CS 330 Project Report: Augmentation by Distillation

Sanjit Neelam
ICME
Stanford University
sneelam@stanford.edu

Abstract

Dataset distillation is a task where the goal is to synthesise a small dataset from a full dataset, such that a model trained on the small set will achieve almost the same validation accuracy to a model trained on the full dataset. Data augmentation is the process of generating artificial data by applying (random) transformations to real data, thereby increasing the size of our training set.

We aim to adapt the framework of dataset distillation to learn an augmentation function: given training datapoint (x, y), we aim to learn a stochastic function $\phi(x)$ which outputs a datapoint so that training with batches of $(\phi(x), y)$ achieves better validation loss compared to training on the original dataset.

More specifically, we modify the algorithm of dataset distillation by matching training trajectories ([CWT$^+$22a]). The authors obtain a low-support synthetic dataset which will induce a classifier f's parameters to update in the same direction as when we train f on the full dataset, by updating individual pixels in images in the synthetic set. We instead learn a stochastic function ϕ, and update the parameters of ϕ such that the training trajectory of f on 'augmented' data $\phi(x)$ matches the training trajectory of the f on the original dataset.

What would be potentially useful is if we can learn ϕ using only a subset of the original dataset. Then at evaluation time we can run e.g 1%, 10% or 50% of the original dataset through the trained ϕ and see how the performance of f compares to performance f trained on 100% of the original dataset without ϕ. This is also in the spirit of dataset distillation, where we try to achieve the same performance as training on the real dataset but by only training on a small subset of the dataset.

Unfortunately, although our proposition is a seemingly straightforward extension of [CWT$^+$22a], we are unsuccessful. In our first experiment we train ϕ as described above, but the images $\phi(x)$ do not look like the distinctive synthetic images obtained using the unmodified algorithm from [CWT$^+$22a]. It is clear that despite trying a range of architectures for ϕ it is unable to learn the desired transformation, even though we have effectively just reparameterised pixels in the synthetic dataset as weights and biases in ϕ. We do not carry out planned subsequent experiments.

We describe in detail our implementation, our attempts to debug and what we would try next, which we hope will be useful to either make progress or seek a different direction if our idea is considered in a future work.

1 Introduction

Data augmentation refers to the process of increasing the amount of training data available by applying transformations to existing training data. It is an important technique because it helps to prevent overfitting by increasing the amount of data available for training networks, particularly deep networks with a large number of parameters.

Augmentations were initially chosen manually; in image classification for example, we might augment our training set by adding a copy of every image rotated anticlockwise by 10 degrees. This transformation is restrictive enough that it won't change the true label of each image, but is sufficiently different from the un-rotated image that it forces the network to learn more about the image's features in order to classify it correctly.

The natural progression is to design an algorithm which learns the best way to augment data for various downstream tasks. Continuing with the example above, we might want to learn the maximum and minimum angles by which we can rotate an image without changing the true label of an image ([MMD$^+$22]). Then, we can augment our training set by rotating each image by various points within this interval. More abstractly, an algorithm which can learn how to augment data could be especially useful for non-image data, for which humans have less intuition about how to design augmentations.

On the other hand, dataset distillation is the task of synthesising a small number of data points from a large dataset, such that a model can be rapidly trained on the few synthesised points and achieve similar performance to a model trained on the original large dataset. The task was introduced by [WZTE18], and they manage to compress 60,000 MNIST training images into just 10 synthetic distilled images, with only one image per class. More recently, with the task of image classification on the more complex CIFAR-10 dataset, [CWT$^+$22a] achieve 71.6% validation accuracy with 50 images class compared to 84.8% when training on the full dataset with 6000 images per class.

Here, a function ϕ which is able to map any image from the MNIST or CIFAR-10 training set to something which is close to the corresponding synthetic distilled image for the class of that image can be considered to be the ultimate data augmentation function: it has encoded a single transformation which enables the model to learn about many similar images from just one or a few images, massively reducing computation without overfitting. Moreover, if ϕ is stochastic then we can get multiple augmented images from a single training image and experience the benefits data augmentation, which would not be the case if ϕ is deterministic.

Thus, given a training example (x, y), we aim to learn a stochastic function $\phi(x)$ which outputs a datapoint y, so that training with batches of $(\phi(x), y)$ achieves better validation loss compared to training on the original dataset. In this way, we adapt the framework of dataset distillaton, in which we are able to learn more quickly from transformed data than from the original data, to learning an augmentation function.

