Supplementary Material for Paper: Leveraging Hierarchical Feature Sharing for Efficient Dataset Condensation

Haizhong Zheng[1], Jiachen Sun[1], Shutong Wu[2], Bhavya Kailkhura[3], Z. Morley Mao[1], Chaowei Xiao[2], and Atul Prakash[1]

[1] University of Michigan, Ann Arbor
[2] University of Wisconsin, Madison
[3] Lawrence Livermore National Laboratory

A Appendix Overview

In this appendix, we provide more details on our experimental setting and additional evaluation results. In Section B, we discuss the computation cost caused by data parameterization. In Section C, we introduce the detailed setup of our experiment to improve the reproducibility of our paper. In Section D, we conduct additional studies on how the pruning rate influences the model performance. In addition, we visualize images generated by HMNs in Section D.5.

B Discussion

While data parameterization methods demonstrate effective performance in data condensation, we show that generated images per class (GIPC) play an important role in data parameterization. The payoff is that HMNs, along with other SOTA data parameterization methods [2,3,5] invariably generate a higher quantity of images than those condensed and stored in pixel space with a specific storage budget, which may potentially escalate the cost of data condensation. A limitation of HMNs and other data parameterization methods is that determining the parameters of the data container to achieve high-quality data condensation can be computationally demanding. Besides, more generated images can lead to longer training time with condensed datasets. In Section D.2, we show that, even though HMNs generate more training images, training on condensed datasets generated by HMNs achieves better test accuracy within the same training time.

Another difference between data parameterization and conventional DC methods using images as data containers is that data parameterization methods need to generate images before training with condensed datasets. It is important to note that this additional step incurs only a minimal overhead, as it merely requires a single forward pass of HMNs. For example, on a 2080TI, the generation time for a 1 IPC, 10 IPC, and 50 IPC CIFAR10 HMN is 0.036s, 0.11s, and 0.52s, respectively (average number through 100 repeats).

C Experiment Setting and Implementation Details

C.1 HMN architecture design.

In this section, we introduce more details on the designs of the Hierarchical Memory Network (HMN) architecture, specifically tailored for various datasets and storage budgets. We first introduce the three-tier hierarchical memories incorporated within the network. Subsequently, we present the neural network designed to convert memory and decode memories into images utilized for training.

Table 1: The detailed three-tier memory settings. We use the same setting for CIFAR10 and SVHN. #Instance-level memory is the number of memory fitting the storage budget. #Instance-level memory (Over-budget) indicates the actual number of instance-level memory that we use for condensation, and we prune this number to #Instance-level memory after condensation. I-10 stands for ImageNet-10.

Dataset	SVHN & CIFAR10			CIFAR100			Tiny		I-10
IPC	1	10	50	1	10	50	1	10	1
Dataset-level memory channels	5	50	50	5	50	50	30	50	30
Class-level memory channels	3	30	30	3	30	30	20	30	25
Instance-level memory channels	2	6	8	2	8	14	4	10	8
#Instance-level memory	85	278	1168	93	219	673	42	185	125
#Instance-level memory (Over-budget)	93	306	1284	102	243	740	46	203	138

Hierarchical memories. HMNs consist of three-tier memories: dataset-level memory $m^{(D)}$, class-level memory $m_c^{(C)}$, and instance-level memory $m_{c,i}^{(I)}$, which are supposed to store different levels of features of datasets. Memories of HMNs for SVHN, CIFAR10, and CIFAR100 have a shape of (4, 4, Channels), and memories for Tiny ImageNet have a shape of (8, 8, Channels). Memories for ImageNet-10 have a shape of (12, 12, Channels). The number of channels is a hyper-parameter for different settings.

We present the detailed setting for the number of channels and the number of memories under different data condensation scenarios in Table 1. Besides the channels of memories, we also present the number of instance-level memories. Since each instance-level memory corresponds to a generated image, the number of instance-level of memories is the GPIC for an HMN. Every HMN has only one dataset-level memory, and the number of class-level memory is equal to the number of classes in the dataset. The number of instance-level memory for the over-budget class leads to an extra 10% storage budget cost, which will be pruned by post-condensation pruning.

