EAI Lab2 Report

RE6121011 數據所 徐仁瓏

Task 1 – create your own CNN model

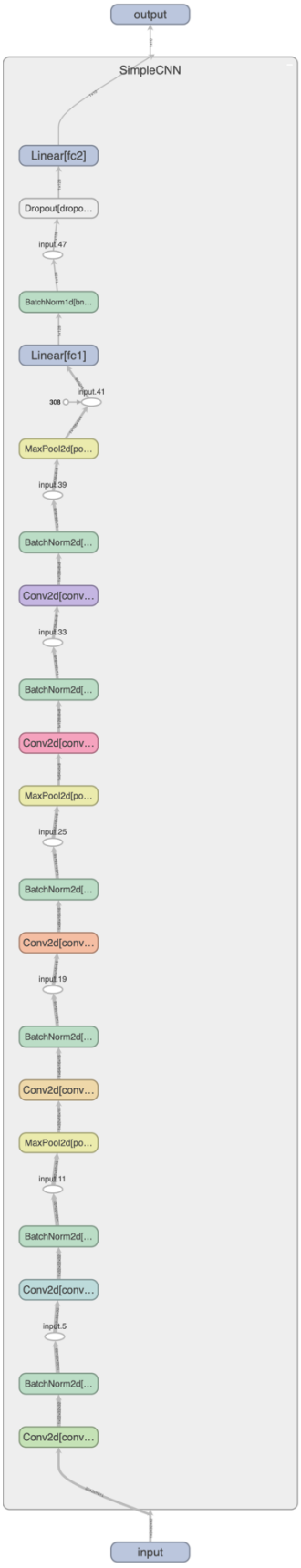
(1) Print model summary(including parameters) and plot model(5%)

