Dataset Distillation via the Wasserstein Metric

Supplementary Material

The supplementary material is structured as follows:

- Appendix A outlines the potential social impact of our work;
- Appendix B provides a possible theoretical explanation why the Wasserstein metric shows superior performance to MMD in our experiments;
- Appendix C discusses the method for Wasserstein barycenter computation in more detail;
- Appendix D presents our algorithmic framework;
- Appendix E presents our implementation details;
- Appendix F examines the computational efficiency our method;
- Appendix G visualizes how our synthetic data is distributed relative to the real data in feature space.
- Appendix H discusses the increased variety in our synthetic images;
- Appendix I provides more visualization examples.

A Discussion on Potential Social Impact

Our method, focused on accurately matching data distributions, inherently reflects existing biases in the source datasets, potentially leading to automated decisions that may not be completely fair. This situation underscores the importance of actively working to reduce bias in distilled datasets, a critical area for further investigation. Despite this, our technique significantly improves efficiency in model training by reducing data size, potentially lowering energy use and carbon emissions. This not only benefits the environment but also makes AI technologies more accessible to researchers with limited resources. While recognizing the concern of bias, the environmental advantages and the democratization of AI research our method offers are believed to have a greater positive impact.

B Theoretical Explanation on the Superior Performance of the Wasserstein Metric

In this section, we provide a possible theoretical explanation for the observed superior performance of the Wasserstein metric over the MMD metric in our experiments (shown in Fig. 4 of the main paper).

It is important to note that the performance of dataset distillation (DD) methods depends largely on various factors in the algorithmic framework, such as the choice of neural networks or kernels [36], image sampling strategies, loss function design [62], and techniques like factorization [29] and FKD [37]. Additionally, high-resolution datasets, which pose challenges to most existing DD methods, often necessitate trading some precision for computational feasibility in algorithm design. Consequently, we do not aim to assert that the Wasserstein metric is consistently superior as a statistical metric for distribution matching in DD, nor do we believe this to be the case. Instead, we provide a theoretical explanation for the observed superior performance of the Wasserstein metric by combining error bound analysis with practical considerations in DD algorithms, hoping to provide some insights into this phenomenon.

We consider two methods for measuring the discrepancy between the synthetic distribution $\mathbb{Q}$ and the real data distribution $\mathbb{P}$: the Wasserstein distance and the empirical Maximum Mean Discrepancy (MMD). Specifically, we focus on the Wasserstein-1 distance W_1, as it provides a meaningful and tractable metric in our context.

B.1 Setup and Notation

Let $\mathcal{X} \subset \mathbb{R}^d$ denote the input space (assumed to be compact), and $\mathcal{Y} \subset \mathbb{R}$ the label space. Let $\mathbb{P}$ be the real data distribution over $\mathcal{X}$, and $\mathbb{Q}$ the synthetic data distribution over $\mathcal{X}$. Let $f : \mathcal{X} \to \mathcal{Y}$ be the labeling function. We consider a hypothesis class $\mathcal{H}$ of functions $h : \mathcal{X} \to \mathcal{Y}$. The loss function is $\ell : \mathcal{Y} \times \mathcal{Y} \to [0, \infty)$, and we denote the composite loss function as $g(x) = \ell(h(x), f(x))$.

B.2 Assumptions

We make the following assumptions:

- **A1.** The composite loss function $g(x)$ is Lipschitz continuous with respect to x, with Lipschitz constant L:

$$|g(x) - g(x')| = |\ell(h(x), f(x)) - \ell(h(x'), f(x'))| \leq L\|x - x'\|. \tag{16}$$

- **A2.** The input space $\mathcal{X}$ is compact.
- **A3.** The kernel $k(x, x')$ used in MMD calculations is a characteristic kernel. That means, $\mathrm{MMD}_k(\mathbb{P}, \mathbb{Q}) = 0$ implies $\mathbb{P} = \mathbb{Q}$.
- **A4.** The composite loss function $g(x)$ lies in the Reproducing Kernel Hilbert Space (RKHS) $\mathcal{H}_k$ associated with the kernel k, with RKHS norm $\|g\|_{\mathcal{H}_k} < \infty$.

B.3 Theoretical Analysis

Our goal is to bound the difference in expected losses between the real and synthetic distributions:

$$\left|\mathbb{E}_{x\sim\mathbb{P}}[g(x)] - \mathbb{E}_{x\sim\mathbb{Q}}[g(x)]\right|. \tag{17}$$

Bounding Using Wasserstein Distance Under Assumption **A1**, the function $g(x)$ is Lipschitz continuous with constant L. By the definition of the Wasserstein-1 distance W_1:

$$W_1(\mathbb{P}, \mathbb{Q}) = \inf_{\gamma\in\Pi(\mathbb{P},\mathbb{Q})} \mathbb{E}_{(x,x')\sim\gamma}[\|x - x'\|], \tag{18}$$

where $\Pi(\mathbb{P}, \mathbb{Q})$ is the set of all couplings of $\mathbb{P}$ and $\mathbb{Q}$.

