

MNIST, CIFAR-10 Training with Several CNN Architectures

Park Jae Hyun

Postech EE

ltbljb1677@postech.ac.kr

1. Introduction

This project is to train the neural network which performs the classification task of MNIST and CIFAR-10 Dataset. For MNIST, 3 architectures are exploited ; LeNet, AlexNet, and ImageNet pretrained Resnet-18. For fair comparison, train, val, and test data sizes are fixed with 50000, 10000, 10000 each and learning curves for training and validation have been made with this data.

2. Implementations

In chapter2, the Implementation detail of each architecture is explained, the basic process to implement a simple neural network, convolutional neural network architecture and training algorithm. Also, the hyperparameter settings and scheduling of training process to maximize the classification performance are discussed. Specific learning results such as learning curves and precision tables are showed in chapter 3.

The implementation is based on pytorch framework and source code is submitted with zip file. One additional thing to note here is that latest version of scikit-learn (0.23.2) does not support the source code and scikit-learn 0.22.1 version must be satisfied (pip install scikit-learn ==0.22.1 command is enough).

2.1. MNIST – LeNet

LeNet architecture is featured with 2 repetition of convolution + pooling layer followed by 2 fully connected layer and softmax for classification scores. The whole training process starts with loading and preprocessing the dataset. Preprocessing step includes data normalization, reshaping and splitting it with train, val, and test data. Data normalization is recommended to improve training speed and reshaping the data to 1 x 28 x 28 is essential to fit in to architecture input size. The whole dataset is loaded on gpu if it's available and the process should be executed before 'DataLoader' call for time efficiency.

After data setting phase, the optimizer setting follows. The optimal starting learning rate for LeNet architecture is manually selected with 0.001. The learning rate is handled by learning rate scheduler and the learning rate decreases with the ratio of 0.85 with every epoch. The learning process exploits Adam optimizer which is selected as an optimal. For minibatch training, batch size for training is chose as 100 and the iteration number is 500 when we assume the size of the training data as 50,000. For every 100 iteration, validation test procedure is executed and the number 100 is arbitrarily selected to collect enough learning curve's points. Loss for the classification task is selected as cross entropy loss (which includes exponentiation of every class scores in pytorch framework).

Different hyperparameters have been tried such as learning rate, class score outputs, decay ratio of learning rate, epochs, and so on. On learning rate's aspect, fine adjustments of learning rate and decay ratio had only a slight effect on total performance in LeNet's case, which seem to be related with training speed only. One thing to note is that learning rate should be in range 0.01 ~ 0.001 and decay ratio in 0.7 to 0.9 to get fine learning effect near minimum with 10 epochs. And epoch number of 10~15 results in reasonable training speed considering shallow architecture of LeNet. For output layer choices, the trials of additional softmax and log softmax layer had been made.

Given that CrossEntropyLoss function of torch.nn already computes exp score of the last layer, additional softmax layer in model's forward definition results in double calculation (this method is frequently seen in many open source implementation). This does not severely change the classification result, but may distort relative levels of the outputs, making gradients less strong and potentially slowing training a bit.

The precision score was 0.9901 and accuracy score was 0.9901 for the test dataset.

2.2. MNIST – AlexNet

AlexNet is featured with convolution, activate, local response normalization, pooling layer and drop out

(pattern of architecture is similar to LeNet but much bigger). Whole process is the same with LeNet's case except for model definition. When training the AlexNet with similarly optimized learning rates, performance scores were 3~4% lower than LeNet's case (given that LeNet's performance score was 98.5 to 99%). Adadelata optimizer has been tried which gave slightly better results than Adam without LR decay, the same or poor results than Adam with LR decay. Manually searched optimal hyperparameters were learning rate with 0.001, decay ratio with 0.9 with 12 epochs, results in precision score 0.9600 and accuracy score 0.9599 for test dataset.

2.3 MNIST – ImageNet Pretrained ResNet18

Processing steps of MNIST data to fit in to ResNet architecture includes resize with bilinear or bicubic interpolation (nearest neighbor interpolation might cause image with unnatural features). It requires, however, huge memory resources to handle the dataset in local memory, preprocessed data should be loaded. If the torchvision's DataLoader method is utilized then every single data in batches should be loaded to gpu one by one, which is time consuming way. On the other hand, manually customizing the data without exploiting torchvision's transform function also takes additional time, so the former way is adopted in this implementation.

Also, in this case, selection between Adadelata and Adam optimizer with LR decay has been made. To manually find optimal hyperparameters of Adam optimizer, several trials are needed while single trial is much more expensive because of the larger model size. The performance training with similar hyperparameters as LeNet and AlexNet's cases, Epoch number of 5 is reasonable consider the model size and learning curves.

2.4 CIFAR10 – AlexNet

The whole training procedure is almost same except for data loading process and Architecture's input size correction. AlexNet's input channel should be three, for CIFAR10 has RGB channel.

