Dataset Distillation for 3D Point Clouds

Karim ElGhandour
karim.elghandour@tum.de

Abstract

As the demand for better deep learning model increases, the need for larger datasets is on the rise. This increase in size is requiring more resources to be able to store such datasets and use them in training. Dataset distillation is a relatively new research focus that attempts to condense a large dataset by learning a smaller synthetic dataset which when used to train a neural network reaches a similar performance as using the original dataset. Several approaches attempt distilling dense data representations such as images whether by using a bi-level optimization method, feature matching, or gradient matching. However, the objective of this paper is to use dataset distillation using gradient matching on a sparse data representation, specifically 3D point clouds with a variable number of input points. Instead of turning on or off voxels in a dense grid, the objective is updating the point coordinates directly starting from points scattered randomly in 3D space to eventually converge to capture the geometric features of the classes. Even though the results do not reach other state of the art approaches, they show a proof of concept that updating coordinates directly is a possible method for distilling data.

1. Introduction

Datasets are incredibly important when training deep learning models to reach state of the art performance. With the massive amounts of data that is easily accessible, it is no surprise that datasets are getting larger across the different fields of Computer Vision and Natural Language Processing. This rapid growth in size adds a lot of challenges when it comes to storing the data and requires much more computational resources to use them to train such models. Research has gone into finding methods that can compress or condense datasets where using a small dataset to train models yields similar performance to using a large dataset. A relatively straight forward method is finding valuable samples in a dataset and using these solely in training. This type of data selection is known as instance selection or core-set [2, 6, 14, 15, 17]. However, it often follows a greedy heuristic which is inefficient and often reaches sub optimal results.

A slightly different approach is to try and learn a synthetic dataset through some kind of knowledge transfer. Approaches similar to this have been implemented to condense neural network models through knowledge distillation [8]. Dataset Distillation techniques [12, 13, 16, 18, 19] aims to learn a synthetic datasets that is several magnitudes smaller in size yet yield a similar performance.

This paper investigates using previous methods in dataset distillation using gradient matching [19] to distill point clouds, which has not been tested in previous research. Gradient matching formulates the goal as minimizing the distance between two sets of gradients of distillation network parameters where one is obtained using real data and the other is obtained using a synthetic dataset. This paper implements a distillation pipeline for point clouds. It also shows a framework that allows implementing flexible architectures to work with sparse and dense operations. Additional analysis of different ablations was also performed to investigate their impacts on the distillation. The paper shows a proof of concept that distilling point clouds by directly updating coordinates in a sparse setting is possible.

2. Related Work

2.1. Core-set

Core-set or Instance Selection [2,6,14,15,17] is a method used to decrease the size of a training set. The most influential samples in a dataset are selected to represent it with the goal that training a model on these samples will yield comparable performance as to training on the entire dataset.

Welling [17] introduced Herding which incrementally adds samples to the core-set trying to minimize the distance in feature space between the centers of the core-set and the original dataset. Mirzasoleiman et al. [10] offers CRAIG which is a method that aims to increment the core-set by minimizing the gradient differences generated on the same neural network by passing the core-set and by passing the original dataset.

Core-set selects raw samples from the original dataset based on greedy approaches as it has to try different combinations of samples to improve. This makes them prone to sub-optimal results.

2.2. Dataset Distillation

Dataset Distillation (DD) or Dataset Condensation (DC), on the other hand, learns and generates a smaller synthetic dataset from an original larger one such that training a model on the synthetic data performs similar to training a model on the real data. It holds similarity to Knowledge Distillation [8] where a larger teacher model transfers knowledge to a smaller student model such that they perform similar to each other.

Wang et al. [16] introduced dataset distillation through performance matching. A differentiable model is trained on a synthetic dataset $\mathcal{S}$ in the inner loops, then in the outer loops the model is validated on the real dataset τ and the validation loss is backpropagated to update $\mathcal{S}$. This optimization method is computationally expensive as it involves bi-level optimization.

Other methods of dataset distillation include model parameter matching [1], and kernel ridge regression [12, 13], and Distribution Matching [18] which directly matches the output features between real and synthetic samples. The focus of this paper is using Gradient Matching [19] that tries to minimize the distance between gradient produced by passing a synthetic dataset through a network and passing the real dataset through the network.