- Summary

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1           [-1, 32, 32, 32]             896
       BatchNorm2d-2           [-1, 32, 32, 32]              64
            Conv2d-3           [-1, 32, 32, 32]           9,248
       BatchNorm2d-4           [-1, 32, 32, 32]              64
         MaxPool2d-5           [-1, 32, 16, 16]               0
            Conv2d-6           [-1, 64, 16, 16]          18,496
       BatchNorm2d-7           [-1, 64, 16, 16]             128
            Conv2d-8           [-1, 64, 16, 16]          36,928
       BatchNorm2d-9           [-1, 64, 16, 16]             128
        MaxPool2d-10             [-1, 64, 8, 8]               0
           Conv2d-11            [-1, 128, 8, 8]          73,856
      BatchNorm2d-12            [-1, 128, 8, 8]             256
           Conv2d-13            [-1, 128, 8, 8]         147,584
      BatchNorm2d-14            [-1, 128, 8, 8]             256
        MaxPool2d-15            [-1, 128, 4, 4]               0
           Linear-16                  [-1, 128]         262,272
      BatchNorm1d-17                  [-1, 128]             256
          Dropout-18                  [-1, 128]               0
           Linear-19                   [-1, 10]           1,290
================================================================
Total params: 551,722
Trainable params: 551,722
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.01
Forward/backward pass size (MB): 1.86
Params size (MB): 2.10
Estimated Total Size (MB): 3.98
----------------------------------------------------------------
```

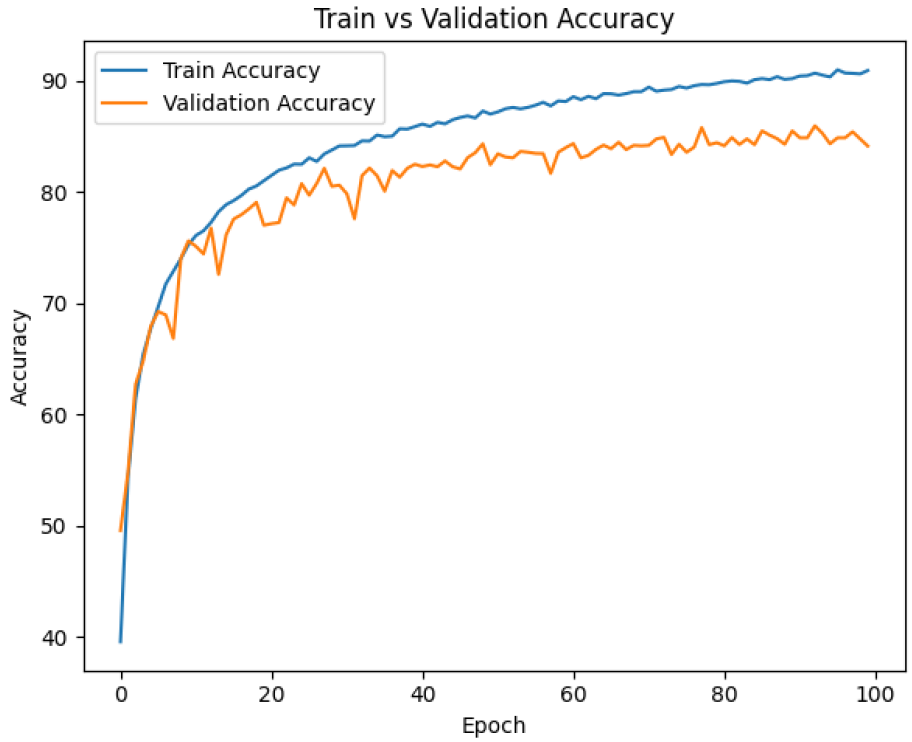
- FLOPs : 39356160.0
- Parameters : 551722.0

- Plot model

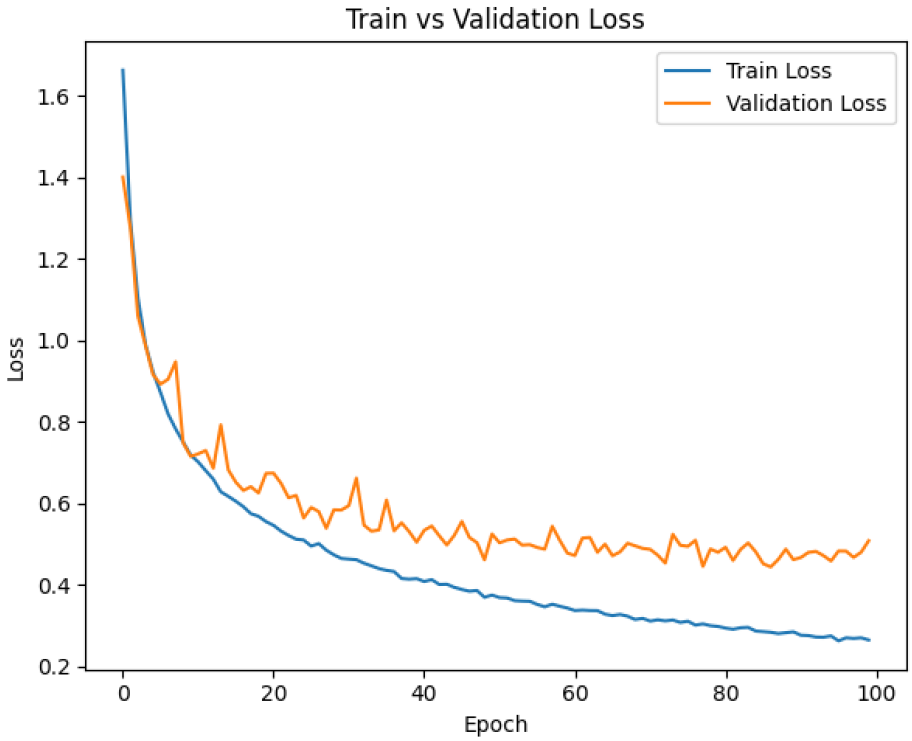

output
SimpleCNN
Linear[fc2]
Dropout[dropo…
input.47
BatchNorm1d[bn…
Linear[fc1]
input.41
308
MaxPool2d[po…
input.39
BatchNorm2d[…
Conv2d[conv…
input.33
BatchNorm2d[…
Conv2d[conv…
MaxPool2d[po…
input.25
BatchNorm2d[…
Conv2d[conv…
input.19
BatchNorm2d[…
Conv2d[conv…
MaxPool2d[po…
input.11
BatchNorm2d[…
Conv2d[conv…
input.5
BatchNorm2d[…
Conv2d[conv…
input

(2) Print test accuracy, plot epoch-train accuracy, epoch-val accuracy, epoch-train loss, epoch-val loss(10%)

- Test Accuracy : 88.63%
- Plot Epoch-Train&Val Accuracy



- Plot Epoch-Train&Val Loss



(3) Describe how do you choose your best model(5%)

Initially, I followed the suggestion from the PPT by applying two layers of Conv-BN-ReLU followed by pooling, but the results were not satisfactory. Therefore, I deepened the model by repeating the process of two Conv-BN-ReLU layers followed by pooling three times. Then, I added two fully connected layers, including a dropout layer, and found that the model's performance improved.

At first, increasing the epochs from 50 to 100 did not yield better results. Adjusting the learning rate to 0.1 with step decay to gradually reduce the learning rate also did not significantly improve the performance. As a result, I finally decided to set the learning rate to 0.001 and did not use a scheduler, allowing it to train for 100 epochs. Ultimately, I achieved a test accuracy of 88.63%.

Task 2 – ResNet 18 implementation

(4) Print model summary(including parameters) and plot model(5%)

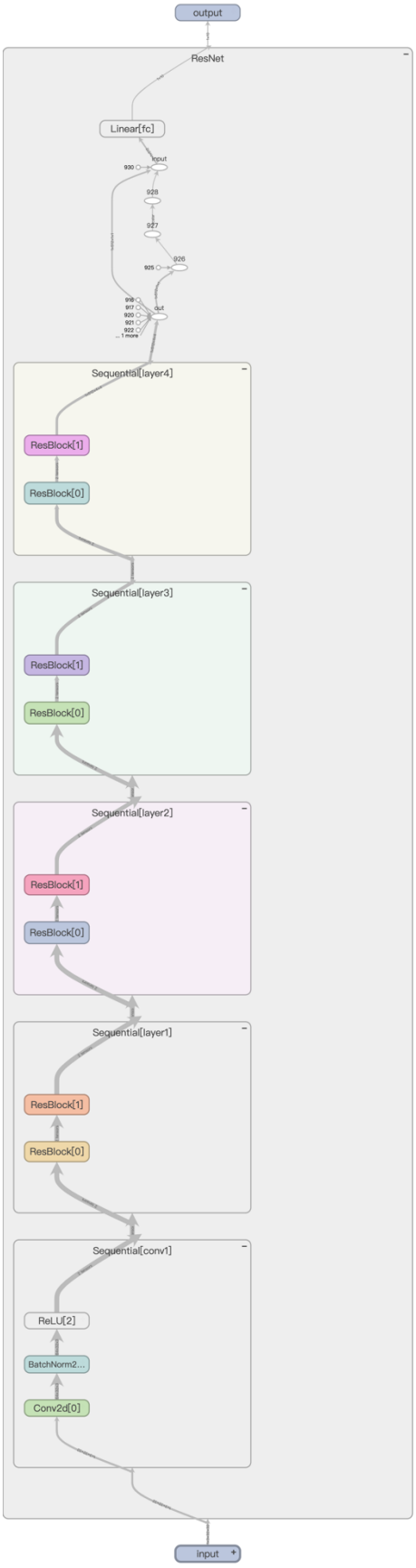
- Summary

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1          [-1, 64, 32, 32]           1,728
       BatchNorm2d-2          [-1, 64, 32, 32]             128
              ReLU-3          [-1, 64, 32, 32]               0
            Conv2d-4          [-1, 64, 32, 32]          36,864
       BatchNorm2d-5          [-1, 64, 32, 32]             128
              ReLU-6          [-1, 64, 32, 32]               0
            Conv2d-7          [-1, 64, 32, 32]          36,864
       BatchNorm2d-8          [-1, 64, 32, 32]             128
          ResBlock-9          [-1, 64, 32, 32]               0
           Conv2d-10          [-1, 64, 32, 32]          36,864
      BatchNorm2d-11          [-1, 64, 32, 32]             128
             ReLU-12          [-1, 64, 32, 32]               0
           Conv2d-13          [-1, 64, 32, 32]          36,864
      BatchNorm2d-14          [-1, 64, 32, 32]             128
         ResBlock-15          [-1, 64, 32, 32]               0
           Conv2d-16         [-1, 128, 16, 16]          73,728
      BatchNorm2d-17         [-1, 128, 16, 16]             256
             ReLU-18         [-1, 128, 16, 16]               0
           Conv2d-19         [-1, 128, 16, 16]         147,456
      BatchNorm2d-20         [-1, 128, 16, 16]             256
```

```
      Conv2d-21          [-1, 128, 16, 16]           8,192
 BatchNorm2d-22          [-1, 128, 16, 16]             256
    ResBlock-23          [-1, 128, 16, 16]               0
      Conv2d-24          [-1, 128, 16, 16]         147,456
 BatchNorm2d-25          [-1, 128, 16, 16]             256
        ReLU-26          [-1, 128, 16, 16]               0
      Conv2d-27          [-1, 128, 16, 16]         147,456
 BatchNorm2d-28          [-1, 128, 16, 16]             256
    ResBlock-29          [-1, 128, 16, 16]               0
      Conv2d-30            [-1, 256, 8, 8]         294,912
 BatchNorm2d-31            [-1, 256, 8, 8]             512
        ReLU-32            [-1, 256, 8, 8]               0
      Conv2d-33            [-1, 256, 8, 8]         589,824
 BatchNorm2d-34            [-1, 256, 8, 8]             512
      Conv2d-35            [-1, 256, 8, 8]          32,768
 BatchNorm2d-36            [-1, 256, 8, 8]             512
    ResBlock-37            [-1, 256, 8, 8]               0
      Conv2d-38            [-1, 256, 8, 8]         589,824
 BatchNorm2d-39            [-1, 256, 8, 8]             512
        ReLU-40            [-1, 256, 8, 8]               0
      Conv2d-41            [-1, 256, 8, 8]         589,824
 BatchNorm2d-42            [-1, 256, 8, 8]             512
    ResBlock-43            [-1, 256, 8, 8]               0
      Conv2d-44            [-1, 512, 4, 4]       1,179,648
 BatchNorm2d-45            [-1, 512, 4, 4]           1,024
        ReLU-46            [-1, 512, 4, 4]               0
      Conv2d-47            [-1, 512, 4, 4]       2,359,296
 BatchNorm2d-48            [-1, 512, 4, 4]           1,024
      Conv2d-49            [-1, 512, 4, 4]         131,072
 BatchNorm2d-50            [-1, 512, 4, 4]           1,024
    ResBlock-51            [-1, 512, 4, 4]               0
      Conv2d-52            [-1, 512, 4, 4]       2,359,296
 BatchNorm2d-53            [-1, 512, 4, 4]           1,024
        ReLU-54            [-1, 512, 4, 4]               0
      Conv2d-55            [-1, 512, 4, 4]       2,359,296
 BatchNorm2d-56            [-1, 512, 4, 4]           1,024
    ResBlock-57            [-1, 512, 4, 4]               0
      Linear-58                  [-1, 10]           5,130
================================================================
Total params: 11,173,962
Trainable params: 11,173,962
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.01
Forward/backward pass size (MB): 13.63
Params size (MB): 42.63
Estimated Total Size (MB): 56.26
----------------------------------------------------------------
```