Similar hyperparameter settings and optimizer usage results in severely poor performance. With Adadelata optimizer with 50 epochs,

2.5. CIFAR10 – ResNets

The whole training procedure for ResNet-20, ResNet-32, ResNet-44, ResNet-56, ResNet-110 takes huge amount of time and the most stable and quickest way to select hyperparameter is to use Adadelata optimizer without any trial of SGD or Adam. In chapter3, the pure

comparison between each model, trained from the same condition of optimizer and epoch number 200, is provided. Even if overfitting of training data degenerates the generalization performance, it is easy to know when to stop simply by looking into the learning curve of validation set (here is why the epoch number is set large enough for this huge training procedure).

2.6. CIFAR10 – ImageNet Pretrained ResNets

When training with pretrained ResNet, SGD optimizer is used with learning rate 0.001 and momentum 0.9. Learning rate scheduler also helps decaying learning rate with lr_scheduler.StepLR with step size 7 and gamma number 0.1. This scheduler decays learning rate to certain point and keep constant just as step function. The total epoch number is set to 30.

Although It is hard to compare fairly with previous experiments with different size of ResNet architecture, well-known hyperparameters and training methods for these tasks are given (here's why we don't use Adadelata optimizer) and we utilized them without any supplementary optimization procedure.

3.1. MNIST Results – LeNet

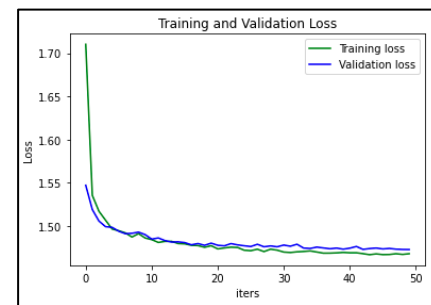


Figure 3.1.1 : Learning Curve (MNIST_LeNet)

Average Precision	0	1	2	3	4	5	6	7	8	9	total
Training	1.0	1.0	1.0	1.0	1.0	0.9998	1.0	0.998	1.0	1.0	0.9997
Test	0.992	0.993	0.982	0.997	0.993	0.985	0.996	0.995	0.983	0.986	0.990

Figure 3.1.2 : Precision Table (MNIST_LeNet)

The figure above represents learning curve and Precision table of manually searched hyperparameters (epoch 15, learning rate 0.001, decay ratio 0.85/epoch, Adam optimizer, CrossEntropyLoss). The Training data's precision is almost 1 because of slight overfitting while keeping generalization performance.

3.2. MNIST Results – AlexNet

When training with learning rate 0.001 without LR decay, the precision is 0.9450 and accuracy is 0.9453 as shown in Figure 3.2.2.

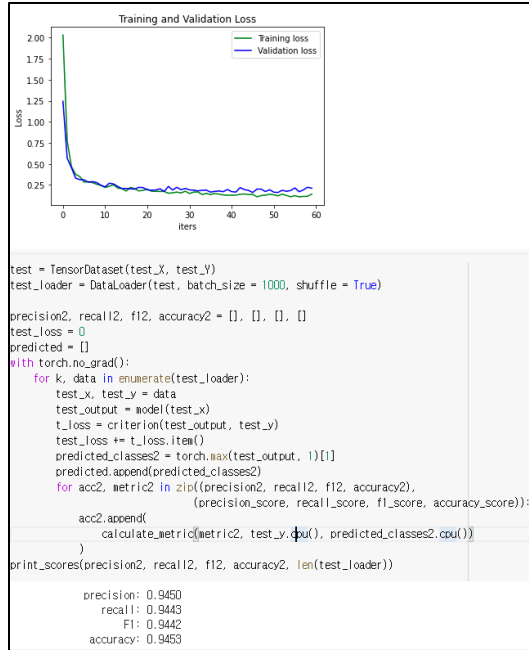


Figure 3.2.1 : Learning Curve (MNIST_ AlexNet) without LR Decay

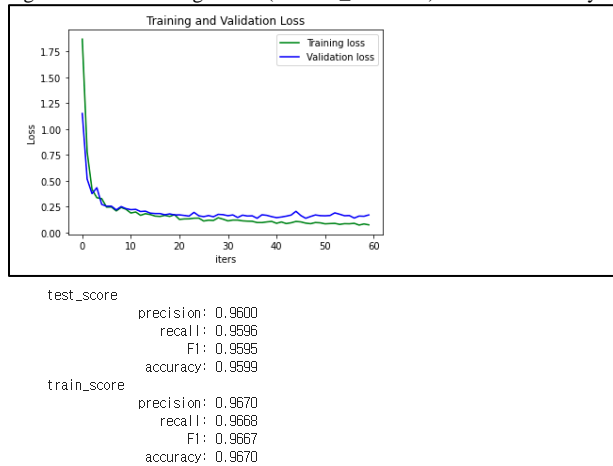


Figure 3.2.2 : Learning Curve (MNIST_ AlexNet) with LR Decay

With learning rate decay, the precision score was 0.9600 and accuracy was 0.9599 for test dataset as shown in Figure 3.2.2.

