dst_train, dst_test, testloader = g
   args.data_path)
    model_eval_pool = get_eval_pool(args.eval_mode, args.model, args.model)
```

```
    accs_all_exps = dict()  # record performances of all experiments
    for key in model_eval_pool:
        accs_all_exps[key] = []
    data_save = []
    for exp in range(args.num_exp):
        logger.info('================== Exp %d ==================' % exp)
        logger.info('Hyper-parameters: \n', args.__dict__)
        print('Evaluation model pool: ', model_eval_pool)

        ''' organize the real dataset '''
        images_all = []
        labels_all = []
        indices_class = [[] for c in range(num_classes)]

        images_all = [torch.unsqueeze(dst_train[i][0], dim=0) for i in range(len(dst_train))]
        labels_all = [dst_train[i][1] for i in range(len(dst_train))]
        for i, lab in enumerate(labels_all):
            indices_class[lab].append(i)
        images_all = torch.cat(images_all, dim=0)
        labels_all = torch.tensor(labels_all, dtype=torch.long, device=args.device)
        for c in range(num_classes):
            logger.info('class c = %d: %d real images' % (c, len(indices_class[c])))
        def get_images(c, n):  # get random n images from class c
            # import pdb; pdb.set_trace()
            idx_shuffle = np.random.permutation(indices_class[c])[:n]
            return images_all[idx_shuffle].to(args.device)
        for ch in range(channel):
            logger.info('real images channel %d, mean = %.4f, std = %.4f' % (
                ch, torch.mean(images_all[:, ch]), torch.std(images_all[:, ch])))
        image_syn = torch.randn(size=(num_classes * args.ipc, channel, im_size[0], im_size[1])
                                requires_grad=True, device=args.device)
        label_syn = torch.tensor([np.ones(args.ipc) * i for i in range(num_classes)], dtype=to
                                 requires_grad=False, device=args.device).view(-1)  # [0,0,0,
        if args.init == 'real':
            logger.info('initialize synthetic data from random real images')
            for c in range(num_classes):
                image_syn.data[c * args.ipc:(c + 1) * args.ipc] = get_images(c, args.ipc).deta
        else:
            logger.info('initialize synthetic data from random noise')
        optimizer_img = torch.optim.SGD([image_syn, ], lr=args.lr_img, momentum=0.5)
# optimizer_img for synthetic data
        optimizer_img.zero_grad()
        criterion = nn.CrossEntropyLoss().to(args.device)
        criterion_sum = nn.CrossEntropyLoss(reduction='sum').to(args.device)
        logger.info('%s training begins' % get_time())
        for it in range(args.Iteration + 1):
            adjust_learning_rate(optimizer_img, it, args.lr_img)
            if it in eval_it_pool:
                for model_eval in model_eval_pool:
                    logger.info(
                        '-------------------------\nEvaluation\nmodel_train = %s, model_eval =
                            args.model, model_eval, it))
                    if args.dsa:
                        args.epoch_eval_train = 1000
                        args.dc_aug_param = None
                        logger.info('DSA augmentation strategy: \n' + args.dsa_strategy)
                        logger.info('DSA augmentation parameters: \n' + str(args.dsa_param.__d
                    else:
```

```
            # This augmentation parameter set is only for DC method. It will be mu
            args.dc_aug_param = get_daparam(args.dataset, args.model, model_eval,
                                            args.ipc)
            logger.info('DC augmentation parameters: \n' + str(args.dc_aug_param))

        if args.dsa or args.dc_aug_param['strategy'] != 'none':
            args.epoch_eval_train = 1000  # Training with data augmentation needs
        else:
            args.epoch_eval_train = 600