Using any coupling $\gamma \in \Pi(\mathbb{P}, \mathbb{Q})$, we have:

$$\begin{aligned}
|\mathbb{E}_{\mathbb{P}}[g(x)] - \mathbb{E}_{\mathbb{Q}}[g(x)]| &= \left|\int_{\mathcal{X}} g(x)\, d\mathbb{P}(x) - \int_{\mathcal{X}} g(x')\, d\mathbb{Q}(x')\right| \\
&= \left|\int_{\mathcal{X}\times\mathcal{X}} (g(x) - g(x'))\; d\gamma(x, x')\right| \\
&\le \int_{\mathcal{X}\times\mathcal{X}} |g(x) - g(x')|\, d\gamma(x, x') \\
&\le L \int_{\mathcal{X}\times\mathcal{X}} \|x - x'\|\, d\gamma(x, x') \\
&= L\mathbb{E}_{(x,x')\sim\gamma}[\|x - x'\|].
\end{aligned} \tag{19}$$

Since this holds for any coupling γ, it holds in particular for the optimal coupling that defines $W_1(\mathbb{P}, \mathbb{Q})$:

$$|\mathbb{E}_{\mathbb{P}}[g(x)] - \mathbb{E}_{\mathbb{Q}}[g(x)]| \le LW_1(\mathbb{P}, \mathbb{Q}). \tag{20}$$

This bound shows that minimizing the Wasserstein-1 distance $W_1(\mathbb{P}, \mathbb{Q})$ directly controls the difference in expected losses via the Lipschitz constant L.

Bounding Using MMD Under Assumption **A4**, the function $g(x)$ lies in the RKHS $\mathcal{H}_k$ associated with the kernel k, with norm $\|g\|_{\mathcal{H}_k}$. The Maximum Mean Discrepancy (MMD) between $\mathbb{P}$ and $\mathbb{Q}$ is defined as [18]:

$$\mathrm{MMD}_k(\mathbb{P}, \mathbb{Q}) = \|\mu_{\mathbb{P}} - \mu_{\mathbb{Q}}\|_{\mathcal{H}_k}, \tag{21}$$

where $\mu_{\mathbb{P}} = \mathbb{E}_{x\sim\mathbb{P}}[k(x, \cdot)]$ is the mean embedding of $\mathbb{P}$ in $\mathcal{H}_k$.

Then, we have:

$$\begin{aligned}
|\mathbb{E}_{\mathbb{P}}[g(x)] - \mathbb{E}_{\mathbb{Q}}[g(x)]| &= |\langle g, \mu_{\mathbb{P}} - \mu_{\mathbb{Q}}\rangle_{\mathcal{H}_k}| \\
&\le \|g\|_{\mathcal{H}_k} \|\mu_{\mathbb{P}} - \mu_{\mathbb{Q}}\|_{\mathcal{H}_k} \\
&= \|g\|_{\mathcal{H}_k} \mathrm{MMD}_k(\mathbb{P}, \mathbb{Q}).
\end{aligned} \tag{22}$$

B.4 Discussion

For a reasonably expressive neural network trained on the compact synthetic data, $\mathbb{E}_{\mathbb{Q}}[g(x)]$ should be close to 0. From Eq. (20) and Eq. (22) we know the key in comparing the error bound for both metrics lies in comparing $LW_1(\mathbb{P}, \mathbb{Q})$ and $\|g\|_{\mathcal{H}_k} \mathrm{MMD}_k(\mathbb{P}, \mathbb{Q})$. When the inputs are raw pixels and h includes a deep neural network, both the Lipschitz constant L and the RKHS norm $\|f\|_{\mathcal{H}_k}$ can be large due to the complexity of h. However, when the inputs are features extracted by an encoder e, which is the case for most DD methods, h can be a simpler function, leading to smaller values for L and $\|g\|_{\mathcal{H}_k}$.

In practice, most existing MMD-based methods [45, 60, 62] approximate distribution matching by aligning only the first-order moment (mean) of the feature distributions. They minimize a loss function of the form:

$$\mathcal{L}_{\text{mean}} = \|\mu_{\mathbb{P}} - \mu_{\mathbb{Q}}\|^2 , \tag{23}$$

where $\mu_{\mathbb{P}} = \frac{1}{N}\sum_i g(\mathbf{x}_i)$ with $\mathbf{x}_i \sim \mathbb{P}$, and $\mu_{\mathbb{Q}} = \frac{1}{M}\sum_j g(\mathbf{s}_j)$ with $\mathbf{s}_j \sim \mathbb{Q}$, are the empirical means of the feature representations from the real and synthetic datasets, respectively.

This mean feature matching is mathematically equivalent to minimizing the MMD with a linear kernel: $k(\mathbf{x}, \mathbf{y}) = \langle \mathbf{x}, \mathbf{y} \rangle$ which simplifies the MMD to:

$$\mathrm{MMD}_k^2(\mathbb{P}, \mathbb{Q}) = \|\mathbb{E}_{\mathbf{x}\sim\mathbb{P}}[\mathbf{x}] - \mathbb{E}_{\mathbf{x}\sim\mathbb{Q}}[\mathbf{x}]\|^2 . \tag{24}$$

However, the linear kernel is generally not characteristic, meaning it cannot uniquely distinguish all probability distributions. As a result, aligning only the means leads to inaccurate distribution matching, neglecting higher-order moments like variance and skewness. This inaccuracy can cause the actual discrepancy between the distributions to remain large, even if the MMD computed with the linear kernel is minimized. Consequently, the inaccurate approximation does not reduce the actual MMD value that would be computed with a characteristic kernel, leaving a significant distributional mismatch unaddressed.