Average Precision (%)	0	1	2	3	4	5	6	7	8	9	total
Training	0.974	0.979	0.969	0.973	0.962	0.978	0.958	0.970	0.964	0.944	0.967
Test	0.967	0.984	0.960	0.969	0.952	0.973	0.947	0.964	0.955	0.931	0.960

Figure 3.2.2 : Precision Table (MNIST_ AlexNet)

Adaptive Learning Rate Method (Adadelta) optimizer

was tried which is known to show promising results on the MNIST digit classification task. The results, however, is the same or less competitive than Adam optimizers with LR decay while more competitive than Adam without LR decay as mentioned in chapter2.

3.3. MNIST Results – ResNet18

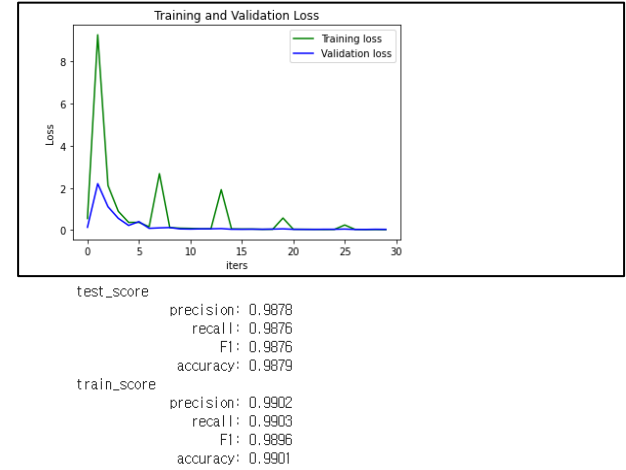


Figure 3.3.1 : Learning Curve (MNIST_ ResNet18) Adadelta

Average Precision (%)	0	1	2	3	4	5	6	7	8	9	total
Training	0.999	0.997	0.977	0.991	0.989	0.971	0.998	0.996	0.989	0.992	0.990
Test	0.994	0.995	0.976	0.985	0.989	0.964	0.997	0.995	0.992	0.991	0.988

Figure 3.3.2 : Precision Table (MNIST_ ResNet18)

The results between two optimization technique looks similar (Figure3.3.1, Figure3.3.2)

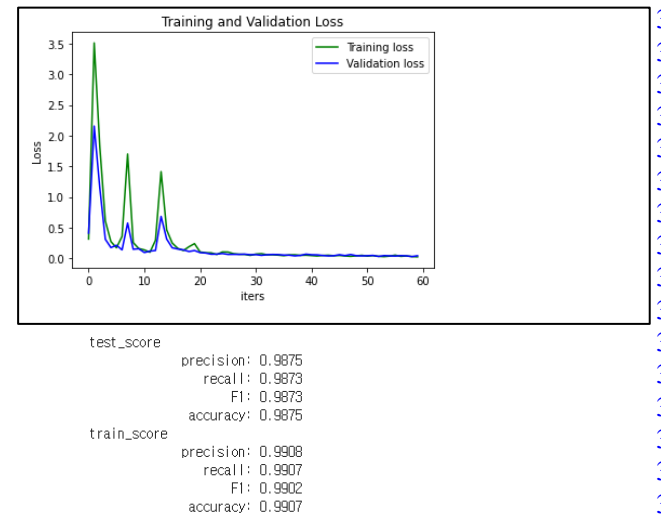


Figure 3.3.3 : Learning Curve (MNIST_ ResNet18) Adam + LR decay

3.4. CIFAR10 Results – AlexNet

```
test_score
precision: 0.7761
recall: 0.7356
F1: 0.7455
accuracy: 0.7360

train_score
precision: 0.8502
recall: 0.8233
F1: 0.8224
accuracy: 0.8244

test precision score : [0.80789755 0.94059406 0.76744186 0.39276211 0.70377185 0.78699862
0.82972136 0.84492588 0.85147508 0.82826087]
train precision score : [0.88057155 0.98059387 0.87597449 0.51013269 0.80104517 0.85945529
0.89279075 0.89572758 0.91040521 0.89855679]
```

Figure 3.4.1 : (CIFAR10 Results) Adadelta / 50 epochs

In this case the learning did not converge at the beginning and the performance metrics in every epoch suggest the training is ongoing even if the epoch number is near 50. So epoch number was set to much higher number of 100, 150, 200, but the score only stays at 0.76 to 0.79.

