

Leveraging Hierarchical Feature Sharing for Efficient Dataset Condensation

Haizhong Zheng¹, Jiachen Sun¹, Shutong Wu², Bhavya Kailkhura³,
Z. Morley Mao¹, Chaowei Xiao², and Atul Prakash¹

¹ University of Michigan, Ann Arbor

² University of Wisconsin, Madison

³ Lawrence Livermore National Laboratory

Abstract. Given a real-world dataset, data condensation (DC) aims to synthesize a small synthetic dataset that captures the knowledge of a natural dataset while being usable for training models with comparable accuracy. Recent works propose to enhance DC with *data parameterization*, which condenses data into very compact parameterized data containers instead of images. The intuition behind data parameterization is to encode *shared features* of images to avoid additional storage costs. In this paper, we recognize that images share common features in a hierarchical way due to the inherent hierarchical structure of the classification system, which is overlooked by current data parameterization methods. To better align DC with this hierarchical nature and encourage more efficient information sharing inside data containers, we propose a novel data parameterization architecture, *Hierarchical Memory Network (HMN)*. HMN stores condensed data in a three-tier structure, representing the dataset-level, class-level, and instance-level features. Another helpful property of the hierarchical architecture is that HMN naturally ensures good independence among images despite achieving information sharing. This enables instance-level pruning for HMN to reduce redundant information, thereby further minimizing redundancy and enhancing performance. We evaluate HMN on five public datasets and show that our proposed method outperforms all baselines.

1 Introduction

Introduced by Wang et al. [37], given a training dataset \mathcal{D} , the aim of *data condensation* (DC), also known as data distillation, is to synthesize a much smaller *synthetic dataset* \mathcal{S} such that \mathcal{S} can be used to train models that are comparable in test data performance to those trained on \mathcal{D} . Given increasing sizes of datasets, DC has emerged as an important goal for compute- and storage-efficient deep learning [2, 11, 14, 23, 24, 32, 48]. Researchers have shown that DC can provide significant efficiencies in diverse applications such as continual learning [29, 30], network architecture search [42], and federated learning [33, 39].

To improve the effectiveness of DC methods, Kim et al. [15] propose **data parameterization**. Instead of condensing data into images, data parameterization condenses data into parameterized data containers. Such a data container

is a parameterized function f_θ such that $f_\theta(\cdot)$ generates the synthetic dataset \mathcal{S} . The goal is that storing f_θ , represented by its parameters θ , is much more compact than storing \mathcal{S} . The intuition behind data parameterization methods is to encode *shared features* among images together into a *data container* to make the data condensation more effective [13, 19].

Recognizing this shared feature insight, it’s important to delve deeper into the inherent structure of these shared features in datasets. We notice that images share common features in a hierarchical way due to the inherent hierarchical structure of the classification system. Even if images differ in content, they can still share features at different hierarchical levels. For example, two images of cats can share common features specific to the cat class, but an image of a cat and another of a dog may still have shared features of the broader animal class.

However, current data parameterization methods that adopt factorization to share features among images overlook this hierarchical nature of shared features in datasets. In this paper, to better align with this hierarchical nature and encourage more efficient information sharing inside data containers, we propose a novel data parameterization architecture, *Hierarchical Memory Network (HMN)*. Figure 1 illustrates how HMN is used for data condensation. An HMN can be used to efficiently generate a synthetic dataset, which can then be used to train a model that is designed to be close in performance to a model that is trained on a

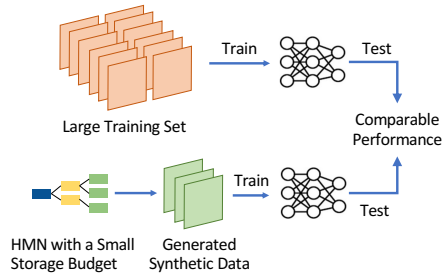


Fig. 1: Illustration of data condensation with HMN. Like other data parameterization methods, HMN is a data container using a small storage budget and can generate images for training.

larger dataset. Zooming in on the HMN, as illustrated in Figure 2, an HMN comprises a three-tier memory structure: *dataset-level memory*, *class-level memory*, and *instance-level memory*. Examples generated by HMNs share information via common dataset-level and class-level memories. Another helpful property of the hierarchical architecture is that HMN naturally ensures good independence among images. We find that condensed datasets contain redundant data, indicating room for further improvement in data condensation by pruning redundant data. In Section 3.2, we show that, compared to other data containers, HMNs are easier to prune. We propose an algorithm to prune HMNs to further reduce the redundancy in HMNs.

We evaluate our proposed methods on four public datasets (SVHN, CIFAR10, CIFAR100, and Tiny-ImageNet) and compare HMN with the other nine baselines. The evaluation results show that, even when trained with a low GPU memory consumption batch-based loss, HMN still outperforms all baselines, including those using high GPU memory trajectory-based losses. For a fair comparison, we also compare HMN with other data parameterization baselines under the

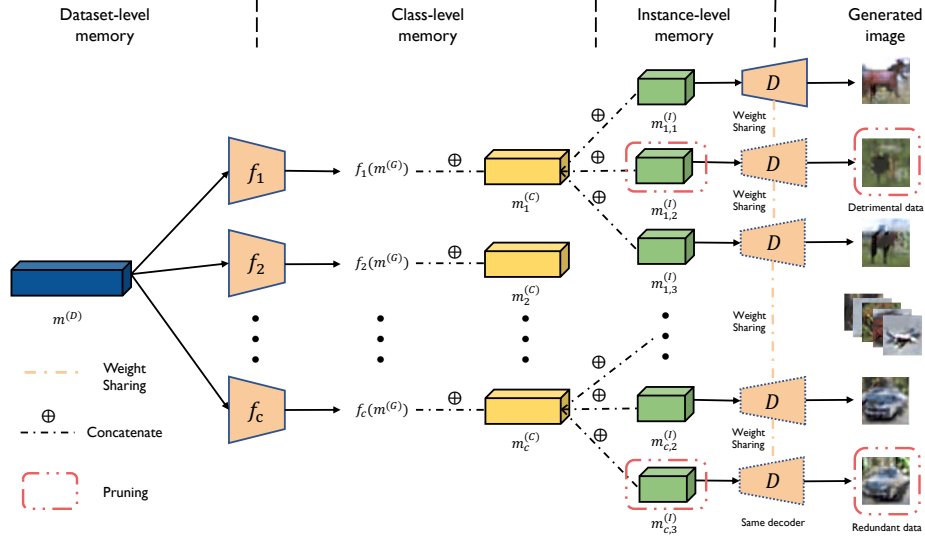


Fig. 2: Illustration of Hierarchical Memory Network and pruning. HMN consists of three tiers of memories (which are learnable parameters). f_i is the feature extractor for each class. D is a single shared decoder to translate a concatenated memory to a synthetic image, though it is applied on a per-image basis, as shown. When we identify redundant or detrimental images, the corresponding instance-level memories are pruned, as indicated by red boxes, saving storage budget.

same loss. We find that HMN outperforms these baselines by a larger margin. For instance, HMN outperforms at least 3.7%/5.9%/2.4% than other data parameterization methods within 1/10/50 IPC (Image Per Class)⁴ storage budgets when trained with the same loss on CIFAR10, respectively. Additionally, we also apply HMN to continual learning tasks. The evaluation results show that HMNs effectively improve the performance on continual learning.

2 Background and Related Work

Problem Definition: As described in Wang et al. [37], given an original training dataset \mathcal{D} , DC methods aim to generate a synthetic dataset \mathcal{S} , which uses a much smaller storage than \mathcal{D} , i.e., $|\mathcal{S}| \ll |\mathcal{D}|$, while achieving a comparable performance (accuracy) of models trained on \mathcal{S} to that on \mathcal{D} .

