Course Work 4 Report

Gwanwoo Choi
Department of Computer Science and Engineering
UNIST
cgw1999@unist.ac.kr

1 Configuration

I progressed this course work in Jupyter Notebook of "Windows10 OS". I utilized pytorch framework and NVIDIA "RTX-3090" GPU. The python version is "3.8.5" and the pytorch version is "1.8.1". I utilized cuda 11.1 version for pytorch.

2 Task 1

In task1, I compared the performance of Random Forest, MLP and CNN for CIFAR-10 and CIFAR-100 datasets. Basically, I downloaded the "CIFAR-10" and "CIFAR-100" datasets from pytorch dataset module. And applied these datasets to all of three different models. "CIFAR-10" and "CIFAR-100" own images which have dimension of "$3 \times 32 \times 32$".

2.1 Random Forest

For Random Forest, I used default hyper parameter settings in scikit learn package. Anyone can find my Random Forest configuration in sklearn.ensemble.RandomForest. The number of trees in the forest is 100. "Gini" criterion is used. Bootstrap method is used and "Out of Bag" score is not used. It takes 35.5 seconds for training "CIFAR-10" and 109.5 seconds for training "CIFAR-100". It takes 0.628 seconds for predicting CIFAR-10 test case and 1.50 seconds for predicting CIFAR-100.

2.2 MLP

For implementing Multi Layer Perceptron, I used pytorch package in python. My MLP has structures that 5 hidden layers and relu activations between them. First hidden layer gets 3072 dimension vector as input and outputs 512 dimension vector. Second hidden layer gets 512 dimension vector as input and outputs 512 dimension vector. Third hidden layer gets 512 dimension vector and 64 dimension vector. Fourth hidden layer gets 64 dimension vector and outputs 64 dimension vector. Fifth hidden layer gets 64 dimension vector and outputs 10 dimension vector for "CIFAR-10" dataset and 100 dimension vector for "CIFAR-100" dataset. It has batch size of 4.

I utilized validation sets to decide the best training epochs.Figure 1 and Figure 2 are the training and validation loss plot for CIFAR-10 and CIFAR-100. So final training epochs are 7 for CIFAR-10 and 9 for CIFAR-100. I used Cross Entropy Loss and stochastic gradient descent algorithm with learning rate 0.001 and momentum 0.9 for both two datasets. It takes 45.5 seconds for training CIFAR-10 and 61.3 seconds for training CIFAR-100. It takes 3.58 seconds for predicting CIFAR-10 test case and 3.59 seconds for predicting CIFAR-100 test case.

2.3 CNN

As above, for implementing Convolutional Neural Network, I used pytorch package in python. My CNN has structures that two convolutional layers, two maxpooling layers and three fully-connected layers. In first convolutional layer, kernel size is (5,5), in_channels is 3, out_channels is 6 and stride

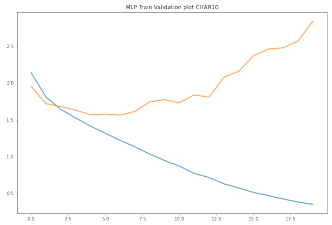

Figure 1: MLP Train-Validation plot CIFAR-10

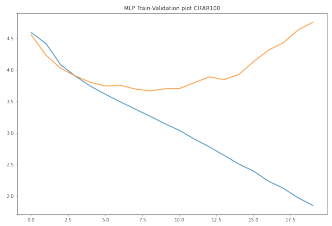

Figure 2: MLP Train-Validation plot CIFAR-100

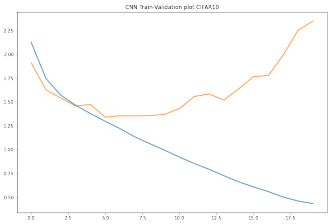

Figure 3: CNN Train-Validation plot CIFAR-10

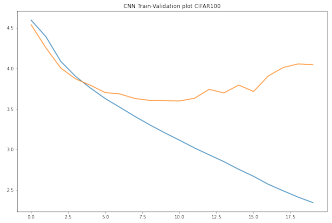

Figure 4: CNN Train-Validation plot CIFAR-100

= 1. Relu activation and Maxpooling layer of kernel size (2,2) and stride = 2 are followed. In second convolutional layer, kernel size is 5 times 5, in_channels is 6, out_channels is 16 and stride = 1. Relu activation and Maxpooling layer of kernel size (2,2) and stride = 2 are followed. In first fully connected layer, input dimension is 400 and output dimension is 120. In second fully connected layer, input dimension is 120 and output dimension is 84. In last fully connected layer, input dimension is 84 and output dimension is 10 for CIFAR-10 and 100 for CIFAR-100. There are relu activation functions between fully connected layers. It has batch size of 4.