With Adam optimizer, the learning didn't proceed after 40 ~ 50 epochs

```
Epoch 200/200, training loss: 0.5001469248533249, validation loss: 1.7206747782230378
precision: 0.7783
recall: 0.7434
F1: 0.7423
accuracy: 0.7442
Training time: 5587.988766670227s
test_score
precision: 0.7796
recall: 0.7362
F1: 0.7454
accuracy: 0.7359

train_score
precision: 0.8895
recall: 0.8726
F1: 0.8700
accuracy: 0.8731
```

Figure 3.4.2 : (CIFAR10 Results) Adadelta / 200 epochs

Adadelta and Adam optimizer provide similar performance only to increase training score which means overfitting is going on.

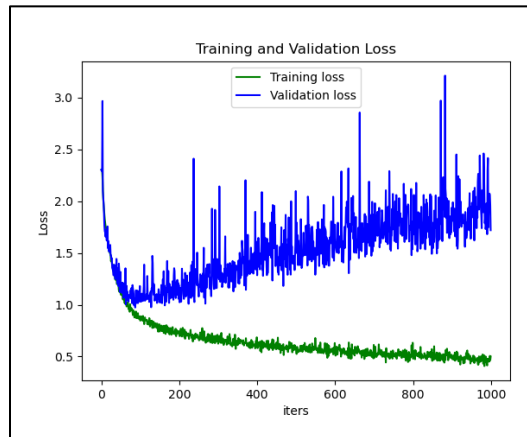


Figure 3.4.3 : Learning Curve (CIFAR-10_ AlexNet)

Average Precision (%)	0	1	2	3	4	5	6	7	8	9	total
Training	0.952	0.979	0.963	0.909	0.935	0.963	0.966	0.974	0.978	0.980	0.9599
Test	0.755	0.875	0.706	0.432	0.668	0.658	0.823	0.853	0.898	0.832	0.750

Figure 3.4.4 : Precision Table (CIFAR-10_ AlexNet)

3.5. CIFAR10 Results – ResNet-20 / 32 / 44 / 56 / 110

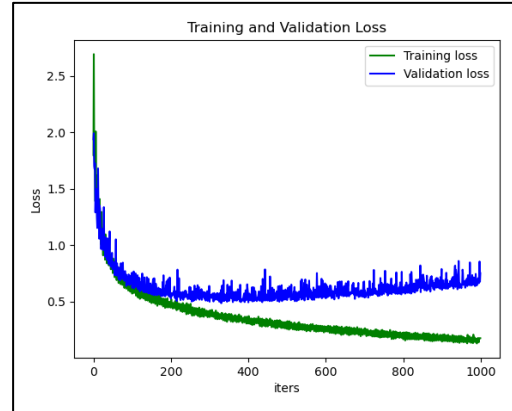


Figure 3.5.1 : Learning Curve (CIFAR-10_ ResNet-20)

```
Training time: 4306.760057210922s
test_score
precision: 0.8411
recall: 0.8272
F1: 0.8286
accuracy: 0.8272

train_score
precision: 0.9430
recall: 0.9354
F1: 0.9326
accuracy: 0.9350

test precision score : [0.82024598 0.92182741 0.81899441 0.59130435 0.84789644 0.85534591
0.87863591 0.91639522 0.97466828 0.78583196]
train precision score : [0.93657874 0.98417529 0.96762666 0.7485767 0.97170006 0.97191963
0.97820383 0.9892992 0.99976937 0.88596803]
```

Average Precision (%)	Air plane	Automobile	Bird	Cat	Deer	Dog	Frog	Horse	Ship	Truck	total
Training	0.937	0.984	0.968	0.749	0.972	0.972	0.978	0.989	0.9997	0.886	0.9430
Test	0.820	0.322	0.819	0.591	0.878	0.855	0.879	0.916	0.975	0.786	0.8411

Figure 3.5.2 : Precision Table (CIFAR-10_ ResNet-20)

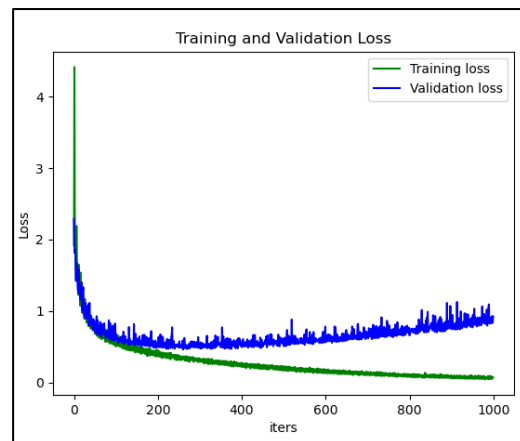


Figure 3.5.3 : Learning Curve (CIFAR-10_ ResNet-32)

```

Training time: 5420.947761297226s
test_score
    precision: 0.8457
    recall: 0.8428
    F1: 0.8409
    accuracy: 0.8429
train_score
    precision: 0.9759
    recall: 0.9747
    F1: 0.9732
    accuracy: 0.9751
test precision score : [0.93532338 0.90143541 0.7840796 0.79445145 0.73355263 0.78690127
0.86177312 0.87524558 0.89688716 0.8861629 ]
train precision score : [0.99977558 0.98831683 0.96774828 0.98677937 0.92839874 0.95989043
0.97594838 0.98477659 0.98553883 0.98093179]

