Project Report: Dataset Distillation for One-Shot Federated Learning
Optional Project in Data Science

Ahmed Jellouli
EPF Lausanne, Switzerland

Supervised by: Dr Luis Barba
Machine Learning and Optimization Laboratory MLO, EPF Lausanne, Switzerland

***Abstract*—Federated learning [1] is traditionally done via some form of weight aggregation across different nodes. This has many limitations, such as the difficulty of updating local Batch-Normalization [2] statistics. In this work, we explore the use of a data-driven approach to Federated Learning. We use Deep Inversion [3] in order to generate image batches from a set of teacher nodes, then train a student using the combined dataset. We experiment with many levels of data heterogeneity as well as number of teachers, and find that naively mixing datasets from different teachers does not lead to a good performance. This is due in part because of a decrease in image quality when splitting the dataset, as well as the existence of similar classes at different teachers. We propose a scheme to generate images using information from multiple teachers, and show that this leads to better images and reasonable student performance.**

I. Introduction

The goal of the project is to explore a different approach to federated learning which consists in reconstructing a representation of the nodes' local datasets via distillation, followed by student training on these datasets. Potential benefits of this method consist in:

- There is only a single round of communication between the central server and the edge nodes (*one-shot* federated learning). This is desirable in scenarios where communication is expensive or incurs high latency.
- All nodes (server and edge nodes) are trained locally, similar to a traditional machine learning problem. This means that all the usual training tricks, such as Batch-Normalization [2] and momentum, can be used without extra overhead.
- Nodes communicate only their final trained model to the server, which is helpful for privacy as no training data needs to be revealed.

The rest of this report is organized as follows: In section II, we introduce the required background in order to understand the rest of the paper. In section III, we present our attempt to reproducing results from the original deep inversion paper. In section IV, we develop on the experiments performed as well as the achieved results. In section V, we briefly present the implementation of the project. Finally, section VI contains ideas that were not implemented due to time constraints.

II. Background

A. Federated Learning

Federated Learning is a distributed learning setting where nodes would like to learn a shared model without giving access to their own private data. This setup is usually centralized, meaning that there is one special node; the server, which can communicate with the other nodes, exchanging model weights and/or gradient updates. Popular methods for Federated Learning include FederatedSGD, FederatedAveraging [1] as well as other variations.
These methods face three main issues:

- Absence of a clear way to aggregate Batch-Norm [2] layer statistics due to the fact that these reflect the distribution of the local datasets, which are usually heavilly non-iid.
- Using momentum at the edge nodes leads to higher local drift at the clients. This means that it will move faster towards its local minima, making the learning of a joint model harder. Momentum has been shown to be important for training deep neural networks [4].
- Reliance on many rounds of communication between the server and the edge nodes, which can be very expensive.

B. Deep Inversion

Deep Inversion is a method that allows to distill training data from a fully trained image classification model (the teacher). This works by minimizing a loss function over a batch of input images.
Let $\hat{x} \in \mathbb{R}^{B \times C \times H \times W}$ (B, C, H, W being respectively the batch size, the number of channels, height and width of the input) and y a vector of labels of length B for the synthesized image.

Then:

$$\begin{aligned} L_{deepinv}(\hat{x}, y) = & \, CrossEntropyLoss(\hat{x}, y) \\ & + \alpha_{TV} L_{TV}(\hat{x}) \\ & + \alpha_{l2} L_{l2}(\hat{x}) \\ & + \alpha_{bn} L_{bn}(\hat{x}) \end{aligned} \tag{1}$$

Where the cross-entropy loss is computed between the teacher output on $\hat{x}$ and y, the L_{TV} and L_{l2} compute respectively the total variance and L2 norm of the input. Finally L_{bn} uses batch norm statistics from the teacher in order to add more realism to the generated images:

$$\begin{aligned} L_{bn}(\hat{x}) = & \sum_{l} \|\mu_l(\hat{x}) - BN_l(running\_mean)\|_2 \\ & + \sum_{l} \left\|\sigma_l(\hat{x})^2 - BN_l(running\_variance)\right\|_2 \end{aligned} \tag{2}$$

Where l is the number of convolutional layers, μ_l and σ_l compute the sampling mean and variance of the l^{th} convolutional layer output, and BN_l denotes the teacher statistics for the corresponding layer.

