Mathematisch-
Naturwissenschaftliche
Fakultät

Lernbasierte Computer Vision

Masterarbeit

Generative Dataset Distillation

Eberhard Karls Universität Tübingen
Mathematisch-Naturwissenschaftliche Fakultät
Lernbasierte Computer Vision
Jovan Cicvaric, jovan.cicvaric@student.uni-tuebingen.de, 2023

Bearbeitungszeitraum: von 17.11.2022 bis 17.05.2023

Betreuer/Gutachter: Prof. Dr. Andreas Geiger, Universität Tübingen
Zweitgutachter: Prof. Dr. Philipp Hennig, Universität Tübingen

Selbstständigkeitserklärung

Hiermit versichere ich, dass ich die vorliegende Masterarbeit selbständig und nur mit den angegebenen Hilfsmitteln angefertigt habe und dass alle Stellen, die dem Wortlaut oder dem Sinne nach anderen Werken entnommen sind, durch Angaben von Quellen als Entlehnung kenntlich gemacht worden sind. Diese Masterarbeit wurde in gleicher oder ähnlicher Form in keinem anderen Studiengang als Prüfungsleistung vorgelegt.

Jovan Cicvaric (Matrikelnummer 5678176), May 15, 2023

Abstract

Dataset distillation is a relatively new area of research. The main goal is to create a synthetic dataset which is much smaller than the original dataset. The synthetic dataset should be tailored specifically for fast model training on the target downstream task while keeping the performance on that task close to the actual performance achieved by training on the original dataset. This thesis proposes a new dataset distillation approach - generative dataset distillation. The idea is to utilize a generative network to generate synthetic samples. We find that off-the-shelf GANs are already a strong baseline for dataset distillation on ImageNet and its subsets. Our goal is to create a small generator capable of synthesizing informative training samples using the generator architecture and dataset distillation objective. This method is plug-and-play and can be combined with most of the other dataset distillation approaches. Our method outperforms classical approaches DM and TTM on CIFAR10 but fails to outperform state-of-the-art methods. Furthermore, on larger resolution datasets, such as ImageNet and its subsets, our method fails to be competitive with classical distillation approaches, indicating that additional investigation is necessary.

Acknowledgments

I am grateful to Prof. Dr. Andreas Geiger for his support and insightful feedback throughout my thesis. I express my gratitude to Kashyap Chitta and Axel Sauer for their guidance, contributions and fruitful discussions throughout this work. I also want to express my appreciation and thankfulness to my family, especially my parents Lenka and Lazar for being role models to me and my sisters Eva and Lena for providing unhesitating support. I want to express gratefulness to all my friends that provided large emotional support throughout my studies.

Contents

1 Introduction

The field of Machine Learning achieved great advances which were heavily fueled by new large datasets. These datasets led to large neural networks that significantly improved previous accomplishments in the field.

1.1 Problem Statement

Dataset Distillation, also known as Dataset Condensation, is a method of compressing large datasets into smaller ones while retaining similar performance on the downstream task and allowing for faster training. The key goals are to create a much smaller synthetic dataset, which would contain all the information relevant for the downstream task training and enable much faster training requiring fewer resources.

1.2 Motivation

In recent years there has been a steady increase in dataset sizes in the Machine Learning field. be seen from Figure 1.1 dataset size in terms of data points has been progressively growing with an increase of 0.11 orders of magnitude per year. Until 2016 the largest dataset was ImageNet [DDS+09], which consists of over 1 million images spread across 1 thousand classes. After 2016 there was a significant jump in dataset size and currently the largest widely used computer vision dataset is JFT-3B [ZKHB21] consisting of nearly 3 billion images spread across 30 thousand classes. This signals that the rate of dataset growth in the field will increase. And of course, growth is not limited only to the computer vision field but also can be seen in language, speech and other fields of Machine Learning.

The increase in dataset sizes poses a problem for multiple reasons. First of all, the cost of acquiring the dataset itself grows, and with it also grows the cost of storing the dataset. These two issues already greatly limit access to the newest datasets for many researchers due to the lack of available resources, which in turn can hurt the field and become the next big obstacle for developing new state-of-the-art methods. Another issue is training times, with larger datasets also come larger models and that together increases training times and memory requirements significantly, once again making state-of-the-art research inaccessible to the wider research community. An additional

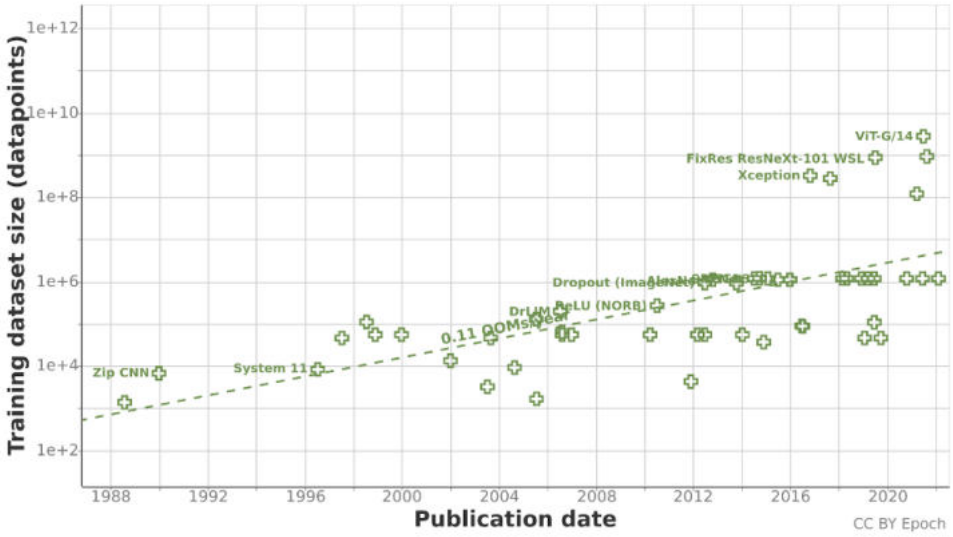


Figure 1.1: **Growth of Computer Vision datasets in recent years.** It can be clearly seen that the size of datasets in terms of datapoints is steadily increasing. Until recently ImageNet was the largest dataset with more than 1 million samples, but after 2016 datasets with more than a billion samples were published and it seems like an upcoming trend. Taken from `https://epochai.org/blog/trends-in-training-dataset-sizes`

concern is increasing CO2 emissions caused by training deep neural networks, it was shown that the large contribution is caused by the length of training [LLSD19]. By creating a smaller synthetic dataset, which would reduce training times, we would effectively reduce the CO2 footprint caused by the Machine Learning field.
In Figure 1.2 a high-level explanation of the dataset distillation can be observed. In this example the MNIST [Den12] dataset consisting of 60000 handwritten digits is distilled into a synthetic dataset containing only one sample per class. Nevertheless, with this small dataset, we are able to achieve 94% accuracy on the original dataset.

But dataset distillation methods come with their own limitations. The main limitation is the fixed small synthetic dataset size that creates a risk of overfitting when training current state-of-the-art models with large number of parameters. This brings our attention to another field that also focuses on generating synthetic samples - generative modelling. There exist many approaches and models in the field, such as DALL-E, Stable diffusion [RBL+22] and StyleGAN-XL [SSG22]. All these models are able to generate an unlimited amount of realistic high-quality images (some examples of generated images can be seen in Figure 1.3). This motivates us to explore the possibilities of using generative models for dataset distillation.

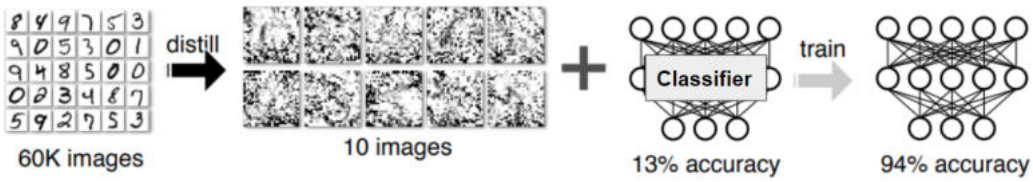


Figure 1.2: **General idea of dataset distillation.** Given a large dataset with 60K samples, we create 10 synthetic images (meaning only 1 image per class) and after using them for training we already get 94% accuracy. Dataset distillation can be used on multiple different datasets and the size of the synthetic dataset can also vary. The usual settings that are considered in the field are 1, 10 and 50 images per class. [WZTE18a]

Figure 1.3: **Examples of corgi images generated by different generated models.** Left is generated using DALL-E model, right is generated using Stable Diffusion.

1.3 Contributions

The contributions of this thesis can be summarized as follows:

- We explore the idea of bridging the gap between dataset distillation and generative modelling fields by introducing the Generative Dataset distillation approach. We evaluate our method on multiple datasets and compare it with baselines from both fields.
- We explore the performance of off-the-shelf StyleGAN-XL for synthetic dataset generation and find it to be a very strong baseline on the ImageNet dataset.
- We investigate and determine the best tactic for sampling images from the generator in order to get the best performance on the downstream task.
- We explore multiple variations of traditional dataset distillation methods and find that using the L1 distance metric can be beneficial for some of them.

- In addition to typical classification benchmarks, we apply dataset distillation to the CarRacing environment by OpenAI [BCP+16], which has not been done before, demonstrating the extremely good distillation performance on a simple dataset, where images of the same class have small variation. We retain more than 80% of the original performance with only 1 image per class, thereby demonstrating the applicability of dataset distillation to simple driving tasks.

2 Related Work

Dataset distillation is a relatively new but rapidly emerging field. In recent years there has been growing interest in researching new methods and scaling distillation methods to higher resolution and more complex datasets. In this section, we will discuss various approaches that have been introduced for tackling the distillation problem, including classical dataset distillation, factorization methods, generative modelling and finally generative attempts at tackling the dataset distillation problem.

2.1 Core-set Selection

Before dataset distillation was introduced, core-set selection methods were used for selecting the most informative samples from the whole dataset and are currently seen as the simplest baselines of dataset distillation.

The most naive approach is using simple Random sampling. With a small sample size, this method performs very poorly, since it does not properly cover the data distribution. On the other hand, it was shown in [ZB23] that as the core set size grows, random sampling performs on par with the dataset distillation methods, which require preliminary training, while random sampling works at no extra cost. More sophisticated core-set selection methods include herding [CWS12], [CMG+18], [BP20] and forgetting [TSdC+18]. Herding is a deterministic process that learns to approximate a probability density function with a collection of samples. It does so by greedily adding samples into the core set so that the mean vector is approaching the whole dataset's mean. The forgetting method is inspired by catastrophic forgetting. It was shown that certain examples are forgotten with higher frequency than others. Using this frequency a significant fraction of the least forgotten samples from the dataset can be omitted without harming the performance. Core-set selection methods are very limited by their nature since they do not compress information from the whole dataset into a smaller subset, they rather try to keep only the most informative samples, which limits the upper bound performance on such selected datasets.

2.2 Classical Dataset Distillation

Authors of the original paper [WZTE18a] introduced dataset distillation as a learning-to-learn problem. The idea is to try and optimize the performance of the model trained

on the synthetic data on the original dataset by updating the synthetic data itself. They explored distilling synthetic datasets for a specific fixed model initialization with only one gradient step look ahead before the performance matching. Since this greatly limits generalization performance algorithm was expanded further to be suitable for random network initializations. In the final iteration, they extend their algorithm to support multiple gradient update steps. This method outperformed all of the core-set selection methods and started the field of dataset distillation.

Although the paper directly uses the dataset distillation objective in its pure form, it creates a bottleneck for scaling up. This objective involves bi-level optimization, which is computationally very heavy. That is why many proxy distillation objectives emerged in the field, such as gradient matching [ZMB20], [ZB21a], [LCJ+22], distribution matching [WZP+22], [ZB23], training trajectory matching [CWT+22] and others.

Gradient matching approaches were the first attempt at using the surrogate objective for distillation. The concept is very simple, we want synthetic and real images to produce the same gradients. These methods still use bi-level optimization, but the performance significantly increased compared to previous work. As it was later shown in [ZB23] gradient matching comes with its own issues. Given a batch of data from the same class, the mean gradient vector with respect to each output neuron in the last layer of a network is equivalent to a weighted mean of features where the weights are proportional to the distance between prediction and ground truth, meaning that the larger weights would be assigned to the samples with false predictions. Effectively these weights would vary for different networks, resulting in possible overfitting to the architecture used during distillation and greatly hurting the cross-architecture generalization.

Distribution matching [ZB23] aimed to solve both - expensive bi-level optimization and architecture overfitting issues. The main idea comes from the core set selection methods where distance metric such as Maximum Mean Discrepancy (MMD) [GBR+12] is often used to select samples. The authors propose matching distributions of real and synthetic data in the lower dimensional feature embedding space. This gives us a simple matching objective that does not require bi-level optimization. Additionally, authors have shown that this method weights all features equally during matching, thus overcoming the issues of gradient matching.

The distribution matching approach helped overcome some of the issues of previous methods, but there was still a lot of space for improvement. Training trajectory matching [CWT+22] at the expense of bi-level optimization significantly increases the performance of the distillation and explicitly makes synthetic data suited for faster training. The objective is to create such synthetic data that would guide the model parameters along the same trajectory in the network parameters space as the real data. And most importantly synthetic data would need fewer gradient update

steps to get to the same point in network parameters space compared to the real data when given the same model initialization. Such an objective is the main reason for the success of the training trajectory matching approach. But bi-level optimization comes at a cost - it is very expensive computationally to scale to higher resolution datasets or to larger synthetic datasets, thus larger limiting the applicability of this objective.

Kernel approaches to the dataset distillation task also proved to be quite successful [ZNB22], [NNXL22], [NCL21a], [NNXL21], [LHAR22], [NCL21b]. These methods achieve good performance and are very efficient due to the existence of closed-form solutions for classifiers that are trained on synthetic data. Some of the kernel methods [ZNB22] optimize distillation even more with the usage of truncated back-propagation through time [Wer90], [TO17] to overcome the limitations of bi-level optimization. Although kernel methods are extremely effective and achieve good performance, they overfit kernel neural networks and most of them still struggle to scale up to larger datasets.

Recently more new matching objectives were introduced. For example, minimizing accumulated trajectory error (FTD) [DJT+23] or Loss curvature matching [SBS+23]. FTD [DJT+23] seeks to mitigate accumulated trajectory error caused by the discrepancy between the distillation and subsequent evaluation and proposes a method that encourages the optimization algorithm to seek a flat trajectory. Loss curvature matching [SBS+23] identifies the worst loss-curvature gap between the original dataset and the synthetic dataset around the local parameter region and minimizes it in order to distil the synthetic dataset with great generalization properties.

In the dataset distillation field mostly image datasets are considered, although some works expand to text data [LL21]. In this work, we will limit ourselves to the image datasets only.

2.3 Factorization

Classical dataset distillation methods achieved good performance but were extremely limited by the small size of synthetic datasets, which might result in overfitting due to the high number of model parameters. Additionally, the samples of each class were independent thus greatly limiting the informativeness of the synthetic data.
The key idea of Factorization methods is to factorize synthetic datasets into latent vectors and decoder networks. Such factorization allows for more generated synthetic samples and consequently higher diversity of the synthetic dataset. By sharing decoder networks across the classes additional information can be condensed under the same memory budget. It is important to note that most of the factorization methods are plug-and-play, meaning that they can be combined with most of the classical

dataset distillation objectives, thus creating many opportunities for experimenting with different combinations.

Authors in [KKO+22] propose a novel view on dataset distillation. Rather than distilling directly synthetic images, they suggest creating condensed data and applying a deterministic multi-formation algorithm. This way they are able to overcome the limited representability of synthetic data, under the same storage limit.

In [LWY+22] it was attempted to factorize the dataset into two components - Bases and Hallucination networks, where the former is fed into the latter in order to produce synthetic samples. Since bases can be combined with any hallucination networks for producing new training samples, exponential informativeness gain is achieved. The authors went further and introduced adversarial constraints which aim at increasing the diversity of generated images and distilling more relevant information into the factorization. The main shortcoming of this method is the additionally introduced generation time, that is spent on the forward pass of Hallucination networks and increased training time caused by the additional adversarial objectives.

In [DR22] authors have made an observation that a shared representation allows for more efficient and effective distillation. They propose learning a set of bases which are shared between classes and combined through learned mapping functions to generate a diverse synthetic dataset. This introduces multiple benefits. One of which is that the size of compressed data does not necessarily grow linearly with the number of classes, thus allowing for more efficient scaling in terms of dataset classes.

Factorization methods made a big step in increasing the performance of the dataset distillation and allowed for easier scaling up. But even though they increased the size of the synthetic dataset under the same memory budget, the size of it is still fixed, thus again limiting the representability of the synthetic data and keeping the risk of overfitting.

2.4 Image Generation

The synthetic image generation field exists for a long time and has achieved significant success by creating high-quality and high-resolution images. But the objective is slightly different, these images are not explicitly tailored for retaining information relative to model training and mostly aim at creating images with high visual fidelity that are indistinguishable from the real images.

2.4.1 GANs

GANs were introduced in 2014 [GPAM+14] and were since the de-facto standard in the image generation field. A large advancement in the field was made by the introduction of probabilistic generative models [LZSE21a], [vdOVK18]. Style-based GANs (StyleGANs) is a specific instance of probabilistic models, exhibiting many desirable properties, such as high image fidelity [KLA19] or fine-grained semantic control [HHLP20]. Recent improvements in the field allowed for application on large and unstructured datasets like ImageNet [SSG22].

GANs consist of generator and discriminator networks. The goal of the generator is to generate synthetic that could be mistaken for real data. The goal of the discriminator is to detect samples that are artificially generated. These two networks are jointly trained, but only the generator network is used during inference to create new synthetic images. Such adversarial training is a two-edged sword, as it allows for training very expressive generators, but makes the optimization process very complicated and unstable.

