Dataset Distillation as Optimal Quantisation

Hong Ye Tan

September 30, 2025

Joint work with Emma Slade @ GSK.ai

Intro

Dataset Distillation: what and why?

Imagine this very specific scenario:

- You have a big dataset.

Dataset Distillation: what and why?

Imagine this very specific scenario:

- You have a big dataset.
- You need to train a lot of models.

Dataset Distillation: what and why?

Imagine this very specific scenario:

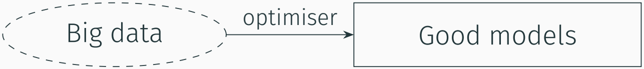
- You have a big dataset.
- You need to train a lot of models.
- You can take a small performance hit as long as it trains quickly.

Dataset Distillation: what and why?

Imagine this very specific scenario:

- You have a big dataset.
- You need to train a lot of models.
- You can take a small performance hit as long as it trains quickly.

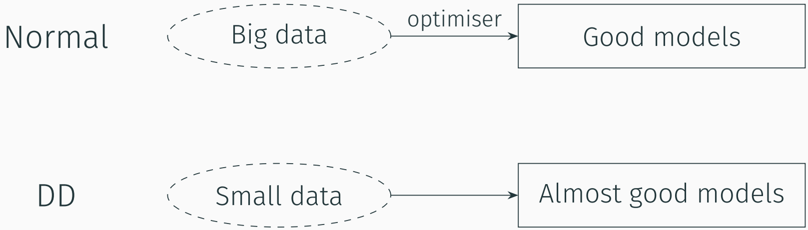
Normal



Dataset Distillation: what and why?

Imagine this very specific scenario:

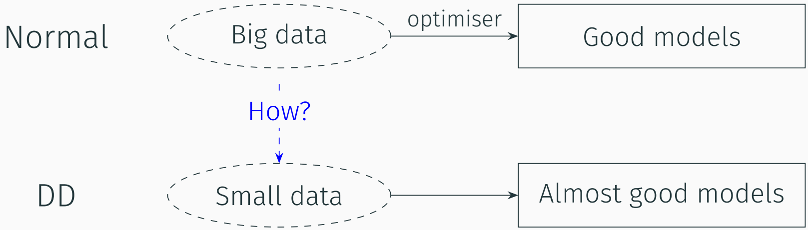
- You have a big dataset.
- You need to train a lot of models.
- You can take a small performance hit as long as it trains quickly.



Dataset Distillation: what and why?

Imagine this very specific scenario:

- You have a big dataset.
- You need to train a lot of models.
- You can take a small performance hit as long as it trains quickly.



How?

Problem: find a small surrogate dataset, so that training on it gives performance similar to training on the whole dataset

How?

Problem: find a small surrogate dataset, so that training on it gives ~~performance similar to training on the whole dataset~~

How?

Problem: find a small surrogate dataset, so that training on it gives good performance

How?

Problem: find a small surrogate dataset, so that training on it gives good performance

Intuition: you want the *most important* data.

Main acceleration: since your surrogate dataset is small, you train faster.

How?

Problem: find a small surrogate dataset, so that training on it gives good performance

Intuition: you want the *most important* data.

Main acceleration: since your surrogate dataset is small, you train faster.
Two main archetypes:

1. Bi-level approaches
2. Disentangled approaches

How?

Problem: find a small surrogate dataset, so that training on it gives good performance

Intuition: you want the *most important* data.

Main acceleration: since your surrogate dataset is small, you train faster.
Two main archetypes:

1. **Bi-level approaches**
2. Disentangled approaches

Bi-level approaches (Terrible)

Dataset distillation is:

Bi-level approaches (Terrible)

Dataset distillation is:

- Optimise over the distilled dataset $\mathcal{S}$:

$$\arg\min_{\mathcal{S}}$$

Bi-level approaches (Terrible)

Dataset distillation is:

- Optimise over the distilled dataset $\mathcal{S}$:
- such that when you train on $\mathcal{S}$,

$$\operatorname*{arg\,min}_{\mathcal{S}} \qquad\qquad \operatorname*{arg\,min}_{\theta} \mathrm{TrainingLoss}_{\mathcal{S}}(\theta))$$

Bi-level approaches (Terrible)

Dataset distillation is:

- Optimise over the distilled dataset $\mathcal{S}$: (Outer loop)
- such that when you train on $\mathcal{S}$, (Inner loop)
- you get good performance. (Test error)

$$\arg\min_{\mathcal{S}} \mathrm{TestError}(\arg\min_{\theta} \mathrm{TrainingLoss}_{\mathcal{S}}(\theta))$$

Limitations of the bi-level approach

1. Does not scale to large training datasets

Limitations of the bi-level approach

1. Does not scale to large training datasets
2. Scales with the number of images in your distilled dataset

Limitations of the bi-level approach

1. Does not scale to large training datasets
2. Scales with the number of images in your distilled dataset
3. Is itself a difficult high-dimensional optimisation problem
4. Opaque mathematical interpretation
5. Requires hand-crafted heuristics

Limitations of the bi-level approach

1. Does not scale to large training datasets
2. Scales with the number of images in your distilled dataset
3. Is itself a difficult high-dimensional optimisation problem
4. Opaque mathematical interpretation
5. Requires hand-crafted heuristics
6. No theoretical guarantees
7. Does not transfer well between model architectures
8. Visually incoherent distillates (not really a limitation)