The M3D method [55] improves the precision of MMD-based distribution matching by using a more expressive kernel such as the Gaussian RBF kernel, which effectively captures discrepancies across all moments, with the MMD equation below:

$$\begin{aligned} \mathrm{MMD}_k^2(\mathbb{P}, \mathbb{Q}) =& \mathbb{E}_{\mathbf{x},\mathbf{x}'\sim\mathbb{P}}[k(\mathbf{x}, \mathbf{x}')] + \mathbb{E}_{\mathbf{s},\mathbf{s}'\sim\mathbb{Q}}[k(\mathbf{s}, \mathbf{s}')] && (25) \\ & - 2\mathbb{E}_{\mathbf{x}\sim\mathbb{P},\mathbf{s}\sim\mathbb{Q}}[k(\mathbf{x}, \mathbf{s})]. && (26) \end{aligned}$$

However, this approach introduces sensitivity to the choice of kernel and its parameters, which may be less favorable because an unsuitable kernel may fail to capture important characteristics of the distributions. Moreover, computing the full MMD with a characteristic kernel requires evaluating the kernel function for all pairs of data points, including those from the extensive real dataset, scaling quadratically with dataset size. As a result, this method generally incurs more computational cost compared to earlier methods such as DM and does not scale to large datasets such as ImageNet-1K.

In general, existing MMD-based methods often struggle to achieve precise distribution matching in a way scalable to large datasets. In contrast, the Wasserstein-1 distance inherently accounts for discrepancies in all moments without relying on a kernel function. Its computational feasibility is ensured by the efficient algorithms for Wasserstein barycenter computation and the reduced dimensionality in the feature space. This may explain why, in our experiments, the Wasserstein-1 distance led to better performance than MMD-based approaches that rely on mean feature matching with linear kernels.

C More Explanations on the Method

In this section, we expand our discussion in Sec. 4.2 in more details, to explain how we adapt the method in [8] for efficient computation of the Wasserstein barycenter.

C.1 Optimizing Weights Given Fixed Positions

The optimization of weights given fixed positions in the optimal transport problem involves solving a linear programming (LP) problem, where the primal form seeks the minimal total transportation cost subject to constraints on mass distribution.

Given the cost matrix C and the transport plan T, the primal problem is formulated as:

$$\min_{\mathbf{T}} \langle C, \mathbf{T} \rangle_F \tag{27}$$

$$\text{subject to} \quad \sum_{j=1}^{m} t_{ij} = \frac{1}{n}, \ \forall i, \tag{28}$$

$$\sum_{i=1}^{n} t_{ij} = w_j, \ \forall j, \quad t_{ij} \geq 0, \ \forall i, j, \tag{29}$$

where $\langle \cdot, \cdot \rangle_F$ denotes the Frobenius inner product.

The corresponding dual problem introduces dual variables α_i and β_j, maximizing the objective:

$$\max_{\alpha, \beta} \left\{ \sum_{i=1}^{n} \frac{\alpha_i}{n} + \sum_{j=1}^{m} w_j \beta_j \right\} \tag{30}$$

$$\text{subject to} \quad \alpha_i + \beta_j \leq c_{ij}, \forall i, j. \tag{31}$$

Given the LP's feasibility and boundedness, strong duality holds, confirming that both the primal and dual problems reach the same optimal value [2]. This equivalence implies that the set of optimal dual variables denoted as $\boldsymbol{\beta}$, acts as a subgradient, guiding the weight updates. Specifically, this subgradient indicates how the marginal costs vary with changes in the weights. To update the weights $\mathbf{w}$ towards their optimal values $\mathbf{w}^\star$, we implement the projected subgradient descent technique. This method ensures that $\mathbf{w}$ remains within the probability simplex, and under appropriate conditions on the step sizes, it guarantees convergence to the optimal solution.

C.2 Optimizing Positions Given Fixed Weights

C.2.1 Gradient Computation

Given the cost matrix C with elements $c_{ij} = \|\tilde{\mathbf{x}}_j - \mathbf{x}_i\|^2$, the gradient of the cost function with respect to a synthetic position $\tilde{\mathbf{x}}_j$ is derived from the partial derivatives of c_{ij} with respect to $\tilde{\mathbf{x}}_j$. The gradient of c_{ij} with respect to $\tilde{\mathbf{x}}_j$ is:

$$\nabla_{\tilde{\mathbf{x}}_j} c_{ij} = 2(\tilde{\mathbf{x}}_j - \mathbf{x}_i). \tag{32}$$

However, the overall gradient depends on the transport plan $\mathbf{T}$ that solves the optimal transport problem. The gradient of the cost function f with respect to $\tilde{\mathbf{x}}_j$ takes into account the amount of mass t_{ij} transported from $\tilde{\mathbf{x}}_j$ to $\mathbf{x}_i$:

$$\nabla_{\tilde{\mathbf{x}}_j} f(\tilde{\mathbf{X}}) = \sum_{i=1}^{n} t_{ij} \nabla_{\tilde{\mathbf{x}}_j} c_{ij} = \sum_{i=1}^{n} t_{ij} 2(\tilde{\mathbf{x}}_j - \mathbf{x}_i). \tag{33}$$

C.2.2 Hessian Computation

The Hessian matrix H of f with respect to $\tilde{\mathbf{X}}$ involves second-order partial derivatives. For $p = 2$, the second-order partial derivative of c_{ij} with respect to $\tilde{\mathbf{x}}_j$ is constant:

$$\frac{\partial^2 c_{ij}}{\partial \tilde{\mathbf{x}}_j^2} = 2\mathbf{I}, \tag{34}$$

where $\mathbf{I}$ is the identity matrix. Thus, the Hessian of f with respect to $\tilde{\mathbf{X}}$ for each synthetic point $\tilde{\mathbf{x}}_j$ is:

$$H_j = \sum_{i=1}^{n} t_{ij} 2\mathbf{I} = 2\mathbf{I} \sum_{i=1}^{n} t_{ij} = 2\mathbf{I} w_j, \tag{35}$$

since $\sum_{i=1}^{n} t_{ij} = w_j$, the amount of mass associated with synthetic point $\tilde{\mathbf{x}}_j$.

