1. Introduction
2. Background

2.1 Machine Learning in Bioinformatic fields

What can be done using ML (several examples using Neural network), may other ways that use old machine learning.

2.2 Performance and dataset

If there’s a paper about data size and training performance… Introduce several famous datasets in biology/medical fields. But compare to large dataset/baseline like imagenet xxx. Or describe the need for dataset like lack sth???

2.3 Federated Learning

What is / why use FedML (privacy/distribute computing or storing resources), challenges, federated learning frameworks. Federated success case in biology field.

1. Datasets and Methods

In this project, the CIFAR-10 dataset has been used for locally simulating different scenarios of federated learning, the OIA-DDR dataset has been used for implementing an example bioinformatic federated learning on the cloud. The performance of the models is not high enough for any actual usage, but this project aims to compare between federated training and the centralized training, current training designs are sufficient for showing their differences.

For each simulation task, we run 5000 epochs (for centralized training) or global rounds (for federated learning) and apply an early stopping of 500. For the cloud implementation, we run 100 epochs/rounds.

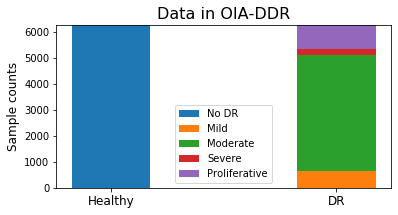
3.1 CIFAR-10 dataset

The CIFAR-10 dataset consists of 50,000 training images and 10,000 testing images, each image is in color format and the size is 32x32. In this project, 4000 training images and 10,000 testing images have been used for scenario testing.

3.2 OIA-DDR dataset

The OIA-DDR dataset provides high-quality diabetic retinopathy images and annotations. The grading annotation label the images to six classes: no DR, mild, moderate, severe, proliferative, and ungradable. In this project, we neglect images from the ‘ungradable’ class, using 6266 images from ‘no DR’ class as healthy samples and 6256 images from the rest 4 classes as DR samples.

Images in this dataset are in different sizes and qualities. We preprocess the images using the methods from public kernels on Kaggle, crop and resize them to size 224x224. (<https://www.kaggle.com/ratthachat/aptos-eye-preprocessing-in-diabetic-retinopathy>) (<https://www.kaggle.com/titericz/circle-to-rectagle-preprocessing-1>)



(Figure xxx each class’s count)

3.3 Training designs

The task for CIFAR-10 is to input images of size 32x32x3 and classify them to 10 classes. The model is provided by Mattias Åkesson. We use ADAM as the optimizer and TensorFlow’s default settings for other parameters. The batch size is set to 50. Labels are in integer format; loss will be calculated using sparse categorical cross entropy.

The task for OIA-DDR dataset is to input images of size 224x224x3 and classify them to 2 classes. VGG16 network (pre-trained on ImageNet dataset) is used. We use ADAM with learning rate initialized at 1e-4 and apply inverse time decay at rate 0.05 in every epoch. The batch size is set to 300. Labels are in string format; loss will be calculated using binary cross entropy. Other settings are by default.

learning\_rate = 1e-4/(1+0.05\*epoch\_number)

In this project, we evaluate the training performance with accuracy:

Accuracy = (Correctly assigned samples)/(All samples)

3.4 Scenarios testing

We simulate and test different scenarios in federated learning in this part. In this simulation system, we assume a folder to be the central node and several folders to be worker nodes and assign data’s index to each folder accordingly.

When the system starts, the central node will initialize and save the model weight in its folder. We apply Federated Averaging in this system. Each epoch consists of the following basic actions:

1. Worker nodes generate local model,
2. Worker nodes load weight from the central node,
3. Worker nodes train the model over their local data and save the history,
4. Worker nodes return weights to the central nodes,
5. Central node averages the weights and evaluate on the testing set,
6. Central node saves weights in its folder.

After simulation, we collect the evaluation results in central node’s folder, we also collect and average the node’s training history in each node’s folder.