```

Average Precision (%)	Air plane	Automobile	Bird	Cat	Deer	Dog	Frog	Horse	Ship	Truck	total
Training	0.9998	0.9884	0.9678	0.987	0.9283	0.960	0.976	0.985	0.986	0.981	0.9759
Test	0.9353	0.9014	0.784	0.794	0.734	0.787	0.862	0.875	0.897	0.886	0.8457

Figure 3.5.4 : Precision Table (CIFAR-10_ ResNet-32)

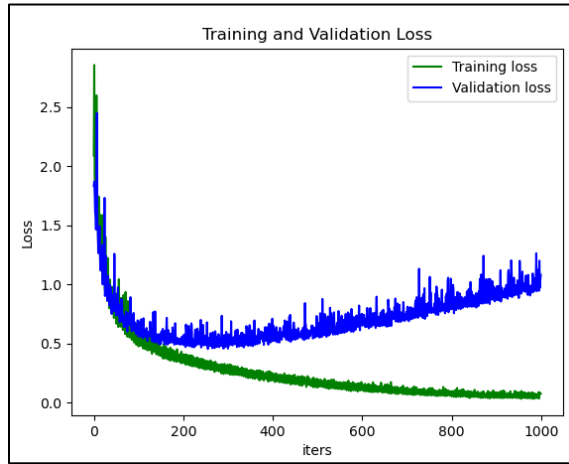


Figure 3.5.3 : Learning Curve (CIFAR-10_ ResNet-44)

```

Training time: 6655.148144006729s
test_score
    precision: 0.8495
    recall: 0.8408
    F1: 0.8408
    accuracy: 0.8413
train_score
    precision: 0.9751
    recall: 0.9733
    F1: 0.9719
    accuracy: 0.9729
test precision score : [0.71460507 0.92038835 0.8729352 0.7047619 0.77905945 0.8338945
0.86745796 0.93318486 0.94885745 0.92169312]
train precision score : [0.87077673 0.99144789 0.9978275 0.95565149 0.95337682 0.99349559
0.99199199 0.99750779 0.99979317 0.99816626]

```

Average Precision (%)	Air plane	Automobile	Bird	Cat	Deer	Dog	Frog	Horse	Ship	Truck	total
Training	0.9998	0.9883	0.9677	0.9868	0.928	0.960	0.976	0.985	0.986	0.981	0.976
Test	0.935	0.9014	0.784	0.794	0.734	0.787	0.862	0.875	0.897	0.886	0.8457

Figure 3.5.4: Precision Table (CIFAR-10_ ResNet-44)

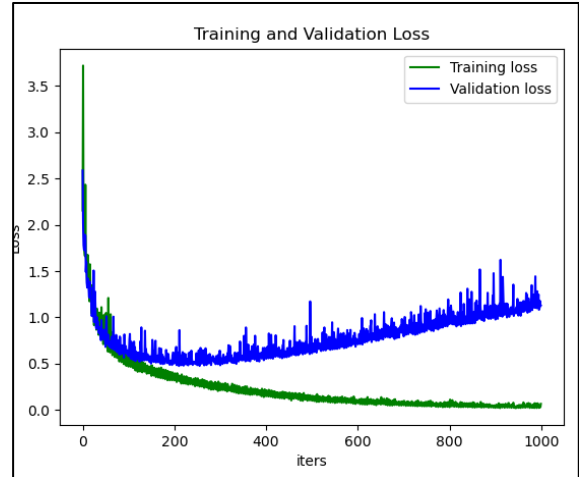


Figure 3.5.5 : Learning Curve (CIFAR-10_ ResNet-56)

```

Training time: 8111.23123550415s
test_score
    precision: 0.8464
    recall: 0.8452
    F1: 0.8445
    accuracy: 0.8448
train_score
    precision: 0.9910
    recall: 0.9911
    F1: 0.9905
    accuracy: 0.9911
test precision score : [0.83558994 0.93462717 0.75607477 0.72895277 0.825 0.80610022
0.8651472 0.91909385 0.8572744 0.9336235 ]
train precision score : [0.98691774 0.99899315 0.98267034 0.99039039 0.98968459 0.99797448
0.99283439 0.99818255 0.97465393 0.99979309]