Limitations of the bi-level approach

1. Does not scale to large training datasets
2. Scales with the number of images in your distilled dataset
3. Is itself a difficult high-dimensional optimisation problem
4. Opaque mathematical interpretation
5. Requires hand-crafted heuristics
6. No theoretical guarantees
7. Does not transfer well between model architectures
8. Visually incoherent distillates (not really a limitation)
9. Need to pre-train lots of expert models
10. High storage requirements from storing checkpoints of expert models

Limitations of the bi-level approach

1. Computational scaling
2. Poor mathematical interpretation
3. Questionable architecture generalisation
4. No mathematical guarantees

Some examples of what people do

$$\arg\min_{\mathcal{S}} \mathrm{TestError}(\arg\min_{\theta} \mathrm{TrainingLoss}_{\mathcal{S}}(\theta)) \tag{DD}$$

Table 1: Test accuracy with 10 images per class.

Dataset	Default
CIFAR-10	84.8
CIFAR-100	56.2

Some examples of what people do

$$\arg\min_{\mathcal{S}} \mathrm{TestError}(\arg\min_{\theta} \mathrm{TrainingLoss}_{\mathcal{S}}(\theta)) \tag{DD}$$

Gradient matching: replace the (intractable) training loss minimiser with a normal training regime

$$\arg\min_{\mathcal{S}} \mathrm{TestError}(\mathrm{Train\_N\_Epochs}_{\mathcal{S}}(\theta)) \tag{GM}$$

Table 1: Test accuracy with 10 images per class.

Dataset	Default	GM
CIFAR-10	84.8	44.9
CIFAR-100	56.2	25.3

Some examples of what people do

$$\arg\min_{\mathcal{S}} \mathrm{TestError}(\arg\min_{\theta} \mathrm{TrainingLoss}_{\mathcal{S}}(\theta)) \qquad \text{(DD)}$$

Distribution matching:

Table 1: Test accuracy with 10 images per class.

Dataset	Default	GM
CIFAR-10	84.8	44.9
CIFAR-100	56.2	25.3

Some examples of what people do

$$\arg\min_{\mathcal{S}} \mathrm{TestError}(\arg\min_{\theta} \mathrm{TrainingLoss}_{\mathcal{S}}(\theta)) \tag{DD}$$

Distribution matching: match the distributions of the synthetic $\mathcal{S}$ and the training $\mathcal{T}$

Table 1: Test accuracy with 10 images per class.

Dataset	Default	GM
CIFAR-10	84.8	44.9
CIFAR-100	56.2	25.3

Some examples of what people do

$$\arg\min_{\mathcal{S}} \mathrm{TestError}(\arg\min_{\theta} \mathrm{TrainingLoss}_{\mathcal{S}}(\theta)) \tag{DD}$$

Distribution matching: match the distributions of the synthetic $\mathcal{S}$ and the training $\mathcal{T}$ … after passing through randomly initialised neural networks (?)

Table 1: Test accuracy with 10 images per class.

Dataset	Default	GM
CIFAR-10	84.8	44.9
CIFAR-100	56.2	25.3

Some examples of what people do

$$\arg\min_{\mathcal{S}} \mathrm{TestError}(\arg\min_{\theta} \mathrm{TrainingLoss}_{\mathcal{S}}(\theta)) \tag{DD}$$

Distribution matching: match the distributions of the synthetic $\mathcal{S}$ and the training $\mathcal{T}$ … after passing through randomly initialised neural networks (?)

$$\arg\min_{\mathcal{S}} \mathbb{E}_{\theta\sim p_\theta} \| \frac{1}{|\mathcal{T}|} \sum_{x\in\mathcal{T}} \psi_\theta(x) - \frac{1}{|\mathcal{S}|} \sum_{x\in\mathcal{S}} \psi_\theta(x) \|^2 \quad (??) \tag{DM}$$

Table 1: Test accuracy with 10 images per class.

Dataset	Default	GM	DM
CIFAR-10	84.8	44.9	48.9
CIFAR-100	56.2	25.3	29.7

Some examples of what people do

$$\arg\min_{\mathcal{S}} \mathrm{TestError}(\arg\min_{\theta} \mathrm{TrainingLoss}_{\mathcal{S}}(\theta)) \qquad \text{(DD)}$$

Matching training trajectories: minimise the ℓ_2 distance of the *parameters* when training with $\mathcal{S}$ vs full training $\mathcal{T}$

$$\arg\min_{\mathcal{S}} \mathbb{E}_{\theta_0 \sim p_\theta} \sum_{t=1}^{T-M} \frac{\|\theta^{\mathcal{S}}_{t+N} - \theta^{\mathcal{T}}_{t+M}\|^2}{\|\theta^{\mathcal{T}}_{t+M} - \theta^{\mathcal{T}}_{t}\|^2}$$

Table 1: Test accuracy with 10 images per class.

