DATASET CONDENSATION WITH LATENT SPACE KNOWLEDGE FACTORIZATION AND SHARING

Anonymous authors
Paper under double-blind review

ABSTRACT

In this paper, we introduce a novel approach for systematically solving dataset condensation problem in an efficient manner by exploiting the regularity in a given dataset. Instead of condensing the dataset directly in the original input space, we assume a generative process of the dataset with a set of learnable *codes* defined in a compact latent space followed by a set of tiny *decoders* which maps them differently to the original input space. By combining different codes and decoders interchangeably, we can dramatically increase the number of synthetic examples with essentially the same parameter count, because the latent space is much lower dimensional and since we can assume as many decoders as necessary to capture different styles represented in the dataset with negligible cost. Such knowledge factorization allows efficient sharing of information between synthetic examples in a systematic way, providing far better trade-off between compression ratio and quality of the generated examples. We experimentally show that our method achieves new state-of-the-art records by significant margins on various benchmark datasets, from low resolution datasets such as SVHN, CIFAR10, CIFAR100, and TinyImageNet to ImageNet-Subset of resolution 224×224.

1 INTRODUCTION

Deep learning has been successful in numerous machine learning problems thanks to the recent progress in parallel processing and the huge amount of real-world data collected from various sources. However, for some machine learning applications it is required to repeatedly rehearse the training process, such as for hyperparameter optimization (Bengio, 2000; Franceschi et al., 2017), neural architecture search (Liu et al., 2018) and continual learning (Lopez-Paz & Ranzato, 2017). In such cases, it requires prohibitive memory and computational cost to keep and rehearse all the examples in huge datasets, giving rise to the need for compressing each dataset into a small set of representative examples. The conventional approaches resort to selecting a coreset (Phillips, 2016; Toneva et al., 2018; Borsos et al., 2020), but their assumption is limited as there may not exist strongly representative examples in the original dataset (Zhao & Bilen, 2021) and they require solving combinatorial optimization problems (Borsos et al., 2020). Recent works thus focus on *dataset condensation* (Wang et al., 2018; Zhao et al., 2021), which aim to directly parameterize and optimize the synthetic dataset with gradient descent. They have shown to perform well compared to coreset selection approaches and generate plausibly looking examples as well.

However, despite their great potential, most of the existing dataset condensation methodologies do not focus on exploiting the regularity in the dataset, such as what the underlying data generating process should be. They usually parameterize a set of synthetic examples directly in the input space and minimize a distance between a real and a synthetic dataset in either the parameter (Zhao et al., 2021) or feature space (Zhao & Bilen, 2022a). Such lack of assumptions on the underlying data generating process prevents them from efficiently sharing knowledge among the synthetic examples. Therefore, those approaches may require much larger space to capture the same amount of information compared to the space required with a properly organized data generating process. This intuition gives rise to a question about whether we can further improve the efficiency of dataset condensation by exploiting the regularity in the given dataset.

To this end, we naturally assume that each datapoint is generated from a *code* in a compact latent space followed by a sharable mapping function from the latent space to the original input space (i.e.

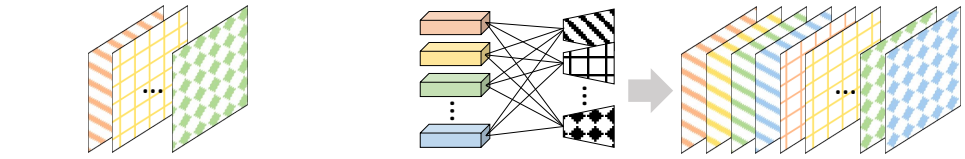


Figure 1: **(Left)** Conventional dataset condensation approaches that directly parameterize the synthetic examples in input space. **(Right)** Our approach that parameterize latent codes and multiple decoders that can generate much more number of examples with essentially the same parameter count.

decoder), both of which are learned in an end-to-end manner. In this way, we can dramatically increase the number of generated synthetic examples with the same parameter count because the latent space can be made to be much lower dimensional than the original input space, and also since we can assume as many decoders as appropriate to capturing various *styles* represented in the dataset. See Figure 1 for the concept. Note that the additional per-class parameter count introduced by the decoders is negligible because those decoders are tiny and can be shared across all the synthetic examples and classes. Furthermore, the knowledge is shared among the generated synthetic examples through the shared latent space and the set of shared decoders applicable to any latent codes interchangeably, dramatically improving the trade-off between compression ratio and quality of the synthetic dataset.

In algorithmic side, we show that the previous approaches that subsample synthetic examples for computational efficiency (e.g. gradient matching (Zhao et al., 2021) or distribution matching (Zhao & Bilen, 2022a)) produce biased gradients, preventing the synthetic examples from being diversified. We further experimentally observe that subsampling of the real examples produce high-variance gradients, preventing the model from finding a good solution. Those observations suggest that we have to perform full batch training for both real and synthetic dataset. We thus adopt distribution matching (Zhao & Bilen, 2022a) as it does not require expensive second order computations unlike gradient matching (Zhao et al., 2021). We also prestore and reuse some of the information needed to perform distribution matching for further improvement of the training efficiency.

We experimentally show that our method, which we name as Knowledge Factorization and Sharing (KFS), achieves new state-of-the-art records on various benchmark datasets such as SVHN, CIFAR10, CIFAR100, TinyImageNet and ImageNet-Subset with varying amount of parameters for constructing a synthetic dataset. We also show that the synthetic examples generated from KFS produce decent performances on network architectures that are different from what they have been trained on (e.g. from ConvNet-3 to DenseNet-121 and EfficientNet-B0), and that it provides superior performance over various amount of training budget at evaluation time without overfitting. We summarize the contribution of this paper as follows:

- We introduce KFS, a novel dataset condensation method which factorizes the synthetic data generating process into a set of learnable latent codes and decoders, thereby dramatically increasing the number of synthetic examples with a limited parameter count.
- We show that the modeling of the data generating process effectively allows knowledge sharing between the synthetic examples such that they achieve far better trade-off between the compression ratio versus the quality of the generated examples.
- We further show that the existing dataset condensation approaches suffer from biased and high-variance gradients, and experimentally verify that full batch training significantly improves the quality of the synthetic dataset.
- Our KFS achieves new state-of-the-art records in all datasets we consider, outperforming the previous state-of-the-art by significant margins.

2 Related Work

Dataset condensation. Wang et al. (2018) firstly introduced a differentiable approach to dataset condensation problem through a bilevel optimization formulation. However, in their work only a few inner-gradient steps are assumed for computational efficiency and additionally a shared initialization is learned and entangled with the learned data. Zhao et al. (2021) overcome the limitation

by matching the network weights and approximating the expensive bi-level optimization with short-horizon approximation, resulting in gradient matching between real and synthetic examples for each step. Zhao & Bilen (2021) further develop the idea by learning a differentiable Siamese networks for augmenting the synthetic examples. Cazenavette et al. (2022) recently propose to match much longer segments of learning trajectories, yielding competitive performances by guiding networks to a more similar state. On the other hand, instead of matching the network weights, Zhao & Bilen (2022a) match distributions of real and synthetic dataset in a feature space. Although they sometimes show inferior performance to gradient matching, their approach is computationally more efficient as they do not require to compute expensive second-order derivatives such as vector-Jacobian products (VJPs) required by gradient matching. Similarly, Wang et al. (2022) propose to layer-wisely align features with auxiliary discriminative loss and bi-level formulation for adjusting parameter updates. On the other hand, Nguyen et al. (2020; 2021) propose to make use of the connection between infinitely wide neural networks and kernel ridge regression, but their approach requires hundreds of GPUs for training. Perhaps the most relevant to our work is recently proposed IDC (Kim et al., 2022). They generate various synthetic data from a single set of condensed examples by segmenting and upscaling the condensed examples. Their approach can be seen as exploiting the regularity in the dataset, for instance, spatially nearby pixels look similar to each other. Although they achieve strong performances on many datasets, it is questionable whether the proposed generative process is sophisticated enough to fully exploit the regularity in the dataset. In this work, we make a more natural and general assumption on the data generating process, yielding new state-of-the-art records on all the datasets we consider.