2.4.2 Diffusion Models

Diffusion-based modelling is a relatively new branch in generative image synthesis [SWMG15]. There has been already a lot of research done in this direction, both for conditional [HSC+21], [NDR+22] and unconditional [ND21], [HJA20] image generation. Some recent studies have shown that in some cases of image generation tasks diffusion models beat GANs [DN21]. Diffusion Models consist of two stages - a forward diffusion process in which images are progressively noised by adding Gaussian noise, and a reverse diffusion process in which the original image is recovered by reversing the noising process. A trained Diffusion Model can be used to generate synthetic data by simply passing randomly sampled noise through the learned denoising pipeline. Diffusion models produce images with cutting-edge quality and do not require adversarial training tactics such as GANs, which are known to cause training instability and introduce other difficulties. Additionally Diffusion model adds the possibility of scalability to a larger resolution.

All these benefits come at a certain price - the image generation process is much slower compared to GANs thus greatly limiting the applicability of diffusion models for on-the-fly synthetic dataset generation.

2.5 Generative Distillation

Our method combines the benefits of all previously introduced approaches. We propose not directly distilling synthetic datasets, but training generative networks that proved to be extremely good at generating realistic synthetic images. We overcome the fixed synthetic dataset size issue of classical dataset distillation by

Classical Dataset Distillation	Generative models	Ours
Fixed synthetic dataset size	Unlimited amount of synthetic images	Unlimited amount of synthetic images
Produces samples informative for downstream task training	Produces high quality images that are not specifically tailored for downstream task	Produces samples informative for downstream task training
Stable training	Unstable GAN training	Stable training using distillation objectives

Figure 2.1: **Comparison between classical dataset distillation, generative models and our approach.** Our approach combines the benefits of both fields and overcomes their downsides. We are able to generate large synthetic datasets, which allows us to overcome the overfitting issue present in classical dataset distillation due to the limited synthetic datasets size. We are able to produce informative training samples and have a stable training due to using distillation objectives for generator training.

leveraging the properties of generative networks. As a training objective, we would use classical dataset distillation objectives, such as distribution matching [ZB23] and training trajectory matching [CWT+22]. Using dataset distillation allows us to train the generator capable of synthesizing informative samples in contrast to the GAN generator, which focuses on creating realistically looking images [SSG22]. Additionally, the distillation objective makes generator training much more stable in comparison to GAN training. By using a single generative network we will be able to share more information across different classes and overcome the overfitting problem, by generating a large synthetic dataset, while still fitting under the memory requirements of the dataset distillation. Thus our method is using the benefits of information sharing introduced by factorization methods [KKO+22], [LWY+22], [DR22] and overcoming overfitting issues. The only previous attempts to merge fields of generative modelling and dataset distillation were conducted in [ZB22] and in concurrent work [CWT+23]. The authors of IT-GAN [ZB22] used pre-trained frozen GANs and distilled latent vectors using dataset distillation objective. In GLaD [CWT+23], the authors also use a pre-trained frozen GAN to extract the learned prior and distil latent vectors. In contrast to [ZB22], they provide generated synthetic samples as the end result of their method. We go a step further and fully train the generator using the distillation objective, which would allow for generating more informative synthetic samples. A summarized comparison between our method, classical dataset distillation and generative modelling can be seen in Figure 2.1.

3 Methods

In this chapter, we describe the technical and methodological details of classical dataset distillation methods such as Distribution Matching (DM) and Training Trajectory Matching (TTM). We go over the StyleGAN generative models, discuss previous attempts at bridging dataset distillation and generative modelling and then finally introduce our method.

3.1 Dataset Distillation

We define the original dataset as D, it consists of $|D|$ training pairs - $D = \{(x_i, y_i)\}_{i=1}^{|D|}$, where x_i denotes a single data sample and y_i denotes the corresponding class label. The goal of dataset distillation is to create a much smaller synthetic dataset $S = \{(\hat{x}_i, \hat{y}_i)\}_{i=1}^{|S|}$, $|S| << |D|$, such that the performance of the network trained on that synthetic dataset S matches the performance of the network trained on the original dataset D. This neural network is parametrized by network parameters θ and denoted as N_θ. In order to learn the optimal network parameters θ^*, loss term L over training dataset D needs to be minimized:

$$\theta^* = \underset{\theta}{argmin}\, L(D, \theta) \tag{3.1}$$

L can be defined as:

$$L(D, \theta) = \frac{1}{|D|} \sum_{(x_i, y_i) \in D} l(N_\theta(x_i), y_i), \tag{3.2}$$

where l is a task-specific loss, in our case it is cross-entropy loss. Cross-entropy loss is usually defined as

$$L_{CE} = -\sum_{i=1}^{|D|} y_i \log(p_i), \tag{3.3}$$

where y_i are ground truth class labels and p_i is the output of the neural network for the input sample x_i.

The goal of dataset distillation is that the performance of the model trained on S is equal to the performance of the model trained on D, this can be formally defined as:

$$E_{(x,y)\sim P_D}[l(N_{\theta^D}, y)] = E_{(x,y)\sim P_D}[l(N_{\theta^S}, y)] \tag{3.4}$$

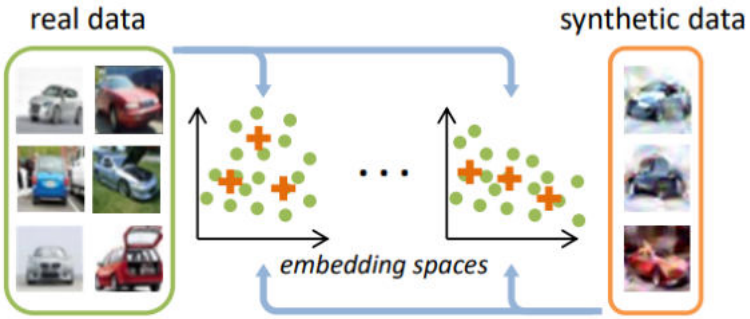


Figure 3.1: **Illustration of Distribution Matching approach.** Randomly sampled real and synthetic data is embedded with the randomly initialized deep neural networks. The synthetic data is learned by minimizing the distribution discrepancy between real and synthetic data in embedding spaces. [ZB23]

where P_D is real data distribution, θ^D are parameters of the model trained on the real dataset D and θ^S are parameters of the model trained on the synthetic dataset S.

The original dataset distillation [WZTE18a] work formulates distillation as learning-to-learn problem:

$$S^* = \underset{S}{argmin}\, L(D, \theta^S) \tag{3.5}$$

$$\text{subject to } \theta^S = \underset{\theta}{argmin}\, L(S, \theta) \tag{3.6}$$

Solving this objective directly is a very challenging task, requiring bi-level optimization and significant amounts of computation. To overcome this limitation, many surrogate objectives emerged in the field.

3.2 Distribution Matching

The distribution matching objective was introduced in [ZB23]. The main idea is to create a synthetic dataset that accurately approximates the distribution of the original dataset. Image dimensionality is too high to be directly used for matching. For example, the CIFAR10 [KH+09] dataset consists of 32x32 dimensional images with 3 channels, which results in a total dimension of 3072. Matching such large vectors can be very computationally challenging. To account for that, authors propose using lower dimensional image embedding. They assume that there exists such a function $F_\theta : R^D -> R^E$, where θ are the function parameters, D is the dimensionality of a flattened image and E is the dimensionality of the target embedding space, which is much smaller than D. This function maps every image into lower-dimensional embedding space. A high-level illustration of the distribution matching objective

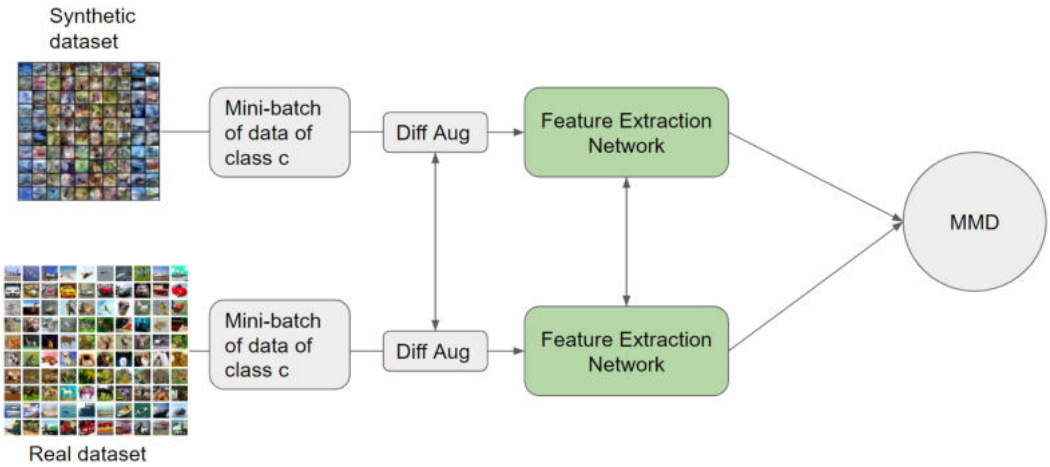


Figure 3.2: **High-level illustration of distribution matching approach.** The same differentiable augmentations are applied to mini-batches of synthetic and real data of class c. Augmented images are fed into a feature extraction network, which is in our case randomly initialized Convolutional neural network. It is important to note that the same extraction network is used for real and synthetic data. Afterwards, extracted features get matched using MMD and gradients get backpropagated to update the synthetic data.

can be seen in Figure 3.1. For the image $x \in R^D$ we denote the respective embedding as $F_\theta(x) \in R^E$. With this, the distribution matching objective can be formally defined. In the original paper Maximum Mean Discrepancy (MMD) [GBR+12] was used to estimate the distance between real and synthetic data in the embedding space, this was largely inspired by core set selection methods that have been using this distance metric. The distance between real and synthetic distribution can be computed as follows:

$$\sup_{\|F_\theta\|_H \leq 1} (E[F_\theta(D)] - E[F_\theta(S)]) \tag{3.7}$$

where H is reproducing kernel Hilbert space. Since this formula assumes access to the original data distribution P_D, which we do not have access to, the empirical estimate of MMD is used in practice:

$$E_{\theta \sim P_\theta} \| \frac{1}{|D|} \sum_{i=1}^{|D|} F_\theta(x_i) - \frac{1}{|S|} \sum_{i=1}^{|S|} F_\theta(\hat{x}_i) \|^2 \tag{3.8}$$

where P_θ is the distribution of embedding network parameters. The authors further follow dataset condensation with differentiable siamese augmentation (DSA) [ZB21b] and apply differentiable siamese augmentations $A(\cdot, w)$, where w is the parametrization of the augmentation function such as rotation degree randomly

sampled from parameter distribution P_w for every training mini-batch, both to the original data D and the synthetic data S. It is important to note that the same augmentation is applied both to mini-batches of real data sampled during the distillation process and to synthetic data. It was shown that this greatly benefits synthetic data by allowing it to learn prior knowledge about the spatial configuration of original samples. With this addition, the final matching objective is described as follows:

$$\min_S E_{\theta \sim P_\theta, w \sim P_w} \| \frac{1}{|D|} \sum_{i=1}^{|D|} F_\theta(A(x_i, w)) - \frac{1}{|S|} \sum_{i=1}^{|S|} F_\theta(A(\hat{x}_i, w)) \|^2 \tag{3.9}$$

This way synthetic data is learned by minimizing the discrepancy between the distribution of the real data embeddings and synthetic data embeddings in the feature space by sampling model parameters θ. Note that as a downstream task image classification problem was selected, the discrepancy between the real and synthetic samples of the same class only is minimized. This formulation limits the Distribution Matching approach only to classification tasks and serves as a barrier towards extending it to regression problems. However, an advantage of this approach is efficiency, as we optimize only synthetic data S and not the model parameters, thus avoiding expensive bi-level optimization like in other works [CWT+22], [WZTE18b]. In Figure 3.3 we can see the TSNE plot with original data samples and synthetic data samples produced by DM. We can clearly see that DM produces synthetic images that fully cover the distribution of the original data, thus creating a synthetic dataset that represents the original data.

Illustration of distribution matching distillation pipeline is illustrated in Figure 3.2 after computing the MMD loss gradients are backpropagated to update the synthetic dataset. A detailed distribution matching algorithm can be seen in Algorithm 1.

Algorithm 1: Dataset Distillation with Distribution matching

Data: Training set D

Input: Deep neural network F_θ parameterized with θ, the probability distribution over parameters P_θ, differentiable augmentation $A(\cdot, w)$ parameterized with w, augmentation parameter distribution P_W, number of training iterations K, learning rate η.

Initialize using random real images $S \sim D$;

for $i = 0, ..., K-1$ **do**

- Sample $\theta \sim P_\theta$;
- Sample mini-batch pairs $B_c^D \sim D$ and $B_c^S \sim S$ and $w \sim P_w$ for every class c ;
- Compute loss: $L = \sum_{c=0}^{C-1} \| \frac{1}{|B_c^D|} \sum_{x \in B_c^D} F_\theta(A(x, w)) - \frac{1}{|B_c^S|} \sum_{\hat{x} \in B_c^S} F_\theta(A(\hat{x}, w)) \|^2$;
- Update $S \leftarrow S - \eta \nabla_S L$;

end

Result: Synthetic set S

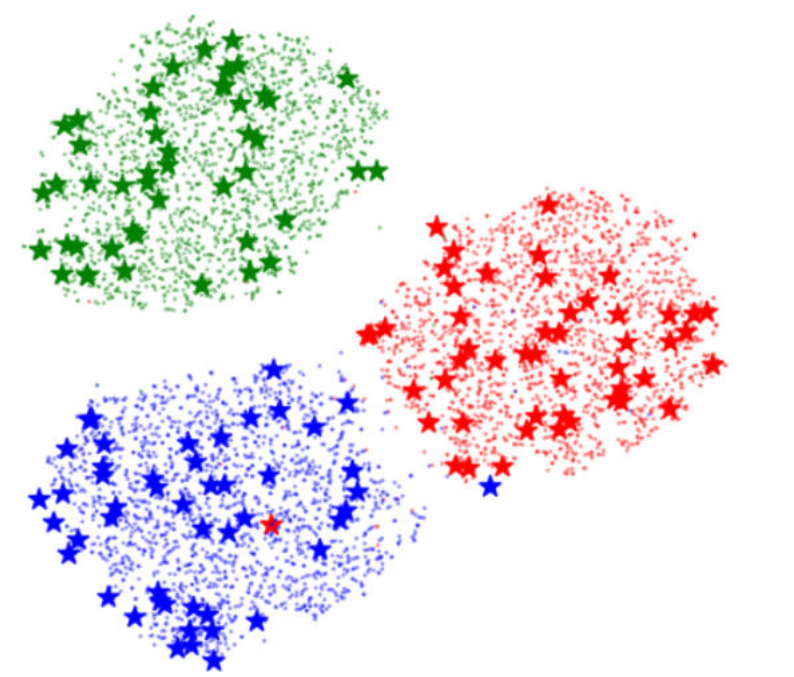


Figure 3.3: **Distribution of synthetic images learned by DM**. The red, green and blue points are the real images of the CIFAR10 first three classes. The stars are corresponding learned synthetic images. It can be seen that the learned synthetic data fairly covers the distribution of real samples. [ZB23]

Examples of synthetic datasets produced by distribution matching for MNIST and CIFAR10 are shown in Figure 3.4

3.2.1 Mapping Functions

For feature extraction (the embedding functions F_θ) deep neural networks are used. The authors have shown that there is no need to use pre-trained feature extraction networks - randomly initialized convolutional neural networks can be used. Some previous works demonstrated that randomly initialized convolutional neural networks already provide strong features for multiple computer vision tasks [SKC+11, CWMG18, AAKW22] and additionally, such networks are shown to perform a distance-preserving mapping of the data - smaller distances between samples of the same class and larger distances across samples of different classes [GSB16]. The authors went further to confirm this and compared DM results using random and pre-trained networks. Results have confirmed the previous assumption - performance is not affected by this choice. Thus using randomly initialized networks saves a lot of computing while producing meaningful representations.

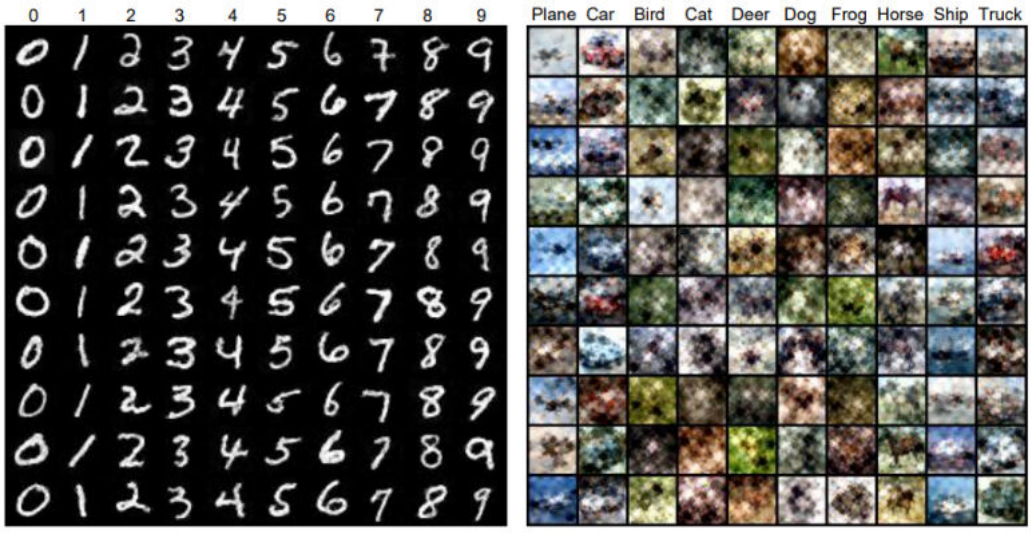


Figure 3.4: Examples of synthetic images generated by DM for MNIST (left) and CIFAR10 (right) datasets. In this example synthetic dataset consists of 10 images of every class

3.2.2 Connection to Gradient Matching

Gradient matching approach [ZB21c] also samples real and synthetic image mini-batches for each class, but rather than matching distributions it aims at matching mean gradients produced by the networks trained on real and synthetic datasets. The authors of the distribution matching paper have shown that given a batch of data from the same class, the mean gradient vector is equivalent to a weighted mean of features where the weights are proportional to the distance between prediction and ground truth. Thus while DM weighs each feature equally, Gradient matching assigns larger weights to the samples whose predictions are inaccurate. This is problematic because different network architectures produce different gradients for different samples, thus produced synthetic dataset will overfit to the network architecture it was distilled with and the performance with the other network architectures may heavily suffer.