Dataset	Default	GM	DM
CIFAR-10	84.8	44.9	48.9
CIFAR-100	56.2	25.3	29.7

Some examples of what people do

$$\arg\min_{\mathcal{S}} \mathrm{TestError}(\arg\min_{\theta} \mathrm{TrainingLoss}_{\mathcal{S}}(\theta)) \quad \text{(DD)}$$

Matching training trajectories: minimise the ℓ_2 distance of the *parameters* when training with $\mathcal{S}$ vs full training $\mathcal{T}$ (so you have to train and store the parameters at every training iteration...

$$\arg\min_{\mathcal{S}} \mathbb{E}_{\theta_0 \sim p_\theta} \sum_{t=1}^{T-M} \frac{\|\theta^{\mathcal{S}}_{t+N} - \theta^{\mathcal{T}}_{t+M}\|^2}{\|\theta^{\mathcal{T}}_{t+M} - \theta^{\mathcal{T}}_{t}\|^2}$$

Table 1: Test accuracy with 10 images per class.

Dataset	Default	GM	DM	MTT
CIFAR-10	84.8	44.9	48.9	65.3
CIFAR-100	56.2	25.3	29.7	40.1

Some examples of what people do

$$\arg\min_{\mathcal{S}} \mathrm{TestError}(\arg\min_{\theta} \mathrm{TrainingLoss}_{\mathcal{S}}(\theta)) \quad \text{(DD)}$$

Matching training trajectories: minimise the ℓ_2 distance of the *parameters* when training with $\mathcal{S}$ vs full training $\mathcal{T}$ (so you have to train and store the parameters at every training iteration...for a lot of different network initializations)

$$\arg\min_{\mathcal{S}} \mathbb{E}_{\theta_0 \sim p_\theta} \sum_{t=1}^{T-M} \frac{\|\theta^{\mathcal{S}}_{t+N} - \theta^{\mathcal{T}}_{t+M}\|^2}{\|\theta^{\mathcal{T}}_{t+M} - \theta^{\mathcal{T}}_{t}\|^2}$$

Table 1: Test accuracy with 10 images per class.

Dataset	Default	GM	DM	MTT
CIFAR-10	84.8	44.9	48.9	65.3
CIFAR-100	56.2	25.3	29.7	40.1

Synthetic images from bi-level methods

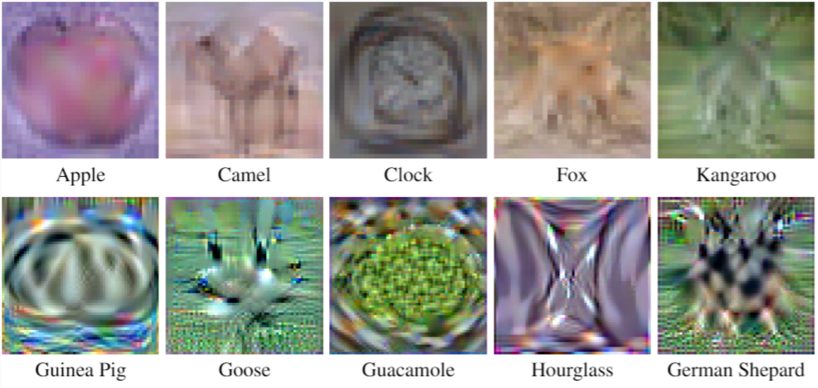


Figure 1: What are these? Directly optimised in image space.

Feature distribution

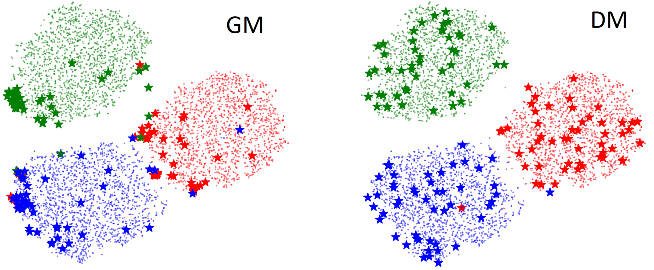


Figure 2: Distribution of synthetic images in CIFAR-10 [Zhao and Bilen '23].

Clustering? Seems like dealing with the data is better than forcing a particular architecture.

Motivation

If distribution matching is "like clustering" $\stackrel{?}{\rightarrow}$ use clustering for DD?

Motivation

If distribution matching is "like clustering" $\overset{?}{\to}$ use clustering for DD?

Yes!

Some other ways you might arrive here:

- (Problem) expensive bi-level formulation.

Motivation

If distribution matching is "like clustering" $\overset{?}{\to}$ use clustering for DD?

Yes!

Some other ways you might arrive here:

- (Problem) expensive bi-level formulation.
  - (Solution) remove either the inner or outer problem.

Motivation

If distribution matching is "like clustering" $\overset{?}{\rightarrow}$ use clustering for DD?

Yes!

Some other ways you might arrive here:

- (Problem) expensive bi-level formulation.
  - (Solution) remove either the inner or outer problem.
- (Problem) dependence on the training data

Motivation

If distribution matching is “like clustering” $\xrightarrow{?}$ use clustering for DD?

Yes!

Some other ways you might arrive here:

- (Problem) expensive bi-level formulation.
    - (Solution) remove either the inner or outer problem.
- (Problem) dependence on the training data
    - (Solution) use pre-trained models that already have the knowledge of the training data

Motivation

If distribution matching is "like clustering" $\stackrel{?}{\rightarrow}$ use clustering for DD?