Generative models. Our work is closely related to deep generative models such as VAE (Kingma & Welling, 2013) and GAN (Goodfellow et al., 2014). They maximize the likelihood of data by learning a compact latent space and a shared decoder, similarly to our work. However, it is questionable whether maximizing the data likelihood would be sufficient for the target task such as classification. Also, whereas it has been shown that modality is crucial for the performance of condensed dataset (Kim et al., 2022), they are known to suffer from the posterior collapse (or mode collapse) problem (Razavi et al., 2019a). Van Den Oord et al. (2017) and Razavi et al. (2019b) overcome the issue by proposing VQ-VAE that models discrete representations in the latent space. The discreteness of the approach makes them closely related to the dataset condensation, but it is questionable whether such generative models can outperform the recent dataset condensation approaches. For instance, Such et al. (2020) combine GAN with meta-learning to generate synthetic examples, but they only slightly outperform Wang et al. (2018). Recently, Zhao & Bilen (2022b) propose to combine a pre-trained GAN generator and dataset condensation learning objective for improving the performance of downstream tasks. However, in addition to the large parameter count of the pre-trained generator, they require additional parameters for learning the discrete codes, making them not directly comparable under the conventional protocol of dataset condensation.

3 Approach

We next present our approach, Knowledge Factorization and Sharing (KFS), which efficiently factorizes the data generating process into latent codes and decoders so that knowledge is effectively shared among the synthetic examples generated.

3.1 Knowledge Factorization and Sharing in Latent Space

Kim et al. (2022) observed that what determines the performance of condensed dataset is its modality rather than resolution. We follow the same principle but aim to find even better trade-off between increasing modality and preserving quality of synthetic examples.

Latent code - decoder factorization. Our approach starts from the assumption that the datapoints lie in a compact latent space whose dimension can be much lower than that of the actual input space. Thanks to its lower dimensionality, we can dramatically increase the number of datapoints given a limited parameter count (e.g., the number of parameters used by synthetic images per each class). We can also preserve the quality of the synthetic images to the extent to which the assumption holds, which has been proven to be generally the case. The only space overhead is the additional parameters for a tiny decoder that maps those latent codes into the input space. Note that this decoder can be shared across all the datapoints and classes, and even if the decoder is relatively large we can flexibly adjust the number of latent codes so that the total parameter counts are comparable.

Let us take an illustrative example. Suppose we want to condense a dataset consisting of images of shape $3\times32\times32 = 3072$ (e.g. SVHN or CIFAR10). Instead of learning those 3072 parameters in the pixel space, we instead learn much smaller codes in, say $12 \times 4 \times 4 = 192$ dimensional latent space and decode them back to the original pixel space with a shared decoder. Given the same parameter count, this allows us to generate 16 times more number of images than the existing approaches ($192 \times 16 = 3072$). Note that the per-class parameter count of shared decoder is negligible. In the above example, we could use a tiny 3-layer deconvolutional architecture with channels decreasing as $12 \to 9 \to 6 \to 3$ and 2×2 kernel, resulting in only 738 parameters in total. Since this decoder can be shared across all classes, dividing it with the number of classes (e.g. 10) results in a negligible amount of per-class parameter count (e.g. 73.8) compared to the actual image size.

Sharing various styles with multiple decoders. With the above factorization, we can now decouple the compressed expression of the instances from how to decode them back. This decoding process can be thought of as adding details to the latent codes so that the decoded instances have sufficient information in the original input space. For instance, given a compressed expression of a car, the decoder may add in or supplement background details or colors to complete the image.

This insight implies that the number of decoders can be as many as the various *styles* that exist in the actual dataset. For instance, given the same compressed expression of an airplane, the background may be ocean, sky, forest, or sunset. By assuming multiple decoders, we could expect each of the decoders to explain each of those distinct styles in the dataset respectively. Furthermore, since the knowledge of how to add styles to the latent codes can be shared across all instances and classes, we can dramatically increase the number of synthetic images without increasing parameter count too much, similarly to Cartesian product between a pair of sets. For instance, if there are 8 shared decoders and 16 latent codes for each of 10 classes, we can generate total $8 \times 16 = 128$ synthetic images with only around 3131 parameter count for each class, which is less than 2% increase from the parameter count of a single image $3 \times 32 \times 32 = 3072$.

3.2 Training with Unbiased and Low-variance Distribution Matching

Basically, our approach is compatible with any learning-based dataset condensation approaches that allows differentiation w.r.t latent codes and decoder parameters by backpropagating through the generated examples, such as gradient matching (Zhao et al., 2021) or distribution matching (Zhao & Bilen, 2022a). In this paper, we propose to use distribution matching as it does not require expensive vector-Jacobian products (VJPs) computations whereas gradient matching requires to compute VJP for every training step.

Distribution matching. Here we introduce notations for further discussion. We denote a set of N real examples for each class $c = 1, \dots, C$ as $\mathcal{X}_c = \{x_{c,1}, \dots, x_{c,N}\}$. Similarly, a set of M latent codes for each class c is $\Theta_c = \{\theta_{c,1}, \dots, \theta_{c,M}\}$ and we collect them into $\Theta = \{\Theta_1, \dots, \Theta_C\}$. Note that the dimension of θ is much lower than that of x. $g(x)$ is a shared feature extractor which takes x as an input. $f(\theta; \phi_d)$ is d-th decoder parameterized by ϕ_d that takes a latent code θ as an input. We assume there are D decoders and collect their parameters into $\phi = \{\phi_1, \dots, \phi_D\}$. With these notations, we consider the following empirical Maximum Mean Discrepancy (MMD) loss (Gretton et al., 2012; Zhao & Bilen, 2022a).

$$L(\Theta, \phi) = \frac{1}{C} \sum_{c=1}^{C} \frac{1}{2} \left\| \frac{1}{N} \sum_{n=1}^{N} g(x_{c,n}) - \frac{1}{DM} \sum_{d=1}^{D} \sum_{m=1}^{M} g(f(\theta_{c,m}; \phi_d)) \right\|_2^2 \tag{1}$$

Following Zhao & Bilen (2022a), we sample $g(\cdot)$ from random initializations rather than from a set of pretrained networks, which is computationally more efficient and works well in practice.

Bias due to subsampling of synthetic examples. Zhao & Bilen (2022a) further approximate the first term (i.e. mean of $g(x)$ over n) and the second term (i.e. mean of $g(f(\theta; \phi))$ over d and m) by random subsampling for computational efficiency. However, whereas the gradient $\nabla_{\Theta,\phi} L(\Theta, \phi)$ is unbiased w.r.t the random subsampling of indices of real data $\{n\}_{n=1}^{N}$ or classes $\{c\}_{c=1}^{C}$, the gradient is actually biased w.r.t the random subsampling of indices of latent codes $\{m\}_{m=1}^{M}$ or decoders $\{d\}_{d=1}^{D}$. Specifically, suppose for simplicity we randomly sample a single pair of $m \sim \text{Uniform}(1, M)$ and $d \sim \text{Uniform}(1, D)$ and denote the corresponding loss as $\hat{L}$. Then we can

easily compute the bias of gradient $\mathbb{E}_{m,d}[\nabla_{\Theta,\phi}\hat{L}(\Theta,\phi)] - \nabla_{\Theta,\phi}L(\Theta,\phi)$ as

$$\nabla_{\Theta,\phi}\frac{1}{C}\sum_{c=1}^{C}\frac{1}{2}\Bigg\{\frac{1}{DM}\sum_{d=1}^{D}\sum_{m=1}^{M} g(f(\theta_{c,m};\phi_d))^{\mathsf{T}} g(f(\theta_{c,m};\phi_d)) - \frac{1}{D^2M^2}\sum_{d=1}^{D}\sum_{d'=1}^{D}\sum_{m=1}^{M}\sum_{m'=1}^{M} g(f(\theta_{c,m};\phi_d))^{\mathsf{T}} g(f(\theta_{c,m'};\phi_{d'}))\Bigg\}. \tag{2}$$

See Appendix D for the derivation. The above result suggests that the bias comes from ignoring the interactions within the latent codes and within the decoders, respectively. The biased gradient fails to minimize the inner product of the embeddings computed from different codes and decoders, such as $g(f(\theta_{c,m};\phi))^{\mathsf{T}} g(f(\theta_{c,m'};\phi))$ and $g(f(\theta_{c,m};\phi_d))^{\mathsf{T}} g(f(\theta_{c,m};\phi_{d'}))$, preventing the codes and decoders from being diversified. Intuitively, if we sample $\theta \in \Theta_c$, then each individual θ will try to explain the whole class set $\mathcal{X}_c$ at a time without considering the existence of other codes $\theta' \in \Theta_c \backslash \{\theta\}$ due to the sampling. The same intuition holds for sampling the decoder $f(\cdot;\phi_d)$. Therefore, we should not sample m and d. However, despite of its importance, most of the recent works do not consider correcting the bias (Zhao et al., 2021; Zhao & Bilen, 2021; 2022a; Cazenavette et al., 2022; Wang et al., 2022; Kim et al., 2022).