3.2.3 L1 Matching Objective

In this work, we have also experimented with different matching objectives. We used additional L1 variations of MMD. It was previously shown that the L2 distance becomes less meaningful in higher dimensional spaces [AHK01]. The authors of that paper have shown that for high dimensional spaces L1 distance is more suitable than Euclidean L2 distance. This is mostly due to the curse of dimensionality - in high dimensional spaces data becomes more sparse. Thus the concept of proximity becomes less meaningful as the dimensionality grows. Let us formally define the

introduced variation of the MMD objective

$$\min_{S} E_{\theta \sim P_\theta, w \sim P_w} \left| \frac{1}{|D|} \sum_{i=1}^{|D|} F_\theta(A(x_i, w)) - \frac{1}{|S|} \sum_{i=1}^{|S|} F_\theta(A(\hat{x}_i, w)) \right| \tag{3.10}$$

3.2.4 Synthetic Dataset Initialization

In [CWT+22] different synthetic data initializations were explored. The authors compare the performance of the synthetic datasets initialized as the random real images and Gaussian noise. Results indicated that even with Gaussian noise initialization solid performance can be achieved, but the performance will still be worse when compared to the real image initialization and the distillation process might be more unstable and require longer distillation times. We go further and explore different tactics for selecting initial real images and their effect on the final performance. We explore two approaches - Entropy based and Clustering based.

Entropy is a metric used to quantify the uncertainty over variable values. Given a discrete random variable Y with underlying distribution P_Y we define entropy as follows:

$$H(Y) = -\sum_{y \in Y} P_Y(y) log P_Y(y) \tag{3.11}$$

In order to select images we will first train the Neural network on the full real dataset, then for every image we will take the output of the trained network and compute the entropy. We explore two approaches - selecting images with larger entropies or with the smallest. In the first case, synthetic images will already contain the samples that the classifier finds the hardest to correctly classify, thus in theory increasing the overall performance of the synthetic dataset. In the second case, synthetic images will be initialized with the easiest-to-classify samples, arguably the advantage here is that the easiest-to-classify samples are already covered by this initialization and the distillation process will be focusing more on adding information relevant to harder samples and might effectively speed up the distillation process.

For clustering-based selection, we will use K-Means clustering. Given a set of samples $(z_1, z_2, ..., z_n)$ K-means clustering algorithm aims at partition these samples into K clusters $\mathbf{Z} = (Z_1, ..., Z_K)$ by minimizing within-cluster sum of squares (WCSS). The objective is formally defined as follows:

$$\underset{\mathbf{Z}}{argmin} \sum_{i=1}^{K} \sum_{z \in Z_i} ||x - \mu_i||^2 \tag{3.12}$$

where μ_i is the centroid of the respective cluster Z_i $\mu_i = \frac{1}{|Z_i|} \sum_{z \in Z_i} z$. In this case, the first step is the same as in the case with Entropy, we train a neural network on the

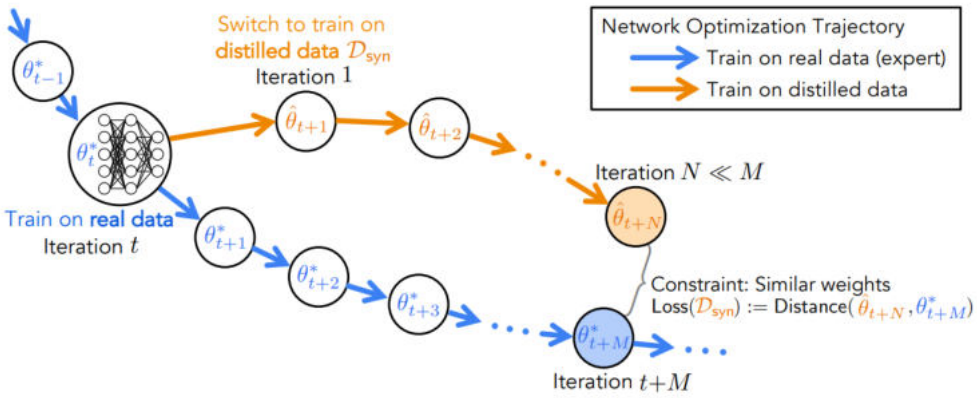


Figure 3.5: **Illustration of the main idea of the Training Trajectory Matching approach.** Long-range parameter matching between network parameters trained on synthetic data and network parameters trained on real data is performed during distillation. Synthetic dataset D_{syn} is distilled in such a way that N training steps on it match the same result (in network parameters space) from much more M steps on real data D. [CWT+22]

full real dataset, but then rather than taking final network outputs, we use feature embeddings produced by the network. Then for each class, we do K-means clustering on the feature embeddings of respective images. The number of clusters reflects the images per class distillation setting. After clusterization, we select samples closest to centroids in the feature space for initialization. In the simplest case of one image per class, the centroid would be equivalent to the class mean. Additionally, we can take the samples furthest away from the centroid for initialization, as they should reflect the samples that are the hardest for the classifier to recognize.

3.3 Training Trajectory Matching

Training Trajectory Matching (TTM) [CWT+22] gives a totally different view on the objective of the distillation process. In this paper, two main criteria for the synthetic data were introduced. Firstly, it should guide the parameters on the network trained on synthetic data along the same trajectory as network parameters trained on the real data. And second - training for the downstream task on the synthetic dataset should be much faster, compared to the training on the large original dataset, this is done by the distillation objective which explicitly induces synthetic data to be tailored for faster training for the downstream task. The training trajectory matching approach is demonstrated in Figure 3.5.

In order to match training trajectories we first need to create a buffer of network

parameters trained on real data. This is one of the downsides of the TTM approach - training networks might take a considerable amount of time and computing and storing buffers might require a lot of additional memory.

Let us formally define expert trajectories of model parameters as τ^*. The single expert trajectory is the time sequence of model parameters $\{\theta_i^*\}_{i=0}^T$ obtained during training the model on the original dataset D. In order to generate these trajectories, multiple models with the same architecture (in our case ConvNet) but with different initializations are trained on the real dataset and a snapshot of their parameters is saved at every training epoch. The name expert trajectories indicate that these trajectories represent the theoretical upper bound for the synthetic dataset. Similarly, student parameters $\hat{\theta}_t$ are defined as the network parameters trained on synthetic images at the training step t. The goal of this approach is to create a synthetic dataset that given the same initial point in the model parameters spaces will induce the same parameter trajectory as the trajectory induced by the real data from the same initial point. So during the distillation process student network $\hat{\theta}_t$ is initialized with randomly sampled expert parameters θ_t^*. Then the student network is updated using the synthetic data S for N steps. N is a hyperparameter which is experimentally chosen. After N training steps on synthetic data, student network parameters $\hat{\theta}_{t+N}$ are matched with expert parameters from the same trajectory after M steps: θ_{t+M}^*. M is also a hyperparameter that is experimentally selected, it is important that $M >> N$, this way we implicitly force synthetic data to be suited for faster model training - the model does fewer training steps on synthetic data, but has to guide model parameters to the same point in the parameters space as the real data with much more update steps. Example of CIFAR10 synthetic datasets produced by TTM can be seen in Figure 3.6.

3.3.1 Loss

To match student and expert parameters normalized L2 Loss is used:

$$L = \frac{||\hat{\theta}_{t+N} - \theta_{t+M}^*||^2}{||\theta_t^* - \theta_{t+M}^*||^2} \tag{3.13}$$

L2 loss is normalized by the distance travelled by expert parameters. This ensures that even during later training epochs, where parameters do not change significantly, we get a strong signal. Additionally, the L1 variation of the loss can be used:

$$L = \frac{||\hat{\theta}_{t+N} - \theta_{t+M}^*||_1}{||\theta_t^* - \theta_{t+M}^*||_1} \tag{3.14}$$

The authors of TTM additionally tried using cosine distance metric or matching output logits between expert and student networks, but simple L2 loss with matching model parameters empirically proved to be the best choice for distillation.

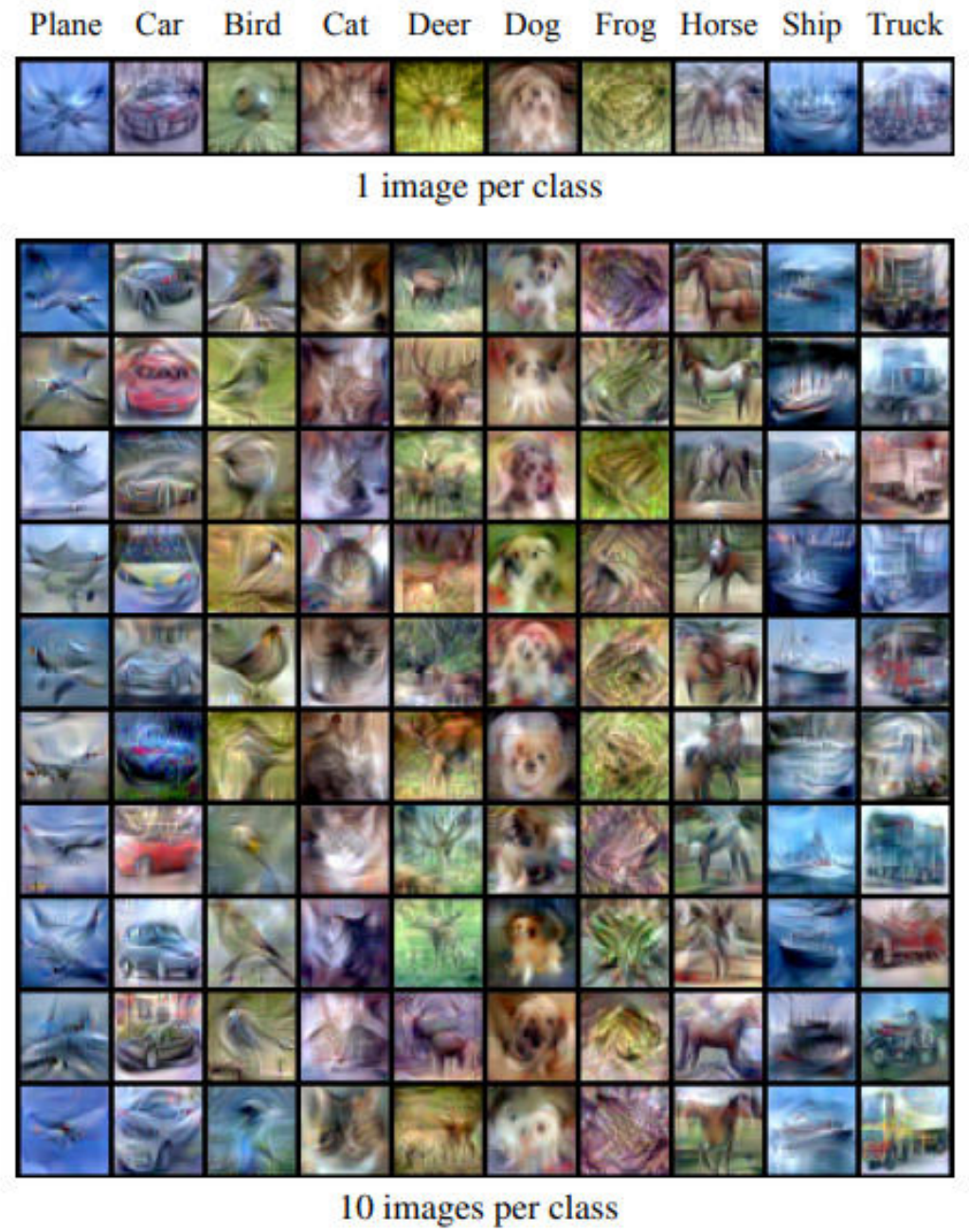


Figure 3.6: **Examples of images generated by TTM for CIFAR10 dataset.** Examples of two distillation cases can be seen here - 1 image per class (above) and 10 images per class (below). It can be seen that compared to images generated by DM, TTM produces synthetic samples that are much closer visually to the original data. [CWT+22]

After computing the loss gradients need to be backpropagated through all N training steps on the synthetic data and only then it can be updated. This makes TTM a bi-level optimization problem, which is extremely costly and the main limitation towards scaling TTM to a higher resolution or more significant number of classes.

3.3.2 Maximal Starting Epoch

The authors of the original work have found that it is important to put an upper bound on the possible starting epoch of the student network. This upper bound is denoted as T^+. They discovered that if the upper bound is too high, the synthetic data receives gradients from points where the expert's parameters movements are small and uninformative. In the opposite case, if it is too low, the synthetic data is missing out on a significant portion of the later training dynamics.

3.3.3 Learnable Learning Rate

Along with updating the synthetic dataset, the learning rate α that is used for downstream task training is updated. Having a learnable learning rate α automatically adjusts for the number of student N and expert M updates. Thus making the optimization process easier and less dependent on the selected hyperparameters.

3.3.4 Mini-batching Synthetic Data

To overcome the limitations of TTM and create synthetic datasets with more samples or/and higher resolutions authors tried using mini-batches of synthetic images. A new mini-batch is sampled for every update of the student network. This way all synthetic images will have been seen by the time of the final loss computation. Mini-batch B^S contains images from multiple classes, just fewer of them. Full TTM algorithm can be found in Algorithm 2.

3.3.5 Parameter Pruning

Parameter pruning as an addition to TTM was introduced in [LTOH22]. The idea is to prune hard-to-match model parameters in the expert and student networks. Whether they are hard to match is determined by similarity metric (L2 or L1 distance) and pre-set threshold. The authors argue that by pruning hard-to-match parameters in the expert and student networks, we can effectively avoid the influence of these parameters on the synthetic data which as a result can improve the performance of the synthetic datasets and cross-architecture generalization.

Algorithm 2: Dataset Distillation with Training Trajectory matching

Data: Training set D

Input: Set of expert trajectories $\{\tau_i^*\}$, differentiable augmentation $A(\cdot, w)$ parameterized with w, augmentation parameter distribution P_W, number of updates on synthetic data N, number of updates on real data M, number of training iterations K, learning rate η, maximum starting epoch T^+.

Initialize with randomly selected real images $S \sim D$;

Initialize trainable learning rate α for training on S ;

for $i = 0, ..., K-1$ **do**

  Sample expert trajectory: $\tau^* \sim \{\tau_i^*\}$ with $\tau^* = \{\theta_t^*\}_{t=0}^T$;

  Choose random starting epoch, $t \leq T^+$;

  Initialize student network with expert parameters: $\hat{\theta}_t = \theta_t^*$;

  for $n = 0, ..., N-1$ **do**

    Sample a mini-batch of synthetic images $B^S \sim S$;

    Sample augmentation parameters $w \sim P_w$;

    Update student network w.r.t. classification loss: ;

    $\hat{\theta}_{t+n+1} \leftarrow \hat{\theta}_{t+n} - \alpha \nabla l(A(B^S, w), \hat{\theta}_{t+n})$

  end

  Compute loss: $L = \frac{\|\hat{\theta}_{t+N} - \theta_{t+M}^*\|_2^2}{\|\theta_t^* - \theta_{t+M}^*\|_2^2}$;

  Update $S \leftarrow S - \eta \nabla_S L$;

  Update $\alpha \leftarrow \alpha - \eta \nabla_\alpha L$;

end

Result: Synthetic set S, learning rate α

3.3.6 Extending to Regression

One of the advantages of the TTM is that it is not limited to classification tasks and can be extended to regression. Of course, this introduces much more complexity and comes with its own issues, but could be an interesting direction for future research in the field. Some other approaches already tried distilling soft labels [ZNB22] and have shown that it can improve the performance, but it introduces additional memory requirements since soft labels also need to be accounted for now and they can take up a significant fraction of space taken by synthetic dataset.

3.3.7 Comparison to Distribution Matching

Both Training Trajectory Matching and Distribution matching have their advantages and disadvantages. Let us shortly go over them.

Distribution Matching avoids expensive bi-level optimization, thus making the distillation process much faster compared to other approaches. This also allows

DM	TTM
Fast distillation	Slow distillation + Need to create buffer
Lower performance	Higher Performance
Scalable to higher resolution/more images	Memory intensive - bi-level optimization
Limited to classification	Can be extended to regression

Figure 3.7: **Side-to-side comparison of the advantages and disadvantages of DM and TTM.** Both approaches have strong advantages - DM allows for faster distillation and scaling up, while TTM achieves much better performance. This pushes us towards the decision of using both methods in our further research.

for scaling up to higher-resolution images and larger synthetic datasets. On the other hand, images produced by DM result in relatively lower performance and DM objective limits approach only to regression tasks.

Training Trajectory matching uses very memory-intensive bi-level optimization for updating synthetic images. This however results in an extremely slow distillation process and an inability to scale to higher-resolution images and larger synthetic datasets. Additionally creating and storing the buffer of expert trajectories requires additional time and memory. But in return, TTM creates a synthetic dataset with relatively high performance and can be easily extended to regression tasks.