C. Adaptive Deep Inversion

One problem with the above approach is the lack of variety in the synthesized images, since the randomness only comes from the initialization of the images and the optimizer. To alleviate this, the authors propose Adaptive Deep Inversion, which uses a student network in order to add diversity to the student images. The idea is to generate images that the student cannot classify easily while the teacher can. This is done by maximizing the Jensen-Shannon divergence between the teachers output and student output:

$$L_{compete}(\hat{x}) = 1 - JS(teacher(\hat{x}), student(\hat{x})) \tag{3}$$

Finally the adaptive deep inversion loss is:

$$\begin{aligned} L_{adaptive\_deepinv}(\hat{x}, y) = & \, L_{deepinv}(\hat{x}, y) \\ & + \alpha_{compete} L_{compete}(\hat{x}) \end{aligned} \tag{4}$$

III. Reproducing Deep Inversion

The first step in the project was to reproduce the results from the original paper [3]. We were able to obtain similar results as the original authors when using adaptive deep inversion but failed to reproduce results without adaptiveness.

A. setup

We use the CIFAR10 [5] dataset for our experiments. We train a ResNet18 [6] on the full dataset and use it as a teacher network. This network reaches 95% test accuracy on the CIFAR10 test set. The task is to train a student network (ResNet18) initialized randomly based on images distilled from the teacher.

For image generation, we use hyperparameters mentioned on the original paper's Github page [7]: $\alpha_{TV} = 1e^{-3}, \alpha_{l2} = 0, \alpha_{bn} = 10, \alpha_{compete} = 10$. Each batch is composed of 256 images and requires 1000 gradient updates. We follow the original authors and use the Mixed Precision Training [8] capabilities of PyTorch [9] in order to speedup the process. Labels for the batches are sampled uniformly at random

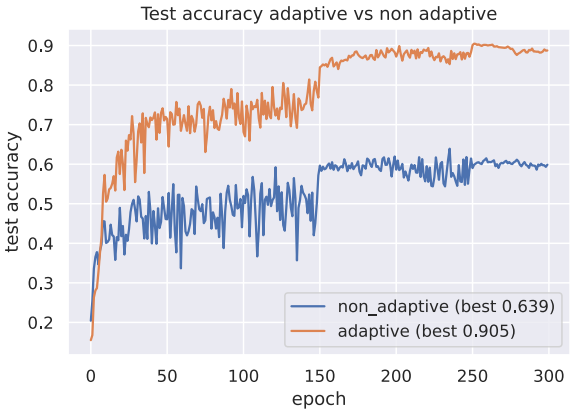


Figure 1: Test accuracy for 1 teacher and one student on CIFAR10, adaptive vs non adaptive.

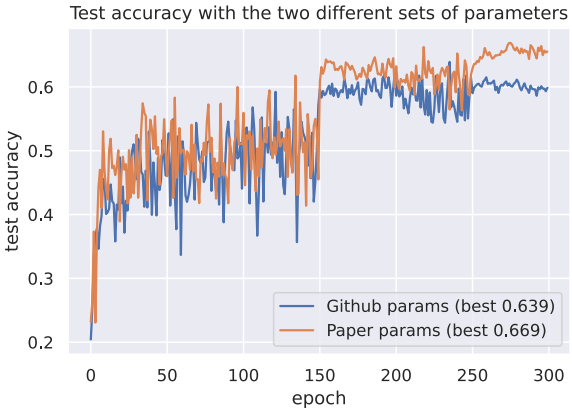


Figure 2: Test accuracy for 1 teacher and one student on CIFAR10, hyperparameters from the paper vs from the Github page.