        accs = []
        for it_eval in range(args.num_eval):
            # get a random model
            net_eval = get_network(model_eval, channel, num_classes, im_size).to(
                args.device)
            # avoid any unaware modification
            image_syn_eval, label_syn_eval = copy.deepcopy(image_syn.detach()), co
                label_syn.detach())
            _, acc_train, acc_test = evaluate_synset(it_eval, net_eval, image_syn_
                                                     testloader, args)
            accs.append(acc_test)
        logger.info('Evaluate %d random %s, mean = %.4f std = %.4f\n-------------
            len(accs), model_eval, np.mean(accs), np.std(accs)))
        # record the final results
        if it == args.Iteration:
            accs_all_exps[model_eval] += accs
if it % 20 ==0:
    ''' visualize and save '''
    save_name = os.path.join(args.save_path, 'vis_%s_%s_%s_%dipc_exp%d_iter%d.png'
        args.method, args.dataset, args.model, args.ipc, exp, it))
    image_syn_vis = copy.deepcopy(image_syn.detach().cpu())
    for ch in range(channel):
        image_syn_vis[:, ch] = image_syn_vis[:, ch] * std[ch] + mean[ch]
    image_syn_vis[image_syn_vis < 0] = 0.0
    image_syn_vis[image_syn_vis > 1] = 1.0
    # Trying normalize = True/False may get better visual effects.
    save_image(image_syn_vis, save_name,
               nrow=args.ipc)
# get a random model
net = get_network(args.model, channel, num_classes, im_size).to(args.device)
net.train()
net_parameters = list(net.parameters())
optimizer_net = torch.optim.SGD(net.parameters(), lr=args.lr_net)  # optimizer_img
optimizer_net.zero_grad()
loss_avg = 0
loss_kai = 0
loss_middle_item = 0
args.dc_aug_param = None  # Mute the DC augmentation when training synthetic data.
acc_watcher = list()
pop_cnt = 0
acc_test = 0.0
while True:
    syn_centers = []
    real_feature_concat = []
    real_feature_concat_mm = []
    real_label_concat = []
    img_real_gather = []
    img_syn_gather = []
```

```
            lab_real_gather = []
            lab_syn_gather = []
            loss = torch.tensor(0.0).to(args.device)
            for c in range(num_classes):
                img_real = get_images(c, args.batch_real)
                lab_real = torch.ones((img_real.shape[0],), device=args.device, dtype=torcl
                img_syn = image_syn[c * args.ipc:(c + 1) * args.ipc].reshape(
                    (args.ipc, channel, im_size[0], im_size[1]))
                lab_syn = torch.ones((args.ipc,), device=args.device, dtype=torch.long) *
                if args.dsa:
                    seed = int(time.time() * 1000) % 100000
                    img_real = DiffAugment(img_real, args.dsa_strategy, seed=seed, param=a
                    img_syn = DiffAugment(img_syn, args.dsa_strategy, seed=seed, param=arg
                img_real_gather.append(img_real)
                lab_real_gather.append(lab_real)
                img_syn_gather.append(img_syn)
                lab_syn_gather.append(lab_syn)
            ImageShape = img_real_gather[0].shape
            if args.dataset == "MHIST":
                img_real_gather = torch.stack(img_real_gather, dim=0).reshape(args.batch_r
                img_syn_gather = torch.stack(img_syn_gather, dim=0).reshape(args.ipc * 2,
ImageShape[1], ImageShape[2], ImageShape[3])
                lab_real_gather = torch.stack(lab_real_gather, dim=0).reshape(args.batch_r
                lab_syn_gather = torch.stack(lab_syn_gather, dim=0).reshape(args.ipc * 2)
            else:
                img_real_gather = torch.stack(img_real_gather, dim=0).reshape(args.batch_r
                img_syn_gather = torch.stack(img_syn_gather, dim=0).reshape(args.ipc * 2,
ImageShape[1], ImageShape[2], ImageShape[3])
                lab_real_gather = torch.stack(lab_real_gather, dim=0).reshape(args.batch_r
                lab_syn_gather = torch.stack(lab_syn_gather, dim=0).reshape(args.ipc * 2)
            output_real, real_features = net(
                img_real_gather)
            output_syn, syn_features = net(
                img_syn_gather)
            loss_middle = args.fourth_weight * criterion_middle(real_features[-1], syn_fea
                        + args.third_weight * criterion_middle(real_features[-2], syn_feat
                        + args.second_weight * criterion_middle(real_features[-3], syn_fea
                        + args.first_weight * criterion_middle(real_features[-4], syn_feat
            loss_real = criterion(output_real, lab_real_gather)
            loss += loss_middle
            loss += loss_real
            if args.dataset =="MHIST":
                last_real_feature = torch.mean(real_features[0].view(2, int(real_features[
                last_syn_feature = torch.mean(syn_features[0].view(2, int(syn_features[0].
                output = torch.mm(real_features[0], last_syn_feature.t())
                last_real_feature = torch.mean(
                    last_real_feature.unsqueeze(0).reshape(2, int(last_real_feature.shape[
                                                           last_real_feature.shape[1]), dim=1)
            else:
                last_real_feature = torch.mean(real_features[0].view(10, int(real_features
                last_syn_feature = torch.mean(syn_features[0].view(10, int(syn_features[0]
                output = torch.mm(real_features[0], last_syn_feature.t())
                last_real_feature = torch.mean(
                    last_real_feature.unsqueeze(0).reshape(10, int(last_real_feature.shape
                                                           last_real_feature.shape[1]), dim=1)
            loss_output = criterion_middle(last_syn_feature, last_real_feature, args) + ar
            loss += loss_output
            loss.backward()
```