Training Loss Functions for Optimizing Synthetic Datasets: A common element of existing approaches to data condensation is to choose a training loss

⁴ IPC measures the equivalence of a tensor storage budget in terms of the number of images. For example, 1 IPC for CIFAR10 stands for: Pixels of an image * IPC * class = $3 * 32 * 32 * 1 * 10 = 30720$ tensors. The same metric is also used in SOTA works [13, 19].

function that helps optimize the performance gap between synthetic and real dataset. Two main types of training loss are used to optimize synthetic datasets: 1) *batch-based loss* [40, 42, 44], and 2) *training trajectory-based loss* [4, 37].

Zhao et al. [44] proposed a batch-based loss method called *gradient matching* that aims to minimize the distance between the gradients of a batch of synthetic data and original data. Another batch-based loss method is distribution matching [42] that aims to minimize the distance between the embeddings of a batch of synthetic dataset \mathcal{S} and original dataset \mathcal{D} .

In contrast to batch-based loss methods, trajectory loss requires training the model on the synthetic dataset for multiple iterations while monitoring how the synthetic dataset updates the model parameters across iterations. MTT [4] employs the distance between model parameters of models trained on the synthetic dataset \mathcal{S} and those trained on the original dataset \mathcal{D} as the loss metric. Trajectory-based losses generally tend to provide better empirical performance than batch-based losses, but have considerably larger GPU memory consumption [4, 10]. *Our work demonstrates that by employing the HMN architecture as a data container, a batch-based loss can achieve comparable and even better performance than current data container methods that are based on memory-intensive trajectory-based loss.*

Data Parameterization for Data Condensation. Given a storage budget, *data parameterization* [13, 15, 19] has been recently proposed to further improve data condensation over just optimizing a synthetic dataset \mathcal{S} . Instead of condensing data into image space, the key idea of data parameterization is to condense data into compact free-parameter data containers that can generate training images. Data parameterization allows different generated training images to share the same weights in the data containers, which improves storage efficiency. IDC [15] proposes to downsample images to improve storage efficiency. HaBa [19] and LinBa [13] concurrently introduce factorization-based methods to improve data condensation by sharing common information among images.

Other Methods: Some recent work [5, 41] explores generating condensed datasets with generative priors [3, 6, 7]. For example, instead of synthesizing the condensed dataset from scratch, GLaD [5] assumes the existence of a well-trained generative model. We do not assume the availability of such a generative model and thus this line of work is beyond the scope of this paper, though it is worthy of future investigation.

Coreset Selection is another technique aimed at enhancing data efficiency [9, 17, 31, 34, 38]. Rather than generating a synthetic dataset as in our work, coreset selection identifies a representative subset from the original dataset. Unfortunately, coresets do not tend to give as much data condensation as synthetic datasets and thus we focus on generating synthetic datasets in this work. Nevertheless, coreset selection methods such as the area under the margin (AUM) [26] that measure the data importance by accumulating output margin across training epochs are useful as an initial step in synthetic data condensation as they can be used to select more representative portion of the dataset \mathcal{D} to initialize condensed data synthesis [10, 20].

3 Methodology

In this section, we present technical details on the proposed data condensation approach. In Section 3.1, we present the architecture design of our novel data container for condensation, Hierarchical Memory Network (HMN), to better align with the hierarchical nature of common feature sharing in datasets. In Section 3.2, we study data redundancy of datasets generated by data parameterization methods and show that techniques inspired by coreset methods can be used to prune an HMN data container to make them more storage-efficient.

3.1 Hierarchical Memory Network (HMN)

A Hierarchical Memory Network (HMN) is a parameterized data container such that given an image index i , it outputs a synthetic image \mathcal{S}_i , where $1 \leq i \leq |\mathcal{S}|$. Here $|\mathcal{S}|$ is still a small fraction of $|\mathcal{D}|$. Naturally, it can also be used to generate the entire synthetic dataset \mathcal{S} .

HMN is inspired by earlier work on data parameterization in that it takes advantage of shared features within the dataset \mathcal{D} . However, HMN goes further in that it exploits shared common features from a hierarchical perspective. For instance, two images of cats can share common features specific to the cat class, but an image of a cat and another of a dog may still have shared features of the broader animal class. Our key insight for HMN is that images from the same class can share class-level common features, and images from different classes can share dataset-level common features. As shown in Figure 2, HMN is a three-tier hierarchical data container to store condensed information. Each tier comprises one or more memory tensors, and memory tensors are learnable parameters. The three tiers are as follows:

1. **Dataset-level memory:** The first tier is a dataset-level memory, $m^{(\mathcal{D})}$, which stores the dataset-level information shared among images in the dataset.
2. **Class-level memories:** The second tier, the class-level memory, $m_c^{(C)}$, where c is the class index. The class-level memories store class-level shared features. The number of class-level memories is equivalent to the number of classes in the dataset.
3. **Instance-level memories:** The third tier stores the instance-level memory, $m_{c,i}^{(I)}$, where c, i are the class index and instance index, respectively. The instance-level memories are designed to store unique information for each image. The number of instance-level memories determines the number of images the HMN generates for training.

Besides the memory tensors, we also have feature extractors f_i for each class and a uniform decoder D to convert concatenated memory to images.

Storage Budget for HMN: Following past work on data containers using parameterization [13, 15, 19], *memory tensors, feature extractors, and uniform decoder D network weights count towards the storage budget*. The generated synthetic images, \mathcal{S} , do not count towards the storage budget, since they can be generated as needed, like an efficient image lookup memory.

Other Design Attempts. In the preliminary stages of designing HMNs, we also considered applying feature extractors between $m_c^{(C)}$ and $m_{c,i}^{(I)}$, and attempted to use different decoders for each class to generate images. However, introducing such additional networks did not empirically improve performance. In some cases, it even causes performance drops. One explanation for these performance drops with an increased number of networks is overfitting: more parameters make a condensed dataset better fit the training data and specific model initialization but compromise the model’s generalizability. Consequently, we decided to only apply feature extractors on the dataset-level memory and use a uniform decoder to generate images.

To generate an image for class c , we first adopt features extractor f_c to extract features from the dataset-level memory⁵. This extraction is followed by a concatenation of these features with the class-level memory $m_c^{(C)}$ and instance-level memory $m_{c,i}^{(I)}$. The concatenated memory is then fed to a shared decoder D , which generates the image used for training. Formally, the i th generated image, $x_{c,i}$, in the class c is generated by the following formula:

$$x_{c,i} = D([f_c(m^{(\mathcal{D})}) \oplus m_c^{(C)} \oplus m_{c,i}^{(I)}]) \quad (1)$$

We treat the size of memories and the number of instance-level memories as hyperparameters for architecture design. We present design details in Appendix C, including the shape of memories, the number of generated images per class, architectures of feature extractors and decoder. Given the same storage budget, HMNs generate more training images than other DC methods. However, more generated training images do not hurt training efficiency on the condensed datasets. In Appendix D.2, we show that training with HMNs achieves better accuracy given the same training time compared to SOTA methods.

Training Loss. HMN can be integrated with either trajectory-based loss or batch-based loss. In this paper, to avoid high GPU memory demands of trajectory-based loss, we use a batch-based loss measure of gradient matching [15] to condense information into HMNs. Given the original dataset \mathcal{T} , the initial model parameter distribution P_{θ_0} , a distance function d , and loss function \mathcal{L} , gradient matching aims to synthesize a dataset \mathcal{S} by solving the following optimization:

$$\min_{\mathcal{S}} \mathbf{E}_{\theta_0 \sim P_{\theta_0}} \left[\sum_{t=0}^{T-1} d(\nabla_{\theta} \mathcal{L}(\theta_t, \mathcal{S}), \nabla_{\theta} \mathcal{L}(\theta_t, \mathcal{T})) \right], \quad (2)$$

where θ_t is learned from \mathcal{T} based on θ_{t-1} , and t is the iteration number. In our scenario, the condensed dataset \mathcal{S} is generated by an HMN denoted as H . In Section 4.2, our evaluation results show that our data condensation approach, even when employing a batch-based loss, achieves better performance than SOTA DC baselines, including those that utilize high-memory trajectory-based losses.

⁵ In some storage-limited settings, such as when storage budget is 1IPC, we utilize the identity function as f_c .