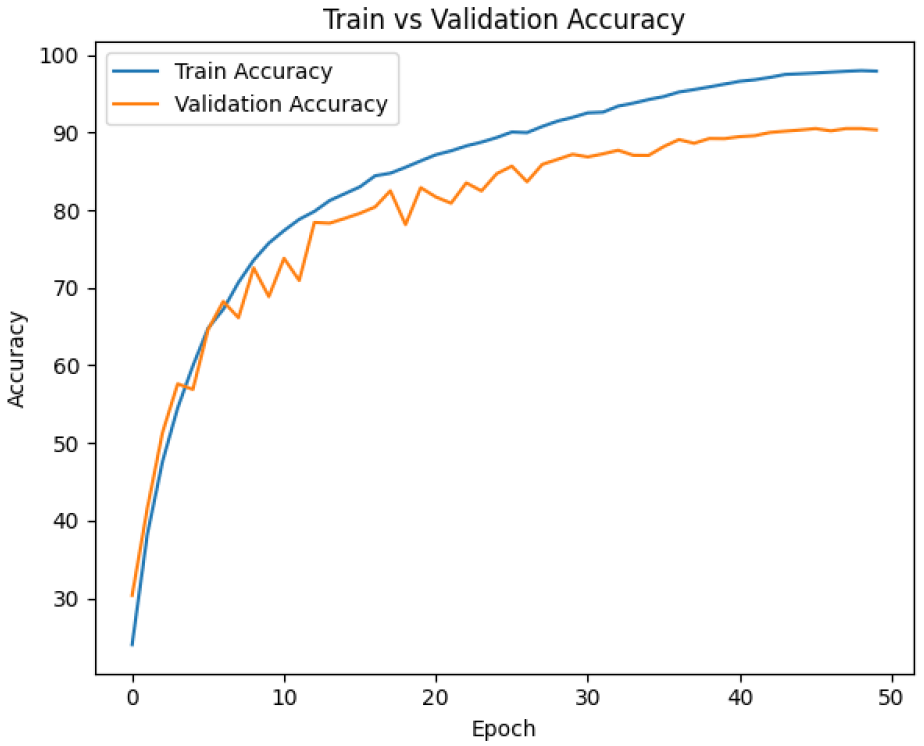
- FLOPs : 557880320.0
- Parameters : 11173962.0