Variance due to subsampling of real examples. We also found that random subsampling of real examples is detrimental to performance due to large variance. Again, suppose for simplicity we randomly sample a single index $n \sim \text{Uniform}(1, N)$ independently for each class c and denote the corresponding loss as $\tilde{L}$. Then the variance of the gradient $\text{Var}_n(\nabla_{\Theta,\phi}\tilde{L}(\Theta,\phi))$ is

$$\frac{1}{C^2}\sum_{c=1}^{C} V_c^{\mathsf{T}}\left\{\frac{1}{N}\sum_{n=1}^{N} g(x_{c,n})g(x_{c,n})^{\mathsf{T}} - \left(\frac{1}{N}\sum_{n=1}^{N} g(x_{c,n})\right)\left(\frac{1}{N}\sum_{n=1}^{N} g(x_{c,n})\right)^{\mathsf{T}}\right\} V_c \tag{3}$$

where $V_c = \frac{1}{DM}\sum_{d=1}^{D}\sum_{m=1}^{M}\nabla_{\Theta,\phi} g(f(\theta_{c,m};\phi_d))$. See Appendix E for the derivation. The middle term in the big braces is nothing but the variance of embeddings of real examples $g(x_{c,n})$ over $n = 1, \ldots, N$. The problem is that since we sample $g(\cdot)$ from random initializations according to the original distribution matching algorithm (Zhao & Bilen, 2022a), the variance of $g(x_{c,n})$ tends to be very high compared to the discriminative power of the representation $g(\cdot)$ across the classes. We empirically observe that such relatively high variance makes the gradient $\nabla_{\Theta,\phi}\tilde{L}(\Theta,\phi)$ too noisy for the training process to find a good solution.

Full batch training. Based on the above analysis, we propose to perform full batch training for both real and synthetic dataset for unbiased and low-variance training. In case of real dataset, we basically need to compute the embedding mean of real examples $\frac{1}{N}\sum_{n=1}^{N} g(x_{c,n})$ across the classes and random weights $w_1, \ldots, w_K$, with K denoting the total training steps. Instead of repeatedly computing them for every experiment, we speed up our experiments by precomputing those means once and storing them in an external device with the corresponding random weights, so that we can quickly restore them for the later experiments. In case of synthetic dataset, we simply distribute the classes over multiple GPUs and accumulate the gradients over all the processes. When the size of each class is too large to fit in a single GPU, we compute the embedding mean over the whole synthetic examples but backpropagate through only a subset of them for handling memory issue, while keeping the gradient unbiased.

Discussion. Although the distribution matching objective in Eq. (1) works well in practice, it is not clear why it should work with repeated sampling of $g(\cdot)$ from random initializations. A simple explanation may be that for each class, a well-condensed synthetic dataset should lead to a similar embedding mean to that of real dataset regardless of those random initializations. However, further justification and understanding of the objective is required, which we leave as a future work.

4 Experiments

We next experimentally validate the efficacy of KFS on various datasets and learning scenarios.

Datasets. We consider the following benchmark datasets such as SVHN (32×32) (Netzer et al., 2011), CIFAR10 (32×32) (Krizhevsky et al., 2009), CIFAR100 (32×32) (Krizhevsky et al., 2009),

Table 1: **Classification accuracies (%) on ConvNet-3.** For our method, we report mean and standard deviation over 15 evaluations (3 training runs and 5 evaluations for each run).

Dataset	SVHN			CIFAR10			CIFAR100		TinyImageNet	
Images / Class	1	10	50	1	10	50	1	10	1	10
Param. / Class	3072	30720	153600	3072	30720	153600	3072	30720	12288	122880
Random	14.6±1.6	35.1±4.1	70.9±0.9	14.4±2.0	26.0±1.2	43.4±1.0	4.2±0.3	14.6±0.5	1.4±0.1	5.0±0.2
Herding	20.9±1.3	50.5±3.3	72.6±0.8	21.5±1.2	31.6±0.7	40.4±0.6	8.4±0.3	17.3±0.3	2.8±0.2	6.3±0.2
DC	31.2±1.4	76.1±0.6	82.3±0.3	28.3±0.5	44.9±0.5	53.9±0.5	12.8±0.3	25.2±0.3	-	-
DSA	27.5±1.4	79.2±0.5	84.4±0.4	28.8±0.7	52.1±0.5	60.6±0.5	13.9±0.3	32.3±0.3	-	-
DM	20.3±2.1	73.5±1.0	84.2±0.0	26.0±0.8	48.9±0.6	63.0±0.4	11.4±0.3	29.7±0.3	3.9±0.2	12.9±0.4
KIP to NN	57.3±0.1	75.0±0.1	80.5±0.1	49.9±0.2	62.7±0.3	68.6±0.2	15.7±0.2	28.3±0.1	-	-
CAFE + DSA	42.9±3.0	77.9±0.6	82.3±0.4	31.6±0.8	50.9±0.5	62.3±0.4	14.0±0.3	31.5±0.2	-	-
Traj. Matching	-	-	-	46.3±0.8	65.3±0.7	71.6±0.2	24.3±0.3	40.1±0.4	8.8±0.3	23.2±0.2
IDC	68.1±0.1	87.3±0.2	90.2±0.1	50.0±0.4	67.5±0.5	74.5±0.1	-	44.8±0.2	-	-
KFS (ours)	**82.9±0.4**	**91.4±0.2**	**92.2±0.1**	**59.8±0.5**	**72.0±0.3**	**75.0±0.2**	**40.0±0.5**	**50.6±0.2**	**22.7±0.2**	**27.8±0.2**
Full dataset		95.4±0.1			84.8±0.1			56.2±0.3		37.6±0.4

Table 2: **Cross architecture experiments.** Conv3, RN10, and DN121 denote ConvNet-3, ResNet-10, and DenseNet-121, respectively. We train on ConvNet-3 and evaluate on the three architectures.

Dataset	Images / Class	1			10			50		
	Test Architecture	Conv3	RN10	DN121	Conv3	RN10	DN121	Conv3	RN10	DN121
SVHN	DSA	27.5±1.4	13.2±1.1	13.3±1.4	79.2±0.5	19.5±1.5	23.1±1.9	84.4±0.4	41.6±2.1	58.0±3.1
	DM	20.3±2.1	10.5±2.8	13.6±1.0	73.5±1.0	28.2±1.5	24.8±2.5	84.2±0.0	54.7±1.3	58.4±2.7
	IDC	68.1±0.1	39.6±1.5	39.9±2.9	87.3±0.2	83.3±0.2	82.8±0.2	90.2±0.1	89.1±0.2	**91.0±0.3**
	KFS (ours)	**82.9±0.4**	**75.7±0.8**	**81.0±0.7**	**91.4±0.2**	**90.3±0.2**	**89.7±0.2**	**92.2±0.1**	**90.9±0.2**	90.2±0.2
	Full dataset	95.4±0.1	93.8±0.5	89.1±0.8	95.4±0.1	93.8±0.5	89.1±0.8	95.4±0.1	93.8±0.5	89.1±0.8
CIFAR10	DSA	28.8±0.7	25.1±0.8	25.9±1.8	52.1±0.5	31.4±0.9	32.9±1.0	60.6±0.5	49.0±0.7	53.4±0.8
	DM	26.0±0.8	13.7±1.6	12.9±1.8	48.9±0.6	31.7±1.1	32.2±0.8	63.0±0.4	49.1±0.7	53.7±0.7
	IDC	50.0±0.4	41.9±0.6	39.8±1.2	67.5±0.5	63.5±0.1	61.6±0.6	74.5±0.1	72.4±0.5	71.8±0.6
	KFS (ours)	**59.8±0.5**	**47.0±0.8**	**49.5±1.3**	**72.0±0.3**	**70.3±0.3**	**71.4±0.4**	**75.0±0.2**	**75.1±0.3**	**76.3±0.4**
	Full dataset	84.8±0.1	87.9±0.2	90.5±0.3	84.8±0.1	87.9±0.2	90.5±0.3	84.8±0.1	87.9±0.2	90.5±0.3

and TinyImageNet (64×64) (Le & Yang, 2015). **Baselines.** We consider coreset selection approaches such as Random and Herding. For dataset condensation approaches, we compare with gradient (or trajectory) matching such as DC (Zhao et al., 2021), DSA (Zhao & Bilen, 2021), Trajectory Matcing (Cazenavette et al., 2022), and IDC (Kim et al., 2022), as well as distribution matching such as DM (Zhao & Bilen, 2022a) and CAFE (Wang et al., 2022). We also compare against KIP (Nguyen et al., 2021). **Experimental setup.** For fair comparison, we use essentially the same parameter count as the baselines. For instance, if there are 10 classes with "Images / Class" = 10 (in Table 1 and Table 2) and the image shape is 3×32×32, then the total per-class parameter count is 30720. In this case our KFS assumes 16 codes of shape 12×4×4 (such that 16×12×4×4 = 30720) with the 8 decoders requiring only 73.8 additional parameters per class (see section 3.1). See Appendix A for further details about the experimental setup.