I utilized validation sets to decide the best training epochs. Figure 3 and Figure 4 are the training and validation loss plot for CIFAR-10 and CIFAR-100. So final training epochs are 8 for CIFAR-10 and 11 for CIFAR-100. I used Cross Entropy Loss and Stochastic gradient descent algorithm with learning rate 0.001 and momentum 0.9 for both two datasets. It takes 58.7 seconds for training

CIFAR-10 and 80.6 seconds for training CIFAR-100. It takes 4.10 seconds for predicting CIFAR-10 test case and 4.01 seconds for predicting CIFAR-100 test case.

Table 1: Accuracy

Model	CIFAR-100	CIFAR-10	run time(CIFAR-10)	run time(CIFAR-100)
RF	16.2%	42.8%	35.5s	109.5
MLP	14.0%	46.5%	45.5s	61.3s
CNN	16.9%	51.3%	58.7	80.6s

2.4 Comparing

You can see the summarize of performance on each models in Table 1. In random forest, the accuracy for CIFAR-10 is 42.8% and accuracy for CIFAR-100 is 16.2%. In MLP, the accuracy for CIFAR-10 is 46.5% and accuracy for CIFAR-100 is 14.0%. In CNN, the accuracy for CIFAR-10 is 51.3% and accuracy for CIFAR-100 is 16.9%. It seems that overall accuracy of three models are similar to each others. But slightly CNN have better accuracies than other two models because it is specialized model for processing visual information. CNN has important characters, weight sharing and locally connected layers, which is the biggest difference with ones of MLP and makes a significant difference.

3 Task 2

3.1 Hyperparameters

In case of Random Forest, I changed criterion from 'Gini' to 'Entropy'. Basically, It takes more time than it with 'Gini' index. The time for training of Random Forest in CIFAR-10 is increased, which is 73.2 seconds and in CIFAR-100 is increased, which is 244.2 seconds. But the accuracy becomes lower than it with "Gini" index. For CIFAR-10, it has accuracy 42.35 and for CIFAR-100, it has accuracy 15.63, which are lower than Random Forest with "Gini" index. It takes 0.627 seconds for predicting CIFAR-10 test case and 1.22 seconds for predicting CIFAR-100 test case.

In case of MLP, I changed the optimizer function. Changed optimization function is Stochastic Gradient Descent algorithm with learning rate of 0.1 and momentum of 0.5. The running time of training for CIFAR-10 is 44.2 seconds and for CIFAR-100 is 58.6 seconds, which are almost same with times of unchanged hyperparameters model. It takes 44.3 seconds for predicting CIFAR-10 test case and 58.6 seconds for predicting CIFAR-100. And the accuracy of CIFAR-10 is 10.0% and accuracy of CIFAR-100 is 1%, which are no different with choosing anything among different classes. So it seems that no learning has taken place at all. One can easily find that by this changed hyperparameters MLP model didn't study well in figure 5 and figure 6 .

In case of CNN, I changed the optimizer funtion. Changed Optimization function is Stochastic Gradient Descent algorithm with learning rate of 0.1 and momentum of 0.5 The running time of training for CIFAR-10 is 59.4 seconds and for CIFAR-100 is 79.9 seconds. It takes 59.4 seconds for predicting CIFAR-10 test case and 79.9 seconds for predicting CIFAR-100 test case. The accuracy of CIFAR-10 is 21.8% and CIFAR-100 is 15.9%. you can see the training-validation graph in Figure7 and Figure8 and find the fact that in this big learning rate and slight small momentum, learning is not done properly. Interestingly, In this case, the accuracies of CIFAR-10 and CIFAR-100 are not quite different with each other. It has a smaller difference in accuracy compared to previouw models.

From the case of MLP and CNN, we can notice that only small enough learning rate can be helpful to train model.