3.2 Post-Condensation Pruning of an HMN

In this part, we first show that data redundancy exists in condensed synthetic datasets. Then, we propose a pruning algorithm on HMN to reduce such data redundancy. While such pruning, in theory, can be applied to other data containers, HMN is particularly suited to such pruning because of instance-level memory (see pruned boxes in Figure 2).

Data Redundancy in Condensed Datasets Real-world datasets are shown to contain many redundant data [26, 35, 46, 47]. Here, we show that such data redundancy also exists in condensed datasets. We use HaBa [19] as an example. We first measure the difficulty of training images generated by HaBa with the area under the margin (AUM) [26], a metric measuring data difficulty/importance. The margin for example (\mathbf{x}, y) at training epoch t is defined as:

$$M^{(t)}(\mathbf{x}, y) = z_y^{(t)}(\mathbf{x}) - \max_{i \neq y} z_i^{(t)}(\mathbf{x}), \quad (3)$$

where $z_i^{(t)}(\mathbf{x})$ is the prediction likelihood for class i at training epoch t . AUM is the accumulated margin across all training epochs:

$$\text{AUM}(\mathbf{x}, y) = \frac{1}{T} \sum_{t=1}^T M^{(t)}(\mathbf{x}, y). \quad (4)$$

Table 1: Coreset selection on the training dataset generated by HaBa on CIFAR10 10 IPC. The data with high AUM is pruned first.

Pruning Rate	0	10%	20%	30%	40%
Accuracy (%)	69.5	69.5	68.9	67.6	65.6

A low AUM value indicates that examples are hard to learn. Those examples with lower AUM value are harder to learn, thus are thought to provide more information for training and are more important [26, 35, 46]. Then, as suggested in SOTA coreset selection work [35], we prune out the data with smaller importance (high AUM). The results of coreset selection on the dataset generated by HaBa for CIFAR10 10 IPC are presented in Table 1. We find that pruning up to 10% of the training examples does not hurt accuracy. This suggests that these 10% examples are redundant and can be pruned to save the storage budget.

Pruning on generated datasets is straightforward, but pruning relevant weights in data containers proposed in prior work can be challenging because of dependency among weights in data containers. Unlike prior containers, pruning an

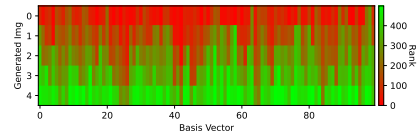


Fig. 3: Rank distribution for different basis vectors in HaBa for CIFAR10 10 IPC. Each column in this figure represents the difficulty rank of images generated using the same basis vector. The color stands for the difficulty rank among all generated images. Green denotes easy-to-learn images, while red indicates hard-to-learn images.

HMN is straightforward since each generated image has its own instance-level memory, which allows us to prune redundant generated images by pruning corresponding instance-level memories (as illustrated by red boxes in Figure 2).

A potential solution for pruning factorization-based data containers is to prune basis vectors in the data containers (each basis vector is used to generate multiple training images). However, we show that directly pruning these basis vectors can lead to removing important data. In Figure 3, we plot the importance rank distribution for training data generated by each basis vector. We observe that the difficulty of images generated by the same basis vector can differ greatly. Thus, simply pruning basis vectors does not guarantee selective pruning of only desired images.

Over-budget Condensation and Post-Condensation Pruning To condense datasets with specific storage budgets and take advantage of the pruning property of HMN to further enhance data condensation, we propose to first condense data into over-budget HMNs, which exceed the storage budget by $p\%$ (p is a hyperparameter). Then, we prune these HMNs to fit the allocated storage budget.

Algorithm 1 Over-budget HMN Double-end Pruning

```

1: Input: Over-budget HMN:  $H$ ; Over-budget images per class:  $k$ ;  $\beta$  search space  $\mathcal{B}$ .
2: Condensed dataset  $\mathcal{S} \leftarrow H()$ ;  $Acc_{best} = 0$ ;  $\mathcal{S}_{best} = \emptyset$ ;
3: Calculate AUM for all examples in  $\mathcal{S}$  based on Equation 4;
4: for  $\beta$  in  $\mathcal{B}$  do
5:    $\tilde{\mathcal{S}} \leftarrow \mathcal{S}.clone()$ ;
6:   Prune  $\lfloor \beta k \rfloor$  of the lowest AUM examples for each class from  $\tilde{\mathcal{S}}$ ;
7:   Prune  $k - \lfloor \beta k \rfloor$  of the highest AUM examples for each class from  $\tilde{\mathcal{S}}$ ;
8:   Retrain model  $f$  on  $\tilde{\mathcal{S}}$ ;
9:    $Acc \leftarrow$  Test accuracy of the model  $f$ ;
10:  if  $Acc > Acc_{best}$  then
11:     $Acc_{best} = Acc$ ;  $\mathcal{S}_{best} = \tilde{\mathcal{S}}$ ;
12:  end if
13: end for
14:  $\Delta\mathcal{S} = \mathcal{S} - \mathcal{S}_{best}$ ;
15:  $\tilde{H} \leftarrow$  Prune corresponding instance-level memories in  $H$  based on  $\Delta\mathcal{S}$ ;
16: Output: Pruned in-budget network:  $\tilde{H}$ .

```

Inspired by CCS [46] showing that pruning both easy and hard data leads to better coreset, we present a double-end pruning algorithm with an adaptive hard pruning rate to prune data adaptively for different storage budgets. As shown in Algorithm 1, given an over-budget HMN containing k more generated images per class than allowed by the storage budget, we employ grid search to determine an appropriate hard pruning rate, denoted as β (Line 4 to Line 12). We then prune $\lfloor \beta k \rfloor$ of the lowest AUM (hardest) examples and $k - \lfloor \beta k \rfloor$ of the highest AUM (easiest) examples by removing the corresponding instance-level memory for each class. The pruning is always class-balanced: the pruned HMNs generate the same number of examples for each class.

Pruning in Algorithm 1 introduces additional computational costs compared to the standard data condensation pipeline. However, in practice, the pruning step is relatively cheap (e.g., an additional 2-3% overhead for data condensation).

4 Experiments

In this section, we compare the performance of HMN to SOTA baselines. HMN is compared to DC baselines for different values of storage budgets. As with other data parameterization techniques, our condensed data does not store images but rather model parameters. **All the model parameters of an HMN in Figure 2, including the three-tier memories and networks, are considered as part of the storage budget.** For the convenience of comparison, following prior work in the data condensation area [13, 15, 19], the unit used for measuring storage budget is IPC (Images Per Class). 1 IPC for CIFAR10 is calculated as $32 * 32 * 3 * 1 * 10 = 30,720$ (assuming that they are stored as 32-bit floating point values). An HMN for CIFAR10 with 1 IPC as the storage budget always has an equal or lower number of parameters than this.

Due to the page limitation, we include additional evaluation results in Appendix D. Appendix D.1 examines the relationship between pruning rate and accuracy. Appendix D.2 studies the convergence speed of training for different condensed datasets. Appendix D.4 presents results from data profiling to study the data redundancy on the condensed datasets synthesized by different DC methods. Finally, Appendix D.5 presents visualizations of the condensed training data generated by HMNs.

4.1 Experimental Settings

Datasets and Training Settings. We evaluate our proposed method on four public datasets: CIFAR10, CIFAR100 [16], SVHN [22], Tiny-ImageNet [12], and ImageNet-10 [12] under three different storage budgets: 1/10/50 IPC (For Tiny-ImageNet and ImageNet-10, due to the computation limitation, we conduct the evaluation on 1/10 IPC and 1 IPC, respectively). Following previous works [13, 19, 40], we select ConvNet, which contains three convolutional layers followed by a pooling layer, as the network architecture for data condensation and classifier training. For the over-budget training and post-condensation, we first conducted a pruning study on HMNs in Appendix D.1, we observed that there is a pronounced decline in accuracy when the pruning rate exceeds 10%. Consequently, we select 10% as the over-budget rate for all settings. Nevertheless, we believe that this rate choice could be further explored, and other rate values could potentially further enhance the performance of HMNs. Due to space limits, we include more HMN architecture details, experimental settings, and additional implementation details in the supplementary material. All data condensation evaluation is repeated 3 times, and training on each HMN is repeated 10 times with different random seeds to calculate mean with standard deviation.