Yes!

Some other ways you might arrive here:

- (Problem) expensive bi-level formulation.
  - (Solution) remove either the inner or outer problem.
- (Problem) dependence on the training data
  - (Solution) use pre-trained models that already have the knowledge of the training data
- (Problem) high-dimensional optimisation

Motivation

If distribution matching is “like clustering” $\xrightarrow{?}$ use clustering for DD?

Yes!

Some other ways you might arrive here:

- (Problem) expensive bi-level formulation.
  - (Solution) remove either the inner or outer problem.
- (Problem) dependence on the training data
  - (Solution) use pre-trained models that already have the knowledge of the training data
- (Problem) high-dimensional optimisation
  - (Solution) actually, remove the outer problem by generating the distilled dataset.
- (Problem?) strange looking distilled images

Motivation

If distribution matching is "like clustering" $\xrightarrow{?}$ use clustering for DD?

Yes!

Some other ways you might arrive here:

- (Problem) expensive bi-level formulation.
  - (Solution) remove either the inner or outer problem.
- (Problem) dependence on the training data
  - (Solution) use pre-trained models that already have the knowledge of the training data
- (Problem) high-dimensional optimisation
  - (Solution) actually, remove the outer problem by generating the distilled dataset.
- (Problem?) strange looking distilled images
  - (Solution?) force the distilled images to look good using generative models.

“Disentangled dataset distillation”

How to use clustering for DD?

- Direct clustering of the big images does not work. (curse of dimensionality)

“Disentangled dataset distillation”

How to use clustering for DD?

- Direct clustering of the big images does not work. (curse of dimensionality)

How about using an encoder-decoder structure? Then you can cluster in the latent space, and you have generative capabilities!

“Disentangled dataset distillation”

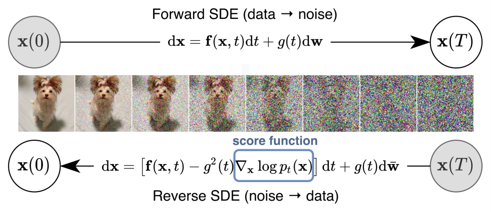
How to use clustering for DD?

- Direct clustering of the big images does not work. (curse of dimensionality)

How about using an encoder-decoder structure? Then you can cluster in the latent space, and you have generative capabilities!

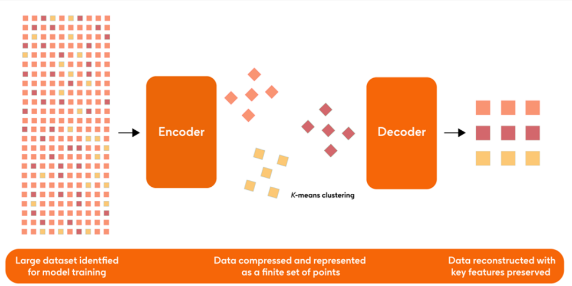
What you want: **latent diffusion models**

- Decoders given by conditional diffusion models



Algorithm: you have a encoder-decoder pair $(\mathcal{E}, \mathcal{D})$.

1. Take your entire training dataset $\mathcal{T}$, and encode it to get a dataset of latent variables $\mathcal{Z} = \mathcal{E}(\mathcal{T})$

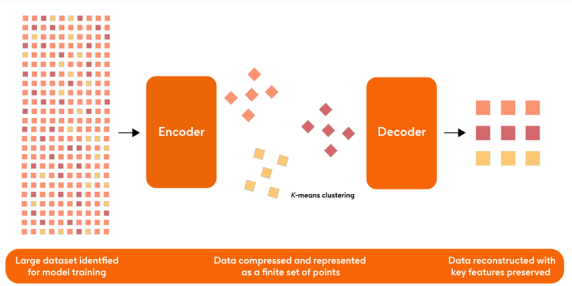


GSK.ai blog!

Dataset Distillation by clustering

Algorithm: you have a encoder-decoder pair $(\mathcal{E}, \mathcal{D})$.

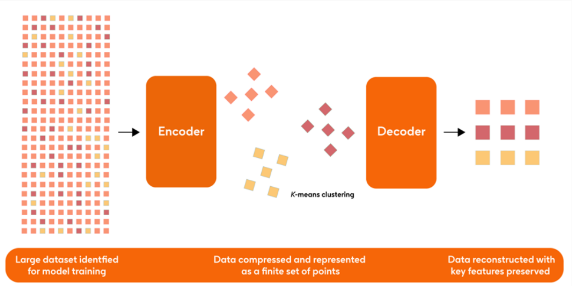
1. Take your entire training dataset $\mathcal{T}$, and encode it to get a dataset of latent variables $\mathcal{Z} = \mathcal{E}(\mathcal{T})$
2. Cluster your latent variables in each class, say with k-means. You get some "centroid" latents in each class $\hat{\mathcal{Z}}$.



GSK.ai blog!

Algorithm: you have a encoder-decoder pair $(\mathcal{E}, \mathcal{D})$.

1. Take your entire training dataset $\mathcal{T}$, and encode it to get a dataset of latent variables $\mathcal{Z} = \mathcal{E}(\mathcal{T})$
2. Cluster your latent variables in each class, say with k-means. You get some "centroid" latents in each class $\hat{\mathcal{Z}}$.
3. Decode these clustered latents $\mathcal{S} = \mathcal{D}(\hat{\mathcal{Z}})$.