3.2 Training Size

I compared to training,validation and test set size. In first condition, I used number of 10000 training,validation and test set size. In second condition, I used number of 1000 training,validation and test set size. Both of them are progressed with CNN. First condition is same as the above one (Figure 3,4). And in second condition, there is a dramatic change in accuracy. All of hyperparameter settings

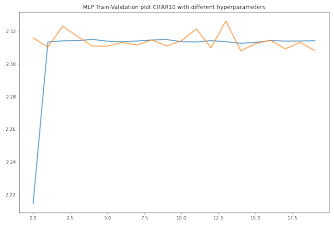

Figure 5: MLP Train-Validation plot CIFAR-10 with different hyperparameters.png

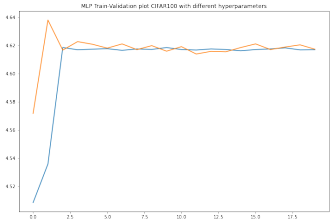

Figure 6: MLP Train-Validation plot CIFAR-100 with different hyperparameters.png

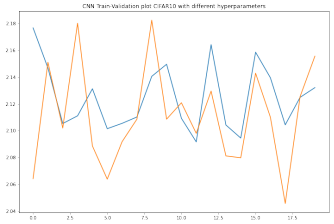

Figure 7: CNN Train-Validation plot CIFAR-10 with different hyperparameters.png

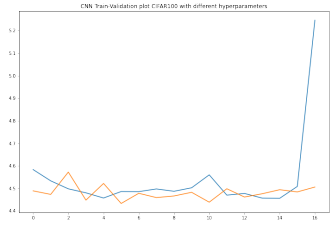

Figure 8: CNN Train-Validation plot CIFAR-100 with different hyperparameters.png

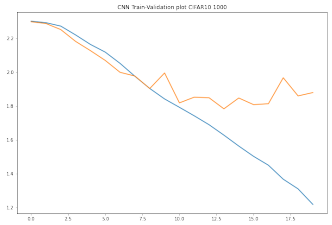

Figure 9: CNN Train-Validation plot CIFAR-10 with 1000 training samples.png

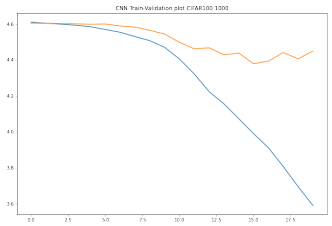

Figure 10: CNN Train-Validation plot CIFAR-100 with 1000 training samples.png

are same with in condition of Figure 3,4. I selected training epochs referring to "Train-Validation plot" (Figure 9,10). In second condition , the accuracy of CIFAR-10 is 29.1 and CIFAR-100 is 3.1, which are significantly lower than first condition. One can see that in Figure 9 and 10, slope of validation loss at the beginning of the graph is less steepest than in Figure 3 and 4. From this, I estimated that overfitting is started at the earliest point of epochs.

3.3 Random Forest with Feature Extractor CNN

I utilize convolutional layers as feature extractor. Originally CNN is composed of convolutional layer and fully connected layer. And in this experiment, I replace this fully connected layer. So finally I have revised CNN, which is composed of convolutional layer and random forest. The structures of convolutional layers and hyperparameters of model are same with in Figure 3 and 4. Only different is replacing fully connected layer as random forest. It has accuracy of 49.1 for CIFAR-10 data set and 19.1 for CIFAR-100. In CIFAR-10, my rivised CNN model has slightly lower accuracy (49.1) than original CNN (51.3). But in CIFAR-100, mine has much higher accuracy (19.1) than original one (16.9). I can't easily say which one is better. But in relatively sparse data set per one class, my rivised CNN workes well than original CNN. It seems because fully connected layer is non-linear deep neural network model and need much more datas per one class. But random forest doesn't really need that much more datas per one class than fully connected layer. So it seems natural that my rivised CNN works well in CIFAR-100 than CIFAR-10 compared to original CNN.

3.4 Fashin MNIST dataset

Fashin MNIST dataset is a classifier datasets that one should be classifier 10 kind of clothes with 60000 size training set and 10000 size test set. One image in Fashine MNIST dataset is gray scale (1 channel) and size of (28,28), so total (1,28,28). Compared to CIFAR-10, Fashion MNIST dataset has much lower complexity because size per one image is small and in Fashin MNIST, there is only one object but in CIFAR-10, there are a lot of objects except target object. This is the basic review of our datasets.