Quantitative analysis. Table 1 shows the performance of the baselines and our KFS from each dataset with varying number of images (or parameter count) per class. We can see that KFS outperforms all the previous methods by significant margins in all datasets and settings. KFS especially performs well when the given parameter count per class is small. For instance, when "Images / Class" = 1, KFS shows 14.8% and 9.8% performance improvements over IDC on SVHN and CIFAR10 respectively, the previous state-of-the-art recently introduced by Kim et al. (2022). This reconfirms the observation from Kim et al. (2022) that increasing the modality in the synthetic dataset is more important than polishing a few high-quality examples when the parameter count is limited. However, KFS provides much higher modality (i.e. more number of synthetic examples) than IDC with essentially the same parameter count because KFS can compress each instance far more tightly by systematically factorizing and sharing the knowledge across the synthetic examples.

Cross-architecture generalization. Table 2 shows how the baselines and KFS perform when the network architecture becomes different at the evaluation time (ResNet-10 and DenseNet-121) from the one that used for condensation (ConvNet-3). We can see that the performance of KFS is more robust to the change of network architectures. For instance, when "Images/Class" = 1 in SVHN dataset, the performance drop of KFS is only 7.2% and 1.9% for ResNet-10 and DenseNet-121 respectively, whereas the drop is 28.5% and 28.2% for IDC. Such robustness means that the learned synthetic examples from KFS are more natural as they are less specifically tailored to a specific network architecture. Similarly, when "Images/Class" = 50 in CIFAR10 dataset, the performance of KFS even improves as the architecture changes, while the performance of other baselines decreases.

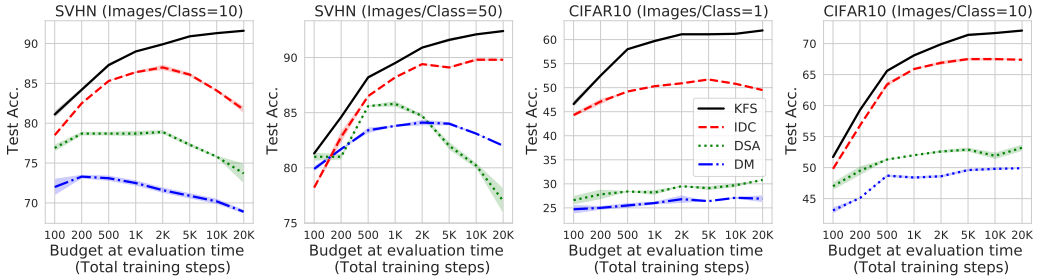


Figure 2: The evaluation results across various amount of budgets defined as the number of training steps allowed for each evaluation. Batchsize is set to $\min(N_{\text{total}}, 256)$ where N_{total} is the number of all examples in each synthetic dataset. We further provide wall-clock time analysis using online decoding in Figure 9.

Figure 3: **(Top row)** Synthetic images (SVHN and CIFAR10) generated by varying latent codes while fixing a decoder. **(Bottom row)** Synthetic images generated by varying decoders while fixing a code.

Training budgets at evaluation time. One may wonder whether the good performance of KFS is solely due to the larger size of the synthetic dataset rather than the quality of each example, such that KFS inherently requires much longer evaluation time than the baselines. To address this question, we consider various amount of budget at evaluation time defined as total training steps allowed. In Figure 2 we can clearly see that KFS provides significantly better performances than the baselines across all range of budgets: from the limited budgets (100-500 training steps) to the longer training steps (1K-20K steps), demonstrating that the quality of synthetic examples generated from KFS is also comparable to or even better than that of the baselines. Interestingly, whereas the baselines often suffer from overfitting (e.g., SVHN dataset), our KFS has no such issue hence can provide more diverse options to users about how much longer to train their models for better performance.

Qualitative analysis. Figure 3 shows the visualization of the synthetic images generated from KFS. We can see from the top row that KFS can generate various modes of the data distribution by varing the learned latent codes while fixing a decoder. For instance, in SVHN dataset the digits show various styles with different adjacent numbers, and in CIFAR10 dataset the orientations and shapes of objects are diverse, demonstrating the ability of the latent codes to capture multi-modality of each data distribution. We also fix a code and vary the decoders in the bottom row. The synthetic examples show various backgrounds and styles of the objects such as strokes and colors. It clearly shows how the factorization between latent codes and decoders allows KFS to efficiently share knowledge between the synthetic examples.

ImageNet-Subset. We also experiment with ImageNet-Subset dataset (Deng et al., 2009; Tian et al., 2020) which is of high resolution (224×224). We used 10-classes version of the dataset. Table 3 shows that our KFS outperforms all the baselines on all the settings and network architectures. The results demonstrate that KFS is also effective for high-resolution images as well. The visualiza-

Table 3: **ImageNet-10 experiments.** We train on ResNetAP-10 and evaluate on the three architectures.

Images / Class	1			10		
Test Architecture	ResNetAP-10	ResNet-18	EfficientNet-B0	ResNetAP-10	ResNet-18	EfficientNet-B0
DSA	-	-	-	52.7	44.1	48.3
DM	-	-	-	52.3	41.7	45.0
IDC	60.5±0.5	55.8±0.4	55.2±1.3	72.8±0.6	73.6±0.4	74.7±0.5
KFS (ours)	**70.7**±0.8	**66.9**±0.4	**67.8**±0.3	**75.4**±0.4	**75.1**±0.7	**74.9**±0.5
Full dataset	86.8±0.2	89.5±0.2	91.8±0.1	86.8±0.2	89.5±0.2	91.8±0.1

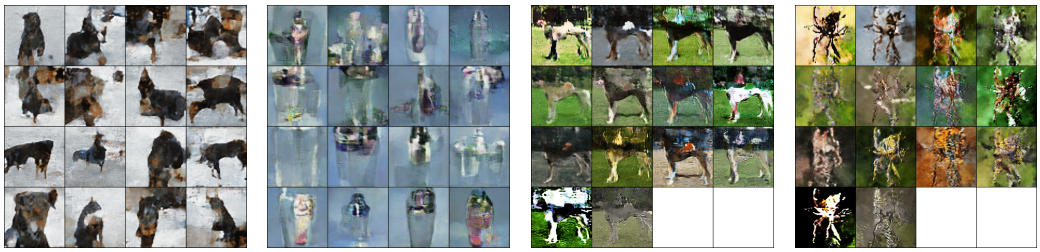


Figure 4: **(First two)** Synthetic images generated by varying latent codes while fixing a decoder. **(Last two)** Synthetic images generated by varying decoders while fixing a code.

tion in Figure 4 shows that our method can effectively capture the modality of the dataset in terms of both various latent codes and decoders. See Appendix B for the experimental setups.

Comparison to generative models. Note that our KFS aims to model synthetic data generating process and hence is closely related to deep generative models such as VAE (Kingma & Welling, 2013) and GAN (Goodfellow et al., 2014). Therefore, we compare against VQ-VAE (van den Oord et al., 2017) whose discrete representation of latent codes is similar to our KFS, and BigGAN (Brock et al., 2019) which is one of the state-of-the-art large-scale deep generative models. We control the parameter count of their decoders by varying the channel sizes. Synthetic datasets are constructed by performing class-conditional ancestral sampling from the prior and decoder.

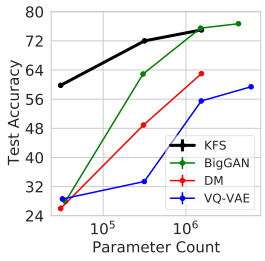


Figure 5: Comparison to generative models (CIFAR10).

Figure 5 shows that our KFS significantly outperforms those generative models when the parameter count is limited and performs similarly to BigGAN when the parameter count is larger. It is interesting to note that the performance of BigGAN significantly outperforms most of the existing dataset condensation methods such as distribution matching (Zhao & Bilen, 2022a) in Figure 5, although BigGAN is not specifically tailored to dataset condensation. The main advantage of existing dataset condensation methods is that the number of samples in the synthetic dataset is comparably much smaller than the generative models, but we definitely need more research about whether we can further benefit from exploiting the generative models for dataset condensation (Zhao & Bilen, 2022b).