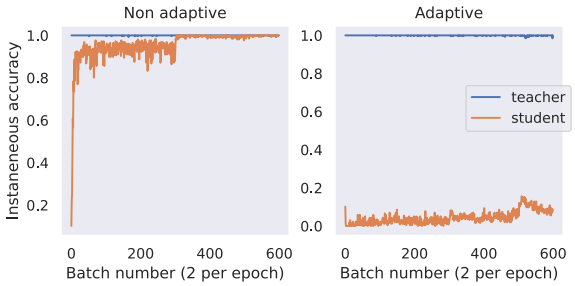
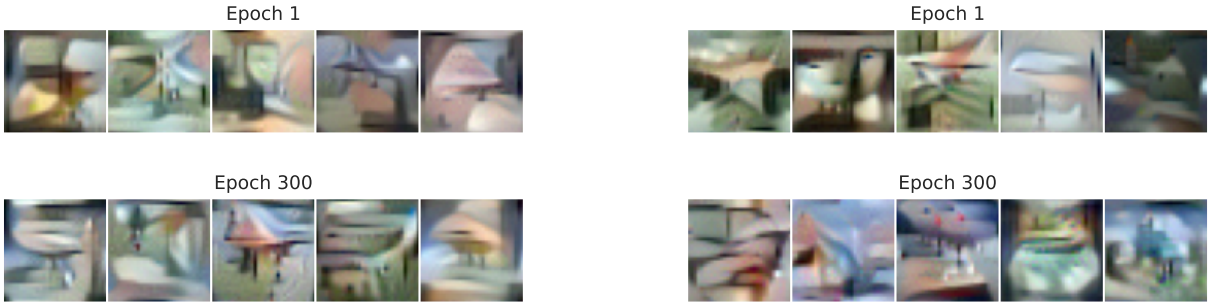


Figure 3: Accuracy on freshly generated batches, adaptive vs non adaptive.



(a) Without adaptiveness

(b) With adaptiveness

Figure 4: Generated airplane images at epochs 1 and 300, adaptive vs non adaptive.

from $0..9$. We generate new batches 2 times per epoch. These batches are used directly to train the student. During other iterations of an epoch, we pick batches uniformly at random from the previously generated images. Initially, we generate 50 batches from the student without adaptiveness.
We train for 300 epochs, an epoch corresponding to approximately 50000 images seen (size of the CIFAR10 training set). We use stochastic gradient descent with a learning rate of 0.1, a momentum of 0.9 and a weight decay of $5e^{-4}$. The learning rate is decayed by a factor of 0.1 at epochs 150 and 250.

B. Results

Figure 1 shows the results for test accuracy on the CIFAR10 test set for both adaptive and non adaptive training. We are able to approximately reproduce the original paper's results for adaptive training: we reach an accuracy of 90.5% versus 93.26% for the original paper. We may explain the difference by the fact that we use a ResNet18 as teacher network while the original paper uses a ResNet34. We find however that when removing the competition loss term, the reached accuracy (63.9%) is very far from the one obtained by the Deep Inversion paper (91.43%). We explore a few ideas in order to find the reason for this.

1) Code bugs: In order to confirm that the problem does not come from a problem with the code, we run two checks:

- We use exactly the same code that we use for adaptive training, while only setting $\alpha_{compete}$ to 0. This is done easily as we use the Hydra library [10] for configuring the main scripts. Training achieves the same results as above.
- We use the original paper's code hosted on Github in order to generate 650 (50 initial + 2 per epoch $\times$ 300 epochs) batches with the same hyperparameters. We then train the student using these batches. This also achieves similar results.

2) Batch generation parameters: The hyperparameters used for image generation are different from that mentioned in the original paper. We hypothesize that this may be the cause of the problem. Therefore we re-train the student with the parameters $\alpha_{TV} = 2.5e^{-5}, \alpha_{l2} = 3e^{-8}, \alpha_{bn} = 10$ and with 2000 gradient updates per batch. Figure 2 shows the result of the training. We can see that the reached accuracy has improved by 3%. However this is still far from the original result.