Decoders. In addition to three-tier memories, each HMN has two types of networks: 1) A dataset-level memory feature extractor for each class; 2) A uniform decoder to convert memories to images for model training. *Dataset-level memory feature extractors* f_c are used to extract features from the dataset-level memory for each class. For 1 IPC storage budget setting, we use the identity function as the feature extractor to save the storage budget. For 10 IPC and 50

IPC storage budget settings, the feature extractors consist of a single deconvolutional layer with the kernel with 1 kernel size and 40 output channels. *The uniform decoder D* is used to generate images for training. For ImageNet-10, the size of the generated image is (3, 96, 96). We use the bilinear interpolation to resize the generated images to (3, 224, 224). In this paper, we adopt a classic design of decoder for image generation, which consist of a series of deconvolutional layers and batch normalization layers: ConvTranspose(Channels of memory, 10, 4, 1, 2) → Batch Normalization → ConvTranspose(10, 6, 4, 1, 2) → Batch Normalization → ConvTranspose(6, 3, 4, 1, 2). The arguments for ConvTranspose is input-channels, output-channels, kernel size, padding, and stride, respectively. The "Channels of memory" is equal to the addition of the channels of the output of f_c, the class-level memory channels, and the instance-level memory channels. When we design the HMN architecture, we also tried the design with different decoders for different classes. However, we find that it experiences an overfitting issue and leads to worse empirical performance.

C.2 Training settings

Baseline Settings In this paper, we evaluate HMN on the same model and architecture and with the same IPC setting as the baselines for a fair comparison. For various baselines, we directly report the numbers represented in their papers. In general, as far as we can tell, the authors of various baselines chose reasonable hyperparameter settings, such as learning rate, learning rate schedule, batch size, etc. for their scheme. Sometimes the chosen settings differ. For instance, LinBa [2] uses 0.1 as the learning rate, but HaBa [5] uses 0.01 as the learning rate. In keeping with past work in this area, we accept such differences, since the goal of each scheme is to achieve the best accuracy for a given IPC setting. The settings that we found to be reasonable choices for HMN are described below. The metrics on which all schemes are being evaluated are the same: accuracy that the scheme is able to achieve for a given IPC setting.

Data condensation. We generally follow the guidance and settings from past work for the data condensation component of HMN. Following previous works [2, 5, 8], we select ConvNet, which contains three convolutional layers followed by a pooling layer, as the network architecture for data condensation and classifier training for all three datasets. For ImageNet-10, following previous work, we choose ResNet-AP (a four-layer ResNet) to condense HMNs. We employ gradient matching [3, 5], a batch-based loss with low GPU memory consumption, to condense information into HMNs. More specifically, our code is implemented based on IDC [5]. For all datasets, we set the number of inner iterations to 200 for gradient matching loss. The total number of training epochs for data condensation is 1000. We use the Adam optimizer ($\beta_1 = 0.9$ and $\beta_2 = 0.99$) with a 0.01 initial learning rate (0.02 initial learning rate for CIFAR100) for data condensation. The learning rate scheduler is the step learning rate scheduler, and the learning rate will time a factor of 0.1 at 600 and 800 epochs. We use the mean squared error loss for calculating the distance of gradients for CIFAR10 and SVHN, and use L1 loss for the CIFAR100 and Tiny ImageNet. To find the

best hard pruning rate β in Algorithm 1, we perform a grid search from 0 to 0.9 with a 0.1 step. All experiments are run on a combination of RTX2080TI, RTX3090, A40, and A100, depending on memory usage and availability.

Model training with HMNs. For CIFAR10, we train the model with datasets generated by HMNs for 2000, 2000, and 1000 epochs for 1 IPC, 10 IPC, and 50 IPC, respectively. We use the SGD optimizer (0.9 momentum and 0.0002 weight decay) with a 0.01 initial learning rate.

For CIFAR100, we train the model with datasets generated by HMNs for 500 epochs. We use the SGD optimizer (0.9 momentum and 0.0002 weight decay) with a 0.01 initial learning rate.

For SVHN, we train the model with datasets generated by HMNs for 1500, 1500, 700 epochs for 1 IPC, 10 IPC, and 50 IPC, respectively. We use the SGD optimizer (0.9 momentum and 0.0002 weight decay) with a 0.01 initial learning rate.

For both Tiny-ImageNet and ImageNet-10, we train the model with datasets generated by HMNs for 300 epochs for both 1 IPC and 10 IPC settings. We use the SGD optimizer (0.9 momentum and 0.0002 weight decay) with a 0.02 initial learning rate.