2 Related work

To the best of our knowledge, no one has explicitly investigated learning an augmentation function using techniques from dataset distillation. Below is a summary of various works in data augmentation and dataset distillation which we used to inform our approach.

2.1 AutoAugment ([CZM$^+$18])

The authors automate the process of finding a data augmentation *policy* which is specific to a target dataset. Here, a policy expresses many choices and orders of augmentation operations (e.g. rotation, then cropping), the probability of applying each augmentation, and parameters specific to each augmentation (e.g. maximum rotation angle, minimum cropped window size). They use reinforcement learning to find a policy such that training a classifier with this policy yields the best validation accuracy.

The issue with AutoAugment and similar methods is that the search for an augmentation policy increases the training complexity and computational cost substantially, especially for large datasets. One way around this is to learn the augmentation policy on a small subset and then apply it to the full dataset, and this is in fact something we intended to try with our augmentation function.

2.2 RandAugment ([CZSL20])

RandAugment avoids an expensive policy search entirely, by proposing a much more simple method to perform augmentations. The method involves randomly sampling N of $K = 14$ possible transformations, and applying each of them with magnitude $M \in \{0, 1, ..., 10\}$. The variables N and M can be selected using standard hyperparameter optimisation techniques, but the authors find that a simple grid search suffices due to the small range of values each variable can take.

RandAugment performed similarly to the state-of-the-art methods at the time. While it is interesting that such a simple method can achieve such good performance, we don't see a way to integrate this method with dataset distillation techniques, which are inherently more complex as they involve a bi-level optimisation in the style of MAML ([FAL17]).

2.3 InstaAug ([MMD$^+$22])

The previous two methods implicitly assume independence between the input and the transformation applied to that input. The authors provide an example of how global augmentations can be restrictive: "we can change a leaf from green to yellow while maintaining its label, but not a lime". To address this, they propose a method 'InstaAug' for learning instance-specific augmentations, such that at inference time the trained augmentation function transforms each input differently based on the characteristics of the input.

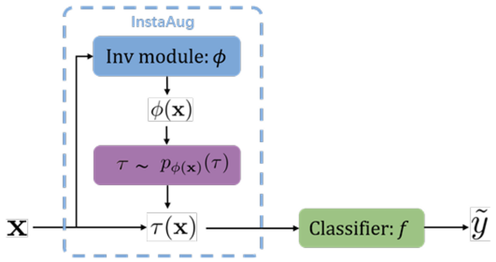

Figure 1: Figure 2 in [MMD$^+$22]. The InstaAug framework.

For example, during training InstaAug can learn maximum and minimum rotation angles $\theta_{\max}$ and $\theta_{\min}$. These are parameters for a rotation transformation, and they are outputs of the invariance module in Figure 2.3, which is a neural network. When it comes to applying rotations to images from the original dataset with a trained invariance module, each image is fed into the invariance module, which outputs a pair $(\theta_{\max}, \theta_{\min})$, and then the actual rotation angle for that image is sampled uniformly from $[\theta_{\min}, \theta_{\max}]$. In this way, the augmentations are instance-specific, rather than being randomly sampled from a set of transformations that is fixed for the entire dataset.

To the best of our knowledge, looking at the way the results are presented in [MMD$^+$22] but not at their code, it seems that each time we train the invariance module, it can only learn how to do one transformation. This is a disadvantage of this method compared to AutoAugment and RandAugment, and it is something we try to avoid in the method we propose below. We do, however, have Figure 2.3 in mind when thinking of our approach; in particular, our approach is also instance-specific.

2.4 Dataset distillation ([WZTE18])

The authors introduce the task of dataset distillation, which we have already described above. Noting that standard minibatch stochastic gradient descent for image classification takes at least thousands of update steps to converge, they aim to learn a relatively tiny synthetic dataset and corresponding learning rate η so that a *single* gradient descent step using the synthetic data can update a classifier f's parameters in a direction which greatly boosts performance of f on the real dataset.

Their algorithm is similar to MAML ([FAL17]). Their tasks are randomly sampled batches of initial weights for f, and for each task they compute the updated weight after one gradient descent step on the synthetic dataset and evaluate the objective function when using the updated f's to classify images from the real dataset (for image classification, this would be cross-entropy loss). Then in the outer loop, they update their synthetic dataset to minimise the sum of these objective function values.