```

Average Precision (%)	Air plane	Automobile	Bird	Cat	Deer	Dog	Frog	Horse	Ship	Truck	total
Training	0.945	0.969	0.885	0.841	0.715	0.689	0.935	0.890	0.898	0.942	0.8711
Test	0.896	0.322	0.828	0.791	0.669	0.656	0.897	0.859	0.863	0.906	0.8287

Figure 3.5.6 : Precision Table (CIFAR-10_ ResNet-56)

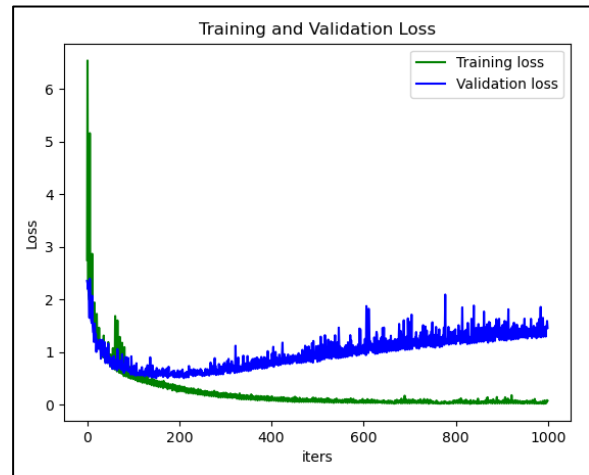


Figure 3.5.7 : Learning Curve (CIFAR-10_ ResNet-110)


```

Training time: 14875.885438203812s
test_score
  precision: 0.8354
  recall: 0.8324
  F1: 0.8322
  accuracy: 0.8324
train_score
  precision: 0.9921
  recall: 0.9917
  F1: 0.9914
  accuracy: 0.9917
test precision score : [0.85671039 0.90980392 0.82045455 0.67772512 0.81218781 0.72381835
0.80828083 0.92962543 0.92213115 0.89363817]
train precision score : [0.99658703 0.99581507 0.99714344 0.97853908 0.99082569 0.98103516
0.98112094 1. 0.99879518 0.99819712]

```

Average	Air plane	Automobile	Bird	Cat	Deer	Dog	Frog	Horse	Ship	Truck	total
Precision (%)											
Training	0.997	0.996	0.997	0.979	0.991	0.981	0.981	1.0	0.999	0.998	0.9921
Test	0.857	0.9098	0.8204	0.978	0.812	0.784	0.808	0.930	0.922	0.894	0.8354

Figure 3.5.8 : Precision Table (CIFAR-10_ ResNet-110)

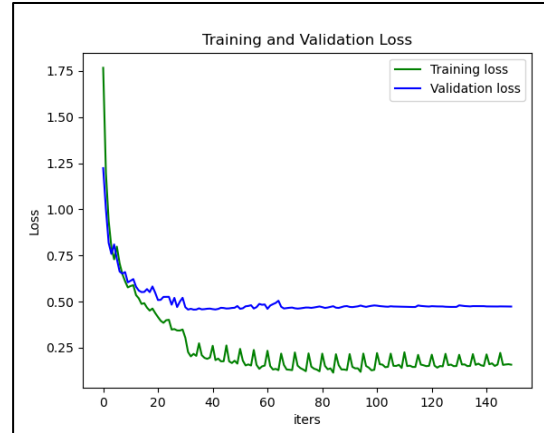


Figure 3.6.3 : Learning Curve (CIFAR-10_ pretrained ResNet-50)

3.6. Pretrained CIFAR10 Results – ResNet – 18 / 50 / 101

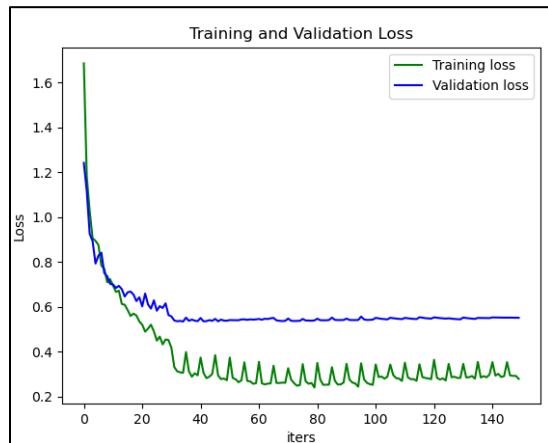


Figure 3.6.1 : Learning Curve (CIFAR-10_ pretrained ResNet-18)

```

Training time: 682.3528068065643s
test_score
  precision: 0.8207
  recall: 0.8210
  F1: 0.8198
  accuracy: 0.8216
train_score
  precision: 0.9056
  recall: 0.9048
  F1: 0.9001
  accuracy: 0.9051
test precision score : [0.83206831 0.894 0.79913138 0.67243133 0.76787372 0.78131868
0.85631068 0.8507014 0.88378906 0.87138584]
train precision score : [0.92699861 0.96391548 0.89981332 0.81446414 0.85040104 0.86279119
0.92820513 0.9104121 0.94437939 0.94983949]