Similar to [5], we use the DSA augmentation [8] and CutMix as data augmentation for data condensation and model training on HMNs. For HMN, for the learning rate scheduler, we use the cosine annealing learning rate scheduler [6] with a 0.0001 minimum learning rate. We preferred it over the multi-step learning rate scheduler primarily because the cosine annealing learning rate scheduler has fewer hyperparameters to choose. We also did an ablation study on the learning rate scheduler choice (see Appendix D.3) and did not find the choice of the learning rate scheduler to have a significant impact on the performance results.

Continual learning. Following the class incremental setting of [3], we adopt distillation loss [4] and train the model constantly by loading weights of the previous stage and expanding the output dimension of the last fully-connected layer [7]. Specifically, we use a ConvNet-3 model trained for 1000 epochs at each stage, using SGD with a momentum of 0.9 and a weight decay of $5e-4$. The learning rate is set to 0.01, and decays at epoch 600 and 800, with a decaying factor of 0.2.

D Additional Evaluation Results

In this section, we present additional evaluation results to further demonstrate the efficacy of HMNs. We study the relationship between pruning rate and accuracy in Section D.1. We then compare the training time with the condensed datasets in Section D.2. Subsequently, we conduct an ablation study on how different learning rate scheduler influences the training on condensed datasets in Section D.3. Additionally, we do data profiling and study the data redundancy on the condensed datasets synthesized by different DC methods in Section D.4. Lastly, we visualize the condensed training data generated by HMNs for different datasets in Section D.5.

D.1 Pruning Rate v.s. Accuracy

In this section, we examine the correlation between accuracy and pruning rates on HMNs. The evaluation results are presented in Figure 2. We observe that the accuracy drops more as the pruning rates increase, and our double-end pruning algorithm consistently outperforms random pruning. Furthermore, we observe that an increasing pruning rate results in a greater reduction in accuracy for HMNs with smaller storage budgets. For instance, when the pruning rate increases from 0 to 30%, models trained on the 1 IPC HMN experience a significant drop in accuracy, plunging from 66.2% to 62.2%. Conversely, models trained on the 50 IPC HMN exhibit a mere marginal decrease in accuracy, descending from 76.7% to 76.5% with the same increase in pruning rate. This discrepancy may be attributed to the fact that HMNs with larger storage budgets generate considerably more redundant data. Consequently, pruning such data does not significantly impair the training performance.

D.2 Training Time Comparison with Condensed Datasets

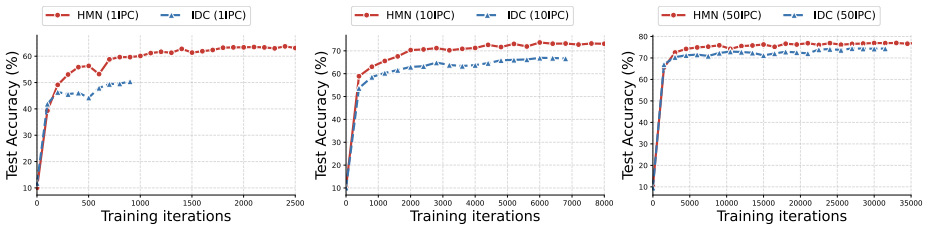


Fig. 1: Test accuracy over training iterations of training with condensed datasets generated by HMN and IDC. Given the same storage budget, although training with HMN needs more training iterations to converge, we find that HMNs achieve better accuracy within the same training time compared to IDC.

One potential limitation of HMNs is that, given the same storage budget, HMNs generate more images than other DC methods, which can potentially increase the cost of training with condensed datasets generated by HMNs. In this section, we conduct a study to study how test accuracy changes with respect to training iterations. We use the same batch size for both methods and follow the training setting suggested in the IDC paper. The comparison results are illustrated in Figure 1. Although condensed datasets generated by HMNs contain more training images, training with HMNs achieves better accuracy within the same training time across different training budgets. For instance, for 1 IPC at the 900th iteration, HMN achieves an accuracy of 60.1% while IDC only achieves 50.4% (at this point, IDC has converged, while HMN's accuracy can still be boosted further with more training iterations).

D.3 Ablation Study on Learning Rate Scheduler

We also train the model with a multi-step learning rate scheduler on CIFAR10 datasets generated by HMNs and found the following hyperparameter settings for a multi-step learning rate scheduler to work well: (a) an initial learning rate of 0.1; (b) The learning rate is multiplied with a 0.1 learning rate decay at 0.3 * total epochs / 0.6 * total epochs / 0.9 * total epochs. As shown in Table 2, we find the difference due to the LR scheduler choice to be overall marginal, and the results with the multistep LR scheduler do not change the findings of our evaluation. Our primary reason for choosing the cosine annealing LR scheduler in our evaluation is that it has fewer hyperparameters to choose from compared to the multistep LR scheduler. The cosine annealing LR scheduler only requires selection of an initial learning rate and a minimum learning rate. Those settings are described in Appendix C.2.