2.5 Dataset distillation by matching training trajectories ([CWT$^+$22a])

The authors attempt the same task as [WZTE18], but here their approach is to synthesise a set of datapoints $\mathcal{S} = \{\hat{x}\}$ which will induce the parameters of a classifier f to be updated in a similar *trajectory* as when we train f on real datapoints $\mathcal{R} = \{x\}$, where $|\mathcal{S}| \ll |\mathcal{R}|$ (see Figure 2.5). They also learn a learning rate α and train f on $\{\hat{x}\}$ with α. At a high-level this method is similar to [WZTE18], but it is a much more recent work and performs better on benchmarks ([CWSH22]).

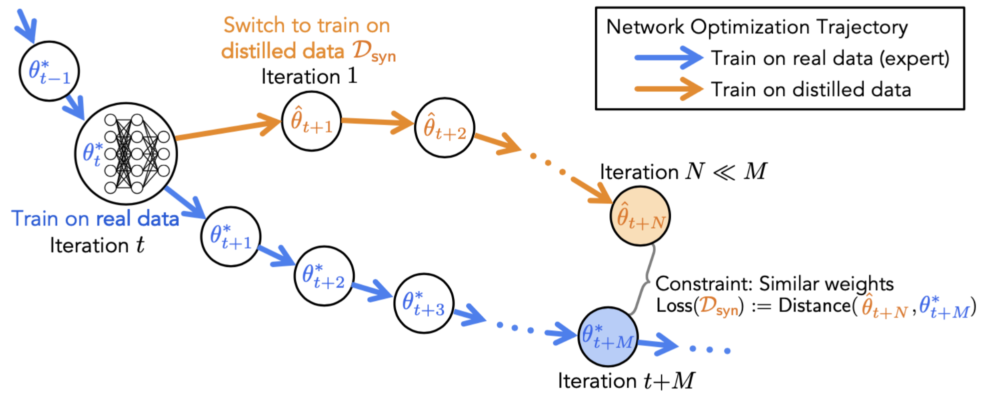

Figure 2: Figure 3 in [CWT$^+$22a].

3 Method

Our method is inspired primarily by InstaAug and dataset distillation by matching training trajectories.

What we would like is for our learned augmentation function to be able to map each image to a distilled image of the same class, with the idea that learning on distilled images will allow the classifier to be able to generalise in fewer iterations. There's no reason to expect that the transformation which maps each image to a distilled image can be represented by a series of commonly used augmentations like rotations, colour shifts or shearing, and certainly not any one of these on their own. So, with the InstaAug framework in mind, we remove the restrictive transformation distribution τ and instead try to learn a more expressive neural network ϕ.

The issue is that an unconstrained neural network has so much freedom in choosing transformations that many transformed samples may end up being incorrectly classified. We can interpret this as the search space of transformations being too large. InstaAug gets around this by allowing what is, relatively, an extremely narrow range of transformations and requiring the entropy of the transformation distribution to lie within a certain bounded interval.

Following [CWT$^+$22b], we aim to update ϕ such that the training trajectory of a classifier f on augmented data $\phi(x)$ matches the training trajectory of the (expert) classifier $\tilde{f}$, which we will train on the original dataset. By its nature this method constrains ϕ and limits the search space of transformations.

Our intuition behind this general approach is that training f with batches of $(\phi(x), y)$ will cause f's parameters to quickly reach the local minimum achieved by training $\tilde{f}$ on the real dataset. We also introduce some stochasticity to ϕ with the hope that it may allow f to outperform $\tilde{f}$. Finally, we hope that by allowing more types of transformations than methods like RandAugment and InstaAug, we might be able to outperform these methods.