Continual learning. Rehearsal-based continual learning approaches construct a compact subset of representative examples the learner has seen so far and train it with the currently given examples to avoid catastrophic forgetting (Rebuffi et al., 2017). In this section, we explore whether the use of dataset condensation approaches can improve the performance of it by enhancing the quality and representativeness of the subset. Following Zhao & Bilen (2022a), we consider class incremental learning scenario where for each of the 5 stages 20 classes from the CIFAR-100 dataset are sequentially given. We condense a dataset only with the currently available classes at each stage. Also, training is done only with the condensed dataset accumulated thus far, not the real examples, following Joseph et al. (2021) and Kim et al. (2022). We use

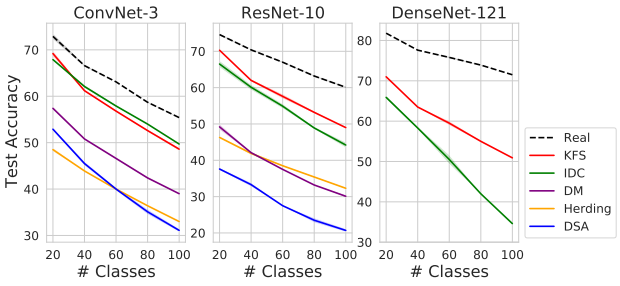
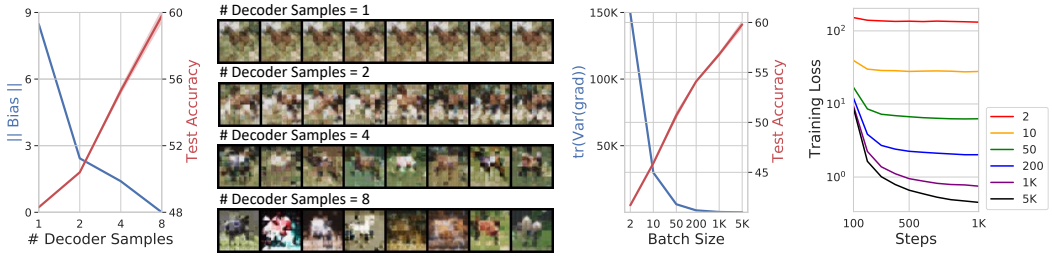


Figure 6: **Continual learning experiments.** "Real" is the upper-bound that trains a model only with real examples.



(a) (b) (c) (d)

Figure 7: **Bias and variance analysis** on CIFAR10 dataset ("Image/Class" $= 1$). **(a,b)** corresponds to the analysis of bias and **(c,d)** to variance. **(a)** Test accuracy and ℓ_2-norm of bias of gradient vs. the number of decoder samples used for training. **(b)** Visualization of the decoders trained with various number of decoder samples. **(c)** Test accuracy and trace of variance of gradient vs. batch size of real examples. Large trace means that the gradient has high variance. **(d)** Convergence of training loss with various batch size of real examples.

ConvNet-3 for constructing the synthetic dataset and evaluate on ConvNet-3, ResNet-10, and DenseNet-121. The baselines include Herding, DSA, and DM, and IDC.

Figure 6 shows the efficacy of our method in this continual learning scenario. Although KFS performs similarly to IDC when the network architecture is ConvNet-3, KFS shows significantly better performance when evaluating on different network architectures such as ResNet-10 and DenseNet-121. Therefore, by using KFS, we can efficiently condense a dataset with a smaller network and evaluate on larger networks without severe interference.

Ablation study. We perform ablation study in Figure 8 by varying the number of decoders used for condensing a dataset. We use CIFAR10 dataset with "Image/Class" $= 1$ setting. We see that using more number of decoders is clearly better for performance, demonstrating the importance of capturing various styles and modalities in the dataset.

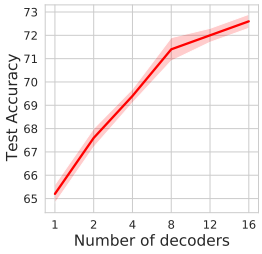


Figure 8: **Ablation study**.

Bias and variance analysis. Lastly, we analyze in Figure 7 the effect of bias and variance of gradients due to subsampling of synthetic and real examples. Firstly, we see the effect of bias by controlling the number of decoder samples used for training. For instance, we may compute the gradient at each iteration by subsampling only a few decoders instead of using all the decoders, leading to a biased gradient. Figure 7(a) shows that the bias actually increases as we subsample fewer number of decoders and it significantly deteriorates the test accuracy. This is because the bias prevents the decoders from being diversified (see Eq. (2)). Figure 7(b) shows that the synthetic examples generated from the decoders trained with fewer number of samples are actually homogeneous across the decoders. Secondly, we analyze the effect of variance by varying the batch size for the real examples. Figure 7(c) shows that the variance becomes higher as we use smaller batch size, and test accuracy quickly gets worse accordingly. We further observe in Figure 7(d) that this high variance makes the training loss to saturate quickly compared to when we use larger or full batch size, preventing the model from finding a good solution.

5 CONCLUSION

In this paper, we introduced a novel method for solving dataset condensation problem in a systematic and efficient way. Instead of parameterizing synthetic examples directly in input space, we proposed to fully exploit the regularity in a given dataset, such that we assume a generative process of synthetic examples based on the factorization between latent codes and decoders. Based on it, we showed how to increase the number of synthetic examples with essentially the same parameter count by combining the codes and decoders interchangeably. We further demonstrated how the knowledge is efficiently shared between synthetic examples with the framework, achieving superior trade-off between compression ratio and quality of the synthetic examples. Potential future research directions include scaling up the method to achieve or even surpass the performances of the full dataset, and further developing the idea to multi-task or meta-learning setting where the goal is to share knowledge of how to compress a dataset between multiple tasks.

REPRODUCIBILITY STATEMENT

For reproducibility, we clearly stated all the experimental setups and hyperparameter configurations in Section A and Section B. We also submit the code for running the main experiments in Table 1, 2, 3. All the numerical results have been obtained from running at least 3 evaluations. Further, we will release our code and synthetic data to be publicly available after acceptance.

REFERENCES

Yoshua Bengio. Gradient-based optimization of hyperparameters. *Neural computation*, 12(8):1889–1900, 2000.

Zalán Borsos, Mojmir Mutny, and Andreas Krause. Coresets via bilevel optimization for continual learning and streaming. *Advances in Neural Information Processing Systems*, 33:14879–14890, 2020.

Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale GAN training for high fidelity natural image synthesis. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=B1xsqj09Fm.

George Cazenavette, Tongzhou Wang, Antonio Torralba, Alexei A Efros, and Jun-Yan Zhu. Dataset distillation by matching training trajectories. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 4750–4759, 2022.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 248–255, 2009. doi: 10.1109/CVPR.2009.5206848.

Luca Franceschi, Michele Donini, Paolo Frasconi, and Massimiliano Pontil. Forward and reverse gradient-based hyperparameter optimization. In *International Conference on Machine Learning*, pp. 1165–1173. PMLR, 2017.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014.

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

KJ Joseph, Salman Khan, Fahad Shahbaz Khan, and Vineeth N Balasubramanian. Towards open world object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 5830–5840, 2021.

Jang-Hyun Kim, Jinuk Kim, Seong Joon Oh, Sangdoo Yun, Hwanjun Song, Joonhyun Jeong, Jung-Woo Ha, and Hyun Oh Song. Dataset condensation via efficient synthetic-data parameterization. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 11102–11118. PMLR, 17–23 Jul 2022.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

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

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

Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. In *International Conference on Learning Representations*, 2018.

David Lopez-Paz and Marc'Aurelio Ranzato. Gradient episodic memory for continual learning. *Advances in neural information processing systems*, 30, 2017.

Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. 2011.

Timothy Nguyen, Zhourong Chen, and Jaehoon Lee. Dataset meta-learning from kernel ridge-regression. *arXiv preprint arXiv:2011.00050*, 2020.

Timothy Nguyen, Roman Novak, Lechao Xiao, and Jaehoon Lee. Dataset distillation with infinitely wide convolutional networks. *Advances in Neural Information Processing Systems*, 34:5186–5198, 2021.

Jeff M. Phillips. Coresets and sketches. *CoRR*, abs/1601.00617, 2016.