2.3. MinkowskiEngine

A dense representation of highly dimensional spaces is extremely inefficient due to sparsity of the points. Thus, it is more efficient to use a sparse representation. Minkowski Engine [3] is an open-source auto-differentiation library for sparse tensors which provides functions that facilitate using convolutions and other functions on sparse tensors. Minkowski Engine works on occupancy grids where it expects items in a dataset to consist of Coordinates representing the non-empty part of a space, a list of associated Features per point which may represent colors, normals or other import attributes, and a list of labels. Operations on sparse tensors such as Convolution differ from dense tensors as they operate solely on occupied cells within the kernel location.

3. Method

Given a large dataset $\tau = \{(x_i, y_i)\}_{i=1}^{|\tau|}$ where $x \in \mathbb{R}^{N\times 3}$, N is the number of points, 3 representing the dimensionality and $y \in \{0, ..., C-1\}$ which is the associated class, the goal is to generate a small dataset $\mathcal{S} = \{(s_i, y_i)\}_{i=1}^{|\mathcal{S}|}$ where $|\mathcal{S}| \ll |\tau|$. Using $\mathcal{S}$ to train a neural network ϕ with parameters θ, correctly predicts labels of previously unseen images from the validation set of τ i.e. $y = \phi_{\theta_s}(x)$. Figure 1 shows the overall pipeline where there is a distillation block that updates the synthetic data and an evaluation block which is a classifier that gets trained from scratch on synthetic data and evaluated using the validation set of the real dataset to test the performance of the distillation.

Figure 1. Full pipeline overview where the synthetic and real datasets pass through a Distillation Block which updates the synthetic data, and then they pass through an Evaluation block which trains a classifier on synthetic data and evaluates it on real data validation set.

3.1. Data Preparation

3.1.1 Pointcloud MNIST

The training dataset τ is expected to be a point cloud. In order to simulate working with point clouds, MNIST dataset [5] is used as it consists of single channel images of handwritten digits with not a lot of intermediate values. The first step is to set a threshold to convert MNIST to binary images. Then, pixel cell locations storing a 1 are used to convert it to an occupancy grid. An additional axis set to 0 is added to simulate 3D conditions making the data 1x28x28 where each single value in the occupancy grid represents a coordinate of a point in 3D space.

3.1.2 Synthetic Dataset and Boundary Check

The synthetic data $\mathcal{S}$ is a randomly initialized $N \times 3$ point cloud where N is the number of points and 3 represent the coordinates $\{Depth, Height, Width\}$. $Depth$ is constant and is kept at 0 for all points, while $Height$ and $Width$'s range is $[0, 27]$. In order for synthetic data coordinates to update, synthetic data is added to the computation graph and a gradient is calculated for each point. For each point in each sample in the synthetic dataset, there are three gradient values, where each value corresponds to one of the axis. In the case of distillation, instead of an optimizer being used to update the network parameters, it is instead used to update the synthetic data directly.

As coordinates shift and move between iterations, they might move out of the designated range of (0,0,0) and (0, 27, 27) and thus it is important at the beginning of each iteration to update the values to fit again within the range and then re-add the synthetic dataset to the computation graph again.

3.1.3 Coordinates as Features

Previous methods of Dataset Distillation [19] used dense representations where the distillation algorithm updated the

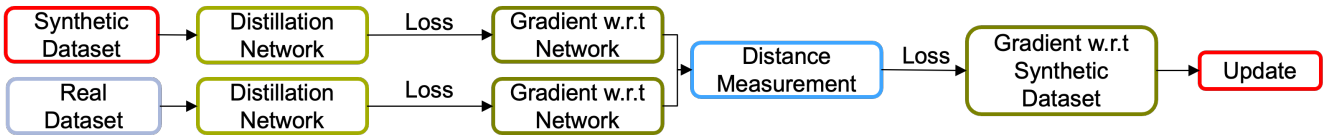

Figure 2. Distillation Block pipeline overview. Synthetic Data and Real data pass through a distillation network separately to calculate losses and gradients w.r.t the network. A distance measurement is used to calculate the differences between both gradients and use the result to calculate new gradients w.r.t synthetic data which passes through an optimizer that updates the synthetic dataset.

pixel values. This is applicable in low dimensionality situations such as images because their nature is dense. When dealing with highly dimensional sparse data such as point clouds, using a dense representation means that features are either 0 to indicate that no point lies in that grid cell, or a 1 to indicate that this is the correct location of a point, and gradients updates the voxel value. A more efficient representation is to use coordinates directly in the features which would allow gradients to update point locations and move them around instead. However, this approach adds an extra challenge as it is important that the coordinates update do not overshoot the points over their target locations.