We implement the general notion above by modifying the algorithm from [CWT$^+$22a]; see Figure 3.

```
Algorithm 1 Dataset Distillation via Trajectory Matching
Input: $\{\tau_i^*\}$: set of expert parameter trajectories trained on $\mathcal{D}_{\text{real}}$.
Input: $M$: # of updates between starting and target expert params.
Input: $N$: # of updates to student network per distillation step.
Input: $\mathcal{A}$: Differentiable augmentation function.
Input: $T^+ < T$: Maximum start epoch.
 1: ~~Initialize distilled data $\mathcal{D}_{\text{syn}} \sim \mathcal{D}_{\text{real}}$~~    initialise phi
 2: Initialize trainable learning rate $\alpha := \alpha_0$ for apply $\mathcal{D}_{\text{syn}}$
 3: for each distillation step... do
 4:     ▷ Sample expert trajectory: $\tau^* \sim \{\tau_i^*\}$ with $\tau^* = \{\theta_t^*\}_0^T$
 5:     ▷ Choose random start epoch, $t \leq T^+$
 6:     ▷ Initialize student network with expert params:
 7:         $\hat{\theta}_t := \theta_t^*$
 8:     for $n = 0 \to N - 1$ do
 9:         ▷ Sample a mini-batch of ~~distilled~~ images: from a subset of real dataset
10:             $b_{t+n}$        then we pass this batch through phi
11:         ▷ Update student network w.r.t. classification loss:
12:             $\hat{\theta}_{t+n+1} = \hat{\theta}_{t+n} - \alpha \nabla \ell(\underline{\mathcal{A}(b_{t+n})}; \hat{\theta}_{t+n})$
13:     end for        Replace this with A(phi(b_{t + n}))
14:     ▷ Compute loss between ending student and expert params:
15:         $\mathcal{L} = \|\hat{\theta}_{t+N} - \theta_{t+M}^*\|_2^2 \;/\; \|\theta_t^* - \theta_{t+M}^*\|_2^2$
16:     ▷ Update ~~$\mathcal{D}_{\text{syn}}$~~ and $\alpha$ with respect to $\mathcal{L}$    we instead update
17: end for                                                                   phi and alpha w.r.t. L
Output: distilled data ~~$\mathcal{D}_{\text{syn}}$~~ and learning rate $\alpha$
        for us, output is trained phi and learning rate alpha
```

Figure 3: Modified algorithm from [CWT+22a].

4 Experiments

4.1 Task

Our final goal is to train an augmentation function for image classification. In the first experiment, however, our task is closer to that of dataset distillation: our aim was to first see if the algorithm in Figure 3, where we have effectively reparameterised the synthetic dataset as a neural network, outputs a network which can produce distilled images.

4.2 Dataset

Following [CWT+22a], we begin with the CIFAR-10 dataset ([KH+09]). CIFAR-10 consists of 32×32 colour (three-channel) images, and we have 6000 images for each of 10 image classes. If experiments with CIFAR-10 are successful, our aim is to run similar experiments with the more complex CIFAR-100 dataset to see if the method is scalable.

4.3 Experiment 1: replicating [CWT+22a]

We train ϕ as described in 3, but the images $\phi(x)$ do not look anything like the distinctive synthetic images obtained using the unmodified algorithm from [CWT+22a]; see Figures 4.3 and 4.3.

Results of each of our attempts, including synthetic images, plots of validation accuracy, synthetic learning rate, and so on can be found at `https://wandb.ai/sanjit-neelam/DatasetDistillation?workspace=user-sanjit-neelam`. In particular, see the links in the captions of the figures below.

While we began with a setup as close to that used by [CWT+22a] as possible, we tried a number of things to try to get our network to learn:

- Fixing the learning rate for the classifier f at 0.01, 0.1, and 1, rather than using the learned learning rate α.

- Fixing the learning rate for ϕ at 1000 (used by [CWT+22a] as the learning rate for the optimiser which updates the pixels of synthetic images), 0.01, and 0.1.
- Trying a variety of different architectures for ϕ. These can be seen at `https://github.com/sanj909/augmentation-by-distillation/blob/5ebaf8ec9b6b54961c8d1d09c6b2693098b2106b/distill.py#L293`. To generate the figures in 4.3, we used `DistNet2` with a learning rate of 1000.

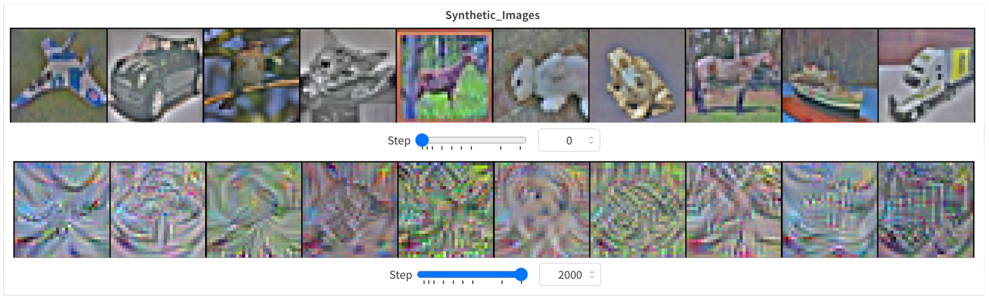

Figure 4: Images we generated using the method in [CWT+22a]. See `https://wandb.ai/sanjit-neelam/DatasetDistillation/runs/2pos2oi6?workspace=user-sanjit-neelam`.