Ali Razavi, Aaron van den Oord, Ben Poole, and Oriol Vinyals. Preventing posterior collapse with delta-VAEs. In *International Conference on Learning Representations*, 2019a.

Ali Razavi, Aaron Van den Oord, and Oriol Vinyals. Generating diverse high-fidelity images with vq-vae-2. *Advances in neural information processing systems*, 32, 2019b.

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

Felipe Petroski Such, Aditya Rawal, Joel Lehman, Kenneth Stanley, and Jeffrey Clune. Generative teaching networks: Accelerating neural architecture search by learning to generate synthetic training data. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 9206–9216. PMLR, 13–18 Jul 2020.

Yonglong Tian, Dilip Krishnan, and Phillip Isola. Contrastive multiview coding. In *European conference on computer vision*, pp. 776–794. Springer, 2020.

Mariya Toneva, Alessandro Sordoni, Remi Tachet des Combes, Adam Trischler, Yoshua Bengio, and Geoffrey J Gordon. An empirical study of example forgetting during deep neural network learning. In *International Conference on Learning Representations*, 2018.

Aaron van den Oord, Oriol Vinyals, and koray kavukcuoglu. Neural discrete representation learning. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper/2017/file/7a98af17e63a0ac09ce2e96d03992fbc-Paper.pdf.

Aaron Van Den Oord, Oriol Vinyals, et al. Neural discrete representation learning. *Advances in neural information processing systems*, 30, 2017.

Kai Wang, Bo Zhao, Xiangyu Peng, Zheng Zhu, Shuo Yang, Shuo Wang, Guan Huang, Hakan Bilen, Xinchao Wang, and Yang You. 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.

Tongzhou Wang, Jun-Yan Zhu, Antonio Torralba, and Alexei A. Efros. Dataset distillation. *CoRR*, abs/1811.10959, 2018.

Bo Zhao and Hakan Bilen. Dataset condensation with differentiable siamese augmentation. In *International Conference on Machine Learning*, pp. 12674–12685. PMLR, 2021.

Bo Zhao and Hakan Bilen. Dataset condensation with distribution matching, 2022a. URL https://openreview.net/forum?id=T2F5aBbSEUQ.

Bo Zhao and Hakan Bilen. Synthesizing informative training samples with gan. *arXiv preprint arXiv:2204.07513*, 2022b.

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

A EXPERIMENTAL SETUP

We provide some additional information about the experimental setup for reproducing our results on SVHN, CIFAR10, CIFAR100, and TinyImageNet.

Table 4: The architecture of Low-Resolution Decoder in pytorch style. H, W are height and weight of target image. $Cs[:]$ is an array that determines channels of each convolutional block. Here the size of the latent codes are assumed to be $Cs[2] \times H/8 \times W/8$.

Output Size	**Layers**
$Cs[2] \times H/8 \times W/8$	Latent Codes
$Cs[1] \times H/4 \times W/4$	`ConvTranspose2d`($Cs[2]$, $Cs[1]$, kernel_size=2, stride=2, padding=0)
$Cs[0] \times H/2 \times W/2$	`ConvTranspose2d`($Cs[1]$, $Cs[0]$, kernel_size=2, stride=2, padding=0)
$3 \times H \times W$	`ConvTranspose2d`($Cs[0]$, 3, kernel_size=2, stride=2, padding=0)
$3 \times H \times W$	`Sigmoid`

Table 5: The architecture of High-Resolution Decoder in pytorch style. H, W are height and weight of target image. $Cs[:]$ is an array that determines channels of each convolutional block. Here the size of the latent codes are assumed to be $Cs[1] \times H/4 \times W/4$.

Output Size	**Layers**
$Cs[1] \times H/4 \times W/4$	Latent Codes
$Cs[0] \times H/2 \times W/2$	`ConvTranspose2d`($Cs[1]$, $Cs[0]$, kernel_size=2, stride=2, padding=0)
$3 \times H \times W$	`ConvTranspose2d`($Cs[0]$, 3, kernel_size=2, stride=2, padding=0)
$3 \times H \times W$	`Sigmoid`

Table 6: The hyperparameter configurations of KFS for each setting. "I/C" denotes "Image / Class", where "C" and "D" in C shape, # C, D type, and # D stand for Code and Decoder, respectively. "Low-R" refers to the decoder in Table 4, and "High-R" refers to the decoder in Table 5. $Cs[:]$ denotes the array of channels, and "O-P." means over-parameterization.

Dataset (Shape)	I/C	C Shape	# C	D Type	# D	$D[:]$	O-P. (%)
SVHN ($3 \times 32 \times 32$)	1	$12 \times 4 \times 4$	13	Low-R.	8	$[6, 9, 12]$	0.47%
	10	$12 \times 4 \times 4$	160	Low-R.	12	$[6, 9, 12]$	2.88%
	50	$12 \times 8 \times 8$	200	High-R.	16	$[6, 12]$	0.38%
CIFAR10 ($3 \times 32 \times 32$)	1	$12 \times 4 \times 4$	13	Low-R.	8	$[6, 9, 12]$	0.47%
	10	$12 \times 4 \times 4$	160	Low-R.	12	$[6, 9, 12]$	2.88%
	50	$12 \times 8 \times 8$	200	High-R.	16	$[6, 12]$	0.38%
CIFAR100 ($3 \times 32 \times 32$)	1	$12 \times 4 \times 4$	16	Low-R.	8	$[6, 9, 12]$	1.92%
	10	$12 \times 4 \times 4$	160	Low-R.	12	$[6, 9, 12]$	0.29%
CIFAR100-CL ($3 \times 32 \times 32$)	20	$18 \times 4 \times 4$	200	Low-R.	16	$[8, 13, 18]$	1.32%
TinyImageNet ($3 \times 64 \times 64$)	1	$12 \times 8 \times 8$	16	Low-R.	8	$[6, 9, 12]$	0.24%
	10	$12 \times 16 \times 16$	40	High-R.	16	$[6, 12]$	0.02%

- See Table 4, 5 for detailed implementations of decoders.
- See Table 6 for code shape, the number of code, decoder type, the number of decoder, channel sizes of decoders and corresponding proportion of over-parameterization on each setting.
- We pre-train a single decoder using autoencoding objective for 2,000 steps with 256 mini-batch. We use Adam optimizer (Kingma & Ba, 2014) with constant learning rate of 0.01. After pre-training, the parameter of pre-trained decoder is copied to all the others.
- We train decoders and codes using the full distribution matching objective in Eq. (1) for 20,000 steps. We use Adam optimizer with constant learning rate of 0.01 and 0.1 for decoders and latent codes, respectively.
- For continual learning (CIFAR100-CL), we first train decoders on phase 0 (i.e. the first 20 classes) and fix the decoders for other phases, while the latent codes are trained for each stage. For each stage, we train decoders (phase 0) and latent codes Adam optimizer for 20000 steps, where the initial learning rates are set to 0.01 and 0.1, respectively. The learning rate is decayed for two times by the factor of 0.2 at 5000 and 15000-th iteration.
- We train classification models for 200 epochs with 256 mini-batch on our condensed datasets. We use SGD with momentum optimizer, where the initial learning rate, momentum, and weight decay are set to 0.01, 0.9, and 0.0005, respectively. We decay the learning rate by factor of 0.2 for two times at 133 and 166-th epoch.

B EXPERIMENTAL SETUP (IMAGENET-SUBSET)

We provide some additional information about the experimental setup for reproducing our results on ImageNet-Subset.

Table 7: The architecture of Low-Resolution Decoder in pytorch style. H, W are height and weight of target image. $Cs[:]$ is an array that determines channels of each convolutional block. Here the size of the latent codes are assumed to be $Cs[2] \times H/8 \times W/8$.

Output Size	**Layers**
$Cs[2] \times H/8 \times W/8$	Latent Codes
$Cs[1] \times H/4 \times W/4$	`ConvTranspose2d`($Cs[2]$, $Cs[1]$, kernel_size=4, stride=2, padding=1)
$Cs[0] \times H/2 \times W/2$	`ConvTranspose2d`($Cs[1]$, $Cs[0]$, kernel_size=4, stride=2, padding=1)
$3 \times H \times W$	`ConvTranspose2d`($Cs[0]$, 3, kernel_size=4, stride=2, padding=1)
$3 \times H \times W$	`Sigmoid`

Table 8: The architecture of High-Resolution Decoder in pytorch style. H, W are height and weight of target image. $Cs[:]$ is an array that determines channels of each convolutional block. Here the size of the latent codes are assumed to be $Cs[1] \times H/4 \times W/4$.