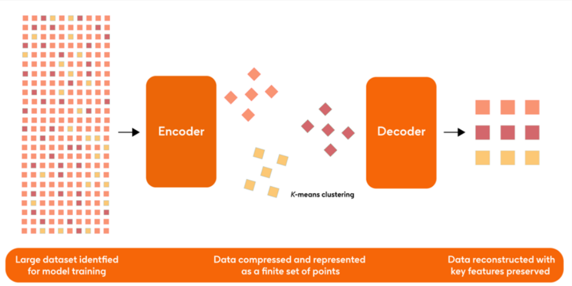
GSK.ai blog!

Dataset Distillation by clustering

Algorithm: you have a encoder-decoder pair $(\mathcal{E}, \mathcal{D})$.

1. Take your entire training dataset $\mathcal{T}$, and encode it to get a dataset of latent variables $\mathcal{Z} = \mathcal{E}(\mathcal{T})$
2. Cluster your latent variables in each class, say with k-means. You get some "centroid" latents in each class $\hat{\mathcal{Z}}$.
3. Decode these clustered latents $\mathcal{S} = \mathcal{D}(\hat{\mathcal{Z}})$.

Oops, someone has done this before [Su et al., CVPR'24].



GSK.ai blog!

DDOQ

A closer look: decoding

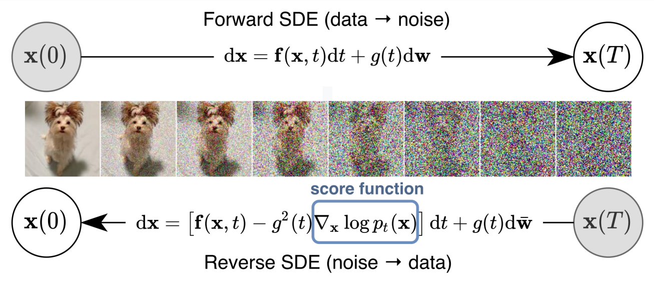
3. Decode these clustered latents $\mathcal{S} = \mathcal{D}(\hat{\mathcal{Z}})$.

What is the decoding the clusters actually doing?

A look at diffusion models.

Generation from latents

Data distribution μ_0 —(1)→ Latents distribution μ_T



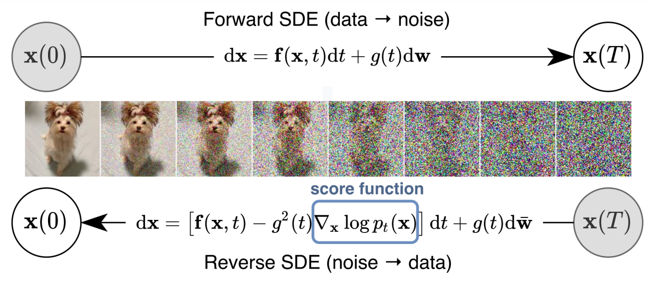
(2) Clustering

Distilled distribution ν_0 ←(3)— Latents cluster distribution ν_T

Generation from latents

Data distribution μ_0 —(1)→ Latents distribution μ_T

(2) Clustering



Distilled distribution ν_0 ←(3)— Latents cluster distribution ν_T

Consistency

Theorem
For the VESDE/VPSDE and any initial data distribution $\mu \in \mathcal{P}_2(\mathbb{R}^d)$ *with compact support bounded by* $R > 0$*, the backwards diffusion process is well posed. Suppose that there are two distributions* μ_T, ν_T *at time* T *that undergo the reverse diffusion process (with fixed initial reference measure* μ*) up to time* $t = \delta \in (0, T)$ *to produce distributions* μ_δ, ν_δ*. There exists a (universal explicit) constant* $C = C(\delta, T, R, d) \in (0, +\infty)$ *such that if* $f : \mathbb{R}^d \to \mathbb{R}^n$ *is an L-Lipschitz function, then the difference in expectation satisfies*

$$\|\mathbb{E}_{\mu_\delta}[f] - \mathbb{E}_{\nu_\delta}[f]\| \leq CL\mathcal{W}_2(\mu_T, \nu_T).$$

Proof: basically stochastic Gronwall's inequality.

Consistency

Theorem (informal)

Suppose you have a bounded data distribution μ_0, say of images. Then you pass it through your forward stochastic process (noising), up to μ_T. If you have another distribution ν_T that approximates μ_T, then after passing both through the reverse stochastic process (generation), your generated distributions ν_δ will approximate μ_δ.

Consistency

Theorem (informal)

Suppose you have a bounded data distribution μ_0, say of images. Then you pass it through your forward stochastic process (noising), up to μ_T. If you have another distribution ν_T that approximates μ_T, then after passing both through the reverse stochastic process (generation), your generated distributions ν_δ will approximate μ_δ.

If you are close in the latent space, then you are close after decoding/generation.