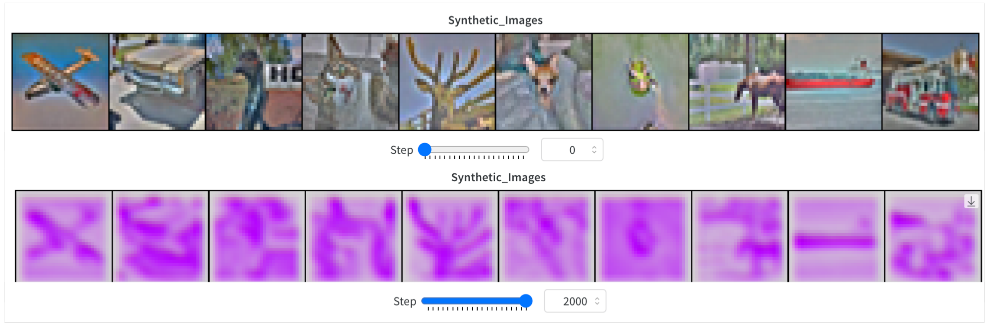

Figure 5: Images we generated using our method in Figure 3, where the subset in line 9 of our algorithm consists of just 10 images (one per class). See `https://wandb.ai/sanjit-neelam/DatasetDistillation/runs/3czybv3c?workspace=user-sanjit-neelam`.

We currently believe that the issue lies in the architecture we use for ϕ, and we would look at this more carefully were we to continue with this approach.

4.4 Undone experiments

Currently, our method gets the same output as [CWT+22a], but in a different form. What would be potentially useful is if we can learn ϕ using only a subset of the original dataset. Then at evaluation time we can run e.g 1%, 10% or 50% of the original dataset through the trained ϕ and see how the performance of f compares to performance f trained on 100% of the original dataset without ϕ. This is also in the spirit of dataset distillation, where we try to achieve the same performance as training on the real dataset but by only training on a small subset of the dataset.

4.5 Code

We forked the repository at `https://github.com/GeorgeCazenavette/mtt-distillation`, which is the official repository for [CWT+22a]. We primarily modified the file `distill.py` from this repository. Our code can be found at `https://github.com/sanj909/augmentation-by-distillation`. In particular, the file `distill.py` in our repository contains detailed comments about the modifications we made to the original file, and what we tried to get the method to work.

5 Conclusion

Unfortunately, although our proposition is a seemingly straightforward extension of [CWT+22a], we are unsuccessful. In our first experiment we train ϕ as described above, but the images $\phi(x)$ do not look like the distinctive synthetic images obtained using the unmodified algorithm from [CWT+22a]. It is clear that despite trying a range of architectures for ϕ it is unable to learn the desired transformation, even though we have effectively just reparameterised pixels in the synthetic dataset as weights and biases in ϕ.

6 Acknowledgements

Thanks to the CS 330 TAs Yoonho Lee, Eric Mitchell and Suraj Nair for their help with aspects of the design and implementation of this project. LaTeX template borrowed from NIPS 2017.

References

[CWSH22] Justin Cui, Ruochen Wang, Si Si, and Cho-Jui Hsieh. Dc-bench: Dataset condensation benchmark. *arXiv preprint arXiv:2207.09639*, 2022.

[CWT+22a] George Cazenavette, Tongzhou Wang, Antonio Torralba, Alexei A. Efros, and Jun-Yan Zhu. Dataset distillation by matching training trajectories. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022.

[CWT+22b] George Cazenavette, Tongzhou Wang, Antonio Torralba, Alexei A Efros, and Jun-Yan Zhu. Dataset distillation by matching training trajectories. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4750–4759, 2022.

[CZM+18] Ekin D Cubuk, Barret Zoph, Dandelion Mane, Vijay Vasudevan, and Quoc V Le. Autoaugment: Learning augmentation policies from data. *arXiv preprint arXiv:1805.09501*, 2018.

[CZSL20] Ekin D Cubuk, Barret Zoph, Jonathon Shlens, and Quoc V Le. Randaugment: Practical automated data augmentation with a reduced search space. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition workshops*, pages 702–703, 2020.

[FAL17] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International conference on machine learning*, pages 1126–1135. PMLR, 2017.

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

[MMD+22] Ning Miao, Emile Mathieu, Yann Dubois, Tom Rainforth, Yee Whye Teh, Adam Foster, and Hyunjik Kim. Learning instance-specific data augmentations. *arXiv preprint arXiv:2206.00051*, 2022.

[WZTE18] Tongzhou Wang, Jun-Yan Zhu, Antonio Torralba, and Alexei A Efros. Dataset distillation. *arXiv preprint arXiv:1811.10959*, 2018.