C.2.3 Newton Update Formula

The Newton update formula for each synthetic position $\tilde{\mathbf{x}}_j$ is then:

$$\tilde{\mathbf{x}}_j^{(\text{new})} = \tilde{\mathbf{x}}_j - H_j^{-1} \nabla_{\tilde{\mathbf{x}}_j} f(\tilde{\mathbf{X}}) \tag{36}$$

$$= \tilde{\mathbf{x}}_j - \frac{1}{2w_j} \sum_{i=1}^{n} t_{ij} 2(\tilde{\mathbf{x}}_j - \mathbf{x}_i). \tag{37}$$

Simplifying, we obtain:

$$\tilde{\mathbf{x}}_j^{(\text{new})} = \tilde{\mathbf{x}}_j - \sum_{i=1}^{n} t_{ij} (\tilde{\mathbf{x}}_j - \mathbf{x}_i)/w_j. \tag{38}$$

This formula adjusts each synthetic position $\tilde{\mathbf{x}}_j$ in the direction that reduces the Wasserstein distance, weighted by the amount of mass transported and normalized by the weight w_j.

D Algorithm details

As discussed in Section 4.3 (Algorithm 1) in the main paper, our method involves computing the Wasserstein barycenter of the empirical distribution of intra-class features. This section details the algorithm employed.

Let us denote the training set as $\mathcal{T} = \{\mathbf{x}_{k,i}\}_{i=1,\ldots,n_k}^{k=1,\ldots,g}$, where g is the number of classes and n_k is the number of images in class k. In the rest of this section, we only discuss the computation for class k, so we omit the index k from the subscript of related symbols for simplicity, e.g., $\mathbf{x}_{k,i}$ is simplified as $\mathbf{x}_i$. A feature extractor $f_e(\cdot)$ embeds the real data of this class into the feature space $\mathbb{R}^{d_f}$, yielding a feature matrix $\mathbf{Z} \in \mathbb{R}^{n_k \times d_f}$, where the ith row $\mathbf{z}_i = f_e(\mathbf{x}_{k,i})$. We employ the algorithm shown in Algorithm 2 to compute the Wasserstein barycenter of the feature distribution. It takes $\mathbf{Z}$ as input and outputs a barycenter matrix $\mathbf{B}^* \in \mathbb{R}^{m_k \times d_f}$, where the jth row $\mathbf{b}_j^*$ is the feature for learning the jth synthetic image, and an associated weight vector (probability distribution) $\mathbf{w}^* \in \mathbb{R}^{m_k}$.

Algorithm 2: Iterative Barycenter Learning for Dataset Distillation

Result: Optimized barycenter matrix $\mathbf{B}^*$ and weights $\mathbf{w}^*$.

Input: Feature matrix of real data $\mathbf{Z} \in \mathbb{R}^{n_k \times d_f}$, initial synthetic dataset positions $\mathbf{B}^{(0)} \in \mathbb{R}^{m_k \times d_f}$, number of iterations K, learning rate η;

Initialize weights $\mathbf{w}^{(0)}$ uniformly;

for $k = 1$ **to** K **do**

`// Optimize weights given positions`

Construct cost matrix $C^{(k)}$ with $\mathbf{B}^{(k-1)}$ and $\mathbf{Z}$;

Solve optimal transport problem to obtain transport plan $\mathbf{T}^{(k)}$ and dual variables $\boldsymbol{\beta}^{(k)}$;

Update weights $\mathbf{w}^{(k)}$ using projected subgradient method: $\mathbf{w}^{(k)} = \text{Project}\left(\mathbf{w}^{(k-1)} - \eta \boldsymbol{\beta}^{(k)}\right)$, ensuring $w_j^{(k)} \geq 0$ and $\sum_j w_j^{(k)} = 1$;

`// Optimize positions given weights`

Compute gradient $\nabla_{\mathbf{B}} f$ as per: $\nabla_{\mathbf{b}_j} f = \sum_{i=1}^{n} t_{ij}^{(k)} 2(\mathbf{b}_j^{(k-1)} - \mathbf{z}_i), \forall j$;

Update positions $\mathbf{B}^{(k)}$ using Newton's method: $\mathbf{b}_j^{(k)} = \mathbf{b}_j^{(k-1)} - H_j^{-1} \nabla_{\mathbf{b}_j} f, \forall j$, where H_j is the Hessian;

end

$\mathbf{B}^* \leftarrow \mathbf{B}^{(K)}$, $\mathbf{w}^* \leftarrow \mathbf{w}^{(K)}$;

E Implementation details

In our experiments, each experiment run was conducted on a single GPU of type A40, A100, or RTX-3090, depending on the availability. We used torchvision [31] for pretraining of models in the squeeze stage, and slightly modified the model architecture to allow tracking of per-class BatchNorm statistics.

We remained most of the hyperparameters in [53] despite a few modifications. In the squeeze stage, we reduced the batch size to 32 for single-GPU training and correspondingly reduced the learning rate to 0.025. In addition, we find from preliminary experiments that the weight decay at the recovery stage is detrimental to the performance of synthetic data, so we set them to 0.