Baselines. We compare our proposed method with eight baselines, which can be divided into two categories by data containers: **1) Image data container.**

Table 2: The performance (test accuracy %) comparison to state-of-the-art methods. The baseline method’s accuracy is obtained from data presented in original papers or author-implemented repos. We label the methods using the trajectory-based training loss with a star (*). I-10 stands for ImageNet-10. We highlight the highest accuracy among all methods and methods with batch-based loss.

Container	Dataset IPC	CIFAR10			CIFAR100			SVHN			Tiny		I-10
		1	10	50	1	10	50	1	10	50	1	10	
Image	DC	28.3 ± 0.5	44.9 ± 0.5	53.9 ± 0.5	12.8 ± 0.3	25.2 ± 0.3	-	31.2 ± 1.4	76.1 ± 0.6	82.3 ± 0.3	4.6 ± 0.6	11.2 ± 1.6	-
	DSA	28.8 ± 0.7	52.1 ± 0.5	60.6 ± 0.5	13.9 ± 0.3	32.3 ± 0.3	42.8 ± 0.4	27.5 ± 1.4	79.2 ± 0.5	84.4 ± 0.4	6.6 ± 0.2	14.4 ± 2.0	-
	DM	26.0 ± 0.8	48.9 ± 0.6	63.0 ± 0.4	11.4 ± 0.3	29.7 ± 0.3	43.6 ± 0.4	-	-	-	3.9 ± 0.2	12.9 ± 0.4	-
	CAFE+DSA	31.6 ± 0.8	50.9 ± 0.5	62.3 ± 0.4	14.0 ± 0.3	31.5 ± 0.2	42.9 ± 0.2	42.9 ± 3.0	77.9 ± 0.6	82.3 ± 0.4	-	-	-
	MTT*	46.3 ± 0.8	65.3 ± 0.7	71.6 ± 0.2	24.3 ± 0.3	40.1 ± 0.4	47.7 ± 0.2	58.5 ± 1.4	70.8 ± 1.8	85.7 ± 0.1	8.8 ± 0.3	23.2 ± 0.2	-
	IDM	45.6 ± 0.7	58.6 ± 0.1	67.5 ± 0.1	20.1 ± 0.3	45.1 ± 0.1	50.0 ± 0.2	-	-	-	10.1 ± 0.2	21.9 ± 0.2	-
Data Paramete- -terization	IDC	50.0 ± 0.4	67.5 ± 0.5	74.5 ± 0.2	-	45.1	-	68.5	87.5	90.1	-	-	60.4
	HaBa*	48.3 ± 0.8	69.9 ± 0.4	74.0 ± 0.2	33.4 ± 0.4	40.2 ± 0.2	47.0 ± 0.2	69.8 ± 1.3	83.2 ± 0.4	88.3 ± 0.1	-	-	-
	LinBa*	66.4 ± 0.4	71.2 ± 0.4	73.6 ± 0.5	34.0 ± 0.4	42.9 ± 0.7	-	87.3 ± 0.1	89.1 ± 0.2	89.5 ± 0.2	16.0 ± 0.7	-	-
	HMN (Ours)	65.7 ± 0.3	73.7 ± 0.2	76.9 ± 0.2	36.3 ± 0.2	45.4 ± 0.2	48.5 ± 0.2	87.4 ± 0.2	90.0 ± 0.1	91.2 ± 0.1	19.4 ± 0.1	24.4 ± 0.1	64.6
Entire Dataset		84.8 ± 0.1			56.2 ± 0.3			95.4 ± 0.1			37.6 ± 0.4		90.8

We use five recent works as the baseline: MTT [4] (as mentioned in Section 2). DC [44] and DSA [40] optimize condensed datasets by minimizing the distance between gradients calculated from a batch of condensed data and a batch of real data. DM [42] aims to encourage condensed data to have a similar distribution to the original dataset in latent space. IDM [45] enhances distribution matching by improving the naive average embedding distribution matching. Finally, CAFE [36] improves the distribution matching idea by layer-wise feature alignment. **2) Data parameterization.** We also compare our method with three SOTA data parameterization baselines. IDC [15] enhances gradient matching loss calculation strategy and employs multi-formation functions to parameterize condensed data. HaBa [19] and LinBa [13] proposes factorization-based data parameterization to achieve information sharing among generated images.

Besides grouping the methods by data containers, we also categorize those methods by the training losses used. As discussed in Section 2, there are two types of training loss: trajectory-based training loss and batch-based training loss. In Table 2, we highlight the methods using a trajectory-based loss with a star (*). In our HMN implementation, we condense HMNs with gradient matching loss used in [15], which is a low GPU memory consumption batch-based loss.

4.2 Data Condensation Performance Comparison

We compare HMN with eight baselines on four different datasets (CIFA10, CIFAR100, SVHN, Tiny ImageNet, and ImageNet-10) in Table 2. We divide all

methods into two categories by the type of data container formats: Image data container and data parameterization container. We also categorize all methods by the training loss. We use a star (*) to highlight the methods using a trajectory-based loss. The results presented in Table 2 show that HMN achieves comparable or better performance than all baselines. It is worth noting that HMN is trained with gradient matching, which is a low GPU memory loss. Two other well-performed data parameterization methods, HaBa and LinBa, are all trained with trajectory-based losses, consuming much larger GPU memory. These results show that batch-based loss can still achieve good performance with an effective data parameterization method and help address the memory issue of data condensation [4, 5, 10]. We believe that HMN provides a strong baseline for data condensation methods. We further study the memory consumed by different methods in Section 4.4.

Data Parameterization Comparison with the Same Loss.

In addition to the end-to-end method comparison presented in Table 2, we also compare HMN with other data parameterization methods with the same training loss (gradient matching loss used by IDC) for a fairer comparison. The results are presented in Table 3. After replacing the trajectory-based loss used by HaBa and LinBa with a batch-based loss, there is a noticeable decline in accuracy (but HaBa and LinBa still outperform the image data container).⁶ HMN outperforms other data parameterization by a larger margin when training with the same training loss, which indicates that HMN is a more effective data parameterization method and can condense more information within the same storage budget. We also discussed the memory consumption of trajectory-based methods in Section 4.4.

Table 3: Accuracy (%) performance comparison to data containers with the same gradient matching training loss on CIFAR10. The evaluation results show that HMN outperforms all other data parameterization methods substantially.

Data Container	1 IPC	10 IPC	50 IPC
Image	36.7	58.3	69.5
IDC	50.0	67.5	74.5
HaBa	48.5	61.8	72.4
LinBa	62.0	67.8	70.7
HMN (Ours)	65.7	73.7	76.9

4.3 Cross-architecture Transferability

To investigate the generalizability of HMNs across different architectures, we utilized condensed HMNs to train other network architectures. Specifically, we condense HMNs with ConvNet, and the condensed HMNs are tested on VGG16, ResNet18, and DenseNet121. We compare our methods with two other data parameterization methods: IDC and HaBa. (Due to the extremely long training time, we are unable to reproduce the results in LinBa). The evaluation results on CIFAR10 are presented in Table 4. We find that HMNs consistently outperform other baselines. Of particular interest, we observe that VGG16 has a better performance than ResNet18 and DenseNet121. A potential explanation may lie in

⁶ We do hyperparameter search for all data containers to choose the optimal setting.

Table 4: Transferability (accuracy %) comparison to different model architectures. Due to the extremely long training time, we cannot reproduce the results on LinBa. Compared with IDC and HaBa, we find that HMN achieves better performance for all model architectures.