These advantages and disadvantages can of both methods encourage us to use both of them in further research in our thesis. A summarized comparison of both approaches can be seen in Figure 3.7

3.4 Factorization

The latest advancements in the field of dataset distillation were made due to the newly introduced factorization approaches [KKO+22], [LWY+22], [DR22]. The main idea rather than directly distilling synthetic images, we could distil a set of latent representations and decoding networks, which combined would produce a synthetic dataset. Previously introduced approaches largely treat each synthetic sample independently and ignore the connections between different samples and classes. As such, the information embraced by each synthetic sample is very limited. This inevitably leads to the loss of dataset information. Moreover, the small size of the synthetic dataset is incompatible with the growing number of parameters in deep models and may result in overfitting. The flexible combinations between latents and

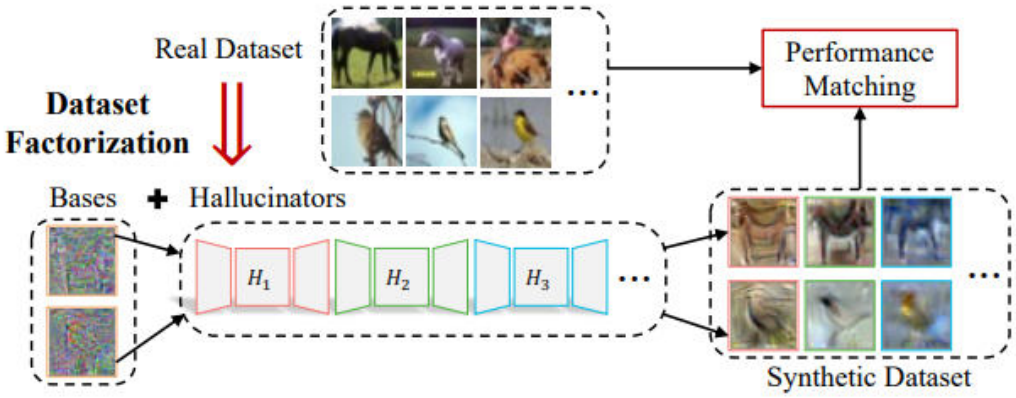


Figure 3.8: **High-level illustration of factorization approaches.** Synthetic data is factorized into latents (bases) and decoder networks (hallucinators). Combined they produce a synthetic dataset which is matched with a real dataset using a distillation objective. Gradients are backpropagated and used to update both latents and decoder networks. [LWY+22]

decoder networks equip the synthetic data with exponential informativeness gain, which largely increases the representability of synthetic data and helps overcome possible overfitting issues.

Another advantage of the factorization approaches is that they are plug-and-play, meaning that they can be combined with most of the distillation objectives.

We will describe in detail the approach introduced in [LWY+22] called Dataset Distillation via Factorization (HaBa) since we conducted experiments with some of the ideas introduced by the authors. An illustration demonstrating the high-level idea of HaBa can be seen in Figure 3.8. The authors described their goal as finding out if it is possible to instead encode shared relationships in synthetic samples rather than doing so at additional storage costs. To achieve this, authors propose factorizing synthetic datasets into Bases and Hallucinators. They redefine dataset distillation problem and define synthetic datasets as:

$$S_{haba} = \{H_{\theta_j}\}_{j=1}^{|H|} \cup \{(\tilde{x}_i, \tilde{y}_i)\}_{i=1}^{|B|}, \tag{3.15}$$

Where H is the set of hallucination networks and B are the bases. Hallucinator H_{θ_j} is parametrized by parameters θ_j. For the downstream training task, the synthetic training dataset is created by passing basses through hallucinator networks. In particular $\hat{x}_{ij} = H_{\theta_j}(\tilde{x}_i)$
In classical dataset distillation the shape of synthetic data $\hat{x}$ had to be the same as the shape of original data x. With the introduction of bases and hallucinators, this changed. Since hallucinator networks are capable of spatial-wise and channel-wise

transformation the shape of the bases can be different from the original images, as long as the output of hallucinators $\hat{x}_{ij} = H_{\theta_j}(\tilde{x}_i)$ has the same shape as original samples. This introduces additional advantages as more bases could be distilled per class, thus increasing the size of the synthetic dataset even more. The authors have shown that using 1 channel in some cases significantly increases the performance.

Hallucinators have encoder-transformation-decoder-based architecture. Encoder is composed of Convolutional blocks, then affine transformation is applied to the extracted feature and finally, the decoder that has symmetric architecture as the encoder produces the final synthetic image for downstream training.

3.4.1 Adverserial Constraints

Ideally, knowledge of every hallucinator should be orthogonal to all other hallucinators for maximal effectiveness of the factorization. The authors introduce different adversarial tactics to encourage such learning behaviour.

First, given the same base $\tilde{x}_i$, outputs produced by two different hallucinators $H_{\theta_1}(\tilde{x}_i)$ and $H_{\theta_2}(\tilde{x}_i)$ are expected to be as different as possible. In order to measure their divergence, a feature extraction network is used. We extract lower dimensional features and measure the distance between them. Training this feature extractor is a challenging task. The authors formalize joint training of hallucinators and feature extractor as a min-max game, where feature extractor is optimized to minimize the distance between $H_{\theta_1}(\tilde{x}_i)$ and $H_{\theta_2}(\tilde{x}_i)$ in embedding space while both bases and hallucinators are optimized to increase that distance. As a feature extractor authors use the same network architecture as the one used for the target downstream task.

In our experiments, we borrow the idea of factorization into bases and hallucinations but do not use adversarial constraints, since they introduce computational overhead and do not benefit the performance in all distillation cases.

3.4.2 Shortcomings

Although the introduced methods help increase the size of the synthetic dataset and overcome some issues, they are still limited in synthetic dataset size and do not fully solve the above-stated problems. Introducing stochasticity into the image-generation process might help us overcome these problems.

3.5 Generative Modelling

Stochastic image generation already exists and is mostly dominated by Generative Adversarial Networks (GANs) and recently introduced diffusion models. We concentrate on GANs due to the much faster image generation, which makes them more

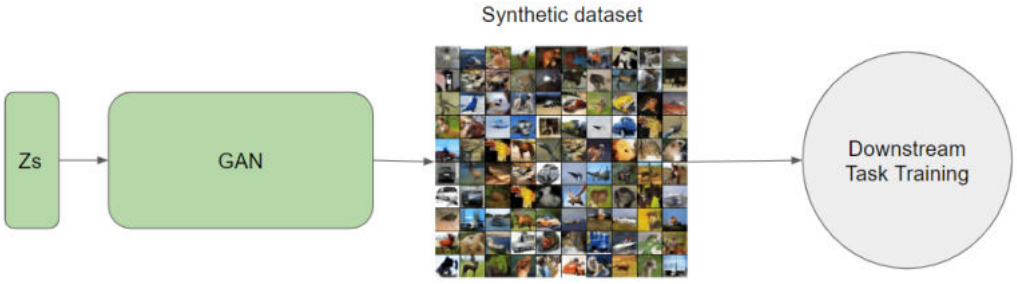


Figure 3.9: **Illustration of how GANs could be used for downstream task training.** Multiple random latents would be sampled and generated images would be used for training the network. This would be done on the fly, i.e. images would not be saved.

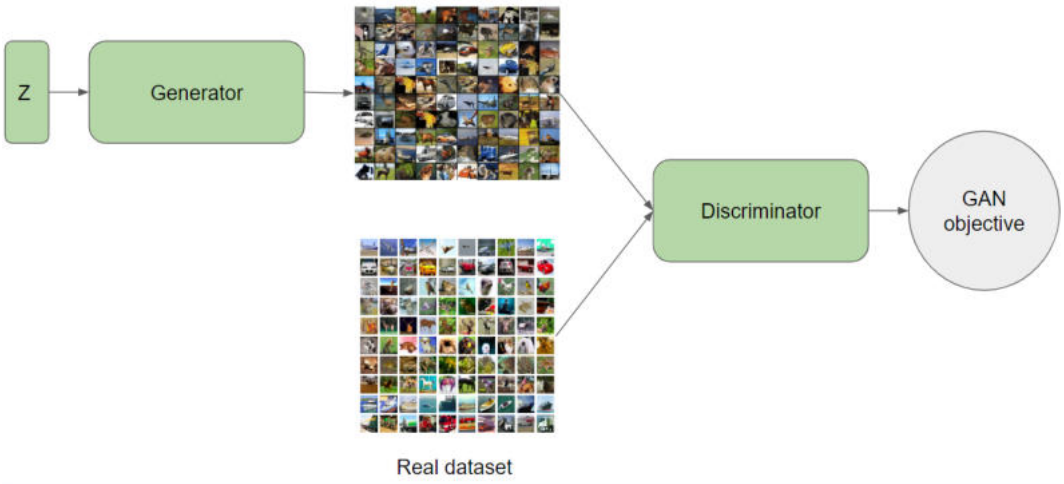


Figure 3.10: **High-level illustration of simplified GAN training loop.** Latents are fed into the generator in order to create fake images. Then fake images together with images from the real dataset are fed into the Discriminator and the output is passed further into the GAN objective.

suitable for generating images on the fly for downstream task training. The idea of using GAN for downstream task training dataset generation is demonstrated in Figure 3.9

It is important to remember that GANs are not trained with the objective of creating images that retain most of the information relevant to the training. A GAN's objective concentrates on the visual fidelity of the images and on generating high-quality samples, which in turn does not actually imply that all of the training-related information is saved. The GAN training pipeline is shown in Figure 3.10.

3.5.1 StyleGAN

StyleGAN was greatly inspired by style-transfer literature. The main novelty was introduced by departing from traditional generator architecture, where random latent vector z was directly fed into the first layer of the generator's feedforward network. The authors stepped away from this approach and proposed starting with a learned constant. Latent vector $z \in Z$ is first fed into a mapping network G_m that produces intermediate latent representation $w \in W$. The dimensionality of the original input latent z and intermediate w was selected to be the same for simplicity, although it also can be chosen differently. Mapping network G_m is an 8-Layer MLP. Then learned affine transformation A is applied to w in order to produce styles $y = (y_s, y_b)$. That is used to control adaptive instance normalization (AdaIN) [HB17],[GLK+17],[DPS+18] operations after each convolution layer of the synthesis network G_{syn}. AdaIN is defined as follows:

$$AdaIN(x_i, y) = y_{s,i}\frac{x_i - \mu(x_i)}{\sigma(x_i)} + y_{b,i}, \tag{3.16}$$

where each feature x_i is separately normalized and then scaled and biased using the corresponding scalar components from style y. Finally, the generator is introduced with explicit noise inputs. These are single-channel images consisting of uncorrelated Gaussian noise, that are fed to each layer of the synthesis network G_{syn}. Illustration of StyleGAN architecture and comparison to traditional generator architecture can be seen in Figure 3.11.

These changes allowed for a great increase in performance, making StyleGAN state-of-the-art at the time for multiple tasks in image generation. Switching to the new Style-based generator led to an automatic and unsupervised separation of high-level attributes and it enabled intuitive, scale-specific control of the image synthesis.

3.5.2 StyleGAN2

The authors of [KLA+20] analysed shortcomings and artefacts produced by the original StyleGAN and described architecture changes that would allow for fixing these issues. First is a common blob-like artefact and the second is the artefact in progressive growing [KALL18b], which is used to stabilize high-resolution StyleGAN training. To fix these issues authors propose two things - redesigning the normalization used in the generator, which would remove the blob-like artefact and start training by focusing on low-resolution images and then progressively shift to higher resolutions — without changing the network topology during training, which would solve the second problem.

In order to solve blub-like artefacts authors pinpoint the source of the issue to the AdaIN normalization. The problem is caused by AdaIN normalizing the mean and

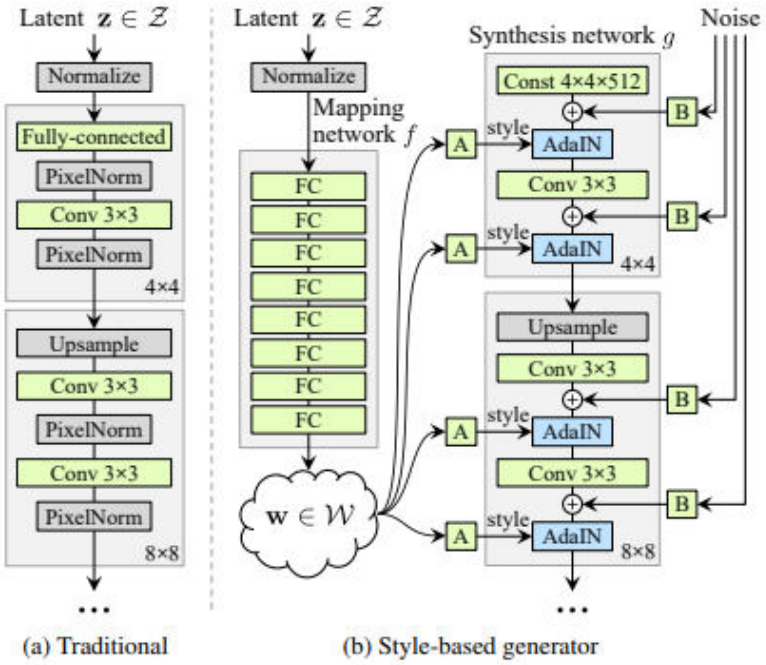


Figure 3.11: **Comparison of style-based generator architecture to traditional generator**. While a traditional generator [KALL18a] feeds the latent vectors through the input layer only, the StyleGAN generator feeds the latent vector to the mapping network, which outputs the t intermediate latent representation w. This representation is later fed into the synthesis network and used to control the generator through adaptive instance normalization (AdaIN) at each convolution layer. Gaussian noise is added after each convolution. [KLA19]

variance on each feature map separately, thus removing any information about the features that are relevant to each other. They have confirmed that normalization is the source by removing the normalization layer from the generator and observing that the blob-like artefact disappears completely. StyleGAN applied bias and noise within a style block, where the style block is part of a network where one style is active, causing their relative impact to be inversely proportional to the style's magnitudes. The authors observed that better results are obtained by moving these operations outside the style block in order to operate directly on normalized data. Additionally, they have noticed that it is sufficient for the normalization and modulation to operate on the standard deviation alone, so there is no need to operate on the mean.

The second artefact is caused by progressive growing, which helps stabilise the training. The main issue is that a progressively trained generator has a strong preference for the placement of details in generated images. The authors argue that the issue is caused by each resolution serving momentarily as the output resolution

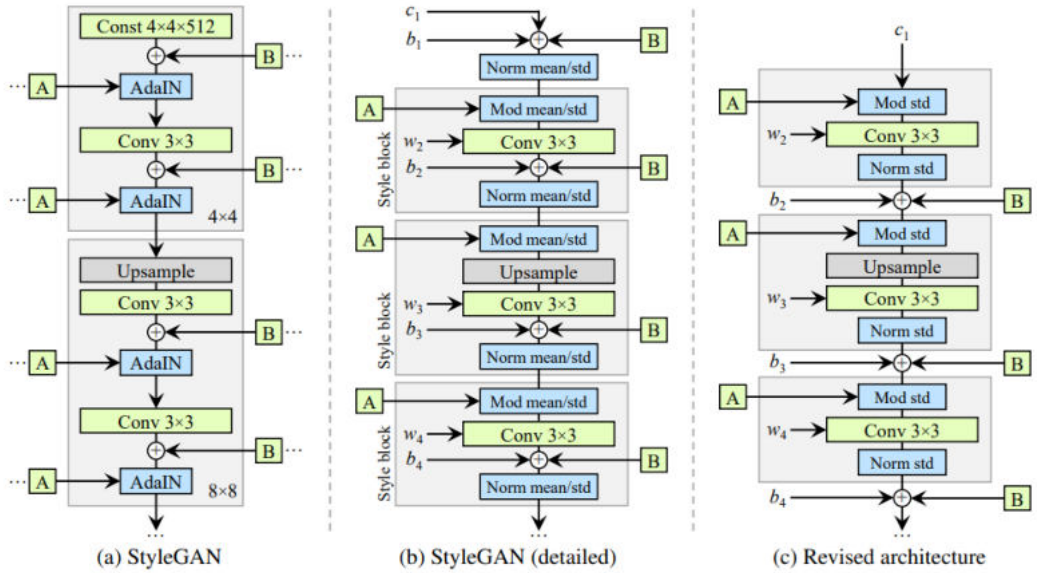


Figure 3.12: **StyleGAN and StyleGAN2 normalization comparison.** Some redundant operations at the beginning were removed, the addition of biases (b) and a noise broadcast operation B was moved to the outside of the active area of a style (style block), and only the standard deviation is adjusted per feature map. [KLA+20]

in progressive growing, which forces the generator to focus on maximal frequency details, leading to the trained network having excessively high frequencies in the intermediate layers. The authors compare different generator and discriminator architectures to select the one that produces high-quality images without progressive growing. They considered MSG-GAN [KW20][22], which connects the matching resolutions of the generator and discriminator using multiple skip connections. This architecture can be seen in Figure 3.13 (a). They additionally evaluate the simplified architecture of MSG-GAN by upsampling and summing the contributions of RGB outputs corresponding to different resolutions (Figure 3.13 (b). They further modify the design to use residual connections (Figure 3.13 (c)), which makes the architecture very similar to LAPGAN [DCSF15]. After evaluation authors decide to use a skip generator and a residual discriminator, without progressive growing.

With these changes, StyleGAN2 improved the quality of generated images further and considerably advanced the state of the art in several datasets. A detailed comparison of StyleGAN and StyleGAN2 normalization can be found in Figure 3.12.