```
        optimizer_img.step()
        optimizer_img.zero_grad()
        loss_avg += loss.item()
        loss_kai += loss_output.item()
        loss_middle_item += loss_middle.item()
        ############ for outloop testing ############

        for c in range(num_classes):
            img_real_test = get_images(c, 128)
            lab_real_test = torch.ones((img_real_test.shape[0],), device=args.device,
            prob, _ = net(img_real_test)
            acc_test += (lab_real_test == prob.max(dim=1)[1]).float().mean()
        acc_test /= num_classes
        acc_watcher.append(acc_test.detach().cpu())
        pop_cnt += 1
        if len(acc_watcher) == 10:
            if max(acc_watcher) - min(acc_watcher) < args.lambda_1:
                acc_watcher = list()
                pop_cnt = 0
                acc_test = 0.0
                break
            else:
                acc_watcher.pop(0)
        image_syn_train, label_syn_train = copy.deepcopy(image_syn.detach()), copy.deep
            label_syn.detach())  # avoid any unaware modification
        dst_syn_train = TensorDataset(image_syn_train, label_syn_train)
        trainloader = torch.utils.data.DataLoader(dst_syn_train, batch_size=args.batch
                                                  num_workers=0)
        acc_inner_watcher = list()
        acc_syn_inner_watcher = list()
        pop_inner_cnt = 0
        acc_inner_test = 0
        # for il in range(args.inner_loop):
        while (1):
            inner_loss, inner_acc = epoch('train', trainloader, net, optimizer_net, cr
                                          aug=True if args.dsa else False)
            acc_syn_inner_watcher.append(inner_acc)
            for c in range(num_classes):
                img_real_test = get_images(c, 128)
                lab_real_test = torch.ones((img_real_test.shape[0],), device=args.devic
                prob, _ = net(img_real_test)
                acc_inner_test += (lab_real_test == prob.max(dim=1)[1]).float().mean()
            acc_inner_test /= num_classes
            acc_inner_watcher.append(acc_inner_test.detach().cpu())
            pop_inner_cnt += 1
            if len(acc_inner_watcher) == 10:
                if max(acc_inner_watcher) - min(acc_inner_watcher) > args.lambda_2:
                    acc_inner_watcher = list()
                    acc_syn_inner_watcher = list()
                    pop_inner_cnt = 0
                    acc_inner_test = 0
                    break
                else:
                    acc_inner_watcher.pop(0)
            epoch('test', trainloader, net, optimizer_net, criterion, args, aug=True i
    loss_avg /= (num_classes * args.outer_loop)
    # if it % 10 == 0:
    logger.info('%s iter = %04d, loss = %.4f, loss_kai = %.4f, loss_middle = %.4f' % (
```

```
                get_time(), it, loss_avg, loss_kai, loss_middle_item))
    logger.info('\n==================== Final Results ====================\n')
    for key in model_eval_pool:
        accs = accs_all_exps[key]
        logger.info('Run %d experiments, train on %s, evaluate %d random %s, mean  = %.2f%%
std = %.2f%%' % (
            args.num_exp, args.model, len(accs), key, np.mean(accs) * 100, np.std(accs) * 100)
if __name__ == '__main__':
    main()
```