For our loss term in Eq. (15), we set lambda (λ) to 500 for ImageNet, 300 for Tiny-ImageNet, and 10 for ImageNette. We set the number of iterations to 2000 for all datasets. Table 6b-6d shows the hyperparameters used in the recover stage of our method. Hyperparameters in subsequent stages are kept the same as in [53].

config	value
optimizer	SGD
learning rate	0.025
weight decay	1e-4
opti. mom.	0.9
batch size	32
scheduler	cosine decay
train. epoch	100

(a) Squeezing setting for all datasets

config	value
lambda	10
optimizer	Adam
learning rate	0.25
opti. mom.	$\beta_1, \beta_2 = 0.5, 0.9$
batch size	100
scheduler	cosine decay
recover. iter.	2,000

(b) Recovering setting for ImageNette

config	value
lambda	300
optimizer	Adam
learning rate	0.1
opti. mom.	$\beta_1, \beta_2 = 0.5, 0.9$
batch size	100
scheduler	cosine decay
recover. iter.	2,000

(c) Recovering setting for Tiny-ImageNet

config	value
lambda	500
optimizer	Adam
learning rate	0.25
opti. mom.	$\beta_1, \beta_2 = 0.5, 0.9$
batch size	100
scheduler	cosine decay
recover. iter.	2,000

(d) Recovering setting for ImageNet-1K

Table 6. Hyperparameter settings for model training and recovering.

F Efficiency Analysis

Having demonstrated the effectiveness of our approach, we now examine its computational efficiency—a crucial factor for practical deployment. To evaluate the time and memory efficiency of our method, we measured the time used per iteration, total computation time, and the peak GPU consumption of our method with a 3090 GPU on ImageNette in the 1 IPC setting and compared these metrics among several different methods. The results are shown in Table 7. As the Wasserstein barycenter can be computed efficiently, our method only brings minimal additional computation time compared with most efficient methods such as [53]. This makes it possible to preserve the efficiency benefits of the distribution-based method while reaching strong performance.

Method	Time/iter (s)	Peak vRAM (GB)	Total time (s)
DC	2.154 $\pm$ 0.104	11.90	6348.17
DM	1.965 $\pm$ 0.055	9.93	4018.17
SRe^{2}L	0.015 $\pm$ 0.029	**1.14**	**194.90**
WMDD	**0.013** $\pm$ **0.001**	1.22	207.53

Table 7. Distillation time and GPU memory usage on ImageNette using a single GPU (RTX-3090) for all methods. 'Time/iter' indicates the time to update 1 synthetic image per class with a single iteration. This duration is measured in consecutive 100 iterations, and the mean and standard deviation are reported. For a fair comparison, we keep the original image resolution and use the ResNet-18 model to distill 2,000 iterations for all methods.

G Feature Embedding Distribution

To provide intuitive insight into why our method achieves superior performance, we visualize how our synthetic data is distributed relative to the real data in feature space. We train a model from scratch on a mixture of both data to map the real and synthetic data into the same feature space. Then we extract their last-layer features and use the t-SNE [43] method to visualize their distributions on a 2D plane. For comparison, we conduct this process for the synthetic data obtained using our method and the SRe^{2}L [53] method as a baseline. Figure 6 shows the result. In the synthetic images learned by SRe^{2}L, synthetic images within the same class tend to collapse, and those from different classes tend to be far apart. This is probably a result of the cross-entropy loss they used, which optimizes the synthetic images toward maximal output probability from the pre-trained model. In contrast, our utilization of the Wasserstein metric enables synthetic images to better represent the

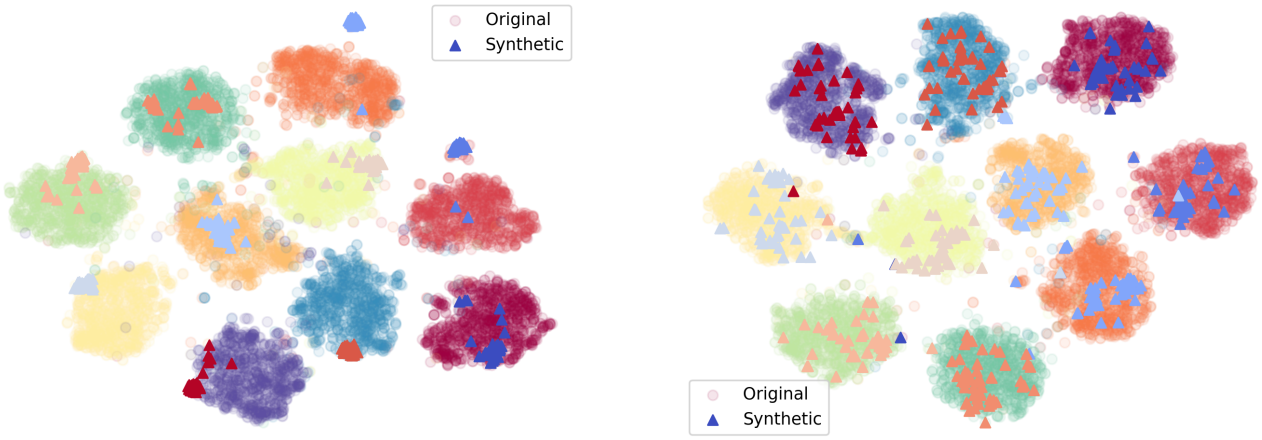
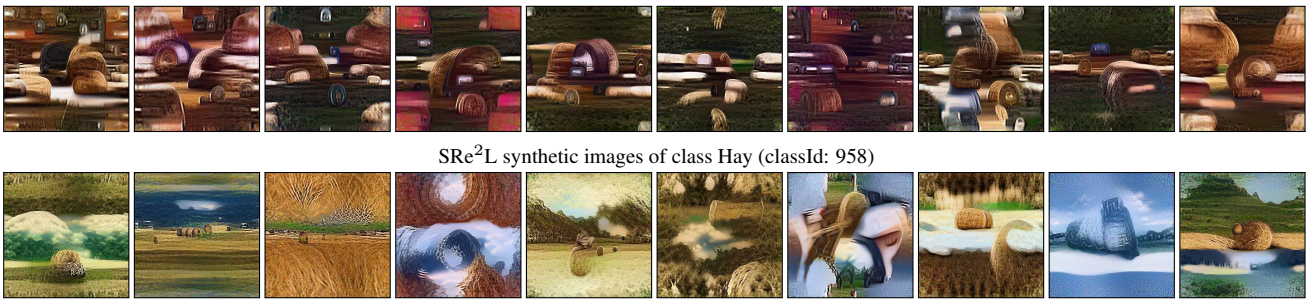