IPC	1			10			50		
Method	HMN	IDC	HaBa	HMN	IDC	HaBa	HMN	IDC	HaBa
ConvNet	65.7	50.0	48.3	73.7	67.5	69.9	76.9	74.5	74
VGG16	58.5	28.7	34.1	64.3	43.1	53.8	70.2	57.9	61.1
ResNet18	56.8	32.3	36.0	62.9	45.1	49.0	69.1	58.4	60.4
DenseNet121	50.7	24.3	34.6	56.9	38.5	49.3	65.1	50.5	57.8

the architectural similarities between ConvNet and VGG16. Both architectures are primarily comprised of convolutional layers and lack skip connections.

4.4 GPU Memory Comparison

As discussed in Section 4.1, GPU memory consumption can be very different depending on the training losses used. We compare the GPU memory used by HMN with two other well-performed data parameterization methods, HaBa and LinBa. As depicted by Table 5, HMN achieves better or comparable performance compared to HaBa

and LinBa with much less GPU memory consumption. Specifically, LinBa is trained with BPTT with a very long trajectory, which leads to extremely large GPU memory consumption. LinBa official implementation offloads the GPU memory to CPU memory to address this issue. However, the context switch in memory offloading causes the training time to be intolerable. For example, LinBa needs about 14 days to condense a CIFAR10 1IPC dataset with a 2080TI, but using

Table 5: Performance and memory comparison between LinBa, HaBa, and HMN trained with suggested loss in corresponding papers on CIFAR10. OOM means "Out of GPU Memory." Reported accuracy numbers use CPU offloading in OOM case.

IPC	Method	Loss	Acc.	Memory
1	HaBa	MTT	48.3	3368M
	LinBa	BPTT	66.4	OOM
	HMN (Ours)	GM-IDC	65.7	2680M
10	HaBa	MTT	69.9	11148M
	LinBa	BPTT	71.2	OOM
	HMN (Ours)	GM-IDC	73.7	4540M
50	HaBa	MTT	74	48276M
	LinBa	BPTT	73.6	OOM
	HMN (Ours)	GM-IDC	76.9	10426M

HMN with gradient matching only needs 15 hours to complete training on a 2080TI GPU. Although this memory saving does not come from the design of HMN, our paper shows that batch-based loss can still achieve very good performance with a proper data parameterization method, which helps address the memory issue of data condensation [4, 5, 10].

Combining an HMN with the trajectory-based loss may further improve the performance of an HMN-based approach, but the cost is too high for the GPUs we have. We leave that investigation to future work since using trajectory-based loss in practice remains challenging from a scalability perspective due to high

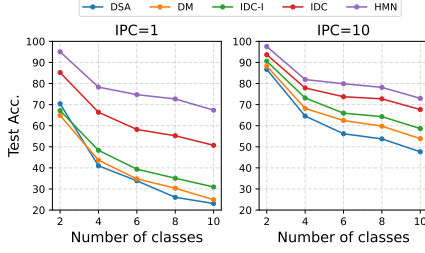


Fig. 4: Continual learning evaluation on CIFAR10. In the class incremental setting with 2 incoming classes per stage, HMN outperforms existing methods (including DSA, DM and IDC) under different storage budgets.

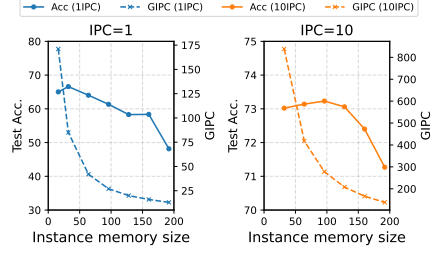


Fig. 5: Instance-memory length vs. Accuracy for CIFAR10 HMNs with 1 IPC/10 IPC storage budgets. GIPC refers to the number of generated images per class. The solid and dashed curves represent the accuracy and GIPC, respectively.

memory needs. For a fair comparison, we compare the different data containers with the same batch-based loss in Table 3.

4.5 Ablation Studies

Instance-Memory Size v.s. Retrained Model Accuracy. In an HMN, every generated image is associated with an independent instance-level memory, which constitutes the majority of the storage budget. Consequently, given a fixed storage budget, an increase in the instance-level memory results in a decrease in the number of generated images per class (GIPC). In Figure 5, we explore the interplay between the instance-memory size, the accuracy of the retrained model, and GIPC. Specifically, we modify the instance-level memory size of CIFAR10 HMNs for given storage budgets of 1 IPC and 10 IPC. (It should be noted that for this ablation study, we are condensing in-budget HMNs directly without employing any coreset selection on the condensed HMNs.)

From Figure 5, we observe that an increase in the instance-level memory size leads to a swift drop in GIPC, as each generated image consumes a larger portion of the storage budget. Moreover, we notice that both excessively small and large instance-level memory sizes negatively affect the accuracy of retrained models. Reduced instance-level memory size can result in each generated image encoding only a limited amount of information. This constraint can potentially deteriorate the quality of the generated images and negatively impact training performance. Conversely, while an enlarged instance-level memory size enhances the volume of information encoded in each image, it precipitously reduces GIPC. This reduction can compromise the diversity of generated images for training. For instance, with a 1IPC storage budget, an increase in the instance-level memory size, leading to a decrease in GIPC from 85 to 13, results in an accuracy drop from 65.1% to 48.2%.

Ablation Study on Pruning. In Table 6, we explore the performance of different pruning strategies applied to over-budget HMNs on the CIFAR10 dataset. The strategy termed "Prune easy" is widely employed in conventional coreset selection methods [9, 25, 35, 38], which typically prioritize pruning of easy examples containing more redundant information. "In-budget" refers to the process of directly condensing HMNs to fit the storage budgets, which does not need any further pruning. As shown in Table 6, our proposed pruning strategy (double-end) outperforms all other pruning strategies. We also observe that, as the storage budget increases, the accuracy improvement becomes larger compared to "in-budget" HMNs. We think this improvement is because a larger storage budget causes more redundancy in the condensed data [10], which makes pruning reduce more redundancy in condensed datasets. Also, the performance gap between the "Prune easy" strategy and our pruning method is observed to narrow as the storage budget increases. This may be attributed to larger storage budgets for HMNs leading to more redundant easy examples. The "Prune easy" is a good alternative for pruning for large budgets.

Table 6: Performance comparison on different pruning strategies on HMN. Double-end is the pruning strategy introduced in Algorithm 1. In-budget stands for HMNs are condensed within the allocated storage budget.

	IPC	1	10	50
Double-end	65.7	73.7	76.9	
Prune easy	65.3	73.1	76.6	
Random	65.2	72.9	75.3	
In-budget	65.1	73.2	75.4	

4.6 Continual Learning Performance Comparison

Following the same setting in DM [43] and IDC [15], we evaluate the effectiveness of HMN in an application scenario of continual learning [1, 8, 27]. Specifically, we split the whole training phase into 5 stages, *i.e.* 2 classes per stage. At each stage, we condense the data currently available at this stage with ConvNet. As illustrated in Figure 4, evaluated on ConvNet models under the storage budget of both 1 IPC and 10 IPC, HMN obtains better performance compared with DSA [40], DM [42], and IDC [15]. Particularly, in the low storage budget scenario, *i.e.* 1 IPC, the performance improvement brought by HMN is more significant, up to 16%. The results indicate that HMNs provide higher-quality condensed data and boost continual learning performance.

5 Conclusion

This paper introduces a novel data parameterization architecture, Hierarchical Memory Network (HMN), which is inspired by the hierarchical nature of common feature sharing in datasets. In contrast to previous data parameterization methods, HMN aligns more closely with this hierarchical nature of datasets. Our evaluation results show that the proposed HMN data container architecture, even when employing a batch-based loss for its optimization, achieves better or comparable performance than SOTA DC baselines, including those that utilize high-memory trajectory-based loss functions.

Acknowledgement

This work was partially supported by DARPA under agreement number 885000 and National Science Foundation Grant No. 2039445. This work was performed under the auspices of the U.S. Department of Energy by the Lawrence Livermore National Laboratory under Contract No. DE-AC52-07NA27344 and was supported by the LLNL-LDRD Program under Project No. 22-SI-004 and 24-ERD-010. Any opinion, findings, and conclusions or recommendations expressed in this material are those of the authors(s) and do not necessarily reflect the views of our research sponsors.