Table 2: Accuracy (%) performance comparison on different LR scheduler on CIFAR10. The evaluation results show that the difference due to the LR scheduler choice is overall marginal.

Data Container	1 IPC	10 IPC	50 IPC
Multi-step	65.7	73.4	76.8
Cosine Annealing	65.7	73.7	76.9

D.4 Data Profiling on SOTA Methods

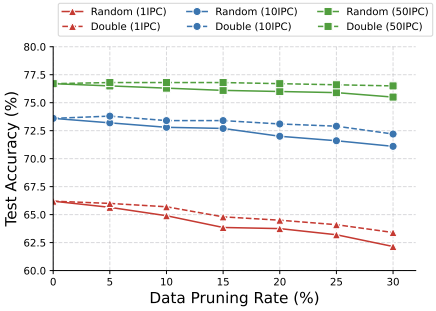


Fig. 2: Relationship between pruning rates and accuracy on HMNs for CIFAR10. All HMNs are over-budget HMNs (10% extra). Different colors stand for different storage budgets. Solid lines stand for random pruning and dashed lines stand for double-end pruning.

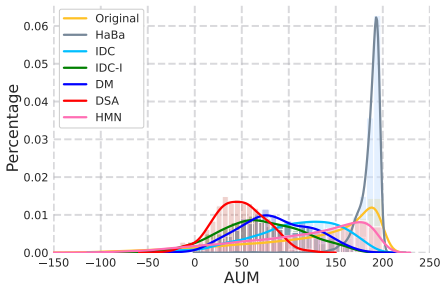


Fig. 3: The distribution of AUM of CIFAR10 training images synthesized by different approaches. Different colors denote different data condensation approaches. Data parameterization based methods have more redundant images.

Figure 3 illustrates the distribution of AUM of images synthesized by different data condensation approaches, as well as the original data, denoted as “Original".

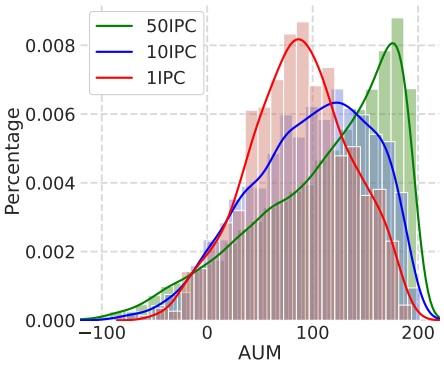


Fig. 4: AUM distribution of images generated by HMNs for CIFAR10 with different storage budgets, denoted by different colors.

Fig. 5: Visualization of the lowest and highest AUM examples generated by a 10 IPC HMN of CIFAR10. Each row represents a class.

We calculate the AUM by training a ConvNet for 200 epochs. We observe that approaches (IDC-I [3], DM [9], and DSA [8]) that condense data into pixel space typically synthesize fewer images with a high AUM value. In contrast, methods that rely on data parameterization, such as HaBa [5], IDC [3], and HMN [4], tend to produce a higher number of high-aum images. Notably, a large portion of images generated by HaBa exhibit an AUM value approaching 200, indicating a significant amount of redundancy that could potentially be pruned for enhanced performance. However, due to its factorization-based design, HaBa precludes the pruning of individual images from its data containers, which limits the potential for efficiency improvements.

Moreover, we conduct a more detailed study on the images generated by HMNs. We calculate the AUM by training a ConvNet for 200 epochs. As shown in Figure 4, many examples possess negative AUM values, indicating that they are likely hard-to-learn, low-quality images that may negatively impact training. Moreover, a considerable number of examples demonstrate AUM values approximating 200, representing easy-to-learn examples that may contribute little to the training process. We also observe that an increased storage budget results in a higher proportion of easier examples. This could be a potential reason why data condensation performance degrades to random selection when the storage budget keeps increasing, which is observed in [1]: more storage budgets add more easy examples which only provide redundant information and do not contribute much to training. From Figure 4, we can derive two key insights: 1) condensed datasets contain easy examples (AUM close to 200) as well as hard examples (AUM with negative values), and 2) the proportion of easy examples varies depending on the storage budget.

[4] We did not evaluate LinBa due to its substantial time requirements.