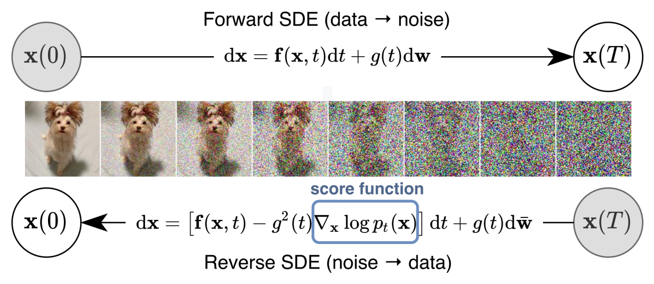
Consequences:

- Gradient steps when training with the distilled dataset are *automatically* similar to those on the full training dataset (supposedly).
- No need to enforce through training!

Generation from latents

Data distribution μ_0 —(1)→ Latents distribution μ_T

(2) Clustering

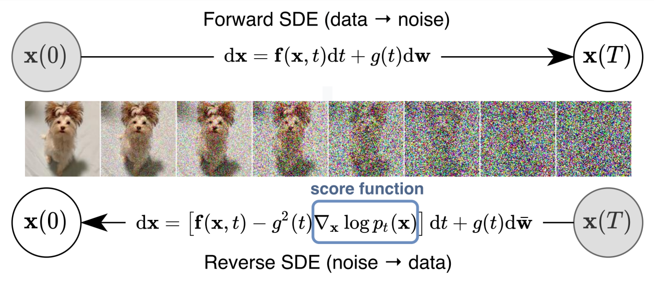


Distilled distribution ν_0 ←(3)— Latents cluster distribution ν_T

Generation from latents

Data distribution μ_0 —(1)→ Latents distribution μ_T

(2) Clustering



Distilled distribution ν_0 ←(3)— Latents cluster distribution ν_T

An even closer look: clustering

2. Cluster your latent variables in each class, say with k-means. You get some "centroid" latents in each class $\hat{\mathcal{Z}}$.

What is clustering actually doing?

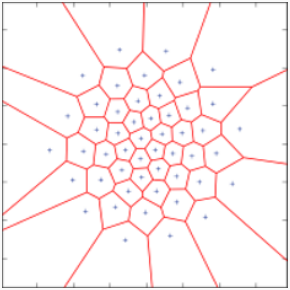
Clustering is optimal quantisation!

Definition
For a probability measure μ, the **quadratic distortion** of a set of points $\{x_1, ..., x_K\}$ is the μ-average distance to the set:

$$\mathcal{G} : (x_1, ..., x_K) \mapsto \int_{\mathbb{R}^d} \min_i \|x - x_i\|^2 \, \mu(\mathrm{d}x) = \mathbb{E}_{X\sim\mu}[\min_i \|X - x_i\|^2]. \quad (1)$$

The **optimal quantisation** of μ (at level K) is a set of points $\{x_1, ..., x_K\}$ that minimizes the quadratic distortion.

Fact: there is a 1-1 correspondence with optimal quantisers $\{x_1, ..., x_K\}$ and Wasserstein-minimisers with finite support.



Example quantisation of a Gaussian

Relating optimal quantisers to Wasserstein minimisers

Proposition
For a set of points $\{x_1, .., x_K\}$*, the empirical measure* $\nu = \sum_i w_i \delta(x_i)$ *minimising* $\mathcal{W}_2(\mu, \nu)$ *satisfying* $\operatorname{supp} \nu \subset \{x_1, .., x_K\}$ *is given by* $\nu = \sum_i \mu(C_i)\delta(x_i)$.[1]

- Sets of points $\leftrightarrow$ approximating measures (automatically)
- Rates: $\mathcal{W}_2(\nu_K, \mu) \sim \mathcal{O}(K^{-1/d})$ as number of points $K \to \infty$.

[1] $C_i \subset \mathbb{R}^d$ are the Voronoi cells, set of points closest to x_i.

Clustering is optimal quantisation!

Guess what is an algorithm for solving optimal quantisation?

[2] Actually, a slight modification thereof: the competitive learning vector quantisation (CLVQ) algorithm.

Clustering is optimal quantisation!

Guess what is an algorithm for solving optimal quantisation?
k-**means clustering**![2] Produces $\{x_1, ..., x_K\}$.

Previous approach: take the approximating measure to be

$$\nu = \frac{1}{K}\sum_{i=1}^{K}\delta(x_i) \quad \left(\approx \frac{1}{|\mathcal{T}|}\sum_{u\in\mathcal{T}}\delta(u) \approx \mu\right)$$

Key: finding $\{x_i\}$ is not enough! Actually, we need to optimise for w_i, x_i in

$$\nu = \arg\min \mathcal{W}_2\left(\sum_{i=1}^{K} w_i\delta(x_i), \mu\right).$$

[2]Actually, a slight modification thereof: the competitive learning vector quantisation (CLVQ) algorithm.

A tiny modification

So we want small $\mathcal{W}_2$ distance to give good approximation of the latents $\nu_T \approx \mu_T$:

$$\nu = \arg\min \mathcal{W}_2 \left(\sum_{i=1}^{K} w_i \delta(x_i), \mu \right).$$

How do we get the $w_i \approx \mu(C_i)$?

A tiny modification

So we want small $\mathcal{W}_2$ distance to give good approximation of the latents $\nu_T \approx \mu_T$:

$$\nu = \text{arg min}\, \mathcal{W}_2 \left(\sum_{i=1}^{K} w_i \delta(x_i), \mu \right).$$

How do we get the $w_i \approx \mu(C_i)$?

