Federated Learning for Pupil Dynamics

Timm Kleipsties
t.kleipsties@stud.uni-hannover.de
Leibniz University
Hanover, Germany

Simon Lübeß
simon.luebess@stud.uni-hannover.de
Leibniz University
Hanover, Germany

Abstract

This paper investigates the applicability of Federated Learning (FL) for the continuous improvement of a neural network designed for pupil segmentation. This network is a core component of a portable device aimed at concussion detection through pupil dynamics analysis [1]. Addressing the challenge of acquiring labeled medical images for training in a real-world deployment scenario, we explore a self-supervised approach using Monte Carlo Dropout inference within a Federated Learning framework. We evaluate and compare different Federated Learning strategies, specifically Federated Averaging and Federated Stochastic Gradient Descent, combined with variations in hyperparameters such as client data points per round and communication rounds. Our experiments, conducted using a U-Net architecture and synthetic infrared eye images, reveal that while Federated Learning with ground truth labels shows potential, our self-supervised MCD approach does not demonstrably outperform a well-pretrained centralized model in this specific use case. Furthermore, practical considerations concerning edge device training and the availability of labeled data for self-supervision in a medical product context lead us to conclude that Federated Learning may not be the most advantageous strategy for the continued improvement of our pupil segmentation model.

1 Introduction

Introduction: FL has emerged as a promising solution to the challenges of data sharing in healthcare. It allows institutions like hospitals to collaborate on developing machine learning models without compromising patient privacy [2]. This is achieved by training models locally on each institution's data and then aggregating the model updates at a central server. The raw data never leaves the individual institutions, ensuring compliance with data protection regulations such as HIPAA and GDPR [2]. FL has the potential to accelerate medical research by enabling the creation of more robust and generalizable models trained on larger and more diverse datasets [2]. One significant advantage of FL is its ability to address the "small sample size" problem often encountered in medical image analysis. By enabling collaborative training on data from multiple institutions, FL can overcome the limitations of small datasets at individual sites and lead to more accurate and reliable models [3].

Background Literature (Related Works): There are plenty examples of successful applications of federated training for segmenting medical imagery. One such example is Lutnick et al. [5] who used FL with FedAvg to train Convolutional Neural Networks (CNN) on medical data for interstitial fibrosis and tubular atrophy (IFTA) and also for glomeruli segmentation. They trained on data from three different hospitals and compared the performance of the models trained using FL with a centrally trained model and models trained only on data from each individual hospital. They showed that FL performs at least on par with a centrally trained model and can even outperform it. Theyo and Godfrey [7] used FL to compare different U-Net architectures (U-Net, U-Net++ and Attention U-Net) on three different data sets for image segmentation. All experiments showed good results. A large difference between these works and our work is, that they can use supervised learning approaches while we test an unsupervised approach. We couldn't find a source for self-supervised federated learning with medical imagery. Psaltis et al. [6] used self-supervised approaches to derive maximum benefit from partially labeled data sets. They deemed self-supervised techniques like contrastive learning too expensive and complex for use on clients, so instead their approach uses supervised learning on the clients and semi- or self-supervised learning on the host-server. They found that their approach yields very good results.
Motivation: The general challenge, from Marcel Schepelmann and his study at the CHI, is designing a device to detect concussions by measuring pupil dynamics, by taking multiple infrared images of an eye and track its behavior. To accomplish that functionality, it was decided to use a Neural Network to predict the segmentation of the pupil for a given infrared image of an eye [1]. Since it is difficult to get hold of large amounts of medical training data (i.e. medical imagery), that is required for machine learning applications, our idea was to embrace the concept of FL for this application and circumvent many privacy concerns. While we still need consent to train on the data, we never transfer the actual image to a centralized server. Instead, we transfer the model itself to the clients, train them with the data, and only transfer the gradients / updated model back to create a global model.
Contribution: In our work, we used synthetic generated images by Jan Kaminski from his ongoing master's thesis at the CHI, which enabled us to use 1000 images for training, testing and distributing to clients. We want to evaluate, if the concept of FL is appropriate for continued training of the neural networks after releasing the product to the public and our results should serve as a guide whether the methodology is feasible for the application. Furthermore, we tackled the challenge of self-supervised learning by implementing a simplified version of Monte Carlo Dropout (MCD) inference in combination with FL.
Our code is available in our corresponding repository[1].