In a pilot test we

|  |  |
| --- | --- |
| Total Training Data | 4000 IID set |
| Total Testing Data | 10,000 IID set |
| Worker Node Number | 10 |
| Local Rounds | 1 |
| Global Rounds | 5000 |
| Early Stopping | 500 |
| Evaluation | By Central Node |
| Node Local Data | Equal |

Table xxx. Default Settings of Federated Simulation

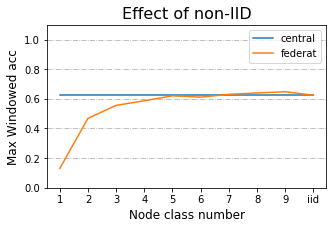
In previous works we already know that a non-IID training set can result in performance degradation in federated learning. There is no obvious standard to quantify the degree of non-IID, so we show it by identifying how many classes each node has. In a pilot test, we see different degrees of degradation (Figure xxx):

1 class non-IID never behaves better than random,

2-4 classes non-IID behave obviously worse than central training,

5-9 classes non-IID can reach similar accuracy as central training,

IID classes can reach similar accuracy as central training.



(Figure xxx. Pilot test)

This figure plots the maximum windowed accuracy of each federated training (window size=20). We run 500 global grounds for each task with the default settings.

Since we are not sure if different non-IID degrees will behave in similar trends in different scenarios, by default we test 2 classes, 5 classes and IID classes in all following scenarios. We neglect 1 class non-IID since we know it will never be trained locally.

***Evaluate methods (TO BE EDITED)***

In default settings the central node performs the evaluation. In this scenario we test the probability of evaluating the model using the node’s data.

Since in this scenario only evaluation data matters, we only perform the testing over IID classes by default settings, train for 200 global rounds. We then distribute the 10,000-testing set to the worker nodes in an unbalanced and non-IID way.

Before performing local training, each node will evaluate last global round’s result with local testing set and save accuracy locally (by each class). When the simulation ends, we manually collect the results. We will first get the average accuracy of each class (weighted by the size of that class in each node’s testing set), then average the class accuracy (weighted by 1 for each class). After that, we compare the result to the evaluation records from the central node, using the same testing set.

***Malicious node***

In this scenario we test the situation when there is a malicious node that returns randomized weights at each global round. We have 10 worker nodes to share the 4000-training data and added in a malicious node who claims to have 40 or 400 training data.

***Local Steps***

In default settings, we set our local step number as 1. That means each node train with local data for 1 round before uploading the weights. In this scenario, we also test 10 and 50 local rounds before uploading. A previous research has tested this scenario with small local steps (Zhao *et al.* 2018), we also want to see the effect with larger local steps.

***Data Dispersion***

In default settings, we distribute the 4000-training data to 10 workers. Here we also distribute the data to 40 workers.

***Delayed Update***

This scenario happens when some nodes update slower than others. We test the situation from 2 aspects:

1. 20%, 50% or 80% nodes delayed, they load the weights at global round n, and upload the weights at global round n+3. New weights won’t be loaded until current weights are uploaded.
2. 20% nodes delayed, they load the weights at global round n, and upload the weights at global round n+3, n+12 or n+30. New weights won’t be loaded until current weights are uploaded.

***Share data***

This special scenario is tested with 1 class non-IID and IID. In the pilot test, 1 class non-IID never behaves better than random. A previous research suggested a data-sharing strategy that can improve the performance of training over non-IID data (Zhao *et al.* 2018). In that research, the central node provides a small IID set (2.5% to 25% as large as total training dataset) to be shared among nodes.

We apply the strategy to our model and 1 class non-IID data, sharing size are 2.5%, 10% and 20% of the total training set. In each node there will be a mixture of shared data and local data. Apart from sharing the data, in each global round, we randomly resample an IID subset from the mixed dataset as the local training set of that round.

Also, we tested this share data strategy on IID data to detect if there are possible negative aspects.