Answer: k-means does this automatically! Produces online approximations to $\mu(C_i)$.

We get better approximation (in measures) *with no additional computation*.

Why do we need the coefficients?

If w_i are uniform $w_i \equiv 1/K$, then this is the *Wasserstein barycentre* problem (as opposed to optimal quantisation)[3]

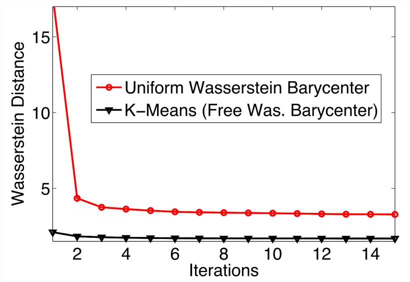
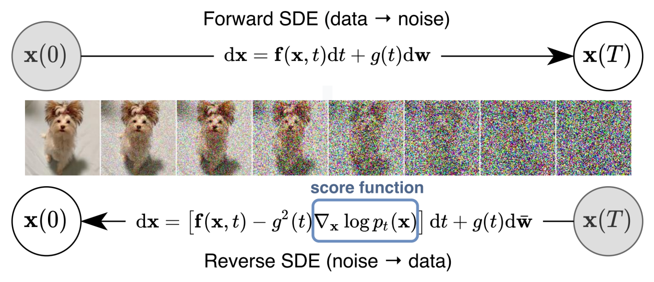


Figure 3: The Wasserstein distance $\mathcal{W}_2$ is higher for the Wasserstein barycentre. Courtesy of [3].

[3][Cuturi & Doucet ICML '14].

Generation from latents

Data distribution μ_0 —(1)→ Latents distribution μ_T



(2) Clustering

Distilled distribution ν_0 ←(3)— Latents cluster distribution ν_T

We have the optimal quantisers $\nu_T \approx \mu_T$ from (2), which implies $\nu_0 \approx \mu_0$ from (3).

Putting everything together

Algorithm: you have a encoder-decoder pair $(\mathcal{E}, \mathcal{D})$.

1. Take your entire training dataset $\mathcal{T}$, and encode it to get a dataset of latent variables $\mathcal{Z} = \mathcal{E}(\mathcal{T})$

Putting everything together

Algorithm: you have a encoder-decoder pair $(\mathcal{E}, \mathcal{D})$.

1. Take your entire training dataset $\mathcal{T}$, and encode it to get a dataset of latent variables $\mathcal{Z} = \mathcal{E}(\mathcal{T})$
2. Cluster your latent variables in each class, say with k-means. You get some "centroid" latents in each class $\hat{\mathcal{Z}}$, and some corresponding weights $W = \{w_z\}_{z \in \hat{\mathcal{Z}}}$.

Putting everything together

Algorithm: you have a encoder-decoder pair $(\mathcal{E}, \mathcal{D})$.

1. Take your entire training dataset $\mathcal{T}$, and encode it to get a dataset of latent variables $\mathcal{Z} = \mathcal{E}(\mathcal{T})$
2. Cluster your latent variables in each class, say with k-means. You get some "centroid" latents in each class $\hat{\mathcal{Z}}$, and some corresponding weights $W = \{w_z\}_{z \in \hat{\mathcal{Z}}}$.
3. Decode these clustered latents $\mathcal{S} = \mathcal{D}(\hat{\mathcal{Z}})$.

Putting everything together

Algorithm: you have a encoder-decoder pair $(\mathcal{E}, \mathcal{D})$.

1. Take your entire training dataset $\mathcal{T}$, and encode it to get a dataset of latent variables $\mathcal{Z} = \mathcal{E}(\mathcal{T})$
2. Cluster your latent variables in each class, say with k-means. You get some "centroid" latents in each class $\hat{\mathcal{Z}}$, and some corresponding weights $W = \{w_z\}_{z \in \hat{\mathcal{Z}}}$.
3. Decode these clustered latents $\mathcal{S} = \mathcal{D}(\hat{\mathcal{Z}})$.
4. When training with the distilled dataset $(\mathcal{S}, W)$, multiply the gradients of each training point with the corresponding weight.

ImageNet-1K: 1.2M images, 1000 classes, 150GB size.

Bi-level approaches: not possible.

- Previous maximum on 100K 128×128 images split into 200 classes, $\sim$ 600MB size.
- Already requires 80GB of GPU memory for 10 images per class.

Comparing the addition of the k-means weights:

IPC	k-means weights?	ResNet-18	ResNet-50	ResNet-101
10	✗	27.9	33.5	34.2
	✓	**33.1**	**34.4**	**36.7**
50	✗	55.2	62.4	63.4
	✓	**56.2**	**62.5**	**63.6**
100	✗	59.3	65.4	66.5
	✓	**60.1**	**65.9**	**66.7**
200	✗	62.6	67.8	68.1
	✓	**63.4**	**68.0**	**68.6**

Table 2: Dataset distillation test accuracy without and with the k-means weights. Adding weights makes it uniformly better.