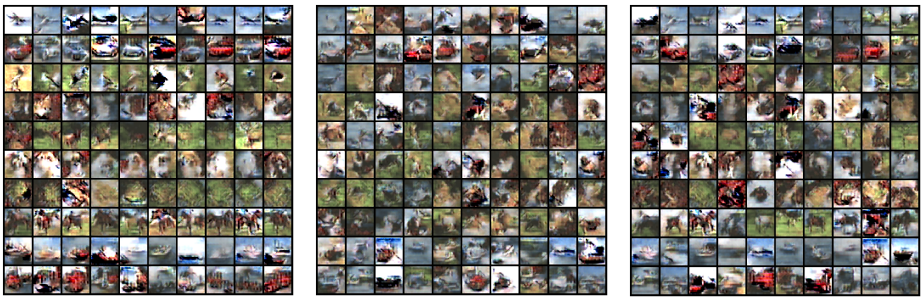
Additionally, in Figure 5, we offer a visualization of images associated with the highest and lowest AUM values generated by an HMN. It is observable that images with low AUM values exhibit poor alignment with their corresponding labels, which may detrimentally impact the training process. Conversely, images corresponding to high AUM values depict a markedly improved alignment with their classes. However, these images may be overly similar, providing limited information to training.

D.5 Visualization

To provide a better understanding of the images generated by HMNs, we visualize generated images with different AUM values on CIFAR10, CIFAR100, and SVHN with 1.1 IPC/11 IPC/55 IPC storage budgets in this section The visualization results are presented in the following images.

Similar to what we observe in Section 3.2 in the main paper, images with a high AUM value are better aligned with their respective labels. Conversely, images with a low AUM value typically exhibit low image quality or inconsistencies between their content and associated labels. For instance, in the visualizations of SVHNs (depicted in Figures 12 13 14), the numbers in the generated images with a high AUM value are readily identifiable, but content in the generated images with a low AUM value is hard to recognize. Those images are misaligned with their corresponding labels and can be detrimental to training. Pruning on those images can potentially improve training performance. Furthermore, we notice an enhancement in the quality of images generated by HMNs when more storage budgets are allocated. This improvement could be attributable to the fact that images generated by HMNs possess an enlarged instance-level memory, as indicated in Table 1. A larger instance-level memory stores additional information, thereby contributing to better image generation quality.

From the visualization, we also find that, unlike images generated by generative models, like GAN or diffusion models, images generated by HMNs do not exhibit comparably high quality. We would like to clarify that the goal of data condensation is not to generate high-quality images, but to generate images representing the training behavior of the original dataset. The training loss of data condensation can not guarantee the quality of the generated images.



High AUM (Easy) data Low AUM (Hard) data Randomly selected data

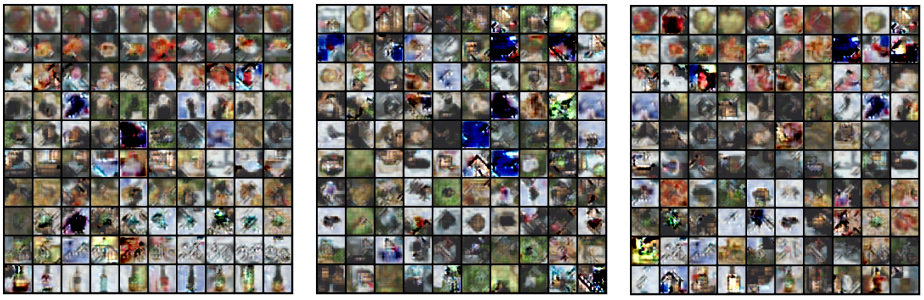
Fig. 6: Images generated by a CIFAR10 HMN with 1.1IPC storage budget. Images in each row are from the same class.

High AUM (Easy) data Low AUM (Hard) data Randomly selected data

Fig. 7: Images generated by a CIFAR10 HMN with 11IPC storage budget. Images in each row are from the same class.

High AUM (Easy) data Low AUM (Hard) data Randomly selected data

Fig. 8: Images generated by a CIFAR10 HMN with 55IPC storage budget. Images in each row are from the same class.