- Plot model

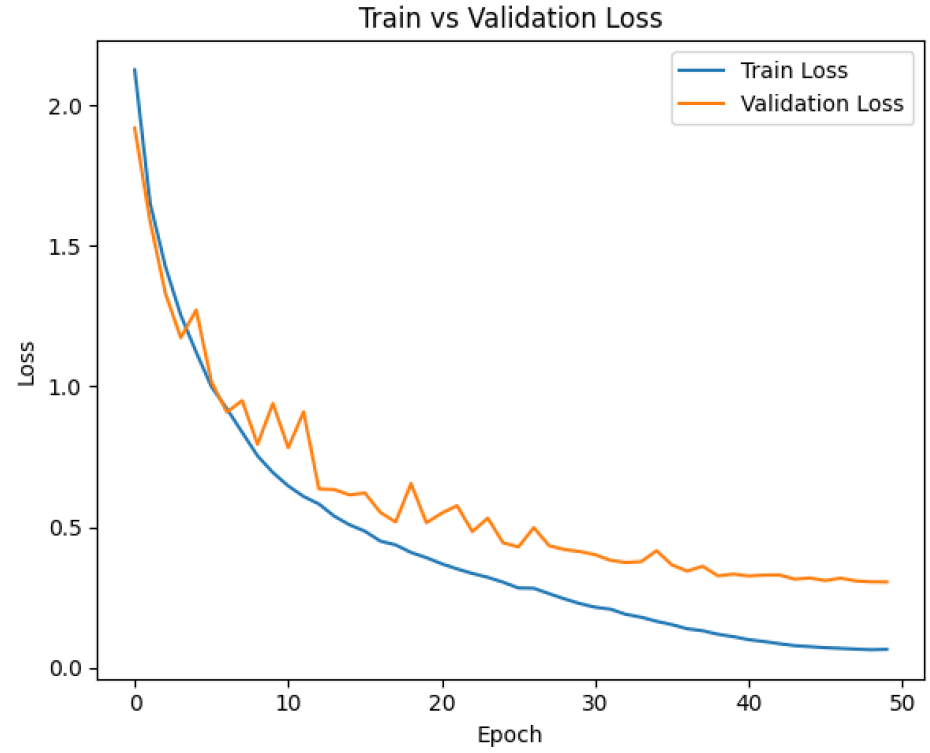

output
ResNet
Linear[fc]
input
930
928
927
926
925
out
Sequential[layer4]
ResBlock[1]
ResBlock[0]
Sequential[layer3]
ResBlock[1]
ResBlock[0]
Sequential[layer2]
ResBlock[1]
ResBlock[0]
Sequential[layer1]
ResBlock[1]
ResBlock[0]
Sequential[conv1]
ReLU[2]
BatchNorm2...
Conv2d[0]
input

(5) Print test accuracy, plot epoch-train accuracy, epoch-val accuracy, epoch-train loss, epoch-val loss(10%)

- Test Accuracy : 92.48%
- Plot Epoch-Train&Val Accuracy



- Plot Epoch-Train&Val Loss



(6) Describe how do you choose your best model(5%)

The model used was ResNet18, and the parameter settings were derived from the results of the ablation study in the following section.