Figure 6. Distribution visualization of ImageNette. The dots present the original dataset's distribution using the model's latent space (e.g., ResNet-101), and the triangles are distilled images. Left: data distilled by SRe^{2}L; Right: data distilled by our method.

distribution of real data, maintaining both intra-class diversity and inter-class relationships that are crucial for effective model training.

H Increased Variety in Synthetic Images

Visualization of the synthetic images at the pixel level corroborates our finding in Appendix G, with the ImageNet-1K Hay class being one such example, as shown in Figure 7. Compared to the SRe^{2}L baseline synthetic images, our method leads to improved variety in both the background and foreground information contained in synthetic images. By covering the variety of images in the real data distribution, our method prevents the model from relying on a specific background color or object layout as heuristics for prediction, thus alleviating the potential overfitting problem and improving the generalization of the model. We provide more visualization on three datasets in Appendix I.



SRe^{2}L synthetic images of class Hay (classId: 958)

Our synthetic images of the class Hay (classId: 958)

Figure 7. Visualizations of our synthetic images vs. SRe^{2}L baseline synthetic images from ImageNet-1K Hay class (classId: 958).

I Visualizations

We provide visualization of synthetic images from our condensed dataset in the supplementary material. In Figure 8, our observations reveal that the synthetic images produced through our methodology exhibit a remarkable level of semantic clarity, successfully capturing the essential attributes and outlines of the intended class. This illustrates that underscores the fact that our approach yields images of superior quality, which incorporate an abundance of semantic details to enhance validation accuracy and exhibit exceptional visual performance.

Additionally, Figure 9 show our synthetic images on smaller datasets. Figure 10 shows the effect of the regularization

strength. In Figure 11, we compare the synthetic data from our method and [53]. It can be seen that our method enables the synthetic images to convey more diverse foreground and background information, which potentially reduces overfitting and improves the generalization of models trained on those images.