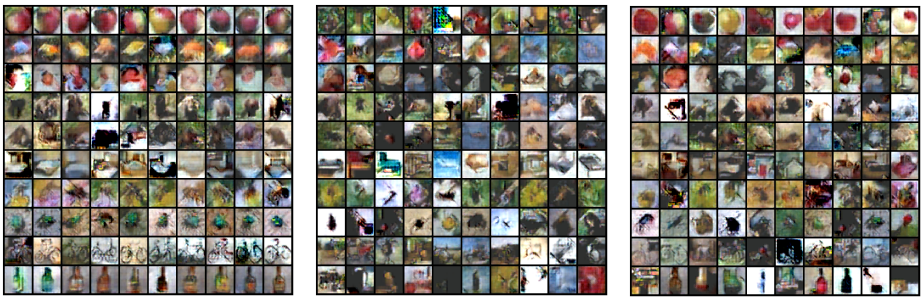


High AUM (Easy) data Low AUM (Hard) data Randomly selected data

Fig. 9: Images generated by a CIFAR100 HMN with 1.1IPC storage budget. Images in each row are from the same class. We only visualize 10 classes with the smallest class number in the dataset.

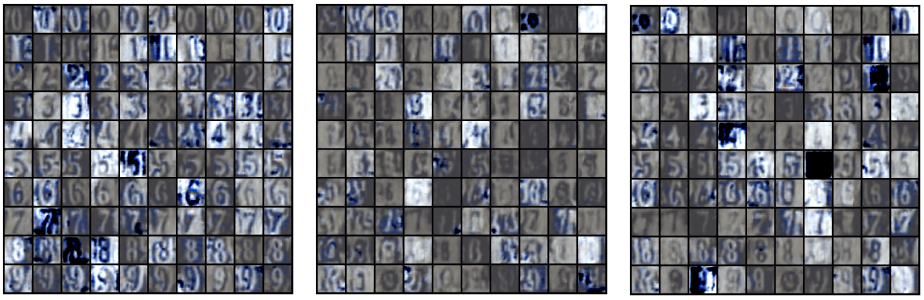
High AUM (Easy) data Low AUM (Hard) data Randomly selected data

Fig. 10: Images generated by a CIFAR100 HMN with 11IPC storage budget. Images in each row are from the same class. We only visualize 10 classes with the smallest class number in the dataset.



High AUM (Easy) data Low AUM (Hard) data Randomly selected data

Fig. 11: Images generated by a CIFAR100 HMN with 55IPC storage budget. Images in each row are from the same class. We only visualize 10 classes with the smallest class number in the dataset.



High AUM (Easy) data Low AUM (Hard) data Randomly selected data

Fig. 12: Images generated by an SVHN HMN with 1.1IPC storage budget. Images in each row are from the same class. Images with a low aum value are not well-aligned with its label and can be harmful for the training.

High AUM (Easy) data Low AUM (Hard) data Randomly selected data

Fig. 13: Images generated by an SVHN HMN with 11IPC storage budget. Images in each row are from the same class.

High AUM (Easy) data Low AUM (Hard) data Randomly selected data

Fig. 14: Images generated by an SVHN HMN with 55IPC storage budget. Images in each row are from the same class.

References

1. Cui, J., Wang, R., Si, S., Hsieh, C.J.: Dc-bench: Dataset condensation benchmark. In: Thirty-sixth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (2022)
2. Deng, Z., Russakovsky, O.: Remember the past: Distilling datasets into addressable memories for neural networks. In: Advances in Neural Information Processing Systems (2022)
3. Kim, J.H., Kim, J., Oh, S.J., Yun, S., Song, H., Jeong, J., Ha, J.W., Song, H.O.: Dataset condensation via efficient synthetic-data parameterization. In: International Conference on Machine Learning. pp. 11102–11118. PMLR (2022)
4. Li, Z., Hoiem, D.: Learning without forgetting. IEEE Transactions on Pattern Analysis and Machine Intelligence **40**(12), 2935–2947 (2018). https://doi.org/10.1109/TPAMI.2017.2773081
5. Liu, S., Wang, K., Yang, X., Ye, J., Wang, X.: Dataset distillation via factorization. In: Advances in Neural Information Processing Systems (2022)
6. Loshchilov, I., Hutter, F.: Sgdr: Stochastic gradient descent with warm restarts. In: ICLR (2017)
7. Rebuffi, S.A., Kolesnikov, A., Sperl, G., Lampert, C.H.: icarl: Incremental classifier and representation learning. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (July 2017)
8. Zhao, B., Bilen, H.: Dataset condensation with differentiable siamese augmentation. In: International Conference on Machine Learning. pp. 12674–12685. PMLR (2021)
9. Zhao, B., Bilen, H.: Dataset condensation with distribution matching. In: Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision. pp. 6514–6523 (2023)