2016311192

정동원

Kaggle ID : **dg52316**

Final Report

1. Description of your method and reason for the choice (more than 1 page, most important)

For the methods I used for semi-supervised learning, I used pseudo-labeling and data augmentation. I used pseudo-labeling because there are only 5000 labeled train data that I can use. For this reason, I had to utilize the 355551 unlabeled data in order to maximize the performance. I used data augmentation for the robust training and performance boost.

A. Pseudo-Labeling

Pseudo-labeling is basically using the predictions of the unlabeled data for the labels of that data. So, if the initial model's performance is low, the predictions of the unlabeled data would not be accurate which leads to incorrect labels for the unlabeled data. In order to solve this issue, I slightly used different method for the pseudo-labeling.

The steps of my pseudo-labeled training is as follows:

a. Train the model with labeled data until convergence (or early stopping triggered)

b. Infer on unlabeled data using the trained model.

c. Use the predictions gotten from (b) as the pseudo-labels of the data.

d. Calculate the loss and adjust the loss by the parameter alpha (loss = loss * a)

e. Backpropagation

f. [every 50 iteration] Train one epoch on labeled train dataset

g. Repeat b to f until convergence (or early stopping triggered)

My pseudo-labeling method is similar to the conventional pseudo-labeling except for step (d) and (f). The reason why I put step d is that at the initial stages, the predictions would not be as accurate, so I would want to train only a little bit by reducing the loss for the backpropagation at the initial stages. As the model get accurate, the loss that is back propagated increases linearly.

The initial alpha is 0. The equation of calculating the loss is like the following:

*Loss = Loss * (step/500) * 2 ('step' is the number of iterations)*

So, as the number of iteration increases, the loss gets larger. After 250 steps, the loss gets bigger than the original loss.

Also, I added a safety measure which is step (f). Every 50 iterations, one epoch of labeled data is trained in order to guide the training into correct path. Since only the unlabeled data has been trained, the training can be misled. In order to fix this, labeled data is trained every 50 iterations.

B. Data Augmentation

I used data augmentation for the labeled training dataset. I didn't use any augmentation on valid and test dataset because I could not get the accurate result if the data is augmented. Also, I did not use data augmentation for the unlabeled training dataset because I thought it would make the training harder. However, if it were done in proper way, it may have led to the increased performance.

I used two data augmentation methods; random horizontal flip and random crop. I thought the two methods are light ways to augment the data. I used torchvision transform module for the two data augmentation methods.

2. **Data Preprocessing / Model and Hyper parameter choices.**

A. Data Preprocessing

For the train dataset, I used 4500 labeled data and for the validation dataset, I used 500 labeled data.

For the whole dataset, I resized the image to (128,128) and changed the image into tensor and normalized the tensors. In order to use the resnet50, I had to increase the image size and normalize them. I used the known mean and variance for resnet50 which is ([0.485, 0.456, 0.406], [0.229, 0.224, 0.225]). . I used torchvision transform module for these preprocessing processes.

For only the labeled train dataset, I used data augmentation methods explained above.

B. Model Choice

I used pretrained resnet50 for the model. The model can be imported from torchvision.models. The reason why I chose this model is because it has been proven to be one of the best model on image classification task. I tried both resnet18 model and resnet50 model but resnet50 showed a lot better performance. That's why I chose resnet50. Also, I used the "pretrained" model, because since there is not much data that I could use for training, it would be better to just finetune the pretrained model rather than training the whole model from the scratch.

C. Hyper Parameter Choice

Hyper parameters that I used are like the below.

momentum	0.9
weight decay	0.005
learning rate	0.001
train batch	256
test batch	256

For the optimizer, I used SGD optimizer from torchvision.optim. The momentum is 0.9 and weight decay is 0.005. For the learning rate, I used 0.001 and for the train and test batch size, I used 256. I tried bigger batch size, but I had not enough GPU memory.

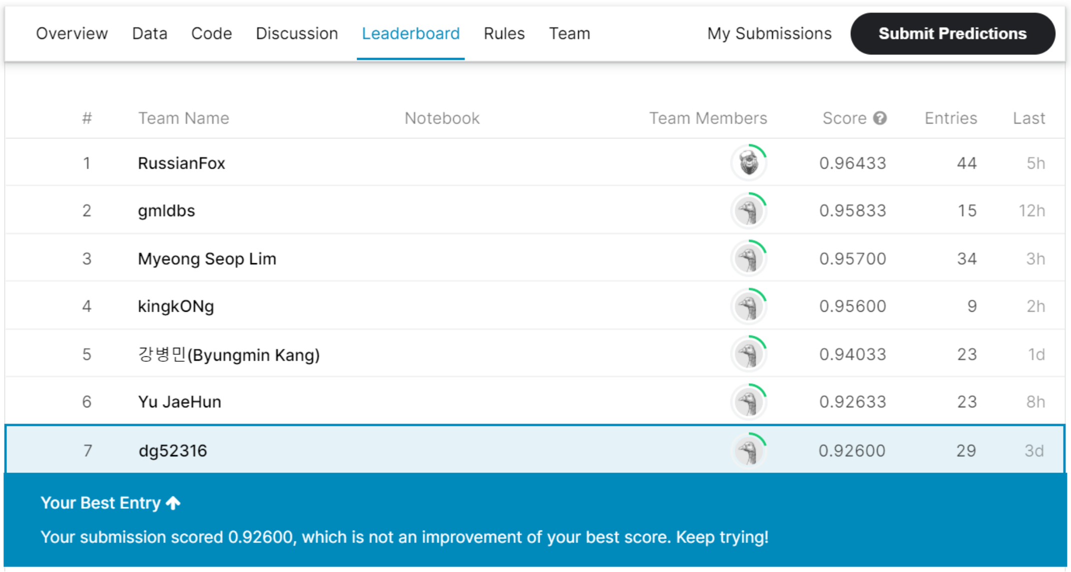
3. Any trial you made for performance

For the first try, I used the default CNN model that was in the model.py. I used only the labeled data to train the model and did not use the unlabeled data to train. Then, I got 0.7 accuracy.

For the second try, I implemented the pseudo-labeling with the default CNN model. I tried a lot of parameter tunings but I could not get over 0.83 accuracy.

For the last try, I upgraded the model to resnet50 and used the pseudo-labeling and data augmentation. Then I got 0.92 accuracy over the test data. Changing the model to pretrained resnet50 boosted the performance.

4. Final results on the public leaderboard (Screenshot of your leaderboard)



#	Team Name	Notebook	Team Members	Score	Entries	Last
1	RussianFox			0.96433	44	5h
2	gmldbs			0.95833	15	12h
3	Myeong Seop Lim			0.95700	34	3h
4	kingkONg			0.95600	9	2h
5	강병민(Byungmin Kang)			0.94033	23	1d
6	Yu JaeHun			0.92633	23	8h
7	dg52316			0.92600	29	3d

Your Best Entry

Your submission scored 0.92600, which is not an improvement of your best score. Keep trying!