3) Image variety: We explore the hypothesis that image variety has a substantial impact on training. Figure 3 shows the accuracy of the teacher and student on the newly generated batches. We observe that without adaptiveness, the accuracy is very close to 1 at batch 180 (epoch 90). This may indicate that all batches generated beyond this point only contain information that the student already knows. Figure 4 further illustrates this. We plot airplane images generated at epochs 1 and 300 for both types of training and clearly observe how adaptive training captures different details at the different epochs. Indeed, the adaptive student accuracy is constantly low in figure 3.

C. Conclusion

We were able to obtain very good results when training adaptively. We could not however obtain the same result without adaptiveness. We therefore chose to continue with the project and use adaptiveness in all our subsequent experiments.

IV. Federated learning with dataset distillation

We present results for one-shot federated learning using dataset-distillation. In all cases, a number of teachers learn on a subset of the CIFAR10 training set. They then all send

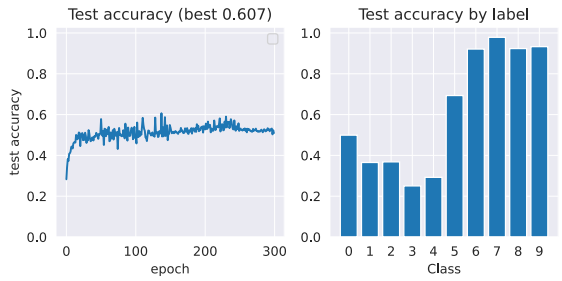


Figure 5: Test accuracy and class-wise accuracy (1 teacher with half the data owned by the student).

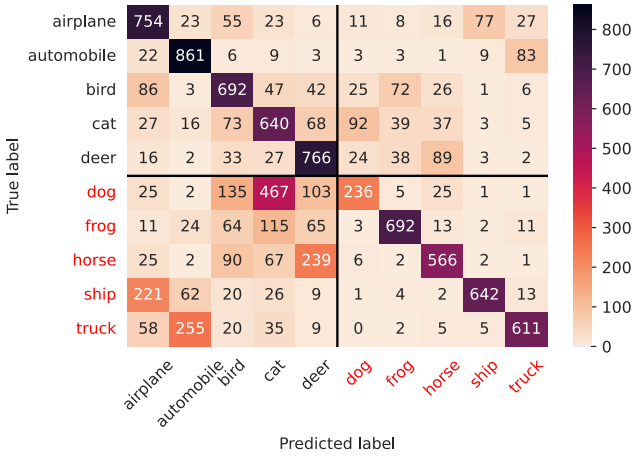


Figure 6: Confusion matrix of the student predictions on the CIFAR10 test set (1 teacher with half of the original dataset owned by the student). In black are labels owned by the teacher, in red those owned by the student.

their model to a student which tries to learn a joint model using datasets distilled from the teachers. We assume further that the student has access to the labels each teacher has.

A. *Loss function when there are unseen labels*

As we will show in section IV-E1, when there are some labels that the teachers hasn't seen it is better not to consider the corresponding output heads when computing the loss. Therefore the adaptive deep inversion loss from equation 4 is masked for each teacher on the labels it has seen.

B. *Intermediate case with 1 teacher*

In this case, a teacher is trained on half of the labels (0 to 4) of the training set. The student has the other half (labels 5 to 9) locally and wants to train using both its local dataset as well as the generated data. Since the goal is only to distill knowledge about half of the labels, we only generate half the data (adaptively) (25 initial batches and 1 new batch per epoch) compared to the experiment in III. The other half corresponds to the local CIFAR10 split. Figure 5 shows the result of the training as well as the final class-wise test accuracy. We observe that the test accuracy is fairly low at 0.607 and that this is caused by poor performance on the classes corresponding to the teacher network. This is counter-intuitive in the sense that there are many more images generated from the teacher network ($25 \times 256 + 300 \times 256 = 83200$ images) than images from the original dataset (25000).