[1] https://github.com/aharabada/FederatedLearning

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM
https://doi.org/10.1145/nnnnnnn.nnnnnnn

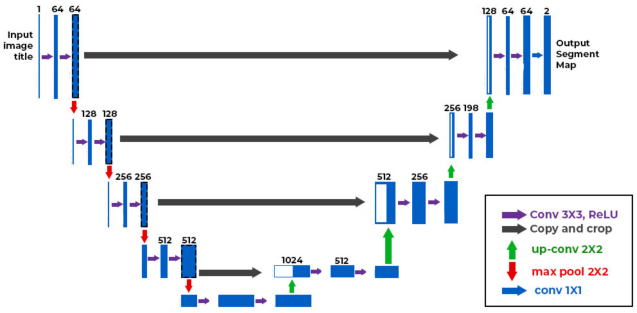

Figure 1: Our proposed Architecture [4]

2 Proposed Methodology

Problem Statement: One of the steps in the detection of concussions is a neural network that creates a segmentation of images of the persons eye. In this segmentation only the pupil will be marked as 1 (white) everything else will be 0 (black). To get an idea, refer to Figure 30 in appendix Results A. In this work, we will focus on this problem.

Furthermore the use-case of the final product forces the use some sort of self-supervised learning, since it is not applicable to request the user to manually draw the true binary mask for a new image. As mentioned, we opted for a simplified version of MCD to create our own labels, by taking the mean of n inferences, and use the uncertainty as the loss for training (refer to equation 4).

Also regarding the use-case we decided to pretrain the model for 50 iterations, as this being used in a medical device, it should have proper initial performance before being shipped.

Later we evaluate multiple experiments with Federated Averaging (FedAvg) and Federated Stochastic Gradient Descent (FedSDG) combined with different sets of hyperparameters to see, whether Federated Learning can improve the models performance.

Flow of Proposed Architecture: The proposed model is a so called U-Net model. In our case this model consists only of Double Convolutions that are chained after each other with decreasing and increasing dimensions: *Convolution Layer, Batch Normalization Layer, ReLU Layer, Dropout Layer, Convolution Layer, Batch Normalization Layer, ReLU Layer, Dropout Layer* and in the encoder a *Max Pooling Layer* after each Double Convolution (as shown in Figure 1). The output is passed to a Sigmoid function to scale the output mask between 0 and 1. The Input is embedded as a square gray-scale normalized image.

For pre-training and testing we used a combination of Dice- and Binary Cross Entropy (BCE) Loss. For MCD inference on clients we used the mean of a percentage of the highest variances over n model predictions on the same input image.

As the metric for evaluation the Intersection over Union (IoU) was chosen.

Explanation of proposed components: The integrated loss function of our proposed system for pre-training and testing is realized as follows (refer to equation 1):

$$L_{\text{DiceBCE}} = \frac{1}{2}L_{\text{BCE}} + \frac{1}{2}L_{\text{Dice}} \tag{1}$$

$$L_{\text{BCE}} = -\frac{1}{N}\sum_{i=1}^{N}\left(y_i \log(\hat{y}_i) + (1-y_i)\log(1-\hat{y}_i)\right) \tag{2}$$

$$L_{\text{Dice}} = 1 - \frac{2\sum_{i=1}^{N} y_i\hat{y}_i + \epsilon}{\sum_{i=1}^{N} y_i + \sum_{i=1}^{N}\hat{y}_i + \epsilon} \tag{3}$$

With L_{DiceBCE} being the combined loss of Binary Cross-Entropy Loss (L_{BCE}) and Dice Loss (L_{Dice}), each contributing equally to the overall loss. L_{BCE} is the Binary Cross-Entropy Loss, which measures the difference between the true labels (y_i) and the predicted probabilities ($\hat{y}_i$). L_{Dice} is the Dice Loss, which calculates the similarity between the predicted and true segmentation maps using the Dice coefficient, with ϵ being a small smoothing value to avoid division by zero. The loss function for MCD inference is constructed as follows (refer to equation 4):

$$L_{\text{MCD}} = \frac{1}{p \cdot M}\sum^{p\cdot M}\max_{\mathcal{T},|\mathcal{T}|=p\cdot M}\sigma^2(x), \quad \mathcal{T} \subset \{1,\ldots,M\} \tag{4}$$

With $\sigma^2(x)$ being the pixel-wise variance of n model activations, p the percentage of chosen highest variances, M the amount of pixels of the output and $\mathcal{T}$ the subset of selected pixels.

In detail we activated the *Dropout Layers* for inference and then passed the same image n times through the model. After that we calculated the variance for each pixel over these n passes and determined the $M \cdot p$ highest of these variances. The final loss is calculated by taking the mean over these values. We decided to choose only p percent of the pixels, because most of the variances will be close to 0, since there will most likely only be noticeable variances at the edges of the predicted pupil.

As for the formula for IoU refer to (5):

$$\text{IoU} = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|} \tag{5}$$

With A being the prediction and B the target.