3.2. Data Distillation

The objective as defined in [19] is to learn synthetic dataset $\mathcal{S}$ such that a model ϕ_{θ_S} trained on the synthetic data perform generally well on the validation set of the original dataset τ compared to a model ϕ_{θ_τ} trained on τ.

3.2.1 Gradient Matching

In Gradient Matching, this objective is achieved by converging in the parameter space ($\theta^S \approx \theta^\tau$). This is done by passing the original dataset τ through the distillation network to calculate the loss and the corresponding gradients, followed by repeating the process with the synthetic dataset $\mathcal{S}$. After obtaining both gradients with respect to the distillation network, a distance function is used between both gradients to calculate a new loss. This loss can then be used to calculate gradients with respect to the synthetic dataset. An optimizer is then used to update the synthetic dataset.

3.2.2 Distillation Architectures

Distillation network architecture used in [19] works well for 2D images, however, it required several modifications to function well with 3D point clouds. Minkowski Engine is used to create the input batches, which provides the flexibility of either using Minkowski operations or directly converting to dense and using pure Pytorch operations instead.

Architectures for both approaches are different due to the different gradient calculations where especially the number of layers for each architecture is changed. However, both architectures involve several network layers each consisting of 3D convolutions, ReLU, and Max Pooling. The output of the layers is flattened and is then passed to a linear layer.

Figure 3. Qualitative results showing the distilled MNIST digits after projecting them back to the image space. Top: Results using full Pytorch convolutions, Bottom: results using Minkowski Engine convolutions.

4. Results and Limitations

	Number of layers		
	2	**3**	**4**
Minkowski	**33.9**	12.78	10.48
Pure Pytorch	25.69	**30.22**	17.43

Table 1. Accuracy resulting from using Minkowski and Pytorch layers with different number of layers (2, 3 and 4). The accuracy is predicting MNIST testset class labels using a classifier trained on synthetic data. Synthetic dataset pointclouds are projected onto the image space and the resultant images were used for training the classifiers.

4.1. Setup

All experiments ran on a system equipped with a NVIDIA RTX 3080 10GB with 64GB memory. Synthetic dataset was randomly initialized for each run and it included 1 sample for each class and the distillation network was initialized using Kaiming [7]. Prior to evaluation, the synthetic pointclouds are projected back into the image space and an image classifier is used for evaluation. For each classification block run, the same classifier network was trained 5 times and the average accuracy was taken.

4.2. Distillation using Pytorch and Minkowski

Table 1 compares the results obtained from using Pytorch layers directly after the input batches are created using Minkowski Engine, and from using full Minkowski Engine

	Mean Absolute Gradient		Accuracy	
	Layerwise Cosine	**Sum of Square**	**Layerwise Cosine**	**Sum of Squares**
ME	4e-4	0.033	11.37	**33.9**
Pytorch	5e-4	0.05	10.23	**30.22**

Table 2. Quantitative results for using different distance measurements. Mean Absolute Gradient is the synthetic data gradient prior to update and Accuracy is the resultant evaluation accuracy. Values are averaged over 1250 iterations. The gradient values from using Layerwise Cosine are too small to move the move the points around in each iteration compared to that of using Sum of Squares distance.

layers prior to the final linear layers. Using Minkowski Engine layers result in better quantitative results reaching an average accuracy of 33.9% compared to using Pytorch layers which reach an average of 30.22%. Additionally, the table also shows that a slight change in architecture heavily impacts results as it yields to unstable small gradients that results in the points getting stuck and not moving between iterations.

Qualitative results in figure 3 shows that using Pytorch layers achieve better overall quality where there are synthetic digits that capture the geometric properties of the correct digits in the original dataset. Trials using Minkowski convolutions tended to suffer from more noise, thus the results were less clear. However, the distillation failed to capture some of the key features of some of the digits such as 8 which tended to suffer in both implementations.

4.3. Distance Measurement

A crucial part is calculating the difference between the real and synthetic data gradients. This can significantly impact performance. Table 2 shows the results of using layerwise negative cosine distance as used in [19] and sum of square distance. Layerwise negative cosine distance generates extremely small gradients such that the update to the synthetic data is insignificant and no convergence occurs. Metrics such as the mean absolute gradient which is the gradient w.r.t the coordinates, and mean absolute distance difference which is the change in coordinates before and after the optimizer update step are good metrics to log as they indicate whether or not the architecture is distilling data or if it is stuck.