For data processing, I found that applying normalization and augmentation significantly improved performance compared to when they were not applied, with normalization having the greatest impact.

Regarding the learning rate, I experimented with 0.1, 0.01, and 0.001, and observed that with only 50 epochs, the performance was better with a learning rate of 0.1 and 0.01 compared to 0.001.

Then, I applied two different adjustment strategies—step decay and cosine annealing—to the learning rates of 0.1 and 0.01. Ultimately, I found that the model performed best with a learning rate of 0.1 and the cosine annealing strategy, achieving an accuracy of 92.17% on the test set. I then retrained the model using this configuration, and the final accuracy on the test set was an excellent 92.48%.

(7) Experiment on the following and compare the result with baseline
   a. Input image normalization (15%)
   b. Data augmentation (15%)
   c. Different base learning rate and update strategy (15%)

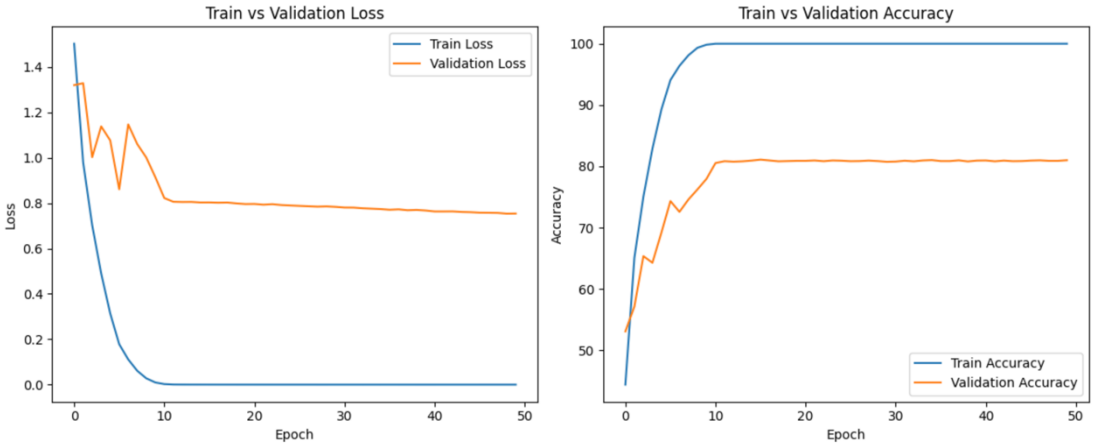
Variations	Learning Rate	Scheduler	Test Accuracy(%)
Baseline	0.01	-	23.07
Normalization	0.01	-	79.73
Augmentation	0.01	-	54.11
Normalization + Augmentation	0.01	-	88.41
Normalization + Augmentatio	0.1	-	87.89
Normalization + Augmentation	0.001	-	83.88
Normalization + Augmentation	0.01	Step Decay	81.32
Normalization + Augmentation	0.01	Cosine Annealing	90.60
Normalization + Augmentation	0.1	Step Decay	65.75
Normalization + Augmentation	0.1	Cosine Annealing	92.17

The following ablation experiments were conducted with a batch size of 512, 50 epochs, a cross-entropy loss function, and an SGD optimizer with a momentum of 0.9, a weight decay of 0.0005, and a learning rate of 0.01 unless otherwise specified.

- Baseline

When no data processing is applied, we can see from the graph that the train accuracy reaches 100%, but the validation accuracy plateaus at 80%, indicating that the model is overfitting. The test accuracy is 23.07%, suggesting that the model may have learned the wrong patterns and completely failed.

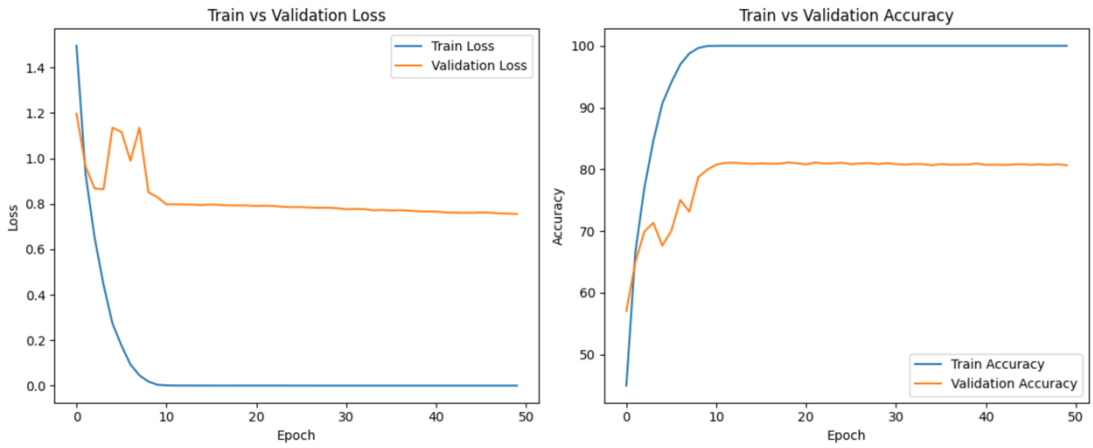
Variations	Learning Rate	Scheduler	Test Accuracy(%)
Baseline	0.01	-	23.07