3 Experiments

3.1 Dataset

The dataset consists of 1000 computer-generated (infrared-like) images of an eye and the corresponding image showing the ideal segmentation of the pupil. These images were generated using tools written by Jan Kaminski for his ongoing research at CHI. For each image, several parameters were generated pseudo-randomly. These are parameters such as the direction that the eye is looking, how far open or closed the eyelid is, and (and arguably most importantly for their work) the dilation of the pupil, e.i. how large or small the pupil is.

We used 400 of the images for pretraining. 80% of these were used for training data and 10% for test- and validation-data respectively. 500 images were split up between the clients. And the remaining 100 were kept for testing the models during the federated learning process.

While the dataset was sufficient for our work the way we received it, we noticed that in about 10%-20% of the images the eyelid completely obscured the pupil, rendering a segmentation impossible for these cases (or the result should be, that no pupil is detected

and the image is completely black). In the dataset however, the segmentation images still show a segmentation for the pupil for these cases, which later imposed a lower bound on the training loss (of about 0.1) for our model (when fully trained), because it is obviously impossible to achieve even a reasonable estimate of the pupil if we cannot see it. Because of this, we later decided to semi-automatically fix the dataset by using the first model we trained on this dataset to detect cases where no pupil is visible and replaced the invalid ground truth-segmentation with one where all pixels are zero/black.

3.2 Implementation Details

Hyperparameters: A visual representation of the used network can be seen in Figure 1. The input images have a single color channel and 144 x 144 pixels. The number of input channels of the down-convolution layers are 1 and 64 for the first two layers. After that it doubles with every consecutive layer, thus 128, 256, 512 and 1024 respectively. Since every down-convolution is followed by a max-pooling-layer, the input size of each down-convolution layer is halved. The last down-convolution layer, the bottleneck-layer, has an input size of 9 x 9. For each down-convolution a kernel of size 3x3 with a stride of 1 is used with a padding of 1. After the bottleneck a transposed convolution-layer is used for up-sampling between the double-convolutions layer. These double each dimension of the layers input data. Meanwhile each input layer has half the number of input channels, thus 512, 256, 128, 64 and 1 as the number of output channels. The output image is an image with a single color channel and a resolution of 144 x 144 pixels. Both the max-pools as well as the transposed convolution-layers use a kernel size of 2 and a stride of 2. The double-convolution layers each consist of two convolution-layers. These are each followed by a batch-normalization, we use ReLU as activation function. After the ReLU follows a dropout-layer. The first layer has the number of output-channels of the preceeding double-convolution-layer as number of input-channels and the number of input-channels of the following double-convolution-layer as output-channels. The second convolution layer has the same number of channels for input as well as output. Both are the number of output-channels as the first convolution for this double-convolution.

We used the Adam optimizer. For pre-training on the host we used a batch size of 64, an initial learning rate of 0.0005 and a weight decay of 1×10^{-5}. We used pytorchs built-in 'ReduceLROnPlateau' scheduler. We configured it, so that it halves the learning rate when the loss didn't improve for five consecutive training iterations.

For training on the clients we always used a batch size of 1 to simplify calculations and the Adam optimizer with a learning rate of 1×10^{-4}.

For the experiments with FedSGD the host model was trained using the SGD-optimizer with a learning rate of 0.005. We initially trained with a learning rate of 0.0005, but this resulted in the gradients having minimal influence (close to none) on the global model, so we decided to raise the learning rate by a factor of 10. For experiments regarding FedAvg we kept the Adam optimizer.

Referring to L_{MCD} (4) a dropout-rate of 0.1 was chosen for the *Dropout Layers*. As percentage of considered variances p we picked 10%. At last, the amount of MCD inferences per client was 5.

Experiment	Rounds	Clients	It./Client	DP./Client/Round
FedAvg 1.1	10	5	1	20
FedAvg 1.2	10	5	1	20
FedAvg 1.3	5	5	1	50
FedAvg 1.4	20	5	1	5
FedSGD 2.1	20	5	1	20
FedSGD 2.2	20	5	1	20
FedSGD 2.3	20	5	1	5

Table 1: Experiment specific hyperparameters (For FedAvg 1.1 & FedSGD 2.1 the groud truth labels were used, MCD otherwise)

For the other hyperparameters that varied between the experiments. Refer to Table 1 to see the individual hyperparameter-configurations.

Explanation of Experiment Hyperparameters: The following passage refers to Table 1. Where the *Rounds* represent the amount of times the FL loop gets executed, *Clients* the amount of clients, *It./Clients* the amount of training iterations on each client per round and *DP./Clients/Round* how many datapoints each client can use for training each round. We decided, limiting the amount of accessible data for each client would simulate realistic usage of the final device.