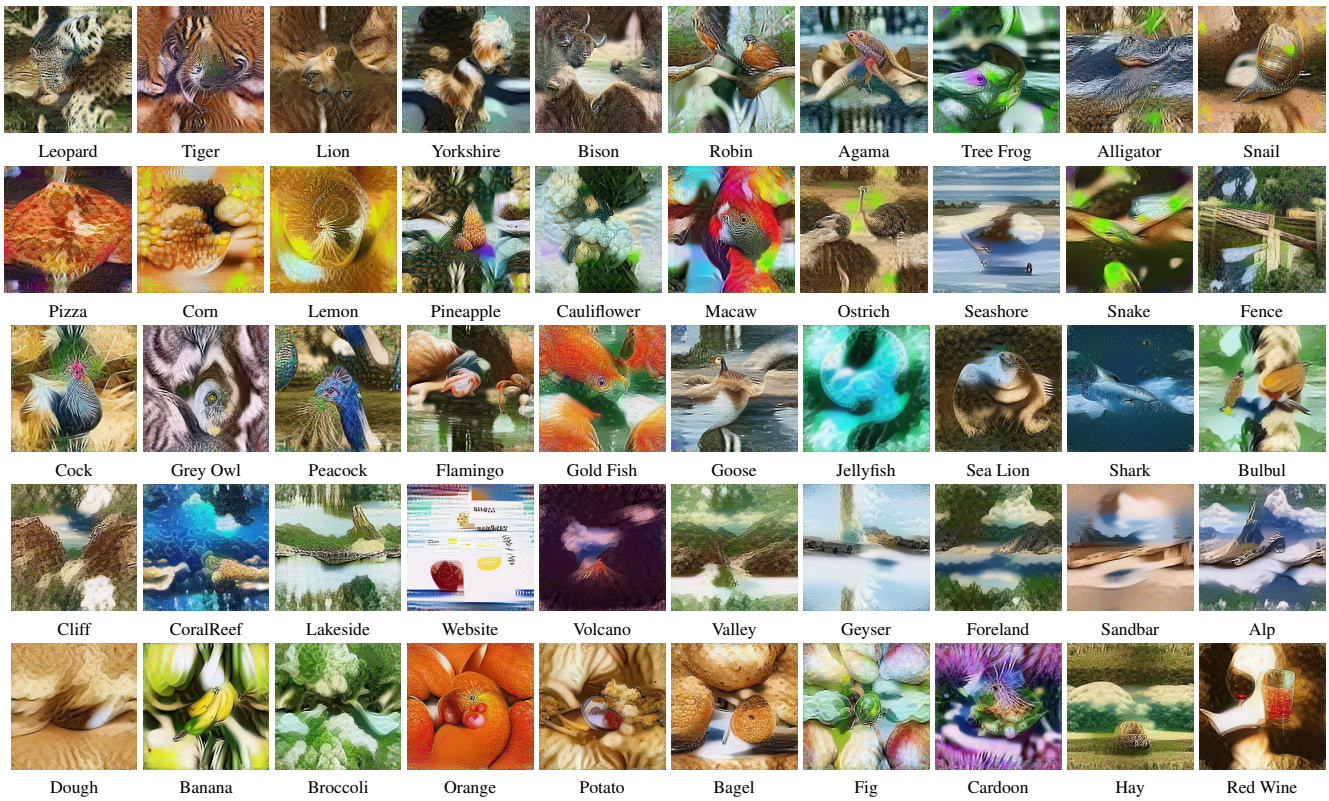


Figure 8. Visualizations of our synthetic images from ImageNet-1K

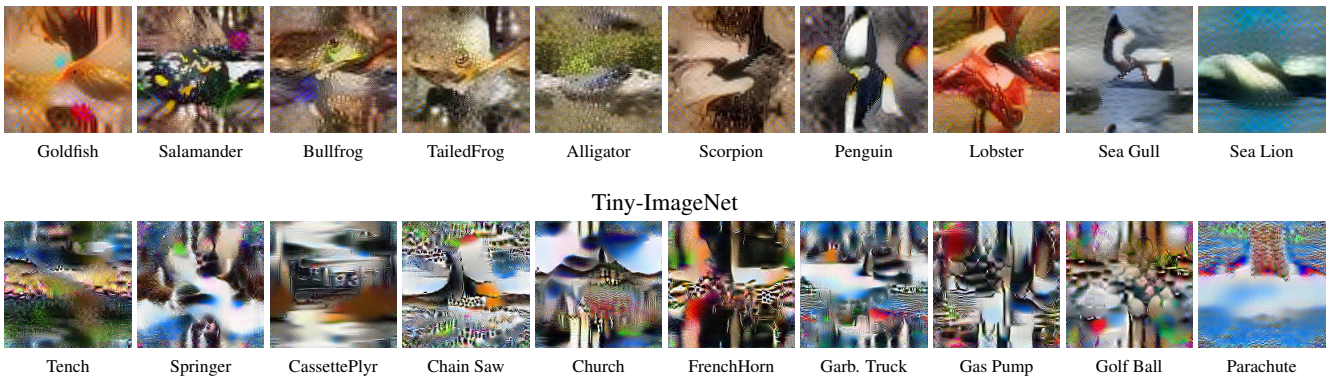


Figure 9. Visualizations of our synthetic images on smaller datasets

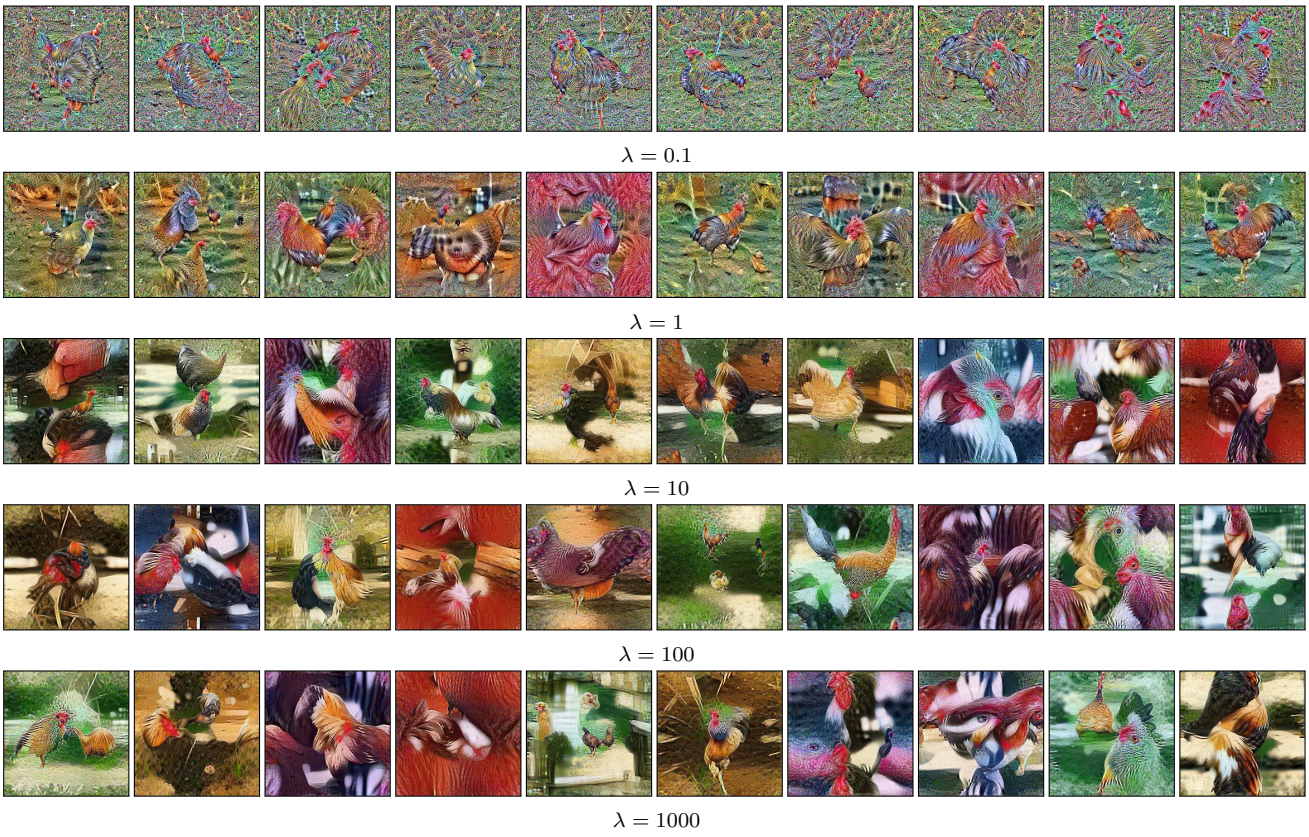
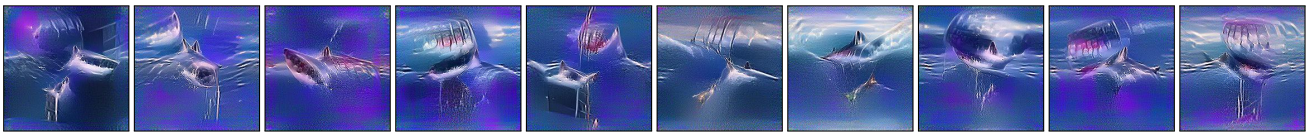
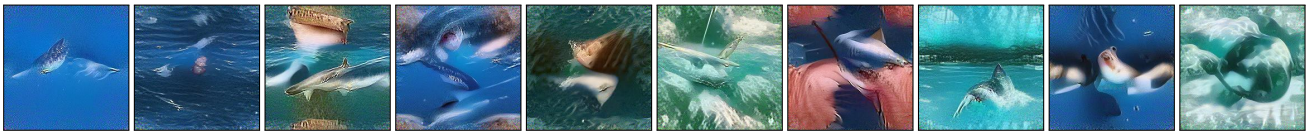