Output Size	**Layers**
$Cs[1] \times H/4 \times W/4$	Latent Codes
$Cs[0] \times H/2 \times W/2$	`ConvTranspose2d`($Cs[1]$, $Cs[0]$, kernel_size=4, stride=2, padding=1)
$3 \times H \times W$	`ConvTranspose2d`($Cs[0]$, 3, kernel_size=4, stride=2, padding=1)
$3 \times H \times W$	`Sigmoid`

Table 9: The hyperparameter configurations of KFS for ImageNet-Subset. "I/C" denotes "Image / Class", where "C" and "D" in C shape, # C, D type, and # D stand for Code and Decoder, respectively. "Low-R" refers to the decoder in Table 7, and "High-R" refers to the decoder in Table 8. $Cs[:]$ denotes the array of channels, and "O-P." means over-parameterization.

I/C	C Shape	# C	D Type	# D	$Cs[:]$	O-P. (%)
1	$3 \times 28 \times 28$	64	Low-R.	14	$[3, 3, 3]$	0.41%
10	$6 \times 56 \times 56$	80	High-R.	14	$[4, 6]$	0.05%

- See Table 7, 8 for detailed implementations of decoders.
- See Table 9 for code shape, the number of code, decoder type, the number of decoder, channel sizes of decoders and corresponding proportion of over-parameterization on each setting.
- We pre-train a single decoder using autoencoding objective for 2,000 steps with 256 mini-batch. We use Adam optimizer (Kingma & Ba, 2014) with constant learning rate of 0.01. After pre-training, the parameter of pre-trained decoder is copied to all the others.
- We train decoders and codes using the full distribution matching objective in Eq. (1) for 50,000 steps. We also use Adam optimizer with constant learning rate of 0.001 and 0.01 for decoders and latent codes, respectively. We decay the learning rates by factor of 0.2 for three times at 30000, 42000, and 48000-th iteration.
- We train classification models for 200 epochs on our condensed datasets. The size of mini-batch is set to 128, 64, and 64 for ResNet10AP, ResNet18BN, and EfficientNet, respectively. We use SGD with momentum optimizer, where the initial learning rate, momentum, and weight decay are set to 0.01, 0.9, and 0.0005, respectively. We decay the learning rate by factor of 0.2 for two times at 133 and 166-th epoch.

C WALL-CLOCK TIME ANALYSIS

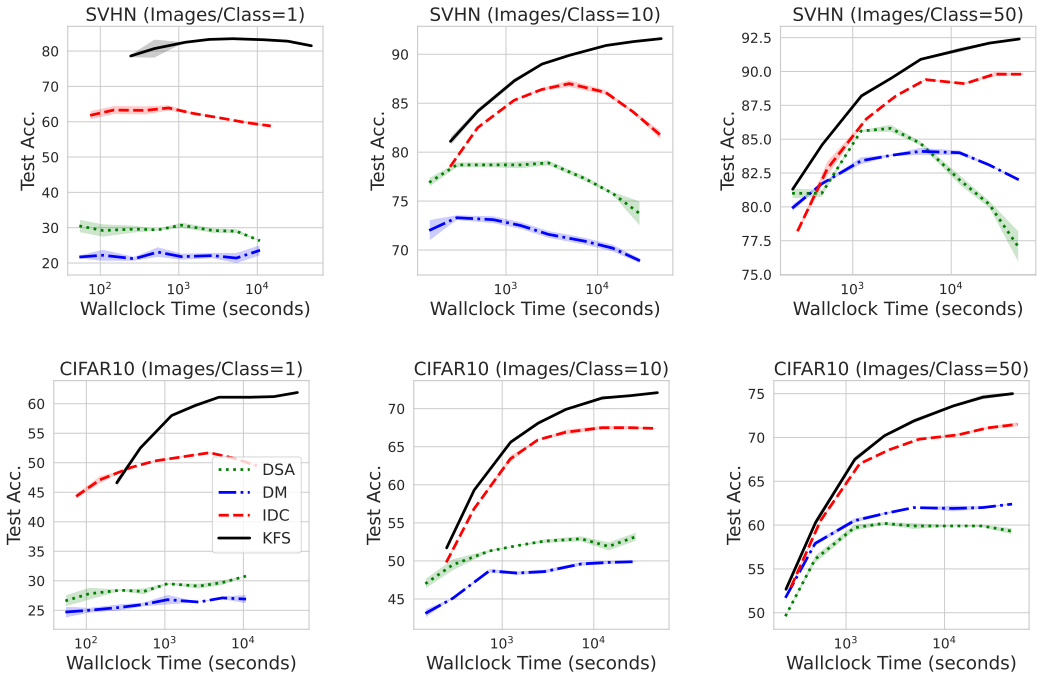


Figure 9: The evaluation results across various amount of wall-clock time for **(Top row)** SVHN and **(Bottom row)** CIFAR10.

D DERIVATION OF EQ. (2)

For notational brevity, let $L_c(\Theta, \phi) := \frac{1}{2}\left\|\frac{1}{N}\sum_{n=1}^{N} g(x_{c,n}) - \frac{1}{DM}\sum_{d=1}^{D}\sum_{m=1}^{M} g(f_{c,m,d})\right\|_2^2$ where $f_{c,m,d} := f(\theta_{c,m}; \phi_d)$. We first compute $\nabla_{\Theta,\phi} L_c(\Theta, \phi)$.

$$
\begin{aligned}
&\nabla_{\Theta,\phi} L_c(\Theta, \phi) \\
&= \nabla_{\Theta,\phi} \frac{1}{2}\left\|\frac{1}{N}\sum_{n=1}^{N} g(x_{c,n}) - \frac{1}{DM}\sum_{d=1}^{D}\sum_{m=1}^{M} g(f_{c,m,d})\right\|_2^2 \\
&= -\frac{1}{NDM}\sum_{n=1}^{N}\sum_{d=1}^{D}\sum_{m=1}^{M} \nabla_{\Theta,\phi} g(f_{c,m,d})^\mathsf{T} g(x_{c,n}) + \frac{1}{D^2M^2}\sum_{d=1}^{D}\sum_{d'=1}^{D}\sum_{m=1}^{M}\sum_{m'=1}^{M} \nabla_{\Theta,\phi} g(f_{c,m',d'})^\mathsf{T} g(f_{c,m,d})
\end{aligned} \tag{4}
$$

On the other hand, suppose we sample $m \sim \text{Uniform}(1, M)$ and $d \sim \text{Uniform}(1, D)$, then

$$
\begin{aligned}
&\mathbb{E}_{m,d}[\nabla_{\Theta,\phi} \hat{L}_c(\Theta, \phi)] \\
&= \mathbb{E}_{m,d}\left[\nabla_{\Theta,\phi} \frac{1}{2}\left\|\frac{1}{N}\sum_{n=1}^{N} g(x_{c,n}) - g(f_{c,m,d})\right\|_2^2\right] \\
&= \mathbb{E}_{m,d}\left[-\frac{1}{N}\sum_{n=1}^{N} g(x_{c,n})^\mathsf{T} \nabla_{\Theta,\phi} g(f_{c,m,d}) + \nabla_{\Theta,\phi} g(f_{c,m,d})^\mathsf{T} g(f_{c,m,d})\right] \\
&= -\frac{1}{NDM}\sum_{n=1}^{N}\sum_{d=1}^{D}\sum_{m=1}^{M} \nabla_{\Theta,\phi} g(f_{c,m,d})^\mathsf{T} g(x_{c,n}) + \frac{1}{DM}\sum_{d=1}^{D}\sum_{m=1}^{M} \nabla_{\Theta,\phi} g(f_{c,m,d})^\mathsf{T} g(f_{c,m,d}))
\end{aligned} \tag{5}
$$

Therefore, subtracting Eq. (4) from Eq. (5), we have

$$
\begin{aligned}
&\mathbb{E}_{m,d}[\nabla_{\Theta,\phi} \hat{L}(\Theta, \phi)] - \nabla_{\Theta,\phi} L(\Theta, \phi) \\
&= \frac{1}{C}\sum_{c=1}^{C}\Big(\mathbb{E}_{m,d}[\nabla_{\Theta,\phi} \hat{L}_c(\Theta, \phi)] - \nabla_{\Theta,\phi} L_c(\Theta, \phi)\Big) \\
&= \nabla_{\Theta,\phi} \frac{1}{C}\sum_{c=1}^{C} \frac{1}{2}\left\{\frac{1}{DM}\sum_{d=1}^{D}\sum_{m=1}^{M} g(f_{c,m,d})^\mathsf{T} g(f_{c,m,d}) - \frac{1}{D^2M^2}\sum_{d=1}^{D}\sum_{d'=1}^{D}\sum_{m=1}^{M}\sum_{m'=1}^{M} g(f_{c,m,d})^\mathsf{T} g(f_{c,m',d'})\right\}.
\end{aligned}
$$