We wanted to exploit the influence of these hyperparameters. Initially we tried to accomplish this by using Automated Machine Learning (AutoML), but that turned out to be too resource- and time-consuming, so we opted to pick specific configurations. To receive comparable results, we evaluated each FL experiment 10 times. At first glance these hyperparameters seem a bit low, but keeping the use-case in mind and this being used in a niche product, we decided these hyperparameters are appropriate.

For FedAvg and FedSTD we started with creating a baseline by using True Labels on the clients. To have a direct comparison the experiments 1.2 and 2.2 follow the same set of hyperparameters as the baselines. In 1.3 the influence of more data per client round is examined. Since this experiment developed a trend really quick, we lowered the amount of rounds to 5. Experiment 1.4 on the other hand was used to test the opposite. To accommodate the fewer datapoints per client per round, the amount of rounds was increased. In 2.3 the impact of less smooth gradients was tested by having fewer datapoints per client.

One might notice there is always just one training iteration per round. We also tested multiple iterations and the results turned out much worse, so we sticked to one iteration.

Environment Details: We used Python 3.12 and PyTorch 2.6.0 with Cuda support on Windows 10. All training was done on the GPU.

Computation Usage: We used two PCs to run the experiments. The first PC has an AMD Ryzen 5 1600x with 6 CPU-cores, 12 threads and 16GB of RAM, as well as an NVIDIA Geforce GTX 1060 with 6GB VRam. The second PC has a Ryzen 7 2700x with 8 CPU-Cores, 16 threads and 32GB of RAM and an NVIDIA Geforce RTX 3080 with 10GB of VRam.

The experiments 1.1 - 1.4 were trained on the NVIDIA Geforce GTX 1060 6GB machine. Experiment 1.1 took around 1.5h, 1.2 &

1.4 around 2h and 1.3 ca. 3h. Experiments 2.1 - 2.3 were trained on the Geforce RTX 3080 10GB machine, where experiment 2.1 took around 2.5h and 2.2 & 2.3 ca. 4h. Pre-training of the host model was also performed on the latter machine and took ca. 30 minutes per 25 iterations.

3.3 Baselines

The pre-trained Model: All experiments were conducted using the same pre-trained model. This model was trained for 50 iterations using 400 images (40%) of the entire dataset. This model had a loss of 0.1742 and a IoU of 0.6599. We chose to pre-train for 50 iterations without FL, as this resulted in already acceptable segmentations, which reasonably simulates our use-case, as a medical product would obviously be shipped in a functional state. Meanwhile, it still leaves enough room for improvement to allow for continued training to show significant improvements.

For comparison we further trained the host model to 75 iterations. However, for the experiments from table 1 the pre-trained model at iteration 50 is used.

Baseline FL with ground truth on clients: To establish a baseline to compare the MCD-results to, we ran all FedSGD as well as FedAvg experiments with ground truth, meaning we assumed the clients have access to the actual labels. We ran all FL baseline experiments for 10 FL cycles with 20 data points per client.

Table 2 shows the L_{DiceBCE} and IoU for FedSGD, FedAvg baseline experiments as well as the pre-trained model at 50 and 75 iterations respectively. These values should serve as a baseline to compare our FL with MCD approach.

Experiment	L_{DiceBCE}	IoU
Pre-training 50 iterations	0.1742	0.6599
Pre-training 75 iterations	0.1203	0.6880
FedAvg 1.1	0.1568 ± 0.0103	0.6869 ± 0.0150
FedSGD 2.1	0.1703 ± 0.0001	0.6637 ± 0.0002

Table 2: L_{DiceBCE} and IoU of baseline experiments on testset. For 1.1 & 2.1 final round with ($\mu \pm \sigma$) and N=10

4 Results

Experiment	L_{DiceBCE} R=1	IoU R=1	L_{DiceBCE} R=n	IoU R=n
FedAvg 1.2	0.1817 ±0.027	0.6787 ±0.039	0.4157 ±0.099	0.2435 ±0.169
FedAvg 1.3	0.1908 ±0.021	0.6267 ±0.038	0.5163 ±0.056	0.0603 ±0.091
FedAvg 1.4	0.1908 ±0.021	0.6267 ±0.038	0.2407 ±0.009	0.5620 ±0.021
FedSGD 2.2	0.1742 ±0.000	0.6599 ±0.000	0.1743 ±0.000	0.6598 ±0.001
FedSGD 2.3	0.1742 ±0.000	0.6598 ±0.000	0.1743 ±0.000	0.6597 ±0.001

Table 3: L_{DiceBCE} and IoU of MCD experiments at rounds 1 and n=final on testset with ($\mu \pm \sigma$) and N=10

Results: The results for training with MCD can be seen in table 3. To display the results in a compacted way, we show the L_{DiceBCE} and IoU after the very first FL round and after the final round of each experiment. For further insights refer to the different plots in appendix Results A. In these graphs all the rounds are plotted for each experiment (and FL baseline) with two plots for L_{DiceBCE} and IoU each. One for the mean and standard deviation over 10 independent evaluations and the other for the individual evaluations.