```

Average	Air plane	Automobile	Bird	Cat	Deer	Dog	Frog	Horse	Ship	Truck	total
Precision (%)											
Training	0.927	0.964	0.8998	0.814	0.850	0.863	0.928	0.910	0.944	0.950	0.9056
Test	0.832	0.894	0.799	0.672	0.768	0.781	0.856	0.850	0.884	0.871	0.821

Figure 3.6.2 : Precision Table (CIFAR-10_ pretrained ResNet-18)

```

Training time: 1104.2644381523132s
test_score
  precision: 0.8498
  recall: 0.8500
  F1: 0.8489
  accuracy: 0.8498
train_score
  precision: 0.9504
  recall: 0.9505
  F1: 0.9476
  accuracy: 0.9503
test precision score : [0.88729508 0.91743119 0.85623003 0.70829171 0.79278446 0.8
0.87584869 0.87525562 0.89864865 0.88717454]
train precision score : [0.97335486 0.9814329 0.96793003 0.88249073 0.92644483 0.9300655
0.96440577 0.95100751 0.97077986 0.95714286]

```

Average	Air plane	Automobile	Bird	Cat	Deer	Dog	Frog	Horse	Ship	Truck	total
Precision (%)											
Training	0.973	0.981	0.948	0.882	0.926	0.930	0.964	0.951	0.971	0.957	0.950
Test	0.887	0.917	0.856	0.708	0.792	0.8	0.876	0.875	0.899	0.887	0.8498

Figure 3.6.4 : Precision Table (CIFAR-10_ pretrained ResNet-50)

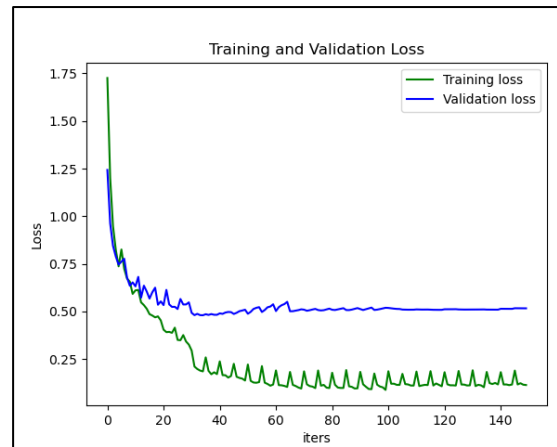


Figure 3.6.5 : Learning Curve (CIFAR-10_ pretrained ResNet-101)

```

Training time: 1541.7182404994965s
test_score
  precision: 0.8421
  recall: 0.8429
  F1: 0.8417
  accuracy: 0.8429
train_score
  precision: 0.9671
  recall: 0.9673
  F1: 0.9653
  accuracy: 0.9670
test precision score : [0.84324835 0.91308793 0.82898853 0.70654397 0.80934223 0.7762605
0.87645914 0.88603696 0.88996139 0.89463019]
train precision score : [0.9702617 0.98909091 0.97635205 0.92287968 0.96058133 0.94835299
0.9717726 0.98299251 0.96782999 0.98119248]

```

Average Precision (%)	Air plane	Automobile	Bird	Cat	Deer	Dog	Frog	Horse	Ship	Truck	total
Training	0.970	0.990	0.976	0.923	0.960	0.948	0.912	0.983	0.968	0.981	0.9671
Test	0.8432	0.913	0.829	0.707	0.8093	0.7763	0.876	0.886	0.890	0.895	0.8421

Figure 3.6.6 : Learning Curve (CIFAR-10_ pretrained ResNet-101)

4. Optimization History

While chapter 3 shows optimized result of each classifiers, this chapter explains the specific optimization procedure to improve performance. The first part is about interpreting learning curves and apply it to hyperparameter decision and second part explains learning rate and decay ratio optimization, and second part explains optimization procedure of learning rate and decay ratio.

4.1. Interpreting Learning Curves

The whole implementations don't include the checkpoint saving of model's state with best metric score during the training. Therefore, the general stop point can be derived by interpreting the learning curves which contain training and validation loss history.

4.2. Learning Rate and Decay Ratio Optimization

The decision rule in here is simple. If the training epoch is chosen, learning rate and decay ratio is selected arbitrarily at the beginning. The first case is when learning speed is too slow and it seems not to be on converging region (Figure 4.2.1). Within this case the learning rate or decay ratio should be higher than before. For example, figure 4.2.1 shows when learning rate and decay ratio of AlexNet with MNIST is set with 0.0001 and 0.8 each. The learning speed in Figure 4.2.1 is too slow considering the fact that the expected accuracy of AlexNet is similar with LeNet's. The process should be interrupted to correct hyperparameters (the accuracy in this case might stuck in range of 0.90 to 0.91). When learning rate and decay ratio is set with 0.002 and 0.8, training performance is deteriorated at the beginning but the training speed is faster than Figure 4.2.1 case. The accuracy was 0.945 in this hyperparameter setting.