References

1. Bang, J., Kim, H., Yoo, Y., Ha, J.W., Choi, J.: Rainbow memory: Continual learning with a memory of diverse samples. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. pp. 8218–8227 (2021)
2. Bartoldson, B.R., Kailkhura, B., Blalock, D.: Compute-efficient deep learning: Algorithmic trends and opportunities. *Journal of Machine Learning Research* **24**, 1–77 (2023)
3. Brock, A., Lim, T., Ritchie, J.M., Weston, N.J.: Neural photo editing with introspective adversarial networks. In: *5th International Conference on Learning Representations 2017* (2017)
4. Cazenavette, G., Wang, T., Torralba, A., Efros, A.A., Zhu, J.Y.: Dataset distillation by matching training trajectories. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. pp. 4750–4759 (2022)
5. Cazenavette, G., Wang, T., Torralba, A., Efros, A.A., Zhu, J.Y.: Generalizing dataset distillation via deep generative prior. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2023)
6. Chai, L., Wulff, J., Isola, P.: Using latent space regression to analyze and leverage compositionality in gans. In: *International Conference on Learning Representations* (2021)
7. Chai, L., Zhu, J.Y., Shechtman, E., Isola, P., Zhang, R.: Ensembling with deep generative views. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. pp. 14997–15007 (2021)
8. Chaudhry, A., Rohrbach, M., Elhoseiny, M., Ajanthan, T., Dokania, P.K., Torr, P.H., Ranzato, M.: On tiny episodic memories in continual learning. *arXiv preprint arXiv:1902.10486* (2019)
9. Coleman, C., Yeh, C., Mussmann, S., Mirzasoleiman, B., Bailis, P., Liang, P., Leskovec, J., Zaharia, M.: Selection via proxy: Efficient data selection for deep learning. In: *International Conference on Learning Representations* (2019)
10. 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)
11. Cui, J., Wang, R., Si, S., Hsieh, C.J.: Scaling up dataset distillation to imagenet-1k with constant memory. In: *International Conference on Machine Learning*. pp. 6565–6590. PMLR (2023)
12. Deng, J., Dong, W., Socher, R., Li, L.J., Li, K., Fei-Fei, L.: Imagenet: A large-scale hierarchical image database. In: *2009 IEEE conference on computer vision and pattern recognition*. pp. 248–255. Ieee (2009)

13. Deng, Z., Russakovsky, O.: Remember the past: Distilling datasets into addressable memories for neural networks. In: *Advances in Neural Information Processing Systems* (2022)
14. Du, J., Jiang, Y., Tan, V.T., Zhou, J.T., Li, H.: Minimizing the accumulated trajectory error to improve dataset distillation. *arXiv preprint arXiv:2211.11004* (2022)
15. 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)
16. Krizhevsky, A., Hinton, G., et al.: Learning multiple layers of features from tiny images. *Tech. rep., Citeseer* (2009)
17. Li, Y., Zhao, P., Lin, X., Kailkhura, B., Goldhahn, R.: Less is more: Data pruning for faster adversarial training. *arXiv preprint arXiv:2302.12366* (2023)
18. 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>
19. Liu, S., Wang, K., Yang, X., Ye, J., Wang, X.: Dataset distillation via factorization. In: *Advances in Neural Information Processing Systems* (2022)
20. Liu, Y., Gu, J., Wang, K., Zhu, Z., Jiang, W., You, Y.: Dream: Efficient dataset distillation by representative matching. *arXiv preprint arXiv:2302.14416* (2023)
21. Loshchilov, I., Hutter, F.: Sgdr: Stochastic gradient descent with warm restarts. In: *ICLR* (2017)
22. Netzer, Y., Wang, T., Coates, A., Bissacco, A., Wu, B., Ng, A.Y.: Reading digits in natural images with unsupervised feature learning. In: *NIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011* (2011), http://ufldl.stanford.edu/housenumbers/nips2011_housenumbers.pdf
23. Nguyen, T., Chen, Z., Lee, J.: Dataset meta-learning from kernel ridge-regression. In: *International Conference on Learning Representations* (2020)
24. Nguyen, T., Novak, R., Xiao, L., Lee, J.: Dataset distillation with infinitely wide convolutional networks. *Advances in Neural Information Processing Systems* **34**, 5186–5198 (2021)
25. Paul, M., Ganguli, S., Dziugaite, G.K.: Deep learning on a data diet: Finding important examples early in training. *Advances in Neural Information Processing Systems* **34** (2021)
26. Pleiss, G., Zhang, T., Elenberg, E., Weinberger, K.Q.: Identifying mislabeled data using the area under the margin ranking. *Advances in Neural Information Processing Systems* **33**, 17044–17056 (2020)
27. 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*. pp. 2001–2010 (2017)
28. 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)
29. Rosasco, A., Carta, A., Cossu, A., Lomonaco, V., Bacciu, D.: Distilled replay: Overcoming forgetting through synthetic samples. In: *Continual Semi-Supervised Learning: First International Workshop, CSSL 2021, Virtual Event, August 19–20, 2021, Revised Selected Papers*. pp. 104–117. Springer (2022)
30. Sangermano, M., Carta, A., Cossu, A., Bacciu, D.: Sample condensation in online continual learning. In: *2022 International Joint Conference on Neural Networks (IJCNN)*. pp. 01–08. IEEE (2022)
31. Sener, O., Savarese, S.: Active learning for convolutional neural networks: A core-set approach. *arXiv preprint arXiv:1708.00489* (2017)

32. Shin, S., Bae, H., Shin, D., Joo, W., Moon, I.C.: Loss-curvature matching for dataset selection and condensation. In: International Conference on Artificial Intelligence and Statistics. pp. 8606–8628. PMLR (2023)
33. Song, R., Liu, D., Chen, D.Z., Festag, A., Trinitis, C., Schulz, M., Knoll, A.: Federated learning via decentralized dataset distillation in resource-constrained edge environments. arXiv preprint arXiv:2208.11311 (2022)
34. Sorscher, B., Geirhos, R., Shekhar, S., Ganguli, S., Morcos, A.S.: Beyond neural scaling laws: beating power law scaling via data pruning. arXiv preprint arXiv:2206.14486 (2022)
35. Toneva, M., Sordoni, A., des Combes, R.T., Trischler, A., Bengio, Y., Gordon, G.J.: An empirical study of example forgetting during deep neural network learning. In: International Conference on Learning Representations (2018)
36. Wang, K., Zhao, B., Peng, X., Zhu, Z., Yang, S., Wang, S., Huang, G., Bilen, H., Wang, X., You, Y.: Cafe: Learning to condense dataset by aligning features. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 12196–12205 (2022)
37. Wang, T., Zhu, J.Y., Torralba, A., Efros, A.A.: Dataset distillation. arXiv preprint arXiv:1811.10959 (2018)
38. Xia, X., Liu, J., Yu, J., Shen, X., Han, B., Liu, T.: Moderate coreset: A universal method of data selection for real-world data-efficient deep learning. In: International Conference on Learning Representations (2023)
39. Xiong, Y., Wang, R., Cheng, M., Yu, F., Hsieh, C.J.: Feddm: Iterative distribution matching for communication-efficient federated learning. arXiv preprint arXiv:2207.09653 (2022)
40. Zhao, B., Bilen, H.: Dataset condensation with differentiable siamese augmentation. In: International Conference on Machine Learning. pp. 12674–12685. PMLR (2021)
41. Zhao, B., Bilen, H.: Synthesizing informative training samples with gan. In: NeurIPS 2022 Workshop on Synthetic Data for Empowering ML Research (2022)
42. 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)
43. 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)
44. Zhao, B., Mopuri, K.R., Bilen, H.: Dataset condensation with gradient matching. In: Ninth International Conference on Learning Representations 2021 (2021)
45. Zhao, G., Li, G., Qin, Y., Yu, Y.: Improved distribution matching for dataset condensation. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 7856–7865 (2023)
46. Zheng, H., Liu, R., Lai, F., Prakash, A.: Coverage-centric coreset selection for high pruning rates. In: International Conference on Learning Representations 2023 (2023)
47. Zheng, H., Tsai, E., Lu, Y., Sun, J., Bartoldson, B.R., Kailkhura, B., Prakash, A.: Elfs: Enhancing label-free coreset selection via clustering-based pseudo-labeling. arXiv preprint arXiv:2406.04273 (2024)
48. Zheng, H., Zhang, Z., Gu, J., Lee, H., Prakash, A.: Efficient adversarial training with transferable adversarial examples. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 1181–1190 (2020)