4.4. Ablation: Using Average Pooling

In [19] Average Pooling is used instead of Max Pooling, which performs well since it was dealing with images which is a dense data representation. Using Average Pooling while working with a sparse representation such as point clouds yields relatively similar performance when using Minkowski Engine which is justified by the fact that Minkowski only uses occupied cells when calculating pooling. This is different, however, when using Pytorch pooling as the empty cells are replaced with zeros which results in a huge bias towards unoccupied cells thus significantly hurting performance.

4.5. Limitations

Initialization seems to have a huge impact on performance as running the same experiment multiple times may lead to significantly different results. This is consistent with the findings in [4, 19] as there are multiple local minima traps that convergence may fall into. Additionally, there are a lot of hyper-parameters tuning necessary to achieve better results for each use case as evident in table 1 which shows that adding or removing layers significantly impacts evaluation. Lastly, as the approach aims at distilling sparse point clouds by updating coordinates, there are 3 features per point that gets optimized which is more difficult than just 1 feature being updated in a single channel dense distillation such as MNIST in [19]. This is also consistent with the low classification performance results for distilling RGB images such as SVHN [11] and CIFAR10 [9] in [19].

5. Conclusion

This paper, to our knowledge, introduces the first trials of dataset distillation on point clouds and on sparse data in general. It introduces a framework based on previous methods of distillation using gradient matching that works with a sparse representation and a variable number of input points. The study shows that using point clouds for distillation is possible. The results, however, did not reach state of the art with MNIST as it showed that added dimensionality adds extra complexity to the distillation since it requires reaching accurate coordinates and not just increasing or decreasing as needed. For the future work, investigating additional hyperparameter tuning may lead to reaching better results in the short term. Additionally, replacing the randomly initialized distillation network with a pretrained classifier trained on the real dataset instead may help investigate if it will lead to more stable gradients and hence better overall results. Lastly, other distillation methods such as distribution matching seems to be more promising and results in a more stable path towards distillation, thus it can help the performance especially with the limitations currently faced due to the added dimensions.

References

[1] 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. 2

[2] Yutian Chen, Max Welling, and Alex Smola. Super-samples from kernel herding. *arXiv preprint arXiv:1203.3472*, 2012. 1

[3] Christopher Choy, JunYoung Gwak, and Silvio Savarese. 4d spatio-temporal convnets: Minkowski convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3075–3084, 2019. 2

[4] Justin Cui, Ruochen Wang, Si Si, and Cho-Jui Hsieh. Dc-bench: Dataset condensation benchmark. *Advances in Neural Information Processing Systems*, 35:810–822, 2022. 4

[5] Li Deng. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012. 2

[6] Sariel Har-Peled and Akash Kushal. Smaller coresets for k-median and k-means clustering. In *Proceedings of the twenty-first annual symposium on Computational geometry*, pages 126–134, 2005. 1

[7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015. 3

[8] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015. 1, 2

[9] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009. 4

[10] Baharan Mirzasoleiman, Jeff Bilmes, and Jure Leskovec. Coresets for data-efficient training of machine learning models. In *International Conference on Machine Learning*, pages 6950–6960. PMLR, 2020. 1

[11] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. 2011. 4

[12] Timothy Nguyen, Zhourong Chen, and Jaehoon Lee. Dataset meta-learning from kernel ridge-regression. *arXiv preprint arXiv:2011.00050*, 2020. 1, 2

[13] Timothy Nguyen, Roman Novak, Lechao Xiao, and Jaehoon Lee. Dataset distillation with infinitely wide convolutional networks. *Advances in Neural Information Processing Systems*, 34:5186–5198, 2021. 1, 2

[14] Ozan Sener and Silvio Savarese. Active learning for convolutional neural networks: A core-set approach. *arXiv preprint arXiv:1708.00489*, 2017. 1

[15] Ivor W Tsang, James T Kwok, Pak-Ming Cheung, and Nello Cristianini. Core vector machines: Fast svm training on very large data sets. *Journal of Machine Learning Research*, 6(4), 2005. 1

[16] Tongzhou Wang, Jun-Yan Zhu, Antonio Torralba, and Alexei A Efros. Dataset distillation. *arXiv preprint arXiv:1811.10959*, 2018. 1, 2

[17] Max Welling. Herding dynamical weights to learn. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 1121–1128, 2009. 1

[18] 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. 1, 2

[19] Bo Zhao, Konda Reddy Mopuri, and Hakan Bilen. Dataset condensation with gradient matching. *arXiv preprint arXiv:2006.05929*, 2020. 1, 2, 3, 4