Discussion of Results: The impacts of hyperparameters for FedAvg showed mixed results. For experiment 1.3 the many datapoints per client resulted in a drastic decay of IoU. Most of the time the IoU stagnated close to 0.0 after just 5 rounds (refer to Figure 12). With fewer datapoints, such as in experiments 1.2 (Figure 8) and 1.4 (Figure 16), the effect was less dramatic. With 20 datapoints per client there even was an average improvement over the baseline for the first two rounds (refer to Figure 8).

The changes over rounds regarding IoU and Loss from FedSGD compared to FedAvg are way smaller. Even the changes for the baseline experiment 2.1 are way smaller than the ones from 1.1. But still, there is a constant improvement over time, unlike in the other FedSGD experiments 2.2 and 2.3 with MCD. Only in experiment 2.2 in round 3 there is an improvement in every of the 10 independent evaluations in terms of IoU over the pre-trained model at 50 iterations.

5 Conclusion

The results imply that our approach of FL with MCD has no overall advantage over the baselines. Even though there are some minor improvements for clients with few data points in the first few rounds (experiment 1.2 & 2.2), it is hard to justify the overhead of federated learning compared to the pre-trained model at 75 iterations. To further underline this conclusion, every of our MCD experiments resulted in an overall decay of IoU. The reason might be our simplified implementation of MCD. Nevertheless, there are more reasons we think that using Federated Learning for our specific use-case is not feasible.

First, the models have to be trained on the clients, which would be a relatively low performance edge device in our application. Even training on an NVIDIA Geforce GTX 1060 6GB proved difficult in certain circumstances, which implies that doing this kind of work on edge-devices might be impossible. Especially if we tried to use more sophisticated self-supervised methods like contrastive learning.

Second, even though there is shown to be an improvement using true labels on the clients, it is unreasonable for our use-case. As this device is intended to be used by non-medical users, and we can't reasonably expect these users to manually create good true-labels for our training purposes.

Finally, in our case, the pre-trained model at 75 iterations was already outperforming every of our FL approaches. As mentioned above, more sophisticated FL approaches or simply more rounds of FL might solve this problem, but still leaves the question if FL makes sense in this use case: Since we have a tool to generate more synthetic training images, that can be used for further improvement of the model, we might be able to achieve a model with high enough precision without continued training after shipping.

References

[1] 2024. CARS 2024—Computer Assisted Radiology and Surgery Proceedings of the 38th International Congress and Exhibition Barcelona, Spain, June 18–21, 2024. *International Journal of Computer Assisted Radiology and Surgery* 19, 1 (2024), 147–148. https://doi.org/10.1007/s11548-024-03128-9

[2] Pallavi Dhade and Prajakta Shirke. 2024. *Federated Learning for Healthcare: A Comprehensive Review.* Retrieved February 28, 2025 from https://www.mdpi.com/2673-4591/59/1/230

[3] Hao Guan et al. 2024. *Federated Learning for Medical Image Analysis: A Survey.* Retrieved February 28, 2025 from https://www.mdpi.com/2673-4591/59/1/230

[4] GeeksforGeeks. 2025. *U-Net Architecture Explained.* Retrieved February 27, 2025 from https://www.geeksforgeeks.org/u-net-architecture-explained/

[5] Brendon Lutnick, David Manthey, Jan U Becker, Jonathan E Zuckerman, Luis Rodrigues, Kuang-Yu Jen, and Pinaki Sarder. 2022. A tool for federated training of segmentation models on whole slide images. *J Pathol Inform* 13 (May 2022), 100101.

[6] Athanasios Psaltis, Anestis Kastellos, Charalampos Z. Patrikakis, and Petros Daras. 2023. FedLID: Self-Supervised Federated Learning for Leveraging Limited Image Data. In *2023 IEEE/CVF International Conference on Computer Vision Workshops (ICCVW).* 1031–1040. https://doi.org/10.1109/ICCVW60793.2023.00111

[7] Elvis Theyo and Wilfred Godfrey. 2023. *Federated Learning for Multi-Institutional Medical Image Segmentation.* Gwalior, India. https://github.com/avocadopelvis/federated-learning/tree/main