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 [13, 15, 19] 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 7: 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 IPC	SVHN & CIFAR10			CIFAR100			Tiny		I-10
	1	10	50	1	10	50	1	10	
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 7. 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) \rightarrow Batch Normalization \rightarrow ConvTranspose(10, 6, 4, 1, 2) \rightarrow Batch Normalization \rightarrow ConvTranspose(6, 3, 4, 1, 2). The arguments for ConvTranspose are 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 [13] uses 0.1 as the learning rate, but HaBa [19] 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 [13, 19, 40], 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 [15, 19], a batch-based loss with low GPU memory consumption, to condense information into HMNs. More specifically, our code is implemented based on IDC [19]. 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 [19], we use the DSA augmentation [40] 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 [21] 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 [15], we adopt distillation loss [18] and train the model constantly by loading weights of the previous stage and expanding the output dimension of the last fully-connected layer [28]. 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 7. 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

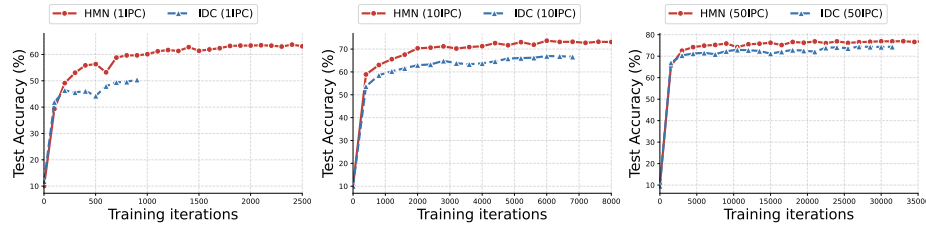


Fig. 6: Test accuracy over training iterations of training with condensed datasets generated by HMN and IDC. Given the same storage budget, although training with HMN needs more training iterations to converge, we find that HMNs achieve better accuracy within the same training time compared to IDC.

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 6. 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 /

Table 8: 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

$0.6 * \text{total epochs} / 0.9 * \text{total epochs}$.

As shown in Table 8, 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.

D.4 Data Profiling on SOTA Methods

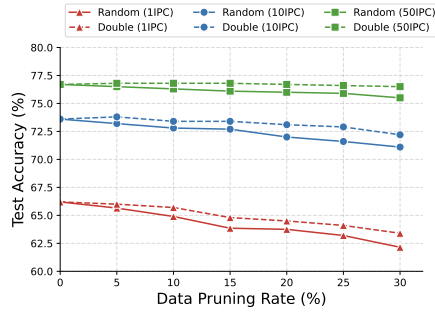


Fig. 7: Relationship between pruning rates and accuracy on HMNs for CIFAR10. All HMNs are over-budget HMNs (10% extra). Different colors stand for different storage budgets. Solid lines stand for random pruning and dashed lines stand for double-end pruning.

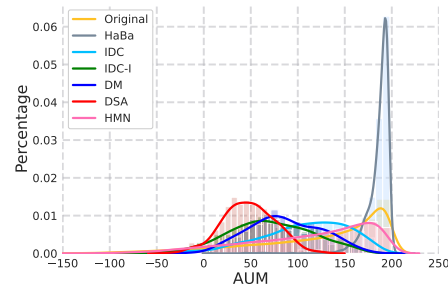


Fig. 8: The distribution of AUM of CIFAR10 training images synthesized by different approaches. Different colors denote different data condensation approaches. Data parameterization based methods have more redundant images.

Figure 8 illustrates the distribution of AUM of images synthesized by different data condensation approaches, as well as the original data, denoted as “Original”. We calculate the AUM by training a ConvNet for 200 epochs. We observe that approaches (IDC-I [15], DM [43], and DSA [40]) 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 [19], IDC [15], and HMN⁷, 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.

⁷ We did not evaluate LinBa due to its substantial time requirements.

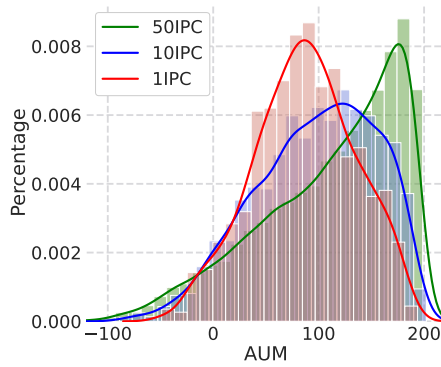


Fig. 9: AUM distribution of images generated by HMNs for CIFAR10 with different storage budgets, denoted by different colors.

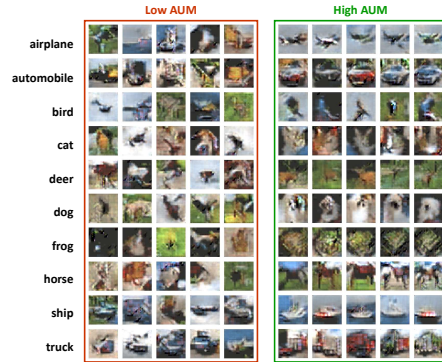


Fig. 10: Visualization of the lowest and highest AUM examples generated by a 10 IPC HMN of CIFAR10. Each row represents a class.

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 9, 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 [10]: more storage budgets add more easy examples which only provide redundant information and do not contribute much to training. From Figure 9, 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.

Additionally, in Figure 10, 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 17 18 19), 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 7. 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.

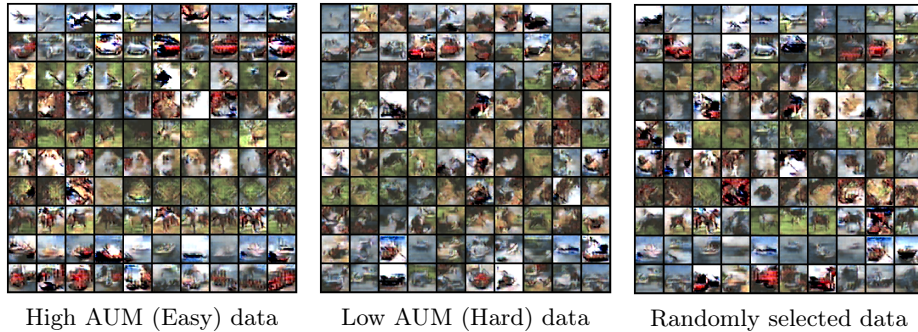


Fig. 11: Images generated by a CIFAR10 HMN with 1.1IPC storage budget. Images in each row are from the same class.

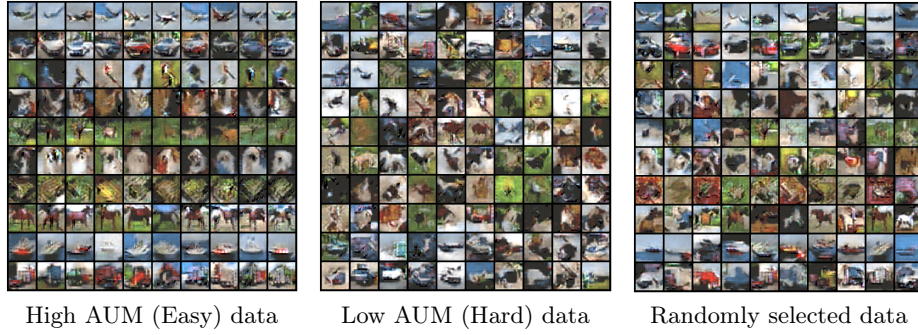


Fig. 12: Images generated by a CIFAR10 HMN with 11IPC storage budget. Images in each row are from the same class.