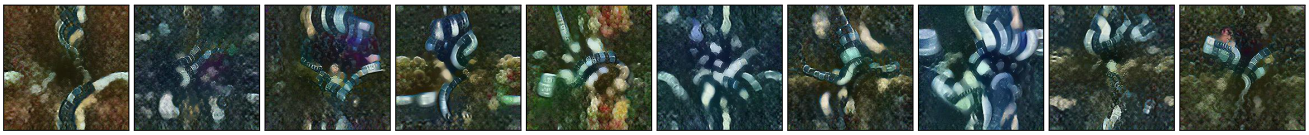
Figure 10. Visualization of synthetic images in ImageNet-1K with different regularization coefficient λ



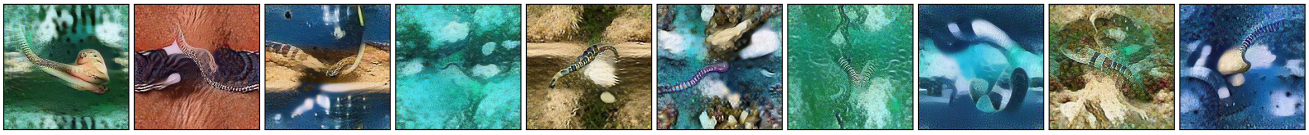
SRe^{2}L synthetic images of class White Shark (classId: 002)



Our synthetic images of the class White Shark (classId: 002)

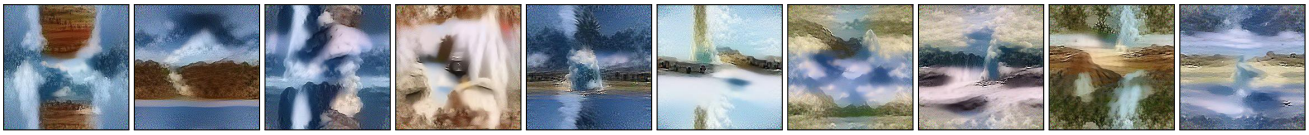


SRe^{2}L synthetic images of class Sea Snake (classId: 065)

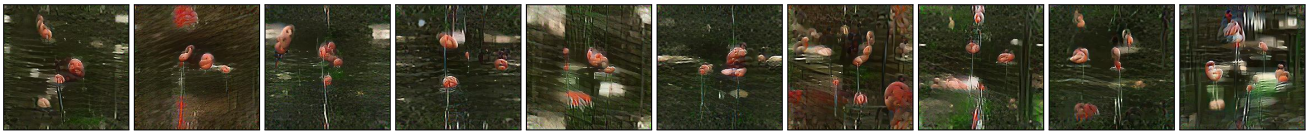


Our synthetic images of the class Sea Snake (classId: 065)

SRe^{2}L synthetic images of class Geyser (classId: 974)



Our synthetic images of the class Geyser (classId: 974)



SRe^{2}L synthetic images of class Flamingo (classId: 130)

Our synthetic images of the class Flamingo (classId: 130)

Figure 11. Comparison of synthetic images obtained from our method vs. SRe^{2}L on ImageNet-1K in 10 IPC setting. Our method yields synthetic images that better cover the diversity of real images within each class.