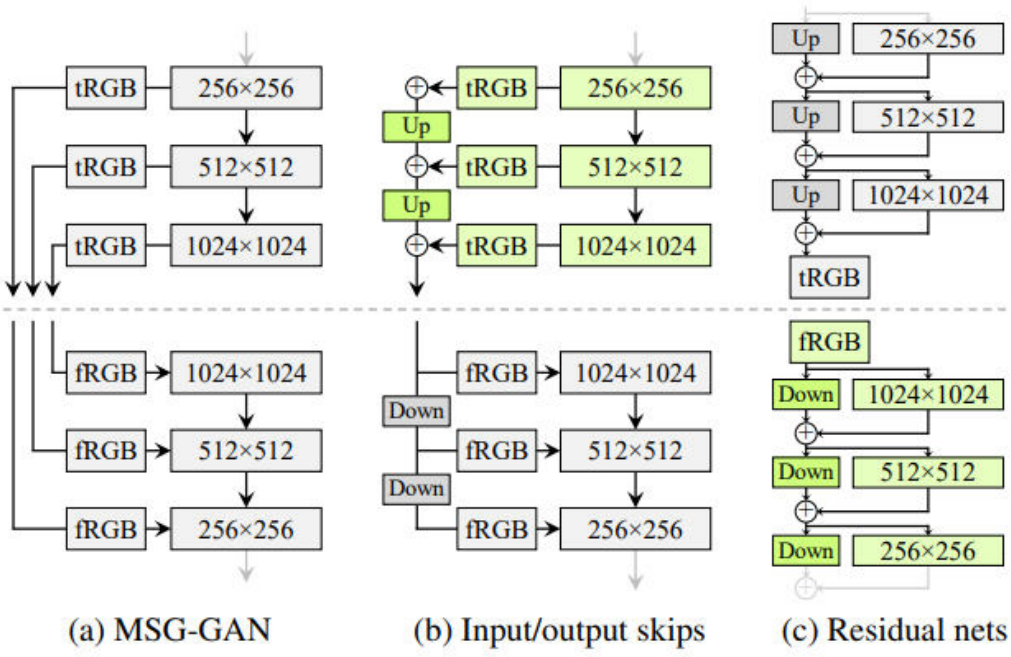


Figure 3.13: **Evaluated StyleGAN2 architectures.** Generator architectures can be seen above the dashed line and discriminator architectures - below. **Up** and **Down** respectively denote bilinear up and downsampling. Architectures selected for a final StyleGAN2 model are shown in green. [KLA+20]

3.5.3 StyleGAN-XL

Even with fixes and improvements from StyleGAN2, StyleGAN was still achieving unsatisfactory results on large unstructured datasets like ImageNet [DDS+09]. Another issue is that scaling up to higher resolutions becomes extremely expensive. That was the motivation behind StyleGAN-XL presented in [SSG22]. First, we will introduce two main building blocks of StyleGAN-XL: StyleGAN3 [KAL+21] and ProejctedGAN [SCMG21].

The novelty of StyleGAN3 compared to previous iterations of StyleGAN is that the synthesis network G_{syn} starts from a spatial map defined by Fourier features [TSM+20], [XWC+20]. This output then passes through N layers of convolutions, non-linearities, and upsampling to generate the final image. The number of layers N is 14, independent of the final output resolution. The authors of StyleGAN3 paper [KAL+21] observe that texture sticking artefacts were caused by a lack of equivariance and carefully designed StyleGAN3 to prevent texture sticking, which allows for revisiting progressively growing training strategy.

ProjectedGAN extended the classical Generator-Discriminator game by introducing

a set of feature projectors $\{P_l\}$. These projectors are used to map real images x and images generated by G to the discriminator's lower dimensional input space. The Projected GAN objective can be then formulated as:

$$\min_{G} \max_{\{D_l\}} \sum_{l \in L} (E_x[logD_l(P_l(x)] + E_z[log(1 - D_l(P_l(G(z))))]) \tag{3.17}$$

where $\{D_l\}$ is a set of independent discriminators operating on different feature projections. The projector networks consist of a pre-trained feature network, cross-channel mixing and cross-scale mixing layers. Four discriminators operate independently on the feature maps produced by these projection networks. Each discriminator uses a simple convolutional architecture and spectral normalization [MKKY18] and the depth of the discriminators varies depending on its input resolution. In the original papers authors settle with EfficientNet-Lite0 [TL20] as a projector network and with FastGAN [LZSE21b] as a generator. When using a StyleGAN generator, they observe that the discriminators can quickly overpower the generator if choosing suboptimal learning rates.

It was shown before that GAN regularization can be detrimental for multi-modal datasets [BDS19]. Therefore authors try to avoid regularization when it is possible. In the StyleGAN3 paper, it was discovered that style mixing is unnecessary for StyleGAN3, thus it is disabled for StyleGAN-XL. For the discriminant authors follows Projected GAN and use spectral normalization without gradient penalties. In addition, all images are blurred with a Gaussian filter for the first 200 thousand images. Discriminator blurring was first introduced in [KAL+21] for StyleGAN3-R and it prevented the discriminator from focusing on high frequencies early on in the training.

The main difference between the FastGAN generator used in Projected GAN and the generator used in StyleGAN is the input latent z dimension. StyleGAN's latent space is relatively high-dimensional. While FastGAN expects input latents to be 100-dimensional, StyleGAN expects 512-dimensional. Such high latent dimensionality is redundant and makes training of a mapping network G_m much harder at the training beginning. Therefore authors reduced StyleGANs input code z to 64 dimensions which allowed them to observe much more stable training in combination with Projected GAN. The intermediate latent w dimension stays unchanged and is equal to 512.

Conditioning mode on class-relevant information is very important for improving overall performance. In order to prevent the training collapse when using trainable class embeddings, authors propose using easing optimization with pre-trained embeddings. Embeddings are extracted using Efficientnet-lite0 [TL20] and the mean per ImageNet class is calculated and used as a class embedding. The network has a low channel count to keep the embedding dimension small for the same reasons as stated above for latent dimensionality. The learned class embeddings are then passed to the trainable projection network in order to match the size of z to avoid

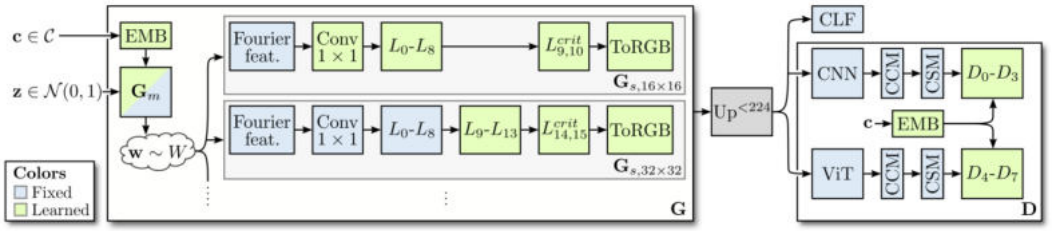


Figure 3.14: **StyleGAN-XL illustration** Latent code z and class label c are fed into the pre-trained embedding and the mapping network G_m to generate intermediate style latents w. During training, layers are gradually added to double the output resolution for each stage of the progressive growing. Only the latest layers are trained while others are frozen. G_m is only trained for the initial 16x16 stage and remains fixed afterwards. The generated image is upsampled when smaller than 224x224 and passed through a CNN and a ViT. At higher resolutions, the CNN receives the unaltered image while the ViT receives a downsampled input to keep memory requirements low. [SSG22]

any imbalance. Both the generator and discriminators are conditioned on class embeddings.

The authors proposed modified progressive growing training. Starting at 16x16 resolution enables breaking hard high-resolution ImageNet tasks into much smaller, easier-to-learn tasks. The training starts at a resolution of 16x16 with 11 layers. Every time the resolution increases, 2 layers are being cut off and 7 new ones are being added. For the final stage at 1024x1024, only 5 new layers are added as the last two are not discarded. This amounts to a total of 39 layers at the maximum resolution. Once new layers are added, the lower-resolution layers remain frozen to prevent mode collapse. A detailed illustration of StyleGAN-XL architecture is shown in Figure 3.14.

StyleGAN-XL became a new state-of-the-art on large-scale image synthesis and was the first to generate images at a resolution of 1024x1024 at such a dataset scale.

3.5.4 IT-GAN

The attempt to bridge Dataset Distillation and Generative modelling fields was conducted in [ZB22]. The main idea is to use a pre-trained GAN, in particular, BigGAN [BDS19], which was trained on the original dataset, freeze its parameters and use a dataset distillation objective to distil latents. The intuition behind this is that some of the samples carry more relevant training information than others. By learning the latent vectors, we effectively cover the distribution of the most informative samples generated by GAN. There were multiple downsides to this approach -

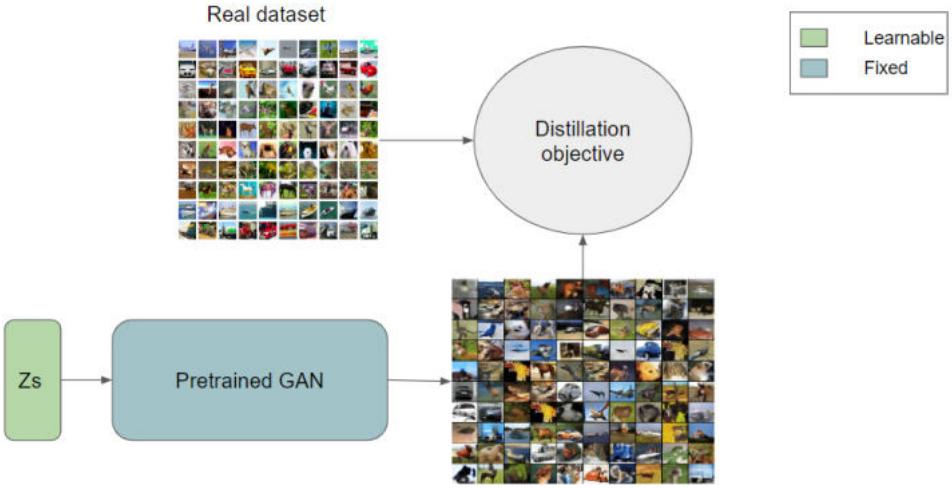


Figure 3.15: **Illustration of IT-GAN approach.** GAN was previously trained on the real dataset and its parameters are frozen. Latent vectors are randomly initialized and distilled using the dataset distillation objective.

the size of the GAN was very large compared to the synthetic datasets produced by classical dataset distillation and performance gain was not that significant. Additionally, the number of latents is fixed, thus again limiting the representability. IT-GAN training pipeline is demonstrated in Figure 3.15 and some of the synthetic samples generated for CIFAR10 and CIFAR100 can be seen in Figure 3.16.

3.6 Generative Dataset Distillation

Now we can finally introduce our approach - Generative Dataset Distillation. The idea is to fully bridge the gap between the dataset distillation and the generative modelling fields by combining the advantages of both fields.

To do so we will, depending on the dataset, use a generator G network from StyleGAN2 or StyleGAN-XL, and adapt the hyperparameters of the network to fit under the strict memory constraints of the dataset distillation. Using a generator architecture that has already been proven to create high-quality images will enable us to train an expressive generator capable of synthesising informative samples. By using a generator we will overcome the issue of classical dataset distillation and factorization methods since we will be able to generate much larger synthetic datasets. The generator G can be initialized both randomly or using the weights from a trained StyleGAN network.

After that, we will follow the steps of a classical dataset distillation methods DM and TTM, allowing us to train a generator that is tailored for producing informative

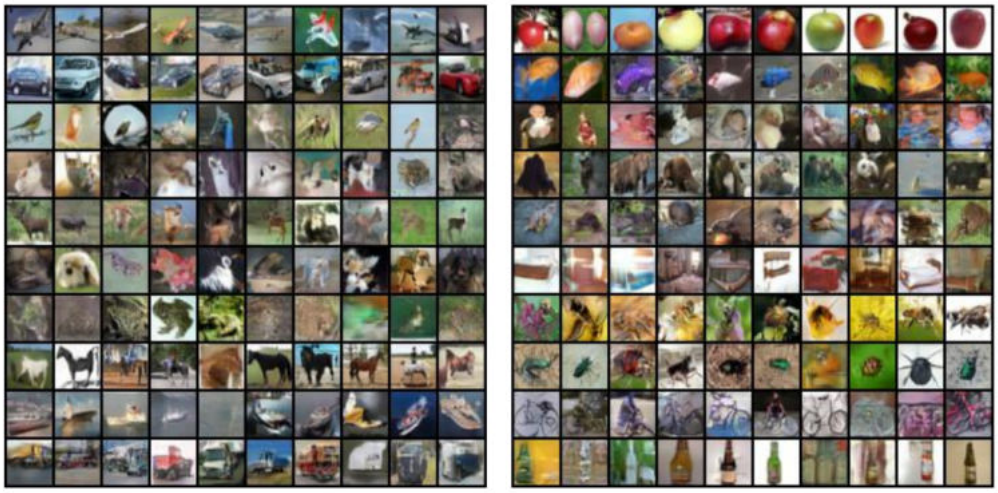


Figure 3.16: **Examples of images generated by IT-GAN for CIFAR10 (left) and CIFAR100 (right) datasets.** We can see that compared to the other distillation methods these images look much more realistic visually.

samples. The main difference will be that prior to matching we will randomly sample latents z and generate a mini-batch of synthetic images B_G^S. After computing the matching loss we will backpropagate all the way to our generator G and update its parameters. An illustration of a generative dataset distillation training pipeline is illustrated in Figure 3.17.

The final product of Generative dataset distillation will be our generator G. For the downstream task training, we will use it to generate training samples. Different sampling strategies will be introduced and evaluated later. Generative dataset distillation is a plug-and-play approach and can be used with many other dataset distillation methods, not only with DM and TTM. To summarize Generative dataset distillation comes with the benefits of distilling informative synthetic samples (DM, TTM), being plug-and-play and enabling encoding the dependencies between different classes and samples (HaBa) and finally having the possibility of generating large synthetic datasets, which would prevent overfitting (StyleGAN). The limitations on the generator are quite strict. For the dataset with 32x32 resolution and 10 classes in the most extreme case, we would have a generator with only 30 thousand parameters. We argue that these limitations serve as a barrier towards having an expressive generator and conclude that the best environment for our method is to be applied to the higher resolution datasets or datasets with more classes, for example, ImageNet with 1 thousand classes. In that scenario, off-the-shelf StyleGAN-XL already fits under distillation memory constraints.

A full generative dataset distillation algorithm can be seen in Algorithm 3.

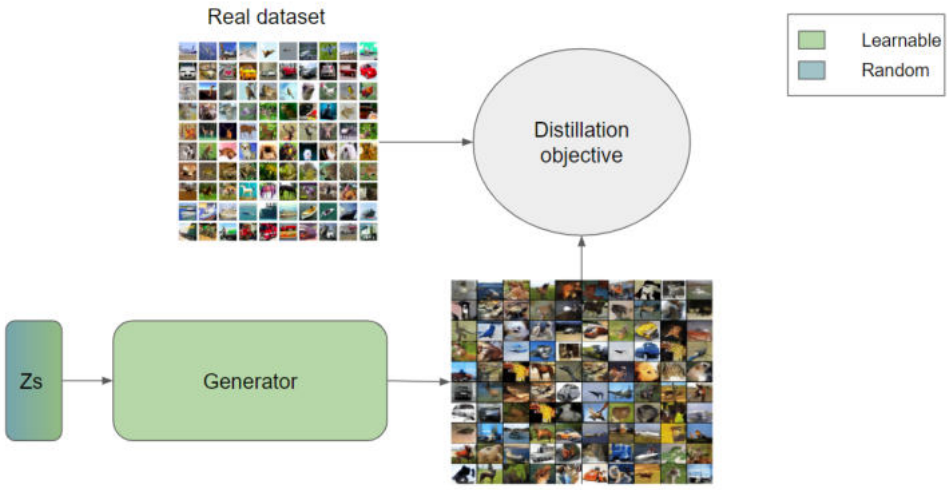


Figure 3.17: **Illustration of proposed Generative Dataset Distillation approach.** Latents which can be random or learnable are fed into a fully trainable generator in order to produce a synthetic dataset. Images are afterwards passed onto TTM or DM method and after loss is computed gradients are backpropagated to update the generator network.

Algorithm 3: Generative Dataset Distillation with Distribution matching

Data: Training set D

Input: Deep neural network F_θ parameterized with θ, the probability distribution over parameters P_θ, differentiable augmentation $A(\cdot, w)$ parameterized with w, augmentation parameter distribution P_W, number of training iterations K, learning rate η.

Initialize generator G;

for $i = 0, ..., K-1$ **do**

- Sample $\theta \sim P_\theta$;
- Sample mini-batch of real data $B_c^D \sim D$ and $w \sim P_w$ for every class c ;
- Sample random latent vectors z;
- Generate mini-batch of synthetic images $B_c^S = G(z, c)$ for every class c ;
- Compute loss: $L = \sum_{c=0}^{C-1} \| \frac{1}{|B_c^D|} \sum_{x \in B_c^D} F_\theta(A(x, w)) - \frac{1}{|B_c^S|} \sum_{\hat{x} \in B_c^S} F_\theta(A(\hat{x}, w)) \|^2$;
- Update $G \leftarrow G - \eta \nabla_G L$;

end

Result: Generator G

3.6.1 Extenstions

There are multiple possible extensions to this approach and we will cover some of them.

First is a method that we denote as Latent Dataset Distillation (**LatentDD**). This method can be considered as a combination of Generative Dataset Distillation and IT-GAN. We will first train the generator using the Dataset Distillation objective, then freeze its parameters and distil the latents. So the main difference is the way the generator was trained. Although, we show later that distilling latents does not improve performance if we sample from the generator enough while introducing additional memory cost for storing the latents.

3.6.2 Sampling

IT-GAN has shown the importance of using the right latent vectors for generating synthetic images, which is why we focus on finding the best sampling tactic for creating synthetic datasets from our generator.

There are a few approaches how to generate images for downstream task training. The simplest one would be just generating a lot of images before the training and then using them to train. Here the main limitation is our GPU memory, since we need to keep a lot of images allocated. Another approach would be generating much fewer images, but after training on them for a few epochs, throwing them away and generating a new batch. This way we overcome memory limitations and effectively show more images to the model during training. The last step of experimenting with sampling would be using different truncation values. Low probability regions in the latent space may not have enough training data to represent it accurately, so when generating images, those regions can be avoided to improve the image quality at the cost of the variation. This trick is commonly used in the generative modelling field and is usually referred to as Truncation Trick.