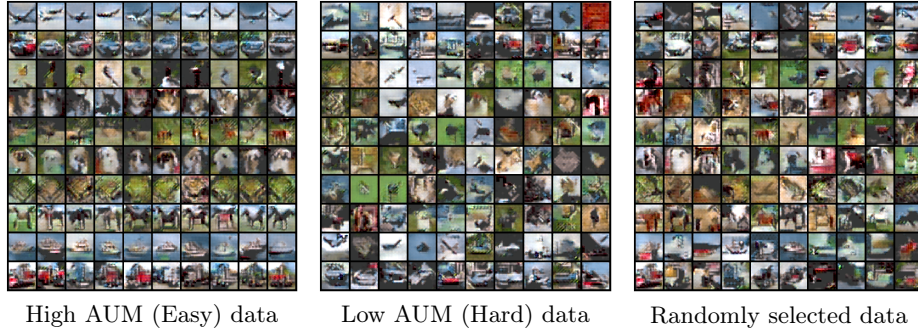


Fig. 13: Images generated by a CIFAR10 HMN with 55IPC storage budget. Images in each row are from the same class.

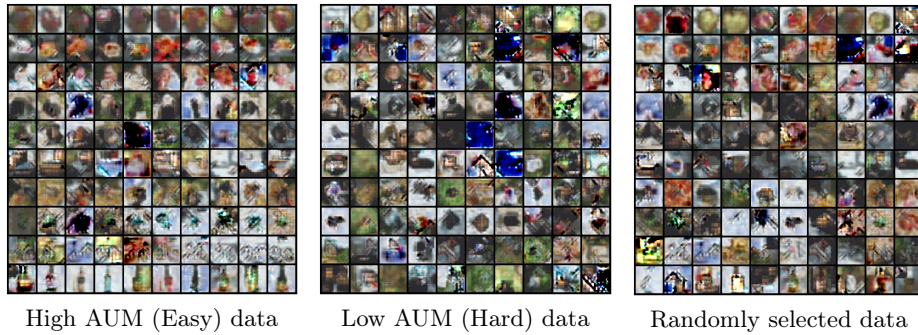


Fig. 14: Images generated by a CIFAR100 HMN with 1.1IPC storage budget. Images in each row are from the same class. We only visualize 10 classes with the smallest class number in the dataset.

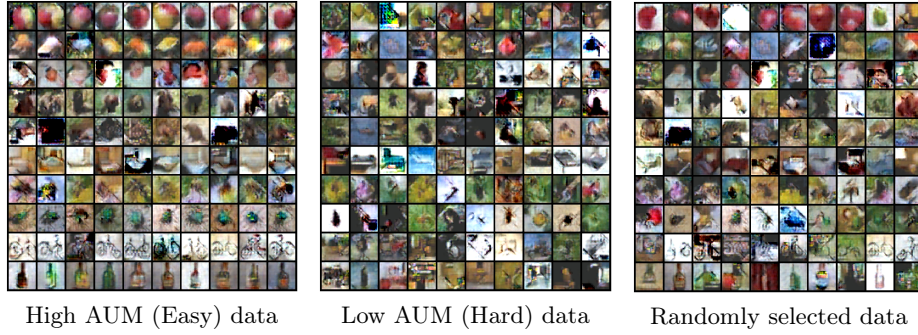


Fig. 15: Images generated by a CIFAR100 HMN with 11IPC storage budget. Images in each row are from the same class. We only visualize 10 classes with the smallest class number in the dataset.

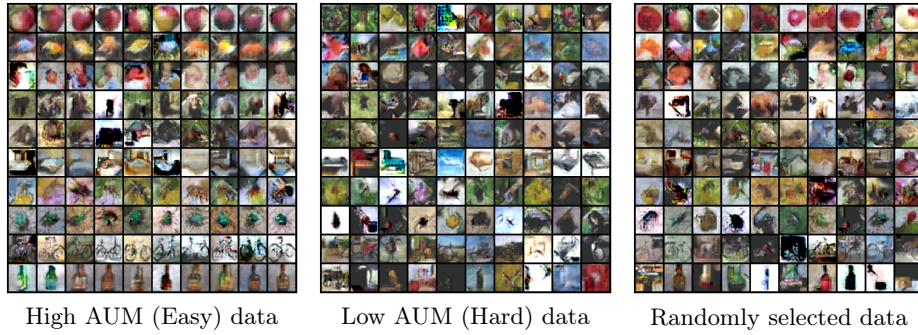


Fig. 16: Images generated by a CIFAR100 HMN with 55IPC storage budget. Images in each row are from the same class. We only visualize 10 classes with the smallest class number in the dataset.

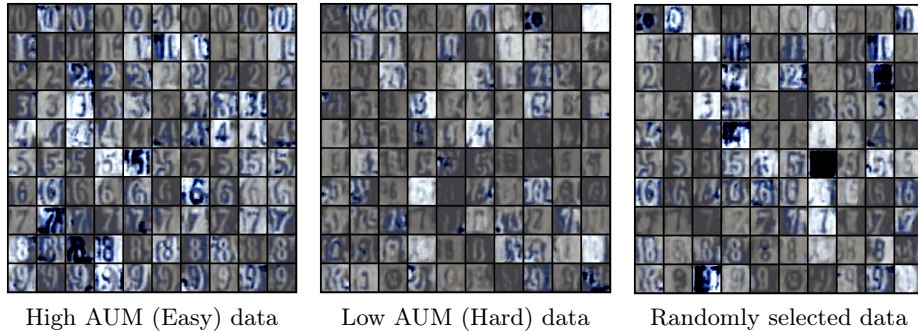


Fig. 17: Images generated by an SVHN HMN with 1.1IPC storage budget. Images in each row are from the same class. Images with a low aum value are not well-aligned with its label and can be harmful for the training.

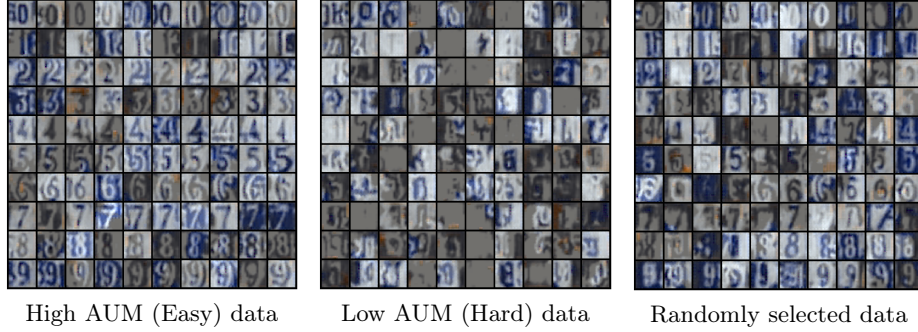


Fig. 18: Images generated by an SVHN HMN with 11IPC storage budget. Images in each row are from the same class.

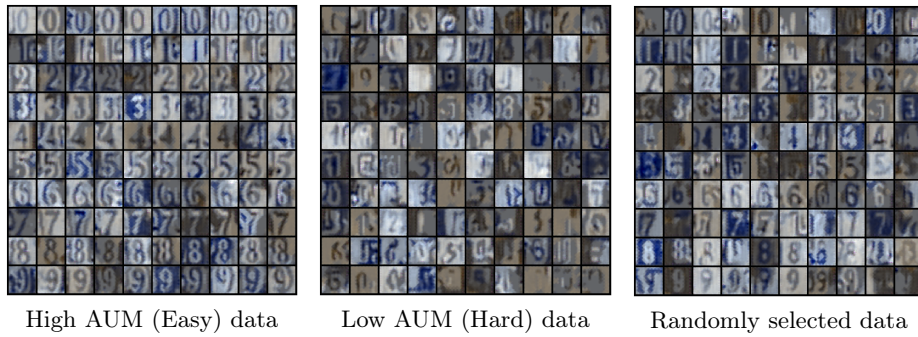


Fig. 19: Images generated by an SVHN HMN with 55IPC storage budget. Images in each row are from the same class.