- Normalization

After applying normalization to the data, although the overfitting situation can still be observed from the graph, the test results show a significant improvement in performance.

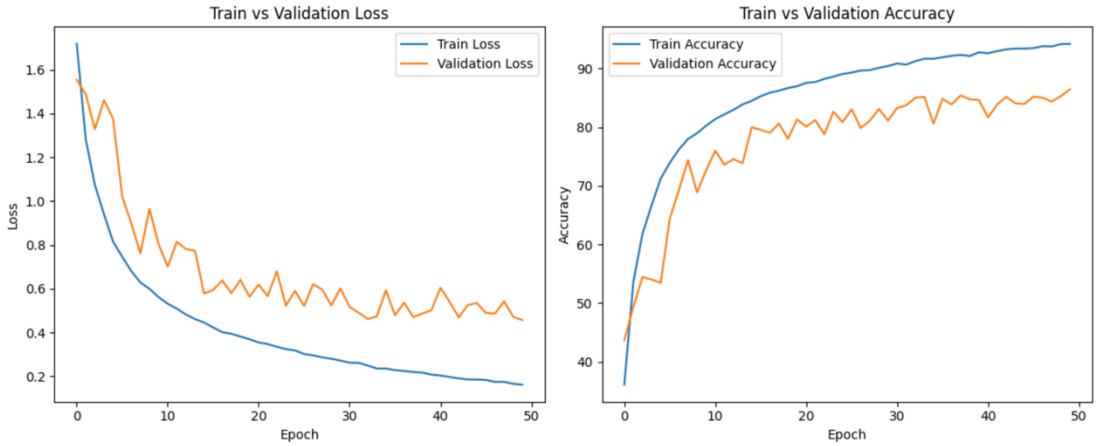
Variations	Learning Rate	Scheduler	Test Accuracy(%)
Baseline	0.01	-	23.07
Normalization	0.01	-	79.73



- Augmentation

When data augmentation was applied, the overfitting issue seemed to be alleviated, but the model's performance on the test set was not as good as before.

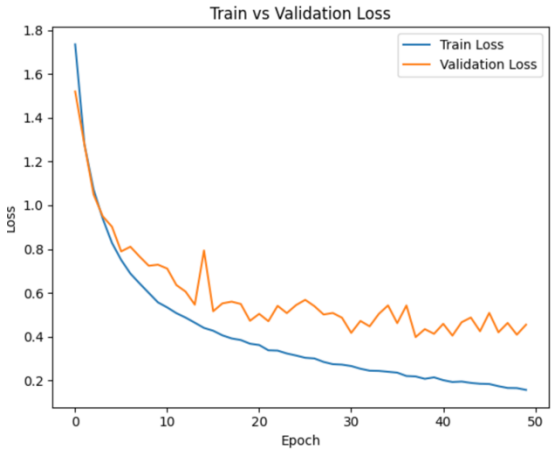
Variations	Learning Rate	Scheduler	Test Accuracy(%)
Baseline	0.01	-	23.07
Normalization	0.01	-	79.73
Augmentation	0.01	-	54.11

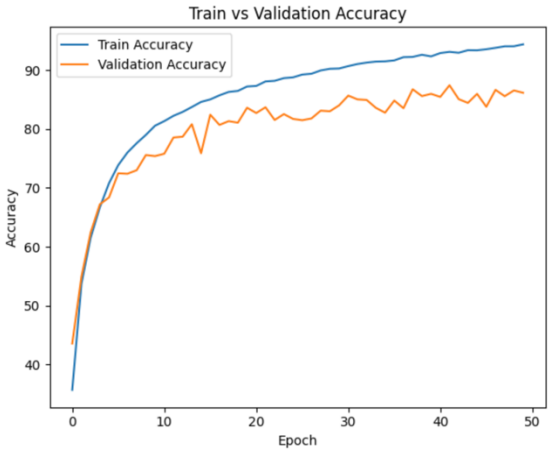


- Both

When both normalization and augmentation were applied to the data, the graph shows that not only was the overfitting issue alleviated, but the model's performance on the test set also improved to 88%.