A Results

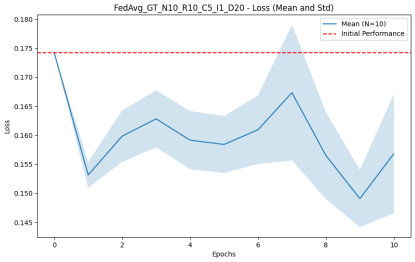

Figure 2: L_{DiceBCE} of baseline experiment 1.1 ($\mu \pm \sigma$) and N=10

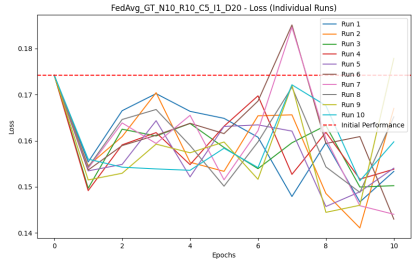

Figure 3: L_{DiceBCE} of baseline experiment 1.1 exact runs

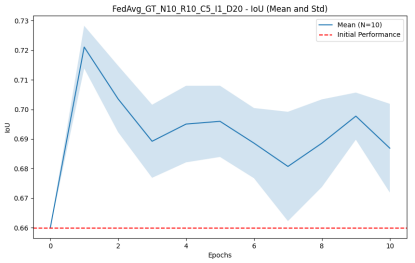

Figure 4: IoU of baseline experiment 1.1 ($\mu \pm \sigma$) and N=10

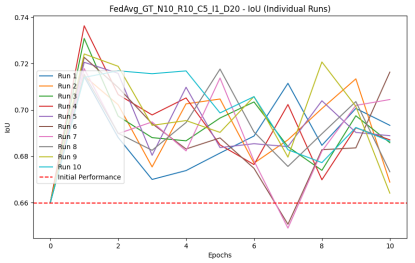

Figure 5: IoU of baseline experiment 1.1 exact runs

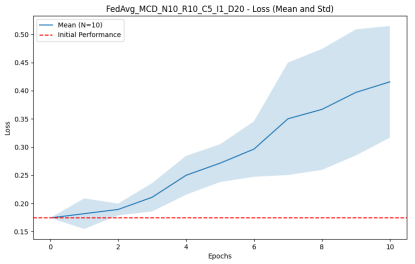

Figure 6: L_{DiceBCE} of experiment 1.2 ($\mu \pm \sigma$) and N=10

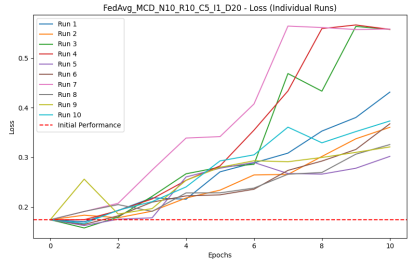

Figure 7: L_{DiceBCE} of experiment 1.2 exact runs

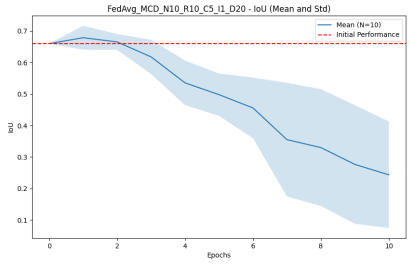

Figure 8: IoU of experiment 1.2 ($\mu \pm \sigma$) and N=10

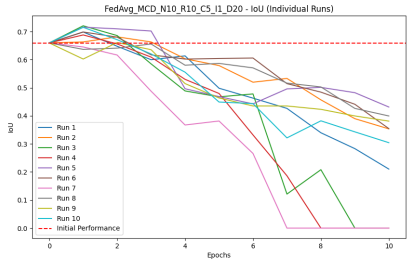

Figure 9: IoU of experiment 1.2 exact runs

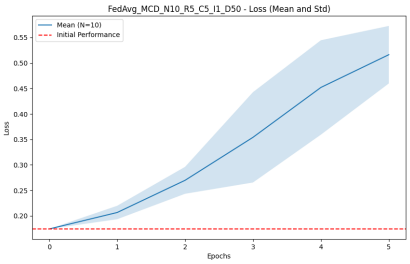

Figure 10: L_{DiceBCE} of experiment 1.3 ($\mu \pm \sigma$) and N=10

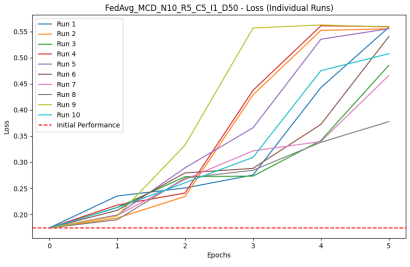