Comparing against other methods

Table 3: Comparison of top-1 classification performance on the ImageNet-1K dataset for baselines versus the proposed DDOQ method at various IPCs. We observe that DDOQ outperforms the previous SOTA method D^{4}M, due to the addition of weights to the synthetic data. The maximum performance for all methods should be 69.8 as the soft labels are computed using a pre-trained ResNet-18 model.

IPC	Method	ResNet-18	ResNet-50	ResNet-101	IPC	Method	ResNet-18	ResNet-50	ResNet-101
	TESLA	7.7	-	-		SRe^{2}L	46.8$_{\pm0.2}$	55.6$_{\pm0.3}$	60.8$_{\pm0.5}$
	SRe^{2}L	21.3$_{\pm0.6}$	28.4$_{\pm0.1}$	30.9$_{\pm0.1}$		CDA	53.5	61.3	61.6
10	RDED	**42.0**$_{\pm0.1}$	-	**48.3**$_{\pm1.0}$	50	RDED	**56.5**$_{\pm0.1}$	-	61.2$_{\pm0.4}$
	D^{4}M	27.9	33.5	34.2		D^{4}M	55.2	*62.4*	*63.4*
	DDOQ	*33.1*$_{\pm0.60}$	*34.4*$_{\pm0.99}$	*36.7*$_{\pm0.80}$		DDOQ	*56.2*$_{\pm0.07}$	**62.5**$_{\pm0.24}$	**63.6**$_{\pm0.13}$
	SRe^{2}L	52.8$_{\pm0.3}$	61.0$_{\pm0.4}$	62.8$_{\pm0.2}$		SRe^{2}L	57.0$_{\pm0.4}$	64.6$_{\pm0.3}$	65.9$_{\pm0.3}$
100	CDA	58.0	65.1	65.9	200	CDA	*63.3*	67.6	*68.4*
	D^{4}M	*59.3*	*65.4*	*66.5*		D^{4}M	62.6	*67.8*	68.1
	DDOQ	**60.1**$_{\pm0.15}$	**65.9**$_{\pm0.15}$	**66.7**$_{\pm0.06}$		DDOQ	**63.4**$_{\pm0.08}$	**68.0**$_{\pm0.05}$	**68.6**$_{\pm0.08}$

What do the weights look like?

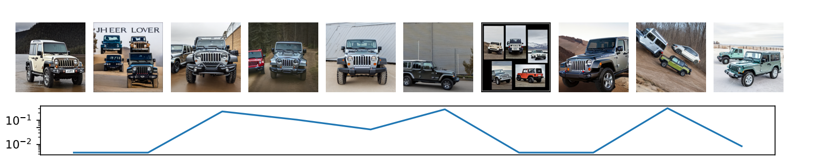


Figure 4: Example distilled images of the "jeep" class in ImageNet-1K along with their k-means weights. There is little to no features that can be used to differentiate the low and high weighted images, mainly due to the high fidelity of the diffusion model. However, the weights are indicative of the distribution of the training data in the latent space of the diffusion model.

Summary

1. Dataset distillation: turning big data into small data
2. By thinking a bit, we get
   i) … rid of any bi-level formulations;
   ii) **Theoretical perspective** as to why clustering-style methods work
   iii) **State of the art** performance for free

Summary

1. Dataset distillation: turning big data into small data
2. By thinking a bit, we get
   i) … rid of any bi-level formulations;
   ii) **Theoretical perspective** as to why clustering-style methods work
   iii) **State of the art** performance for free

What we really use/assume:

1. Generative (diffusion) models as implicitly modelling the underlying data distribution (model expressiveness/trainability)
2. Faithfulness of data distribution $\leftrightarrow$ latent space mapping (manifold hypothesis)

Appendix

Generalisation

Table 4: Generalization performance.

Teacher Network		Student Network			
		ResNet-18	MobileNet-V2	EfficientNet-B0	Swin-T
ResNet-18	D^4M	55.2	47.9	55.4	**58.1**
	DDOQ	**56.2**	**52.1**	**58.0**	57.4
MobileNet-V2	D^4M	47.6	42.9	49.8	**58.9**
	DDOQ	**47.7**	**45.6**	**52.5**	56.3
Swin-T	D^4M	27.5	21.9	26.4	**38.1**
	DDOQ	**28.5**	**24.1**	**29.3**	36.0

Algorithm 1: CLVQ

Data: initial cluster centers $x_1^{(0)}, ..., x_K^{(0)}$, step-sizes $(\gamma_i)_{i \geq 0}$, $i \leftarrow 0$

Initialize weights $\mathbf{w} = (w_1, ..., w_K) = (1/K, ..., 1/K)$;

while *not converged* **do**

  Sample $X_i \sim \mu$;

  Select "winner" $k_{\text{win}} \in \arg\min_{1 \leq k \leq K} \|X_i - x_k^{(i)}\|$;

  Update $x_k^{(i+1)} \leftarrow (1 - \gamma_i) x_k^{(i)} + \gamma_i X_i$ if $k = k_{\text{win}}$, otherwise $x_k^{(i+1)} \leftarrow x_k^{(i)}$;

  Update weights $w_k \leftarrow (1 - \gamma_i) w_k + \gamma_i \mathbf{1}_{k = k_{\text{win}}}$;

  $i \leftarrow i + 1$;

end

Result: quantization $\nu_K = \sum_{i=1}^{K} w_i \delta(x_i^*)$