4 Results

In this chapter, we describe the settings of the conducted experiments, including datasets and network architectures and present our final results.

4.1 Datasets

In our experiments, we concentrated on CIFAR10 [KH+09], CarRacing dataset [BCP+16], ImageNet-1k [DDS+09] and some of the ImageNet subsets.

The CIFAR10 dataset consists of 10 classes of images with 3 channels and 32x32 resolution. It has a total of 60000 images - 50000 training samples and 10000 validation images. CIFAR10 has already been established as one of the main benchmarks for dataset distillation evaluation.

ImageNet dataset is much larger and diverse, it consists of more than 1 million images and 1 thousand classes. This alone makes it already an incredibly hard task for dataset distillation. For ImageNet and its subsets, we consider resolution 64x64. Original images have higher resolution, so in all of our experiments, we first normalize images to have zero mean and standard deviation 1, then resize the image and apply the centre crop. Examples of ImageNet subset samples can be observed in Figure 4.1.

ImageNet subsets that we consider are - ImageWoof (a subset of dog classes), ImageNette (simpler ImageNet) and ImageYellow (yellow objects). All subsets consist of 10 classes.

CarRacing is an environment created by OpenAI and commonly used in Reinforcement Learning and Imitation Learning communities. The original image resolution is 96x96 with 3 channels, but for the sake of resources economy, we downsample images to 32x32 and use them for distillation. CarRacing can take be used for both - classification and regression tasks. We use it in the classification setting with 6 classes - no action, gas, steer left, steer right, steer left and brake, steer right and brake. Our dataset consists of more than 12 thousand saved samples of expert actions. We find this dataset very interesting for two reasons - first, it can serve as a pathway to proving the applicability of dataset distillation to driving tasks and second due to the very low variance between images we expect dataset distillation to perform incredibly well. Examples of downsampled images can be seen in Figure 4.2.

ImageNette

ImageYellow

ImageWoof

Figure 4.1: **Examples of samples from ImageNet subsets**.

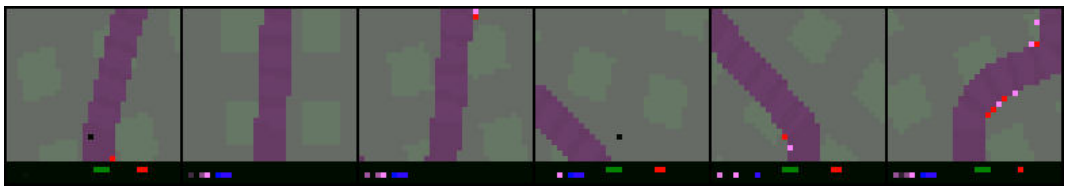


Figure 4.2: **Examples of samples from the CarRacing dataset.**

4.2 Evaluation

For evaluation on CIFAR10 and ImageNet, we use the networks that are the de-facto standard in the field - Convolutional Neural Networks (CNN), with 3 (CIFAR10) or 4 (ImageNet) convolutional blocks, and each block consists of a 128-kernel convolutional layer, instance normalization [UVL17], ReLU activation [NH10] and average pooling. For CarRacing, we use again a convolutional neural network with 3 convolutional blocks, consisting of a convolutional layer and ReLU activation. These blocks are followed by three fully connected layers with ReLU activations.

For the sake of fairness and to account for the faster training goal synthetic dataset is supposed to be tailored for, all of the downstream task training on CIFAR10 and ImageNet subsets is done for 1000 gradient update steps. There is no consensus in the field on how synthetic data should be evaluated, but most of the approaches follow this rule.

4.3 CIFAR10

We start off with the experiments on CIFAR10. Since CIFAR10 images have a low resolution of only 32x32, memory constraints, in this case, are very strict. In this set of experiments as a generator, we used previously introduced StyleGAN2 generator architecture. And as a distillation objective, we use both DM and TTM objectives. For the 1 image per class case, we are able to use only 30 thousand parameters for our generator. This limitation is roughly equal to only 120KB, which is extremely low. In order to extend to higher memory budgets we freeze the generator that was trained using DM/TTM objective and distil additional latent vectors. By distilling the latent vectors we might be able to select only the most informative samples and get rid of any redundant information. We denote this approach as Latent DM/TTM. Results comparing GenDD with classical DD approaches can be seen in Figure 4.3 for DM and in Figure 4.4 for TTM.

When we look at GenDM we see a significant increase in performance over the original DM. LatentDM on the other hand does not improve much over GenDM, while consuming additional memory. This indicates that if we sample from a generator enough, we do not need to learn the distribution of the most informative ones since they will likely already be covered. For GenTTM the margin of improvement over pure TTM is lower compared to GenDM, nevertheless, there is still significant performance gain. LatentTTM in this case even decreases the performance. This motivates us to depart from the LatentDD model and concentrate on GenDD.

Although GenDM and GenTTM bring improvement over classical DM and TTM, they are still behind the state-of-the-art distillation methods in terms of performance. We believe that the reason for this is a strict memory limit, which restricts the expressiveness of the generator and effectively limits the possible performance of

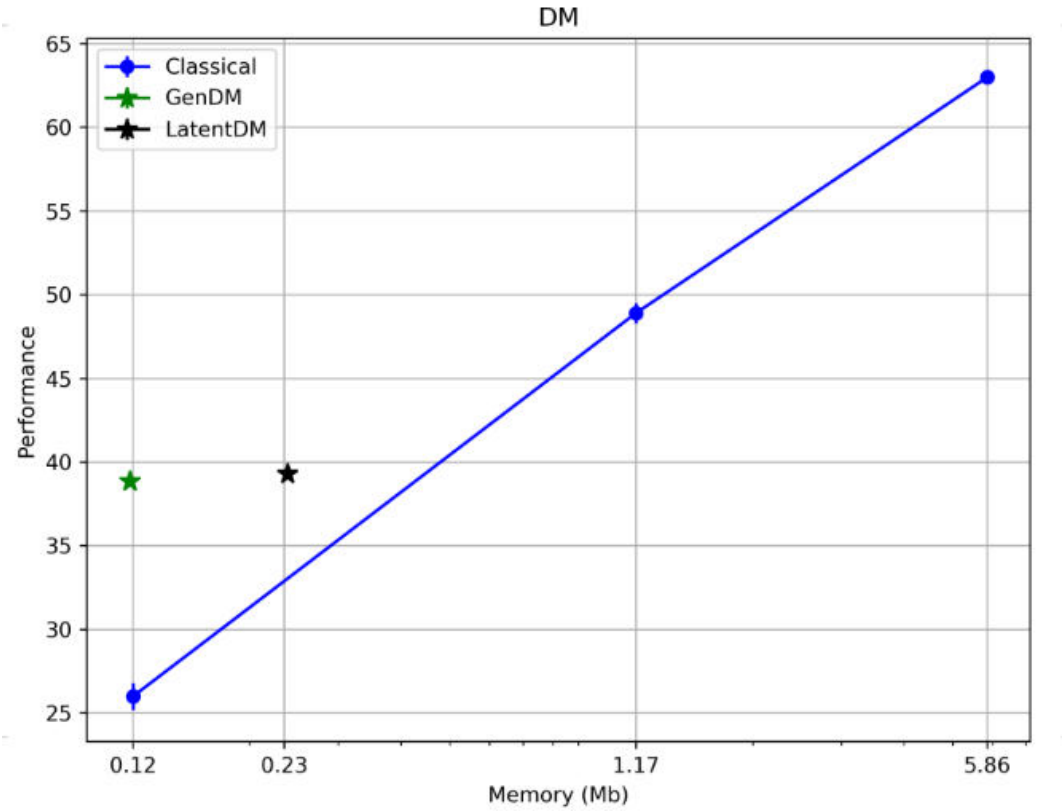


Figure 4.3: **Comparison between the performance of GenDM and LatentDM with pure DM on CIFAR10 dataset.** GenDM significantly improves performance over pure DM, while LatentDM does not introduce any significant improvement while doubling the memory consumption compared to GenDM. Even though GenDM improves over DM, it is still behind in performance compared to the current state-of-the-art methods.

our method on such small datasets. Examples of synthetic images generated by GenDM and GenTTM are illustrated in Figure 4.5.

4.4 Small GANs

To further investigate the magnitude of the effect memory constraint has on the generator, we train two StyleGAN-XL models on ImageNet subsets - Imagenette, Imagewoof and Imageyellow. We train smaller StyleGAN-XL that take up 23MB which roughly corresponds to 50 images per class and then larger StyleGAN-XL that take up 123MB, which corresponds to more than 500 images per class. Performance comparison of models trained on images generated by these two StyleGAN-XLs can

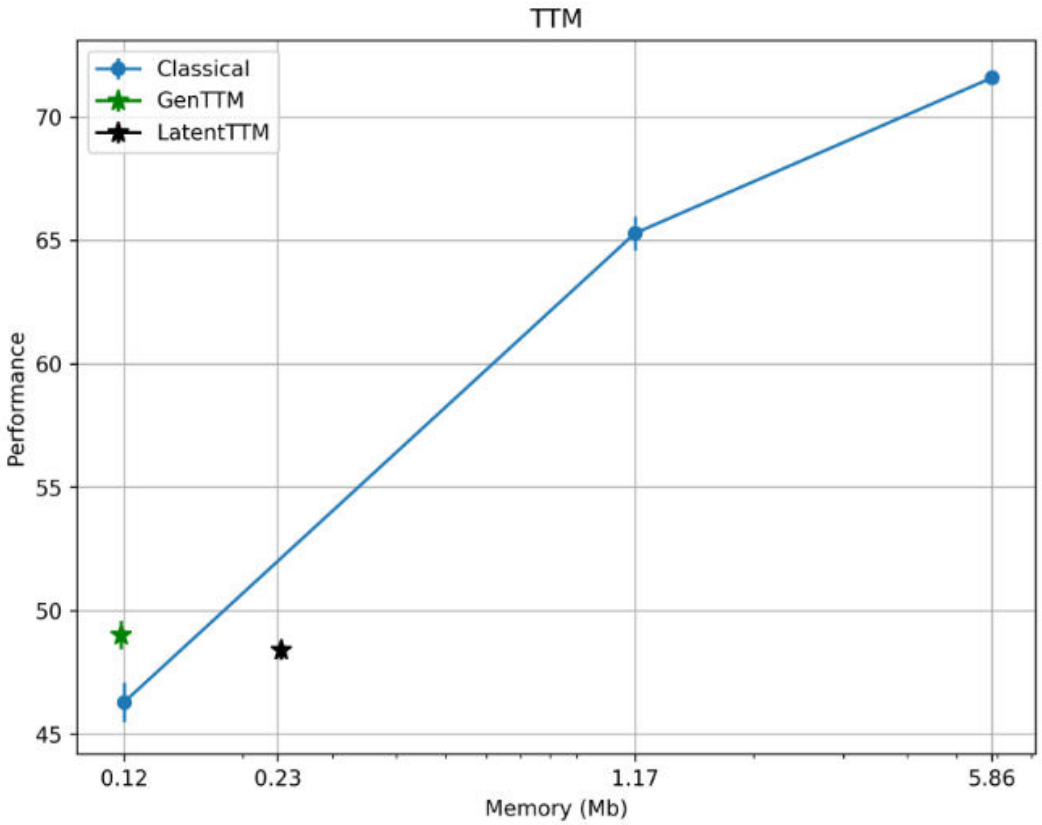


Figure 4.4: **Comparison between the performance of GenTTM and LatentTTM with pure TTM on CIFAR10 dataset.** GenTTM improves performance over pure TTM, while LatentTTM decreases the performance and at that doubles the memory consumption compared to GenTTM. Even though GenTTM improves over TTM, it is still behind in performance compared to the current state-of-the-art methods.

Dataset	StyleGAN-XL(23MB)	StyleGAN-XL(123MB)	StyleGAN-XL(original)
Imagenette	36.33 ± 0.09	57.9 ± 1.14	68.44 ± 1.11
Imagewoof	15.67 ± 0.25	41.0 ± 0.91	50.40 ± 0.44
Imageyellow	26.47 ± 0.61	39.93 ± 1.64	65.76 ± 0.88

Table 4.1: **Comparison of performance between original StyleGAN-XL and smaller StyleGAN-XLs trained on ImageNet subsets** It can be clearly seen that the size of the model has a huge impact on the performance on a downstream task, confirming our point about very limited expressiveness of a generator due to the memory budget.

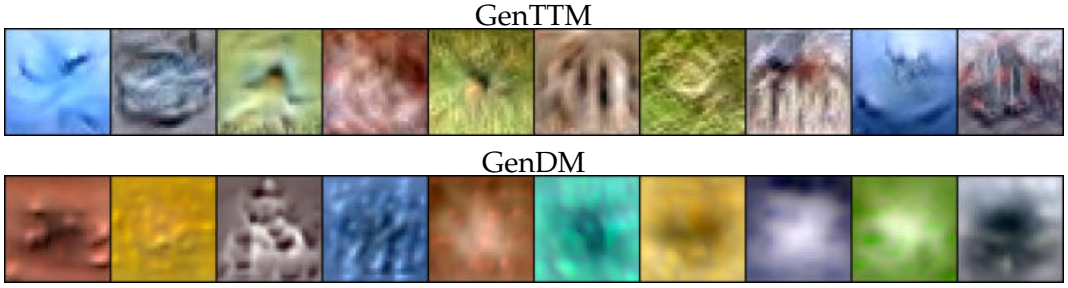


Figure 4.5: **Examples of Images generated by GenTTM (top) and GenDM (bottom) on CIFAR10**. TTM images are much closer to the original images, while GenDM ends up with blobs meaningless for human understanding.

Dataset	StyleGAN-XL(23MB)	StyleGAN-XL(123MB)	StyleGAN-XL(original)
Imagenette	48.24	15.59	9.99
Imagewoof	125.13	6.89	8.63
Imageyellow	66.55	17.65	9.68

Table 4.2: **Comparison of FID scores between original StyleGAN-XL and smaller StyleGAN-XLs trained on ImageNet subsets** It can be clearly seen that the size of the model has a huge impact on the FID scores, confirming our point about very limited expressiveness of a generator due to the memory budget.

be seen in Table 4.1. FID scores can be seen in Table 4.2. Comparisons also include the original StyleGAN-XL that was trained on full ImageNet.

From these results, we can conclude that the current memory budget on smaller datasets is too small to train a proper StyleGAN-XL. The smallest StyleGAN-XL performs poorly both in terms of performance on downstream tasks and FID score. Additionally, all small StyleGAN-XLs that we have trained collapsed, even with tree step progressive growth from 16x16 to 64x64 resolution. This again motivates us to explore Generative dataset distillation on datasets where memory requirements are not so strict.

Examples of images generated for ImageYellow can be seen in Figure 4.7 for small StyleGAN-XL and in Figure 4.6 for large StyleGAN-XL. For the smaller StyleGAN-XL mode collapse can be clearly observed.

4.5 StyleGAN-XL

We use off-the-shelf StyleGAN-XL that was trained on ImageNet with 64x64 resolution to evaluate the distillation performance on the ImageNet and its subsets. With a total size of over 523MB, it falls slightly above 10 images per class memory

Figure 4.6: **Example of images generated by StyleGAN-XL(123MB) trained on ImageYellow subset.**

Method	ImageNet	ImageNette	ImageWoof	ImageMeow	ImageSquawk	ImageFruit	ImageYellow
StyleGAN-XL	15.04± 0.09	68.44± 1.11	50.40± 0.44	49.40± 0.52	69.52± 0.79	43.28± 1.15	65.76± 0.88
Full Dataset	20.84 ± 0.05	78.72± 1.34	50.37 ± 2.50	56.72± 1.97	80.96± 0.88	54.96 ± 0.90	78.08 ± 0.87

Table 4.3: **Performance on dataset Generated using off the shelf StyleGAN-XL** StyleGAN-XL performs surprisingly well, although it's important to note that it takes up 523MB, which is much higher compared to the distillation methods

budget on ImageNet. A comparison between the generator memory budget for CIFAR10 and ImageNet is illustrated in Figure 4.8. None of the previous Distillation

Figure 4.7: **Example of images generated by small StyleGAN-XL(23MB) trained on ImageYellow subset.**Mode collapse can be clearly observed on this examples. This indicates an inability to train a proper generative network with such strict memory constraints.

approaches does not even scale to such a synthetic dataset size. Only a few of them actually try distilling ImageNet and they only consider 1ipc and 2ipc cases [ZNB22]. It is important to say that the comparison on the 10-class subsets is not fair, since StyleGAN-XL is much larger than the budget for 50 images per class - 23MB. Nevertheless, we find this evaluation interesting as StyleGAN-XL was not trained for this task and this also might provide us with new insight into how cross-class information can benefit synthetic images. Comparison between the performance of full original datasets and StyleGAN-XL generated dataset can be seen in Table 4.3.

Figure 4.8: **Memory budget comparison between CIFAR10 and ImageNet**. A much larger budget on ImageNet gives us the opportunity to train a much larger and more expressive generator. 10 images per class case already allows for allocating 512MB on ImageNet, which is close to the size of the original StyleGAN-XL.

For subsets, we generated 1500 images per class and trained the classifier on them for 1 thousand gradient steps. For the full ImageNet, we generated 200 images per class.

From the results, we can see that off-the-shelf StyleGAN-XL is already an incredibly strong baseline for ImageNet with a performance of 15.04%. It also performs fairly well on subsets. Most interesting there is the performance on ImageWoof, which contains 10 dog classes from ImageNet. StyleGan-XL generated images achieve slightly higher performance on full ImageWoof. This can be explained by StyleGAN-XL sharing information from other dog classes, since ImageNet has more than 10 of those and using this additional information to encode more information into synthetic images. Examples of images generated by StyleGAN-XL are shown in Figure 4.9.