1) More data does not help: We confirm that generating half of the data than the original experiment is not the cause of the problem. To do this, we run the experiment with full data generation. We reach an accuracy of 61.6% (0.9% improvement compared to the previous case)

2) Similar classes across the two datasets confuse the student: Figure 6 shows the confusion matrix of the predictions of the student on the CIFAR10 test set. We notice that most errors occur between labels that are close semantically but that exist in different datasets, for e.g. cat and dog, deer and horse. This means that the filters learned by the network for some class also respond to the other. This may be seen in Figure 7 as the level of detail of the distilled images is low.

Figure 7: Quality difference in original versus distilled images.

3) Weighted loss function: As an attempt to compensate for the lower images quality, we experiment with down weighting samples from the original training set when computing the classification loss by giving more weight to labels corresponding to the teacher. We experiment with a weight of 2 and 10. Figure 8 shows the class-wise accuracy. We observe that for a weight of 2, the student is still doing better on classes it has locally, however accuracy improves by 3%. For a weight of 10, the accuracy is more balanced but overall is close to the original case.

C. *IID case with 10 teachers*

We split the dataset uniformly at random between 10 teacher nodes. The student would like to learn from all the teachers. To do this, we iteratively cycle through the teachers and generate one batch from each one. We then use the

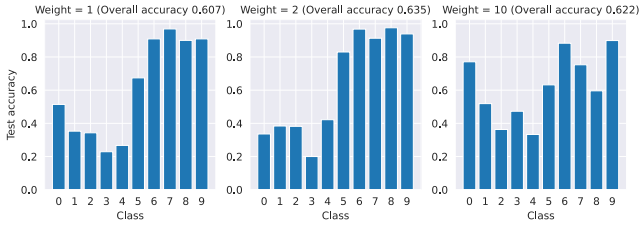


Figure 8: Class-wise accuracy for different 3 different loss weights for the teacher data (see IV-B3).

same training procedure as III-A in order to train the student. Figure 9 (in blue) shows the result of the training. We reach an accuracy of 74% which is better than the accuracy of the worst teacher (68.2%) but worse than the accuracy of the best teacher (77.8%). An interesting observation is that the reached accuracy is almost equal to the average accuracy of the teachers.

1) Less noisy images do not help: Looking at some of the images generated from the teachers (Figure 10b) and comparing them to the case of 1 teacher (Figure 10a), we observe a significant drop in image quality. The images are noisier and we can see less details from the corresponding class. In order to remedy this, we set the number of gradient updates per batch to 3000 and re-run the experiment. Figure 10c shows that the resulting images do have better quality. In particular, the noise is diminished greatly. However, this does not help the student training (Figure 9 in orange). The variance of the test accuracy seems to have diminished but the overall achieved accuracy is almost equal to the previous case. It is also worth mentioning that using 3000 updates per batch hurts the running time significantly (from 10 hours per experiment to 24 hours).

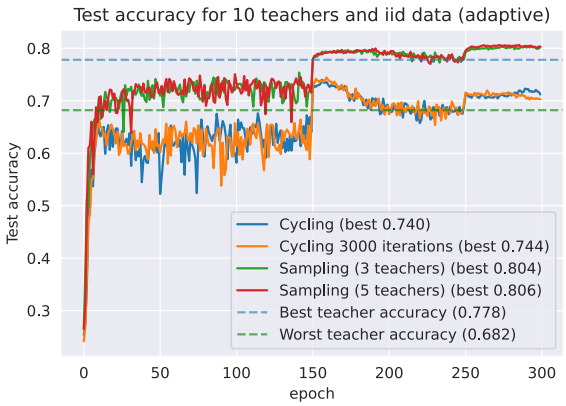


Figure 9: 10 teacher IID training resutls. "Cycling" refers to cycling through teachers and generating one batch from each one (see IV-C). "3000 iterations" refers to using 3000 gradient updates per batch (see IV-C1). "Sampling" refers to using the average loss from many teachers (see IV-C2).