For Fashin MNIST dataset, I revise my CNN (in Figure 3 and 4) slightly because of different input size. In First convolutional layer, the in_channel is changed from 3 to 1 and out_channel is changed from 6 to 4. In second convolutional layer, the in_channel is changed from 6 to 4. In first hidden layer on fully connected layer, input dimension is changed from 400 to 256 and output dimension is changed from 120 to 100. In second hidden layer, input dimension is changed from 120 to 100 and

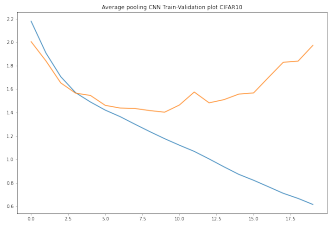

Figure 11: Average pooling CNN Train-Validation plot CIFAR10.png

output dimension is changed from 84 to 50. In third hidden layer, input dimension is changed 84 to 50.

The consumed training time is 83.5 seconds and testing time is 3.58 seconds. The accuracy is dramatically increased compared to CIFAR-10. I achieved accuracy 85.4%, which is really higher than 51.3%. This result notices that it is important to find the appropriate model capacity for the data. From the result in this experiment, we can drive the conclusion, our original CNN has relatively lower capacity for admitting CIFAR-10 dataset.

3.5 Different CNN models

Average Pooling CNN

I have a curious about the performance difference between average pooling CNN and max pooling CNN. I know that Max Pooling is superior than Average Pooling so I want to test it. I just changed my original CNN structure (in Figure 3 and 4) of Max Pooling layer to Average Pooling layer.

There are few differences between max pooling CNN and average pooling CNN. First, the number of training epoch is changed. In max pooling CNN, the training epoch is 8 but in average pooling CNN, the training epoch is 10. It is because optimal point of validation plot is changed as you can see in Figure 11. And the accuracy becomes 50.56, which is lower than original 51.26. So we can conclude that learning is well progressed with max pooling layer.

Batch Normalization CNN

From the insight previous experiment, It seems better to increase the capacity of CNN. In expanded CNN, There are 6 convolutional layers and 4 batch normalization layers and 4 linear layers. There exists relu activation function between layers. In first convolutional layer, input channel=3,output channel=6,kernel=3,stride=1. In second convolutional layer, input channel=6, output channel=12, kernel=3,stride=1. In third convolutional layer, input channel=12, output channel=24,kernel=3,stride=1. In fourth convolutional layer, input channel=24, output channel=48, kernel=3, stride=1. In fifth convolutional layer, input channel=48, output channel=144, kernel=2, stride=1. In sixth convolutional layer, input channel=144, output channel=432, kernel=1,stride=1. Between from first convolutional layer to fifth convolutional layer, there are maxpooling layers, batch normalization layers. Between every convolutional layer, there exists relu activation function.

In first hidden layer on fully connected layers, input dimension is 432 and output dimension is 256. In second hidden layer, input dimension is 256 and output dimension is 120. In third hidden layer, input dimension is 120 and output dimension is 84. In fourth hidden layer, input dimension is 84 and output dimension is 10. Between hidden layers, there exists relu activation function.

I use SGD optimizer and set learning rate as 0.005 and momentum as 0.9. It has a lot time to training because it has very large capacity. It takes 175 seconds for training and 7.41 seconds for testing. Figure 12 show that the train-validation plot of our batch normalization CNN. I choose training epochs as 11 referring to Figure 11. And the accuracy for CIFAR-10 is 46.13, what I didn't expect.