E DERIVATION OF EQ. (3)

We want to compute

$$\begin{aligned}&\mathrm{Var}_n(\nabla_{\Theta,\phi}\tilde{L}(\Theta,\phi))\\&=\mathbb{E}_n\left[\nabla_{\Theta,\phi}\tilde{L}(\Theta,\phi)\nabla_{\Theta,\phi}\tilde{L}(\Theta,\phi)^\mathsf{T}\right]-\mathbb{E}_n\left[\nabla_{\Theta,\phi}\tilde{L}(\Theta,\phi)\right]\mathbb{E}_n\left[\nabla_{\Theta,\phi}\tilde{L}(\Theta,\phi)\right]^\mathsf{T}\end{aligned}$$

Suppose we sample $n\sim\mathrm{Uniform}(1,N)$ for each class c independently. Then we have

$$\begin{aligned}&\mathbb{E}_n\left[\nabla_{\Theta,\phi}\tilde{L}(\Theta,\phi)\right]\\&=\mathbb{E}_n\left[\nabla_{\Theta,\phi}\frac{1}{C}\sum_{c=1}^{C}\frac{1}{2}\left\|g(x_{c,n})-\frac{1}{DM}\sum_{d=1}^{D}\sum_{m=1}^{M}g(f_{c,m,d})\right\|_2^2\right]\\&=\mathbb{E}_n\left[\frac{1}{C}\sum_{c=1}^{C}\left(-\frac{1}{DM}\sum_{d=1}^{D}\sum_{m=1}^{M}\nabla_{\Theta,\phi}g(f_{c,m,d})^\mathsf{T}g(x_{c,n})+\frac{1}{D^2M^2}\sum_{d=1}^{D}\sum_{d'=1}^{D}\sum_{m=1}^{M}\sum_{m'=1}^{M}\nabla_{\Theta,\phi}g(f_{c,m',d'})^\mathsf{T}g(f_{c,m,d})\right)\right]\\&=\frac{1}{C}\sum_{c=1}^{C}\frac{1}{DM}\sum_{d=1}^{D}\sum_{m=1}^{M}\nabla_{\Theta,\phi}g(f_{c,m,d})^\mathsf{T}\left(-\mathbb{E}_n[g(x_{c,n})]+\frac{1}{DM}\sum_{d'=1}^{D}\sum_{m'=1}^{M}g(f_{c,m',d'})\right)\\&=\frac{1}{C}\sum_{c=1}^{C}V_c^\mathsf{T}\left(-\mathbb{E}_n[g(x_{c,n})]+\frac{1}{DM}\sum_{d'=1}^{D}\sum_{m'=1}^{M}g(f_{c,m',d'})\right)\end{aligned}$$

where $V_c:=\frac{1}{DM}\sum_{d=1}^{D}\sum_{m=1}^{M}\nabla_{\Theta,\phi}g(f_{c,m,d})$. Therefore,

$$\begin{aligned}&\mathbb{E}_n\left[\nabla_{\Theta,\phi}\tilde{L}(\Theta,\phi)\right]\mathbb{E}_n\left[\nabla_{\Theta,\phi}\tilde{L}(\Theta,\phi)\right]^\mathsf{T}\\&=\frac{1}{C}\sum_{c=1}^{C}V_c^\mathsf{T}\left(-\mathbb{E}_n[g(x_{c,n})]+\frac{1}{DM}\sum_{d=1}^{D}\sum_{m=1}^{M}g(f_{c,m,d})\right)\cdot\\&\quad\frac{1}{C}\sum_{c'=1}^{C}\left(-\mathbb{E}_n[g(x_{c',n})]+\frac{1}{DM}\sum_{d'=1}^{D}\sum_{m'=1}^{M}g(f_{c',m',d'})\right)^\mathsf{T}V_{c'}\\&=\frac{1}{C^2}\sum_{c=1}^{C}\sum_{c'=1}^{C}V_c^\mathsf{T}\Bigg(\mathbb{E}_n[g(x_{c,n})]\mathbb{E}_n[g(x_{c',n})]^\mathsf{T}-\frac{2}{DM}\sum_{d=1}^{D}\sum_{m=1}^{M}\mathbb{E}_n[g(x_{c,n})]^\mathsf{T}g(f_{c',m,d})\\&\qquad+\frac{1}{D^2M^2}\sum_{d=1}^{D}\sum_{d'=1}^{D}\sum_{m=1}^{M}\sum_{m'=1}^{M}g(f_{c,m,d})^\mathsf{T}g(f_{c',m',d'})\Bigg)V_{c'}\end{aligned}\tag{6}$$

On the other hand, we have

$$\begin{aligned}&\mathbb{E}_n\left[\nabla_{\Theta,\phi}\tilde{L}(\Theta,\phi)\nabla_{\Theta,\phi}\tilde{L}(\Theta,\phi)^\mathsf{T}\right]\\&=\mathbb{E}_n\Bigg[\left\{\frac{1}{C}\sum_{c=1}^{C}\frac{1}{DM}\sum_{d=1}^{D}\sum_{m=1}^{M}\nabla_{\Theta,\phi}g(f_{c,m,d})^\mathsf{T}\left(-g(x_{c,n})+\frac{1}{DM}\sum_{d'=1}^{D}\sum_{m'=1}^{M}g(f_{c,m',d'})\right)\right\}\cdot\\&\qquad\left\{\frac{1}{C}\sum_{c=1}^{C}\frac{1}{DM}\sum_{d=1}^{D}\sum_{m=1}^{M}\nabla_{\Theta,\phi}g(f_{c,m,d})^\mathsf{T}\left(-g(x_{c,n})+\frac{1}{DM}\sum_{d'=1}^{D}\sum_{m'=1}^{M}g(f_{c,m',d'})\right)\right\}^\mathsf{T}\Bigg]\\&=\frac{1}{C^2}\sum_{c=1}^{C}\sum_{c'=1}^{C}V_c^\mathsf{T}\Bigg(\mathbb{E}_n[g(x_{c,n})g(x_{c',n})^\mathsf{T}]-\frac{2}{DM}\sum_{d=1}^{D}\sum_{m=1}^{M}\mathbb{E}_n[g(x_{c,n})]^\mathsf{T}g(f_{c',m,d})\\&\qquad+\frac{1}{D^2M^2}\sum_{d=1}^{D}\sum_{d'=1}^{D}\sum_{m=1}^{M}\sum_{m'=1}^{M}g(f_{c,m,d})^\mathsf{T}g(f_{c',m',d'})\Bigg)V_{c'}\end{aligned}\tag{7}$$

Subtracting Eq. (6) from Eq. (7), we have

$$
\begin{aligned}
&\mathrm{Var}_n(\nabla_{\Theta,\phi}\tilde{L}(\Theta,\phi)) \\
&= \frac{1}{C^2}\sum_{c=1}^{C}\sum_{c'=1}^{C} V_c^{\mathsf{T}}\left(\mathbb{E}_n\left[g(x_{c,n})g(x_{c',n})\right]^{\mathsf{T}} - \mathbb{E}_n[g(x_{c,n})]\mathbb{E}_n[g(x_{c',n})]^{\mathsf{T}}\right)V_{c'} \\
&= \frac{1}{C^2}\sum_{c=1}^{C} V_c^{\mathsf{T}}\left(\mathbb{E}_n[g(x_{c,n})g(x_{c,n})^{\mathsf{T}}] - \mathbb{E}_n[g(x_{c,n})]\mathbb{E}_n[g(x_{c,n})]^{\mathsf{T}}\right)V_c \\
&= \frac{1}{C^2}\sum_{c=1}^{C} V_c\left\{\frac{1}{N}\sum_{n=1}^{N} g(x_{c,n})g(x_{c,n})^{\mathsf{T}} - \left(\frac{1}{N}\sum_{n=1}^{N} g(x_{c,n})\right)\left(\frac{1}{N}\sum_{n=1}^{N} g(x_{c,n})\right)^{\mathsf{T}}\right\}V_c^{\mathsf{T}}
\end{aligned}
$$

because $g(x_{c,n})$ and $g(x_{c',n})$ are independent of each other as we sample n independently for each class.