2) Aggregating losses helps: We hypothesize next that in order to have an improvement in training, we must find a way to aggregate information from different teachers in order to generate the batches. Since the data is IID, a natural choice is to average the losses between several teachers before performing the backwards pass when generating a batch of images. Algorithm 1 explains the procedure.

Algorithm 1 Sampling Deep Inversion

Require: *n_iter* the number of gradient updates
Require: n the number of teachers to sample
Require: *teachers* the set of teachers
Require: *losses* the loss for each teacher
Require: y labels to generate a batch for
  initialize $\hat{x}$ randomly
  $i \leftarrow 0$
  while $i \neq$ *n_iter* $- 1$ **do**
    $s \leftarrow$ random sample of size n from *teachers*
    $loss \leftarrow \frac{1}{n} \sum_{t \in s} losses^t(\hat{x}, y)$
    backwards(loss)
    $i \leftarrow i + 1$
  end while
  return $\hat{x}$

We set $n = 3$ and we run the experiment again. The result is shown in Figure 9 (in green). We observe a clear performance improvement, reaching 80% test accuracy. This outperforms the best teacher model by 2%. Figure 10d shows a sample generated from the training procedure. Although the images are still noisy, we can clearly observe that the images are more realistic than when generating data from one teacher.

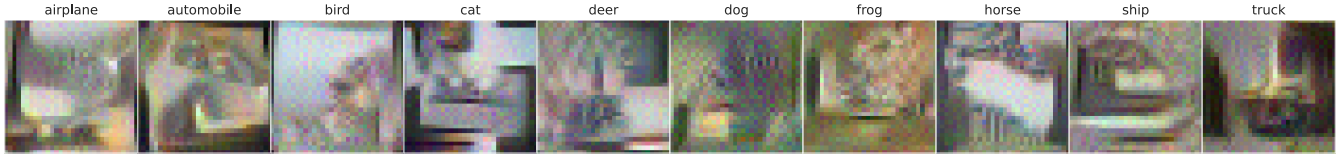
3) Using more teachers does not help: Finally we set $n = 5$ in the algorithm 1 and re-run the experiment. The result can be seen in Figure 9 in red. Although we observe less variance in the test accuracy, there is no significant gain in accuracy due to using a larger number of teachers.

D. Heterogeneous 10 teachers

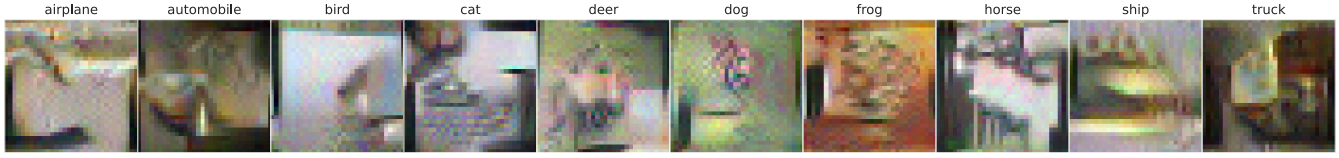
We test the case where there are imbalances in the data distribution. To do this we split the data in the following manner: half of the training set is split randomly and the other half is distributed by class to each teacher (1 teacher has only one label from this half).
We use algorithm 1 with a modified loss for each teacher. We keep all terms in equation 4 the same except the $CrossEntropyLoss$, which is weighted for each teacher according the the percentage the label represents from the teacher dataset:

$$w_l^t = \frac{\#\text{samples of labels } l \text{ at teacher } t}{\#\text{samples at teacher } t} \tag{5}$$

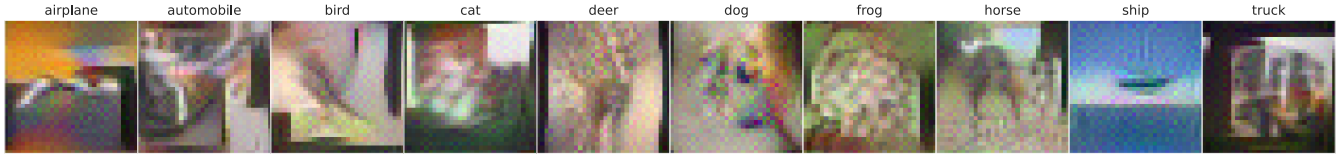
(a) 1 teacher.