Figure 4.9: **Examples of images generated using StyleGAN-XL.**

4.6 Sampling

Sampling plays a very important role in image generation using GANs. Commonly used in the field is a previously introduced truncation trick. We additionally explore two settings: First, all data is generated prior to the downstream task training or second, we generate data in smaller batches which are thrown away once the new batch is generated, this helps us overcome memory constraints. All batches in the second case are generated after an equal number of training epochs. All experiments in this section were done on the ImageWoof subset. Results can be seen in Figure 4.10 and in Figure 4.11.

From the first figure, we can see that the more samples we generate the higher

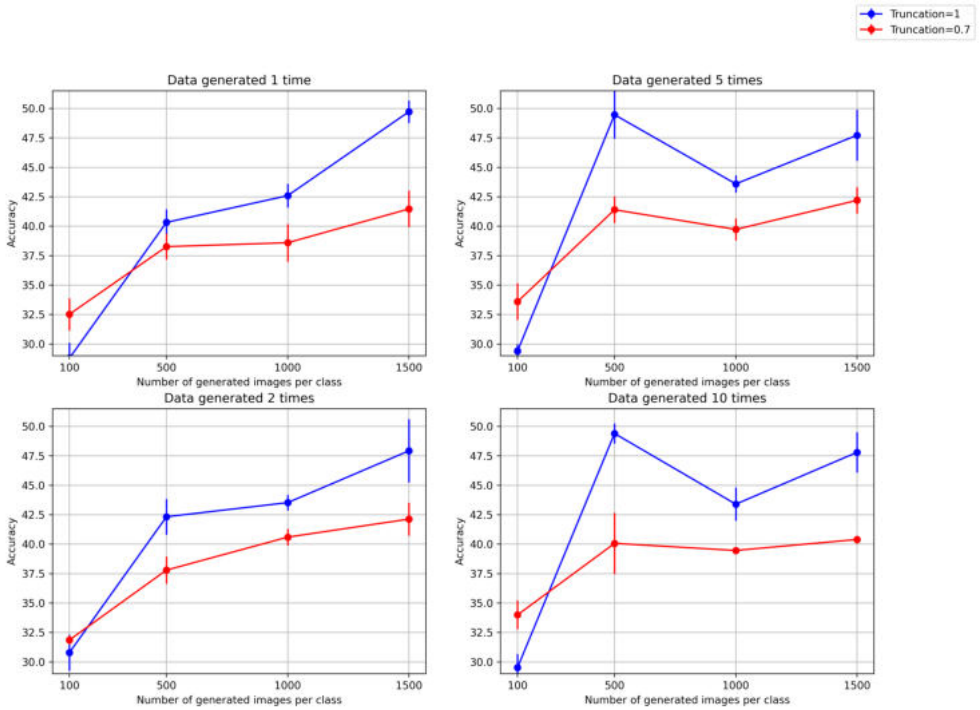


Figure 4.10: **Comparison of downstream task performance with different sampling tactics.** The evaluation was done for two truncation values - 1(blue) and 0.7(red). Each plot represents a different number of data generations, where 1 means that data was generated only prior to the training. Image generations are apart from each other by an equal number of training steps. We can see that regenerating data does not benefit the performance and that smaller truncation only benefits the case with the small synthetic dataset. The best performance is achieved with 1500 images per class generated and with a truncation value of 1.

the performance on the downstream task. Truncation only proves to be valuable on the low number of generated images. This is expected as with the low number of generated images we are unable to fully cover the distribution, thus making truncation valuable by allowing to select only the most informative samples.

From the second figure, we can conclude that repeated generation can in some cases increase the performance, but not significantly and at a great cost since regenerating synthetic datasets is time-consuming and thus desired training speed up will not be achieved.

Based on these findings we decide to generate synthetic data in large batches prior to the training process and use the truncation value of 1.

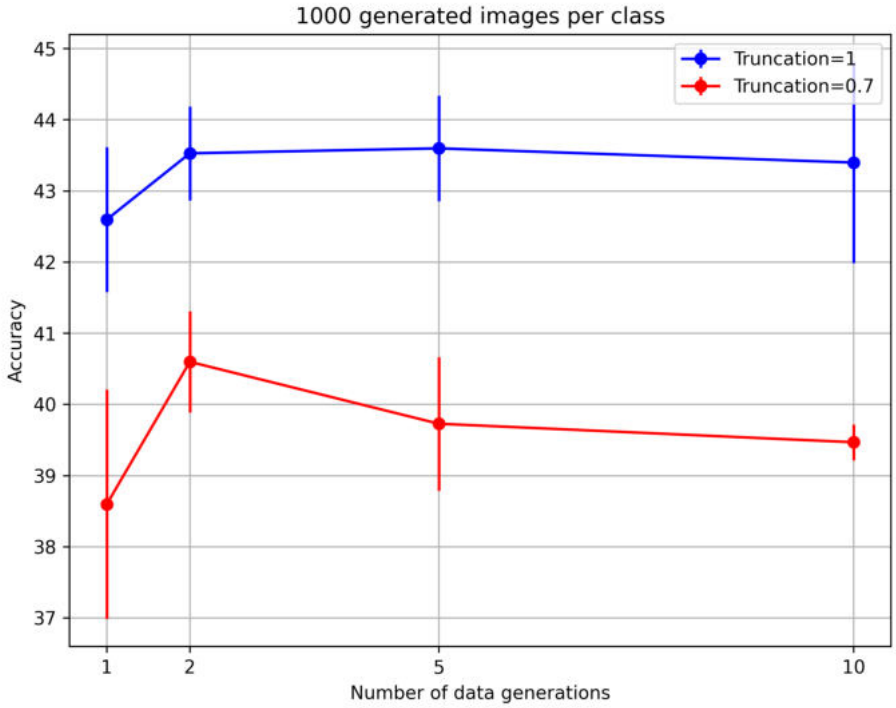


Figure 4.11: **Performance comparison between different synthetic dataset generation strategies on Imagewoof**. It can be seen that repeated data generation does not benefit the overall performance while increasing the training time significantly. This motivates us to generate data for downstream tasks only once - prior to the training.

4.7 ImageNet and Subsets

Dataset	StyleGAN-XL	GenDMv1	DM
ImageNet	15.04± 0.09	13.33 ± 0.08	1.7* ± 0.1
Imagenette	57.9 ± 1.14	61.87 ± 1.24	65.63 ± 1.83

Table 4.4: **GenDM performance on ImageNet and its subsets**. Our generative DM approach decreases performance on ImageNet, while on the Imagenette subset, it increases performance, but still underperforms when compared to the pure DM. This might indicate a lack of supervision during generator training. * DM results for ImageNet are only for synthetic dataset with two images per class.

Based on our previous finding we shift to the full ImageNet dataset with 64x64 resolution and Imagenette subset. Here we are able to use the same generator as in StyleGAN-XL, which already proved capable of generating not only visually high-quality images but also informative samples which achieve good performance on downstream tasks. We used the GenDM approach due to the large memory requirements of GenTTM, which would limit the scope of possible experiments. We tried different training strategies - training from scratch, fine-tuning pre-trained StyleGAN-XL, and using progressive growing with the first stage of GAN training and the second stage of GenDM training. Results from the best performing model can be seen in Table 4.4.

Unfortunately, as we can see our approach fails to outperform StyleGAN-XL baseline on full ImageNet and while improving over StyleGAN-XL on Imagenette, it still performs worse than pure DM. This might indicate that our GenDM lacks supervision for ImageNet training and leaves space for additional future research in this direction. What is also important to note is that GenDM showed great improvement over DM on CIFAR10 with a much smaller model, this might indicate that the DM objective is not suitable for training larger networks.

4.8 Tweaks

Method	**ImageNette**	**ImageWoof**	**ImageMeow**	**ImageSquawk**	**ImageFruit**	**ImageYellow**
DM [ZB23]	65.63± 1.83	36.04± 0.83	40.31± 1.02	59.79± 0.20	39.83± 0.38	61.26± 0.36
DM (L1)	69.03± 0.71	39.33± 0.50	45.37± 0.26	63.63± 0.43	43.50± 1.09	62.64± 0.35
DM (HaBa)	47.89± 3.75	24.93± 4.47	25.32± 1.60	37.25± 1.41	23.60± .97	44.81± 1.91
TTM [CWT+22]	70.17 ± 0.55	37.97± 0.19	45.36± 0.56	62.40± 0.26	46.59± 0.76	66.72± 0.12
TTM (L1)	69.57± 0.09	38.68± 0.14	44.52± 0.24	61.67± 0.31	46.39± 0.47	65.68± 0.44
TTM (HaBa)	68.44± 0.36	35.25± 1.18	42.92± 2.9	59.72± 0.96	38.88± 3.46	63.36± 1.72
TTM (prune)	70.25± 0.68	38.56± 0.77	45.13± 0.92	62.17± 0.52	46.20± 0.75	66.84± 0.59
Full Dataset	78.47± 0.63	49.7± 1.98	57.2± 1.04	80.32 ± 1.11	55.11 ± 1.34	76.83 ± 1.37

Table 4.5: **Evaluation of different tweaks for classical distillation method.** We tested multiple tweaks from different papers and also tested using L1 distance for matching. Most of the tweaks don't increase performance and some of them decrease it, while using L1 distance with DM significantly improves performance over multiple datasets.

During our experiments, we explored many possible modifications to the classical dataset distillation methods, such as parameter pruning [LTOH22] or HaBa factorization [LWY+22] and also the L1 loss introduced by us. Experiments were conducted on multiple ImageNet subsets on 64x64 resolution. The synthetic set size is set to 50 images per class. The results can be seen in Table 4.5.

Most of the tweaks do not benefit the performance. By HaBa we denote factorizing synthetic datasets into Bases and Hallucinators, but without additional adversarial losses. DM combined with the L1 distance metric significantly improves over pure

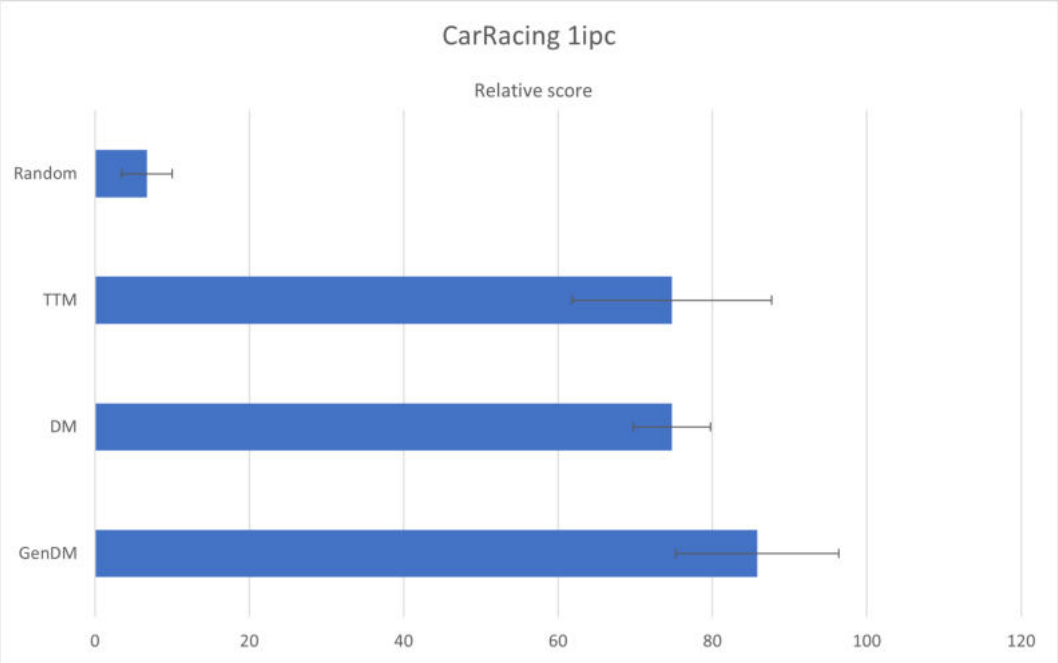


Figure 4.12: **Performance comparison between different distillation methods on CarRacing**. With 1ipc we are already able to retain more than 80% of the score of a model trained on the original dataset. Not only that but GenDM outperforms classical DM and TTM and achieves an incredible performance of over 85 %.

DM, proving our hypothesis that using the L1 metric in high dimensional space over L2 would benefit the final performance. Interestingly enough such a benefit is not present when using TTM with L1 distance.

4.9 Supplementary Experiments

In this section, we will cover preliminary experiments that we conducted in order to understand the dataset distillation baselines better and test the applicability to different datasets.

4.9.1 CarRacing

Here we will present the results of experiments conducted on the CarRacing dataset. We evaluated existing dataset distillation methods TTM and DM and GenDM approach. Additionally, we evaluated different variations of DM loss, including classical MMD, L1, Manhattan and Cosine. Performance on CarRacing is measured in terms of score, which increases when tiles on the road are crossed and decrease as

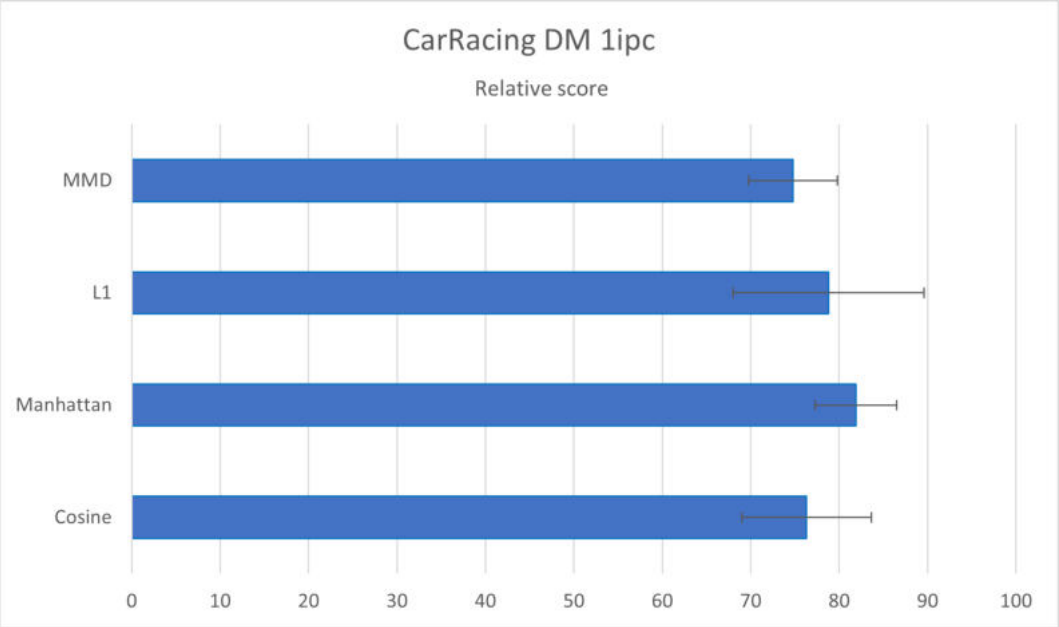


Figure 4.13: **Performance comparison between the different DM losses on CarRacing.** Here we again confirm previous findings. The L1 distance metric can be very beneficial for the final performance of DM images. Additionally, the Manhattan distance proves also helpful in the case of simple datasets with low sample variance.

time goes on. Our expert model achieves a score of 519. All distillation scores will be presented relative to this number. Results can be seen in Figure 4.12 and in Figure 4.14.

From the first figure, we can observe that already with just 1 image per class distillation methods achieve incredible performance, by retaining more than 80% of the original score. This can be explained by the low variance between the simplest, which allows dataset distillation to condense them into one extremely informative sample. Our GenDM achieves the best performance on this task and proves that dataset distillation can be extended further from the classical classification datasets.

From the second figure, we can see that DM performance again highly benefits from using the L1 distance metric in the loss. The same goes for Manhattan distance. Such improvement can be again accounted to the simplicity of the dataset, thus by using metrics which penalize small differences heavier than the original L2 metric, we get rid of more redundant information.

Examples of synthetic datasets produced by different distillation methods can be seen in Figure 4.14

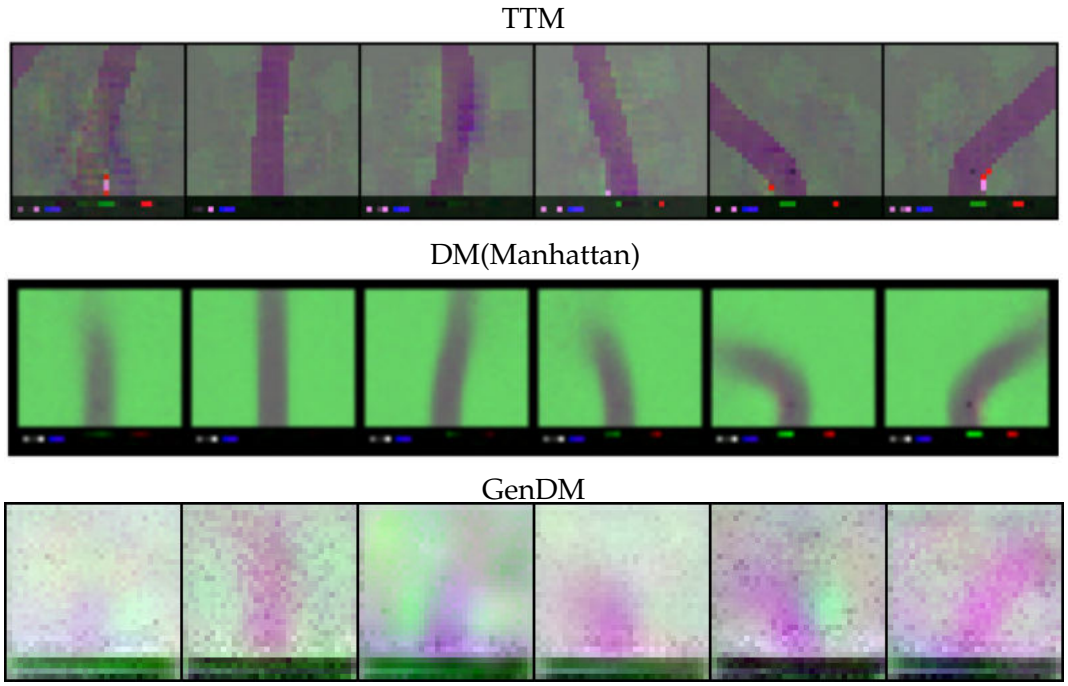


Figure 4.14: **Examples of synthetic CarRacing Images.** Here GenDM generates more realistic images compared to the CIFAR10 dataset, where they mostly have blob-like structures.