Variations	Learning Rate	Scheduler	Test Accuracy(%)
Baseline	0.01	-	23.07
Normalization	0.01	-	79.73
Augmentation	0.01	-	54.11
Normalization + Augmentation	0.01	-	88.41

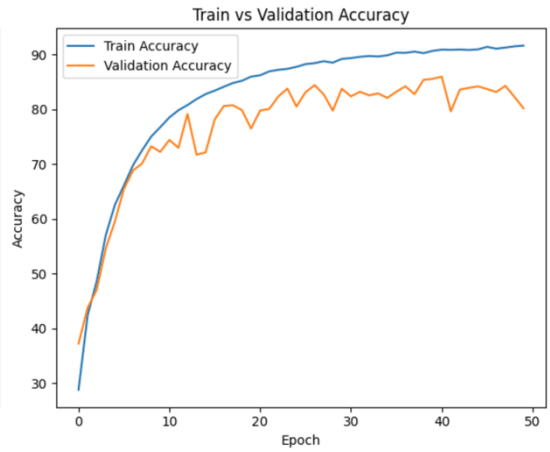




- Increasing the learning rate

After increasing the learning rate, the results were found to be similar to when the learning rate was 0.01, where the test accuracy reached 87.89%.

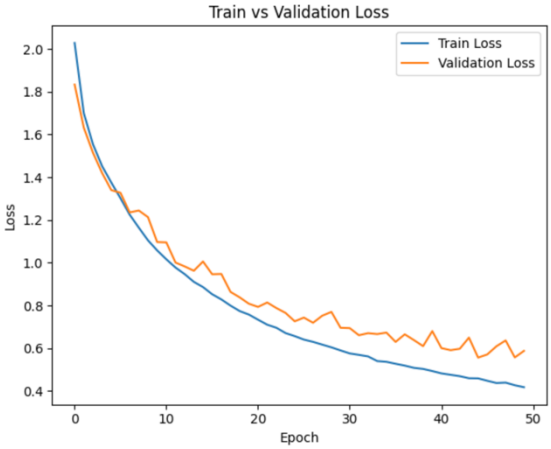
Variations	Learning Rate	Scheduler	Test Accuracy(%)
Baseline	0.01	-	23.07
Normalization	0.01	-	79.73
Augmentation	0.01	-	54.11
Normalization + Augmentation	0.01	-	88.41
Normalization + Augmentatio	0.1	-	87.89

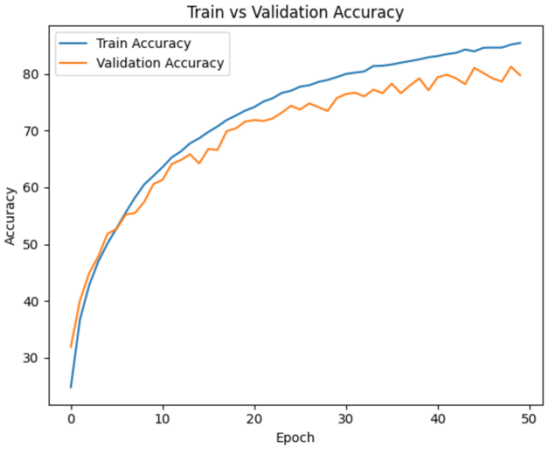


- Decreasing the learning rate

After decreasing the learning rate, the two curves became closer on the graph, indicating that the overfitting issue was further alleviated. However, it seems that the model has not yet fully converged, possibly due to an insufficient number of epochs.

Variations	Learning Rate	Scheduler	Test Accuracy(%)
Baseline	0.01	-	23.07
Normalization	0.01	-	79.73
Augmentation	0.01	-	54.11
Normalization + Augmentation	0.01	-	88.41
Normalization + Augmentatio	0.1	-	87.89
Normalization + Augmentation	0.001	-	83.88

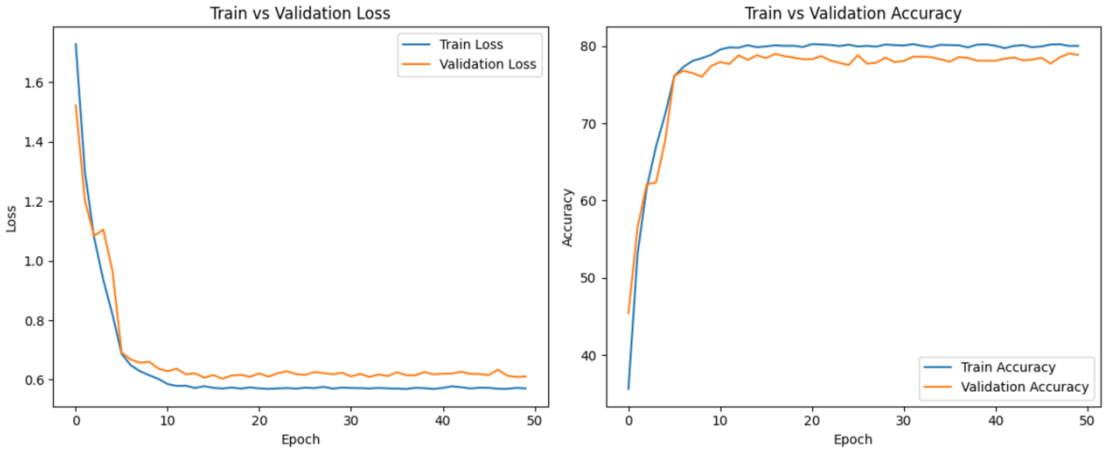




- Learning rate = 0.01 + Step Decay

From the previous observations, it became evident that starting with a high learning rate and using a gradually decreasing learning rate strategy would help the model converge within 50 epochs. Therefore, I introduced step decay. However, the results showed a decrease in performance, possibly because the step decay reduced the learning rate too quickly, causing the learning rate to become too low before the model had fully converged.