Figure 11: L_{DiceBCE} of experiment 1.3 exact runs

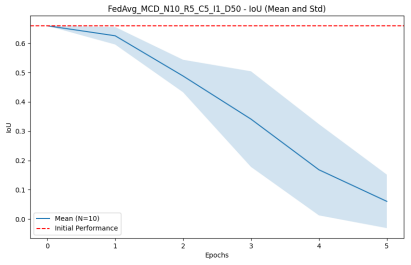

Figure 12: IoU of experiment 1.3 ($\mu \pm \sigma$) and N=10

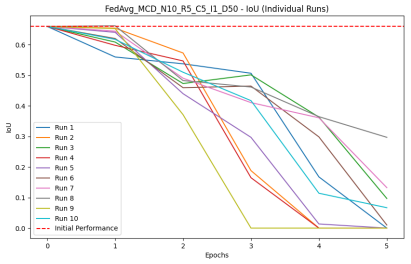

Figure 13: IoU of experiment 1.3 exact runs

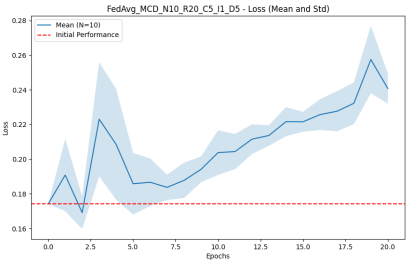

Figure 14: L_{DiceBCE} of experiment 1.4 ($\mu \pm \sigma$) and N=10

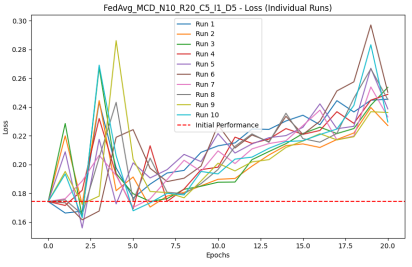

Figure 15: L_{DiceBCE} of experiment 1.4 exact runs

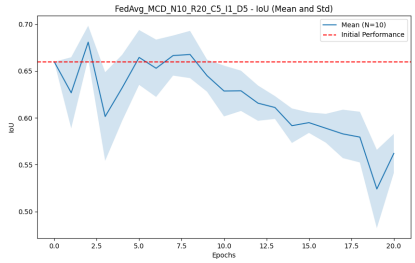

Figure 16: IoU of experiment 1.4 ($\mu \pm \sigma$) and N=10

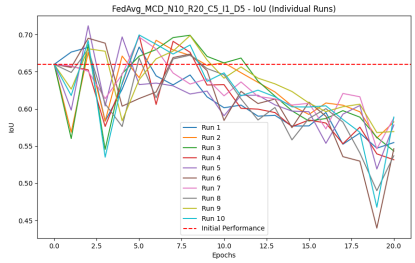

Figure 17: IoU of experiment 1.4 exact runs

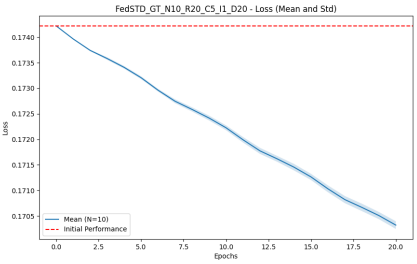

Figure 18: L_{DiceBCE} of experiment 2.1 ($\mu \pm \sigma$) and N=10

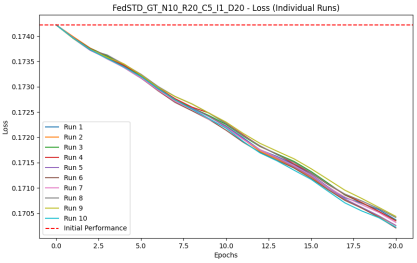

Figure 19: L_{DiceBCE} of experiment 2.1 exact runs

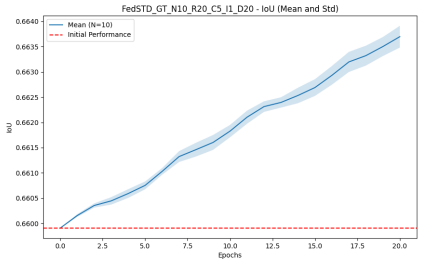

Figure 20: IoU of experiment 2.1 ($\mu \pm \sigma$) and N=10

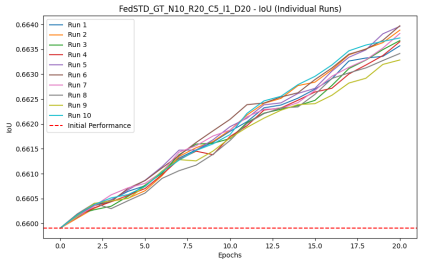