4.9.2 Initializatons

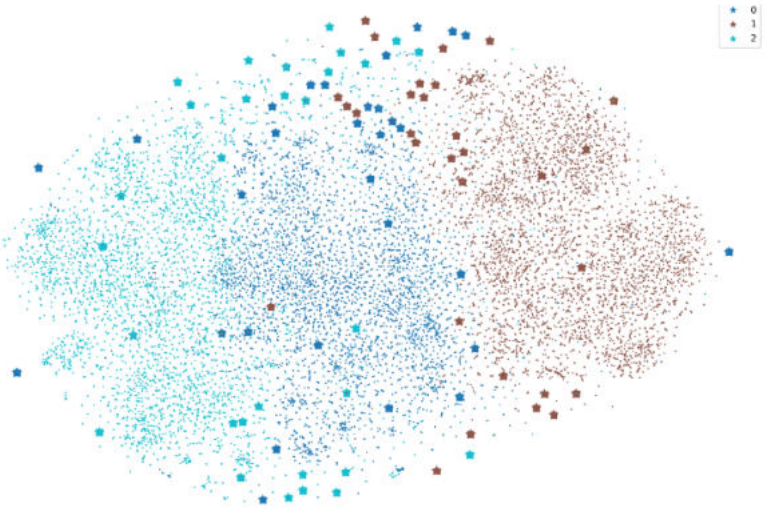
We experiment with different initializations based on the Entropy score and clustering. We evaluate this initialization on CIFAR10 using DM.

TSNE illustration of distributions of different initializations can be seen in Figure 4.15. High entropy scores select mostly samples on the class edges and on borders between two classes, while low entropy concentrates on high-confidence samples that are mostly clustered.

Initialization	**Initial performance**	**Final performance**
High Entropy	9.16 ± 0.2	26.16 ± 0.81
Low Entropy	19.94 ± 0.47	26.86 ± 1.06
Clustering	8.99 ± 0.35	26.25 ± 0.63
Random	13.64 ± 1.88	26.26 ± 0.58

Table 4.6: **Performance comparison of different initialization on CIFAR10.** We can see that initialization only affect the initial performance, while the final performance doesn't change much between different approaches.

High entropy initialization



Low entropy initialization

Figure 4.15: **TSNE illustration of different initializations.** In the Figure can be seen the TSNE plot of the first 3 CIFAR10 classes and images selected for initialization are marked with stars. We can observe that High entropy initialization (upper) concentrates on borders between classes, where hardest-to-classify samples are, while low entropy initializations (bottom) are mostly clustered in high confidence areas.

Results can be seen in Table 4.6. We observe that initialization has a strong effect on the initial performance. Samples selected by the lowest entropy scores result in the best initial performance. But the final performance of the distilled images does not change much between different initialization methods. It also did not help in speeding up the distillation process itself, thus we decided not to explore this direction further.

5 Conclusion

In this work, we introduced Generative dataset distillation. Our approach combines the advantages of classical dataset distillation, factorization methods and generative modelling. Rather than creating a synthetic dataset that is limited in size and thus prone to overfitting, we train a generative network, that would be used for synthetic samples generation. We borrow the generator architecture from StyleGAN2 and StyleGAN-XL, which already proved to be capable of generating state-of-the-art samples in terms of image quality. As a training objective, we used distribution matching and training trajectory matching, which have been shown to produce informative synthetic samples.

Our experiments on CIFAR10 demonstrated an increase in performance over pure DM and TTM but underperformed compared to the state-of-the-art methods, which we argue is caused by the strict memory constraints on the small datasets. To check this hypothesis we trained smaller and larger GANs on ImageNet subsets and compared both performance on the downstream task and FID scores. Our results indicate that small GAN is extremely hard to train, resulting in mode collapse on every try, increasing generator size fixes this issue and significantly improves FID and performance. This motivated us to focus on the datasets with more classes and larger image resolution and we chose ImageNet for our further experiments.

ImageNet is a long-standing goal of dataset distillation, but most of the approaches do not even scale to such dataset size, while the ones that scale perform poorly and are able to create only extremely small synthetic datasets. We showed that off-the-shelf StyleGAN-XL is already an incredibly strong baseline for ImageNet distillation, even though it was trained with GAN objective. Unfortunately, our GenDM approach did not achieve such good results - it performs worse than off-the-shelf StyleGAN-XL on ImageNet and on Imagenette although it improves performance over pure GAN, it still falls short in comparison to DM. We argue that the reason behind this might be the lack of supervision - the DM objective is not capable of training such large generative networks, thus it requires us either to add additional supervision (GAN objective) or shift from DM to other matching objectives, that require much more compute.

In our experiments, we also explore different sampling strategies and the effect of

the truncation value on performance. We find that not using truncation is optimal while creating a large synthetic dataset prior to the training. Regenerating synthetic samples after some amount of updates do not increase the performance while adding time to the training. We find truncation helpful only in the case of very small synthetic datasets.

We conducted experiments comparing the effect of different tweaks on the classical dataset distillation approaches. We tried factorization, parameter pruning and different distance metrics for matching objectives. Most of the tweaks do not contribute to the results while using the L1 distance metric for DM on CIFAR10 significantly increases the performance for multiple datasets. We argue that this might be caused by the meaninglessness of the L2 metric in higher dimensional spaces.

Finally, we conclude with experiments on the CarRacing task. Dataset distillation was not previously applied to this task. We show that dataset distillation is able to achieve incredible performance on simple datasets with low variance allowing to condense datasets of more than 12k samples into just 6 while retaining more than 85% of the original performance. This also demonstrates that dataset distillation is applicable beyond classical classification tasks.

As further possible research directions, we outline using additional supervision with GenDM on ImageNet or trying to use different matching objectives. We believe that generative dataset distillation still has a lot of potential and there are multiple possibilities that have not been tried out in this work. We also find extremely interesting continuing research in the direction of autonomous driving, as CarRacing was a good step in that direction.

Bibliography

[AAKW22] Ehsan Amid, Rohan Anil, Wojciech Kotłowski, and Manfred Warmuth. Learning from randomly initialized neural network features, 02 2022.

[AHK01] Charu C. Aggarwal, Alexander Hinneburg, and Daniel A. Keim. On the surprising behavior of distance metrics in high dimensional spaces. In *Proceedings of the 8th International Conference on Database Theory*, ICDT '01, page 420–434, Berlin, Heidelberg, 2001. Springer-Verlag.

[BCP+16] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

[BDS19] Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale gan training for high fidelity natural image synthesis, 2019.

[BP20] Eden Belouadah and Adrian Popescu. Scail: Classifier weights scaling for class incremental learning. In *2020 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 1255–1264, 2020.

[CMG+18] Francisco M. Castro, Manuel J. Marín-Jiménez, Nicolás Guil, Cordelia Schmid, and Karteek Alahari. End-to-end incremental learning. *CoRR*, abs/1807.09536, 2018.

[CWMG18] Weipeng Cao, Xizhao Wang, Zhong Ming, and Jinzhu Gao. A review on neural networks with random weights. *Neurocomputing*, 275:278–287, 2018.

[CWS12] Yutian Chen, Max Welling, and Alexander J. Smola. Super-samples from kernel herding. *CoRR*, abs/1203.3472, 2012.

[CWT+22] George Cazenavette, Tongzhou Wang, Antonio Torralba, Alexei A. Efros, and Jun-Yan Zhu. Dataset distillation by matching training trajectories, 2022.

[CWT+23] George Cazenavette, Tongzhou Wang, Antonio Torralba, Alexei A. Efros, and Jun-Yan Zhu. Generalizing dataset distillation via deep generative prior, 2023.

[DCSF15] Emily Denton, Soumith Chintala, Arthur Szlam, and Rob Fergus. Deep generative image models using a laplacian pyramid of adversarial networks, 2015.

[DDS+09] 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*, pages 248–255. Ieee, 2009.

[Den12] Li Deng. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

[DJT+23] Jiawei Du, Yidi Jiang, Vincent Y. F. Tan, Joey Tianyi Zhou, and Haizhou Li. Minimizing the accumulated trajectory error to improve dataset distillation, 2023.

[DN21] Prafulla Dhariwal and Alex Nichol. Diffusion models beat gans on image synthesis. *CoRR*, abs/2105.05233, 2021.

[DPS+18] Vincent Dumoulin, Ethan Perez, Nathan Schucher, Florian Strub, Harm de Vries, Aaron Courville, and Yoshua Bengio. Feature-wise transformations. *Distill*, 2018. https://distill.pub/2018/feature-wise-transformations.

[DR22] Zhiwei Deng and Olga Russakovsky. Remember the past: Distilling datasets into addressable memories for neural networks. In *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

[GBR+12] Arthur Gretton, Karsten M. Borgwardt, Malte J. Rasch, Bernhard Schölkopf, and Alexander Smola. A kernel two-sample test. *Journal of Machine Learning Research*, 13(25):723–773, 2012.

[GLK+17] Golnaz Ghiasi, Honglak Lee, Manjunath Kudlur, Vincent Dumoulin, and Jonathon Shlens. Exploring the structure of a real-time, arbitrary neural artistic stylization network, 2017.

[GPAM+14] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014.

[GSB16] Raja Giryes, Guillermo Sapiro, and Alex M. Bronstein. Deep neural networks with random gaussian weights: A universal classification strategy? *IEEE Transactions on Signal Processing*, 64(13):3444–3457, 2016.

[HB17] Xun Huang and Serge Belongie. Arbitrary style transfer in real-time with adaptive instance normalization, 2017.

[HHLP20] Erik Härkönen, Aaron Hertzmann, Jaakko Lehtinen, and Sylvain Paris. Ganspace: Discovering interpretable gan controls, 2020.

[HJA20] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models, 2020.

[HSC+21] Jonathan Ho, Chitwan Saharia, William Chan, David J. Fleet, Mohammad Norouzi, and Tim Salimans. Cascaded diffusion models for high fidelity image generation, 2021.

[KAL+21] Tero Karras, Miika Aittala, Samuli Laine, Erik Härkönen, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Alias-free generative adversarial networks, 2021.

[KALL18a] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of gans for improved quality, stability, and variation, 2018.

[KALL18b] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of gans for improved quality, stability, and variation, 2018.

[KH+09] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[KKO+22] 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 International Conference on Machine Learning (ICML)*, pages 11102–11118, 2022.

[KLA19] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks, 2019.

[KLA+20] Tero Karras, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Analyzing and improving the image quality of stylegan, 2020.

[KW20] Animesh Karnewar and Oliver Wang. Msg-gan: Multi-scale gradients for generative adversarial networks, 2020.

[LCJ+22] Saehyung Lee, Sanghyuk Chun, Sangwon Jung, Sangdoo Yun, and Sungroh Yoon. Dataset condensation with contrastive signals, 2022.

[LHAR22] Noel Loo, Ramin Hasani, Alexander Amini, and Daniela Rus. Efficient dataset distillation using random feature approximation, 2022.

[LL21] Yongqi Li and Wenjie Li. Data distillation for text classification. *arXiv preprint arXiv:2104.08448*, 2021.

[LLSD19] Alexandre Lacoste, Alexandra Luccioni, Victor Schmidt, and Thomas Dandres. Quantifying the carbon emissions of machine learning. *CoRR*, abs/1910.09700, 2019.

[LTOH22] Guang Li, Ren Togo, Takahiro Ogawa, and Miki Haseyama. Dataset distillation using parameter pruning. *arXiv preprint arXiv:2209.14609*, 2022.

[LWY+22] Songhua Liu, Kai Wang, Xingyi Yang, Jingwen Ye, and Xinchao Wang. Dataset distillation via factorization. In *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

[LZSE21a] Bingchen Liu, Yizhe Zhu, Kunpeng Song, and Ahmed Elgammal. Towards faster and stabilized gan training for high-fidelity few-shot image synthesis, 2021.

[LZSE21b] Bingchen Liu, Yizhe Zhu, Kunpeng Song, and Ahmed Elgammal. Towards faster and stabilized gan training for high-fidelity few-shot image synthesis, 2021.

[MKKY18] Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral normalization for generative adversarial networks, 2018.

[NCL21a] Timothy Nguyen, Zhourong Chen, and Jaehoon Lee. Dataset meta-learning from kernel ridge-regression. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021.

[NCL21b] Timothy Nguyen, Zhourong Chen, and Jaehoon Lee. Dataset meta-learning from kernel ridge-regression, 2021.

[ND21] Alex Nichol and Prafulla Dhariwal. Improved denoising diffusion probabilistic models, 2021.

[NDR+22] Alex Nichol, Prafulla Dhariwal, Aditya Ramesh, Pranav Shyam, Pamela Mishkin, Bob McGrew, Ilya Sutskever, and Mark Chen. Glide: Towards photorealistic image generation and editing with text-guided diffusion models, 2022.

[NH10] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *International Conference on Machine Learning*, 2010.

[NNXL21] Timothy Nguyen, Roman Novak, Lechao Xiao, and Jaehoon Lee. Dataset distillation with infinitely wide convolutional networks. In *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS)*, pages 5186–5198, 2021.

[NNXL22] Timothy Nguyen, Roman Novak, Lechao Xiao, and Jaehoon Lee. Dataset distillation with infinitely wide convolutional networks, 2022.

[RBL+22] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models, 2022.

[SBS+23] Seungjae Shin, Heesun Bae, Donghyeok Shin, Weonyoung Joo, and Il-Chul Moon. Loss-curvature matching for dataset selection and condensation, 2023.

[SCMG21] Axel Sauer, Kashyap Chitta, Jens Müller, and Andreas Geiger. Projected gans converge faster, 2021.

[SKC+11] Andrew M. Saxe, Pang Wei Koh, Zhenghao Chen, Maneesh Bhand, Bipin Suresh, and Andrew Y. Ng. On random weights and unsupervised feature learning. In *Proceedings of the 28th International Conference on International Conference on Machine Learning*, ICML'11, page 1089–1096, Madison, WI, USA, 2011. Omnipress.

[SSG22] Axel Sauer, Katja Schwarz, and Andreas Geiger. Stylegan-xl: Scaling stylegan to large diverse datasets, 2022.

[SWMG15] Jascha Sohl-Dickstein, Eric A. Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. *CoRR*, abs/1503.03585, 2015.

[TL20] Mingxing Tan and Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks, 2020.

[TO17] Corentin Tallec and Yann Ollivier. Unbiasing truncated backpropagation through time, 2017.

[TSdC+18] 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. *CoRR*, abs/1812.05159, 2018.

[TSM+20] Matthew Tancik, Pratul P. Srinivasan, Ben Mildenhall, Sara Fridovich-Keil, Nithin Raghavan, Utkarsh Singhal, Ravi Ramamoorthi, Jonathan T. Barron, and Ren Ng. Fourier features let networks learn high frequency functions in low dimensional domains, 2020.

[UVL17] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Instance normalization: The missing ingredient for fast stylization, 2017.

[vdOVK18] Aaron van den Oord, Oriol Vinyals, and Koray Kavukcuoglu. Neural discrete representation learning, 2018.

[Wer90] P.J. Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.

[WZP+22] 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 (CVPR)*, pages 12196–12205, 2022.

[WZTE18a] Tongzhou Wang, Jun-Yan Zhu, Antonio Torralba, and Alexei A. Efros. Dataset distillation. *CoRR*, abs/1811.10959, 2018.

[WZTE18b] Tongzhou Wang, Jun-Yan Zhu, Antonio Torralba, and Alexei A. Efros. Dataset distillation. *arXiv preprint arXiv:1811.10959*, 2018.

[XWC+20] Rui Xu, Xintao Wang, Kai Chen, Bolei Zhou, and Chen Change Loy. Positional encoding as spatial inductive bias in gans, 2020.

[ZB21a] Bo Zhao and Hakan Bilen. Dataset condensation with differentiable siamese augmentation. *CoRR*, abs/2102.08259, 2021.

[ZB21b] Bo Zhao and Hakan Bilen. Dataset condensation with differentiable siamese augmentation. In *International Conference on Machine Learning*, 2021.

[ZB21c] Bo Zhao and Hakan Bilen. Dataset condensation with gradient matching. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021.

[ZB22] Bo Zhao and Hakan Bilen. Synthesizing informative training samples with gan. *NeurIPS 2022 Workshop on Synthetic Data for Empowering ML Research*, 2022.

[ZB23] Bo Zhao and Hakan Bilen. Dataset condensation with distribution matching. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 6514–6523, 2023.

[ZKHB21] Xiaohua Zhai, Alexander Kolesnikov, Neil Houlsby, and Lucas Beyer. Scaling vision transformers. *CoRR*, abs/2106.04560, 2021.

[ZMB20] Bo Zhao, Konda Reddy Mopuri, and Hakan Bilen. Dataset condensation with gradient matching. *CoRR*, abs/2006.05929, 2020.

[ZNB22] Yongchao Zhou, Ehsan Nezhadarya, and Jimmy Ba. Dataset distillation using neural feature regression. In *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS)*, 2022.