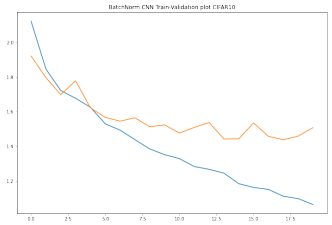

Figure 12: BatchNorm CNN train-Validation plot CIFAR10.png

```
[ Epoch:  1  loss : 2.123488855743408  validation : 1.922220755624771 ]
[ Epoch:  2  loss : 1.847041289949417  validation : 1.7974870515823365 ]
[ Epoch:  3  loss : 1.722896962738037  validation : 1.6994842162489892 ]
[ Epoch:  4  loss : 1.6780086580514908  validation : 1.7792864717721941 ]
[ Epoch:  5  loss : 1.6263284903287887  validation : 1.6227938150035278 ]
[ Epoch:  6  loss : 1.530385835814476  validation : 1.567649412882328 ]
[ Epoch:  7  loss : 1.4940176797151565  validation : 1.5458382598876954 ]
[ Epoch:  8  loss : 1.4390786898255348  validation : 1.5651298232913018 ]
[ Epoch:  9  loss : 1.3852745001867413  validation : 1.513569126188755 ]
[ Epoch:  10  loss : 1.3523011207195162  validation : 1.5244578917413395 ]
[ Epoch:  11  loss : 1.3294162674635648  validation : 1.476987187063694 ]
[ Epoch:  12  loss : 1.2836331535562873  validation : 1.5096081928908824 ]
[ Epoch:  13  loss : 1.2675001603856684  validation : 1.5386201639771462 ]
[ Epoch:  14  loss : 1.2453327784895898  validation : 1.4428167230963707 ]
[ Epoch:  15  loss : 1.1845606321901083  validation : 1.443777369967103 ]
[ Epoch:  16  loss : 1.1628896341532469  validation : 1.5358038329556585 ]
[ Epoch:  17  loss : 1.1513855375275015  validation : 1.4580205759882927 ]
[ Epoch:  18  loss : 1.1122918186552821  validation : 1.4382364712685347 ]
[ Epoch:  19  loss : 1.0982387056551874  validation : 1.4590478633582593 ]
[ Epoch:  20  loss : 1.064433833194739  validation : 1.5083202617056668 ]
Time : 464.83689880371094
```

Figure 13: BatchNormalization loss.png

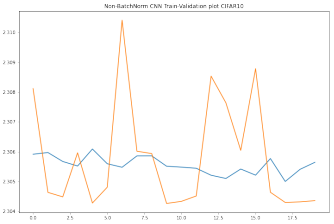

Figure 14: Non-BatchNorm CNN Train-Validation plot CIFAR10.png

```
[ Epoch:  1  loss : 2.3059146825790404  validation : 2.308116782951355 ]
[ Epoch:  2  loss : 2.305971097660065  validation : 2.3046347409248353 ]
[ Epoch:  3  loss : 2.305671356163024  validation : 2.3044833335876467 ]
[ Epoch:  4  loss : 2.305518301868439  validation : 2.3059616770744324 ]
[ Epoch:  5  loss : 2.306089134216309  validation : 2.304276773262024 ]
[ Epoch:  6  loss : 2.3055954542160033  validation : 2.3048077439308168 ]
[ Epoch:  7  loss : 2.3054769198417664  validation : 2.3104114677429197 ]
[ Epoch:  8  loss : 2.3058586487770008  validation : 2.306015175628662 ]
[ Epoch:  9  loss : 2.3058597346305847  validation : 2.3059360852241517 ]
[ Epoch:  10  loss : 2.305512041568756  validation : 2.3042623230934143 ]
[ Epoch:  11  loss : 2.3054816542625427  validation : 2.3043332612991333 ]
[ Epoch:  12  loss : 2.3054438499450685  validation : 2.3045131094932556 ]
[ Epoch:  13  loss : 2.3052094740867615  validation : 2.3085350371360778 ]
[ Epoch:  14  loss : 2.3051008979797363  validation : 2.3076388228416445 ]
[ Epoch:  15  loss : 2.3054178698539736  validation : 2.3060439331054687 ]
[ Epoch:  16  loss : 2.305212100982666  validation : 2.3087861057281494 ]
[ Epoch:  17  loss : 2.305769320011139  validation : 2.3046378457069396 ]
[ Epoch:  18  loss : 2.3050008732795715  validation : 2.3042957469940184 ]
[ Epoch:  19  loss : 2.3054078779220581  validation : 2.3043159049034116 ]
[ Epoch:  20  loss : 2.3056436090469406  validation : 2.3043569643020063 ]
Time : 388.19938921928406
```

Figure 15: Non-BatchNormalization loss.png

3.6 Ablation Study

In actually, first I experiment the Batch Normalization CNN without Batch Normalization. I first tried just bigger capacity CNN. But I found that such a huge model, learning is not efficient. In this case, I thought that adapting batch normalization to my huge CNN. Before adapting batch normalization, model shows a little change in terms of loss. But after applying batch normalization, the training speed was increased astonishingly.

I applied same condition except batch normalization layer. The optimization function, learning rate and momentum values are same with before one. One can see that the difference between Figure 12,13,14 and 15 and notice that without batch normalization, such a huge capacity CNN suffers from learning. And losses of training and validation set are not changed and just oscillated. From this, we can conclude that batch normalization is essential for learning our model.