3.5 Cloud implementation (NOT yet finished)

In this section……

1. Results

4.1 Scenarios testing

***Evaluate Methods (need to edit, maybe change)***

***Malicious node***

***Local Steps***

***Data Dispersion***

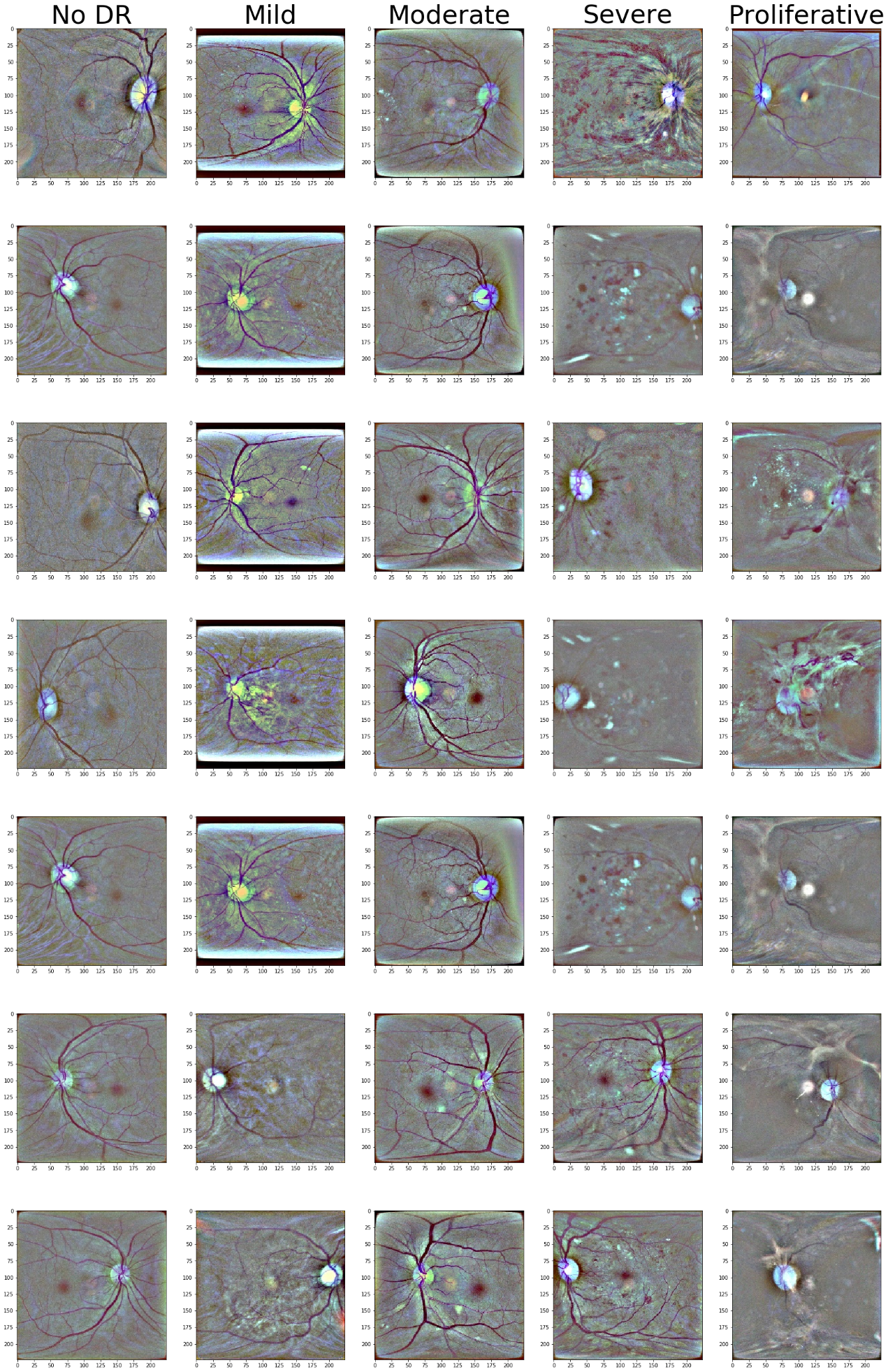
***Delayed Update***

***Share data***

4.2 Cloud implementation

1. Discussion
2. Conclusion
3. Acknowledgement
4. Reference

Appendix 1 OIA-DDR dataset after preprocessing



Appendix 2