(b) 10 teachers.



(c) 10 teachers with noise removed (more updates).



(d) 10 teachers with sampling (3 teachers per batch).

Figure 10: Difference in image quality between different batch generation methods.

The justification for this is that each teacher should have more weight if it has more data from a given label. This results in an accuracy of 68.3%.

E. Fully heterogeneous case with 2 and 5 teachers

Finally we experiment with a fully heterogeneous data distribution. This means that the teacher labels do not intersect. In the extreme case, if there are 10 teachers, each one only has one label therefore the teacher models would not achieve any real learning. This means that dataset distillation cannot be used to learn a joint model. We therefore run the experiment with 2 and 5 teachers.

1) Masking of the unused heads helps: For two teachers, we verify that masking the output heads and leaving only those corresponding to classes the teacher has seen improves performance. Figure 11 shows the result of training both with and without the masking. We observe a 9% increase in accuracy in the former case. When using masking and two teachers, we obtain a test accuracy of 64.6%

2) Image quality degradation when using more teachers: Figure 12 shows a sample of the images distilled using 1, 2 and 5 teachers. We observe a drop in the level of detail and realism as we use more teachers. Similar to the 10 teacher case, we run the experiment with more gradient updates per batch (3000 instead of 1000) and obtain very similar results.

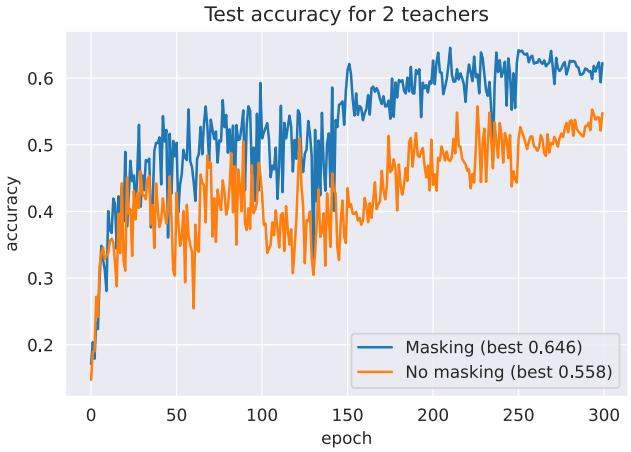


Figure 11: Test accuracy for two teachers (completely heterogeneous).

In this case however, there is no clear way of doing weighted averaging of the teacher losses, as the local datasets do not intersect (in terms of labels).

3) 5 teachers: In this case, training only reaches 43.8 test accuracy.

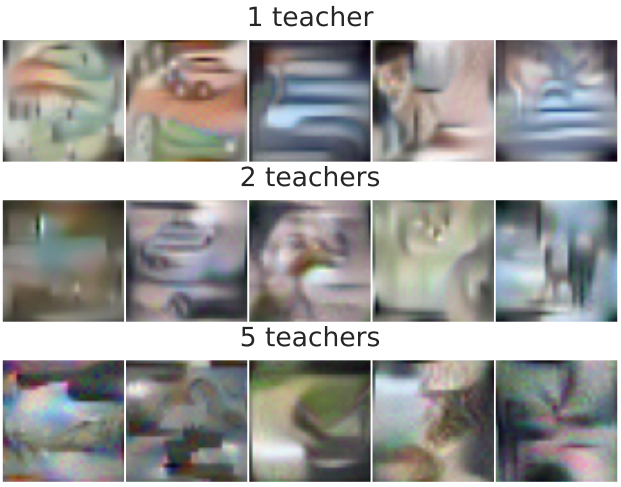


Figure 12: Quality of images. 1, 2 and 5 teachers (completely heterogeneous).