Figure 21: IoU of experiment 2.1 exact runs

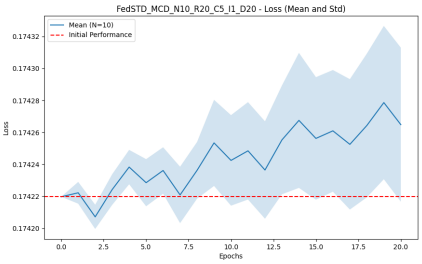

Figure 22: L_{DiceBCE} of experiment 2.2 ($\mu \pm \sigma$) and N=10

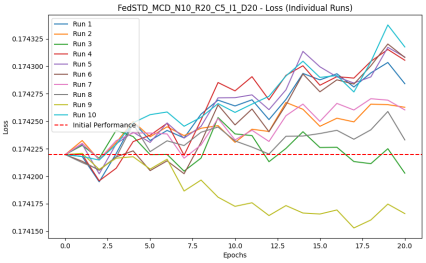

Figure 23: L_{DiceBCE} of experiment 2.2 exact runs

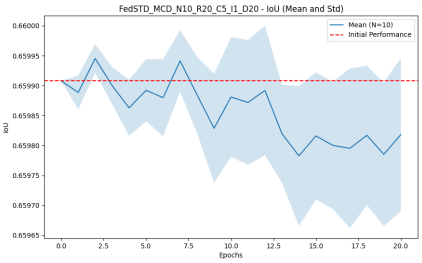

Figure 24: IoU of experiment 2.2 ($\mu \pm \sigma$) and N=10

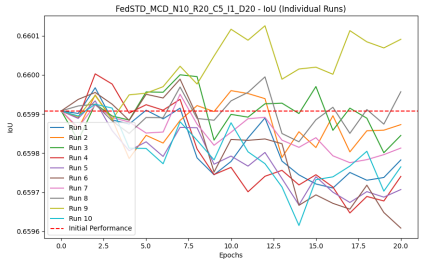

Figure 25: IoU of experiment 2.2 exact runs

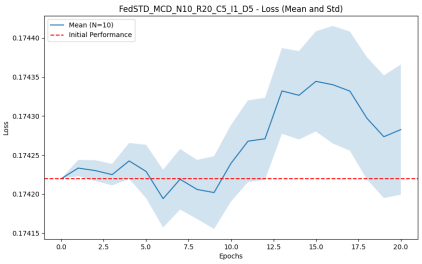

Figure 26: L_{DiceBCE} of experiment 2.3 ($\mu \pm \sigma$) and N=10

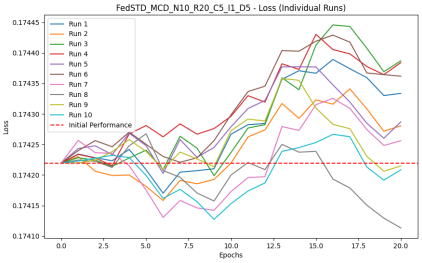

Figure 27: L_{DiceBCE} of experiment 2.3 exact runs

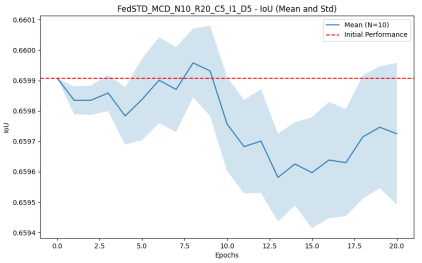

Figure 28: IoU of experiment 2.3 ($\mu \pm \sigma$) and N=10

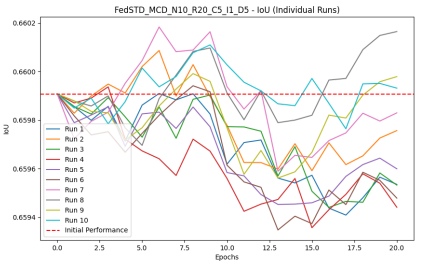

Figure 29: IoU of experiment 2.3 exact runs

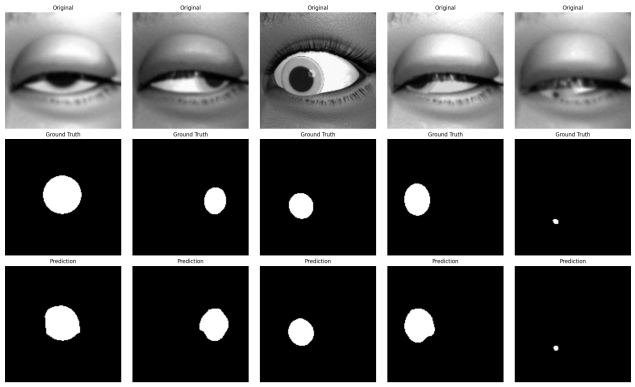

Figure 30: Example segmentation with pre-trained model at 75 training iterations