Current Learning rate is : 8.5e-05
Epoch 1/15, training loss : 0.49013912975788115, validation loss : 0.4667865341901779
precision: 0.7481
recall: 0.7423
F1: 0.7262
accuracy: 0.7443

100%  500/500 [00:29<00:00, 16.78it/s]

Current Learning rate is : 7.225e-05
Epoch 2/15, training loss : 0.3498448771238327, validation loss : 0.3491745020449162
precision: 0.8620
recall: 0.8591
F1: 0.8621
accuracy: 0.8596

100%  500/500 [01:38<00:00, 5.05it/s]

Current Learning rate is : 6.14125e-05
Epoch 3/15, training loss : 0.2883421914279461, validation loss : 0.2942144887149334
precision: 0.8869
recall: 0.8851
F1: 0.8797
accuracy: 0.8853

100%  500/500 [00:30<00:00, 16.17it/s]

Current Learning rate is : 5.220062499999999e-05
Epoch 4/15, training loss : 0.24249373570084573, validation loss : 0.2445982374250888
precision: 0.9051
recall: 0.9046
F1: 0.8994
accuracy: 0.9045

100%  500/500 [00:37<00:00, 13.45it/s]

Current Learning rate is : 4.4370531249999995e-05
Epoch 5/15, training loss : 0.22876361913979054, validation loss : 0.229600370323658
precision: 0.9092
recall: 0.9086
F1: 0.9040
accuracy: 0.9092

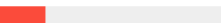
21%  107/500 [00:06<00:48, 8.05it/s]

Figure 4.2.1 : Case1 while Training AlexNet with MNIST

Current Learning rate is : 0.0016
Epoch 1/15, training loss : 0.5102547764778137, validation loss : 0.42437326416373256
precision: 0.6825
recall: 0.6775
F1: 0.6638
accuracy: 0.6817

100%  500/500 [03:55<00:00, 2.12it/s]

Current Learning rate is : 0.0012800000000000003
Epoch 2/15, training loss : 0.3455515293776989, validation loss : 0.3165196818113327
precision: 0.8824
recall: 0.8802
F1: 0.8747
accuracy: 0.8828

100%  500/500 [00:29<00:00, 16.89it/s]

Current Learning rate is : 0.0010240000000000002
Epoch 3/15, training loss : 0.2802585558593273, validation loss : 0.283626816123724
precision: 0.9052
recall: 0.9042
F1: 0.8994
accuracy: 0.9054

100%  500/500 [02:57<00:00, 2.82it/s]

Current Learning rate is : 0.0008192000000000002
Epoch 4/15, training loss : 0.22993629708886146, validation loss : 0.2391429553925991
precision: 0.9194
recall: 0.9186
F1: 0.9144
accuracy: 0.9196

100%  500/500 [00:29<00:00, 16.94it/s]

Figure 4.2.2 : while Training AlexNet with MNIST

While continuously increasing the learning rate and maintaining the decay ratio similar with 0.8, we met the second case and increasing the learning rate deteriorate the classifier performance from this point (Figure 4.2.2 shows the signs of deterioration at first epoch). Within the ranges between the two cases mentioned here, the hyperparameters should be chosen delicately based on trainers' inspections.

5.1. What we've learned through this project

MNIST dataset is the most basic dataset for testing simple deep learning framework. Contrary to the initial expectation, processing the dataset in delicate way and handling hyperparameters were quite tough.

Also, we could learn how to load classic networks such as AlexNet, ResNet, VGGNet and so on. Fine tuning with pretrained model was possible with slight modification of the input and output layers.

5.2. Discussion

There are several discussion topics in this pa. The first issue was the poor performance for MNIST especially for AlexNet. It is known that MNIST classification task is done efficiently with CNN Architecture and most precedent implementations have shown 99% to 100% for all the performance metric score. In this pa, however, AlexNet's classification ability was seriously low, even if we change the fixed format of architecture such as dropout rate, number of fully connected nodes or additional fully connected layers. A few possible ways to improve classification performance might be using augmented data, better data normalization, weight initialization etc.

Next, there are a few striking losses during the training of ResNet18 for MNIST dataset. At first it seems implementation mistakes relevant with loss accumulation while iterating batch training. However, same implementation frame worked well in the other models.

5.3. To be studied

The hyperparameter trials were fixed to Adadelta or Adam's optimizer while tuning the optimal epoch, learning rate and so on. The optimization of different factors must be tried additionally in this programming assignment.

Also, overall source code should be trimmed enough to split the collection subject model for the experiment and main execution code. Most importantly, remembering the best prediction point and save checkpoint parameters should be additionally implemented to get best choice between large epoch numbers.