V. Implementation

We implement the experiments using PyTorch [9] for the training logic, and Hydra [10] for configuring the experiments. Our code is available publicly [11] on Github.

VI. Ideas for future work

We mention a few directions that we did not explore due to time constraints, but that could be interesting for the future:

- *IID training*: IID training using dataset distillation shows promising results. We believe that by further tweaking the loss function and the hyperparameters of batch generation, performance could reach state-of-the-art level.
- *Heterogeneous training*: Much still can be done for imbalanced training. Using some form of aggregation between teachers seems to be necessary. However, this aggregation does not have to be in the form of averaging, or if it is, the weights can be optimized further.
- *Fixed teacher sample for one batch*: When generating one batch, it would be interesting to test whether picking a set of teachers for all iterations instead of one sample for each iteration, would improve performance (i.e. move line 4 from algorithm 1 before the loop).
- *Dealing with class similarity*: It could also be interesting to create a similarity vector for each class. When generating an image for a particular class, we would also minimize the other teacher scores for the most similar classes.

VII. Conclusion

In this work, we explore the use of dataset distillation for the purpose of one-shot federated learning. We find that images generated from multiple teachers do not mix well for the purpose of knowledge transfer to a randomly initialized student. This is due to a decrease in the realism of the images, as well as problems due to semantically close classes being in different teachers. We propose a weighted averaging scheme of the loss function when generating a batch, and reach reasonable performance in the IID and mildly heterogeneous case.

References

[1] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y. Arcas, "Communication-Efficient Learning of Deep Networks from Decentralized Data," in *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, ser. Proceedings of Machine Learning Research, A. Singh and J. Zhu, Eds., vol. 54. PMLR, 20–22 Apr 2017, pp. 1273–1282. [Online]. Available: https://proceedings.mlr.press/v54/mcmahan17a.html

[2] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *CoRR*, vol. abs/1502.03167, 2015. [Online]. Available: http://arxiv.org/abs/1502.03167

[3] H. Yin, P. Molchanov, Z. Li, J. M. Alvarez, A. Mallya, D. Hoiem, N. K. Jha, and J. Kautz, "Dreaming to distill: Data-free knowledge transfer via deepinversion," *CoRR*, vol. abs/1912.08795, 2019. [Online]. Available: http://arxiv.org/abs/1912.08795

[4] I. Sutskever, J. Martens, G. Dahl, and G. Hinton, "On the importance of initialization and momentum in deep learning," in *Proceedings of the 30th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, S. Dasgupta and D. McAllester, Eds., vol. 28, no. 3. Atlanta, Georgia, USA: PMLR, 17–19 Jun 2013, pp. 1139–1147. [Online]. Available: https://proceedings.mlr.press/v28/sutskever13.html

[5] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," University of Toronto, Toronto, Ontario, Tech. Rep. 0, 2009.

[6] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015. [Online]. Available: http://arxiv.org/abs/1512.03385

[7] H. Yin, P. Molchanov, Z. Li, J. M. Alvarez, A. Mallya, D. Hoiem, N. K. Jha, and J. Kautz. Deep inversion online repository. [Online]. Available: https://github.com/NVlabs/DeepInversion

[8] P. Micikevicius, S. Narang, J. Alben, G. F. Diamos, E. Elsen, D. García, B. Ginsburg, M. Houston, O. Kuchaiev, G. Venkatesh, and H. Wu, "Mixed precision training," *CoRR*, vol. abs/1710.03740, 2017. [Online]. Available: http://arxiv.org/abs/1710.03740

[9] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style,

high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds. Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[10] O. Yadan, "Hydra - a framework for elegantly configuring complex applications," Github, 2019. [Online]. Available: https://github.com/facebookresearch/hydra

[11] A. Jellouli. Code accompanying the report. [Online]. Available: https://github.com/Ahmedjjj/Fed_distill