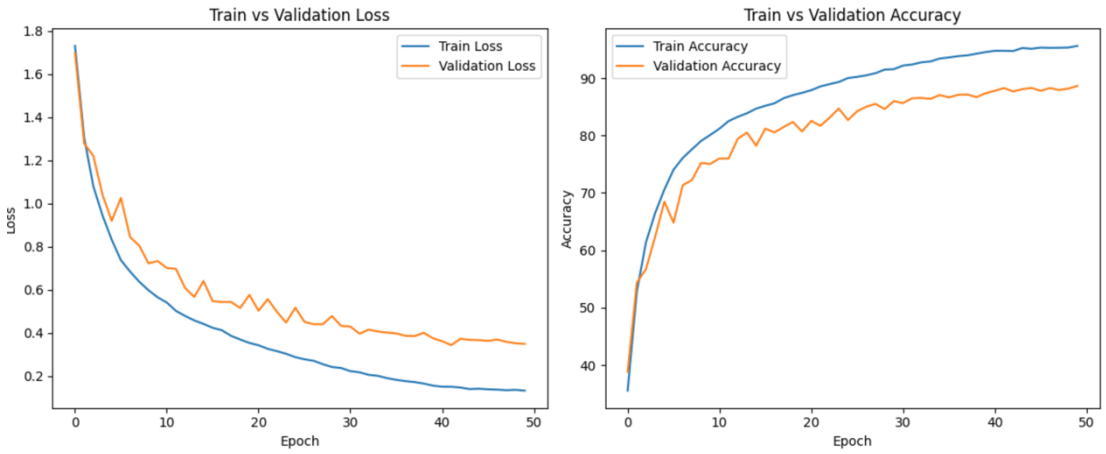
Variations	Learning Rate	Scheduler	Test Accuracy(%)
Baseline	0.01	-	23.07
Normalization	0.01	-	79.73
Augmentation	0.01	-	54.11
Normalization + Augmentation	0.01	-	88.41
Normalization + Augmentatio	0.1	-	87.89
Normalization + Augmentation	0.001	-	83.88
Normalization + Augmentation	0.01	Step Decay	81.32



- Learning rate = 0.01 + Cosine Annealing

After introducing cosine annealing, the results improved. This could be because the learning rate decreased more gradually, allowing the model to continue converging steadily over time.

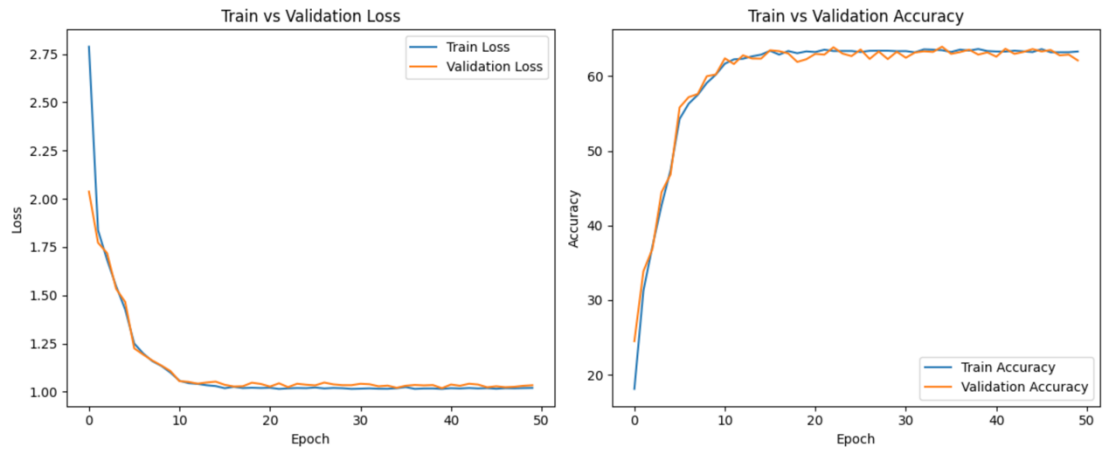
Variations	Learning Rate	Scheduler	Test Accuracy(%)
Baseline	0.01	-	23.07
Normalization	0.01	-	79.73
Augmentation	0.01	-	54.11
Normalization + Augmentation	0.01	-	88.41
Normalization + Augmentatio	0.1	-	87.89
Normalization + Augmentation	0.001	-	83.88
Normalization + Augmentation	0.01	Step Decay	81.32
Normalization + Augmentation	0.01	Cosine Annealing	90.60



- Learning rate = 0.1 + Step Decay

Next, I tried adjusting the initial learning rate to 0.1. It was observed that the model did not train well on the training set, getting stuck around 65%. This may be due to the same issue as previously mentioned: the step decay caused the learning rate to drop too quickly, preventing the model from converging properly, as the learning rate became too low before reaching the minimum.

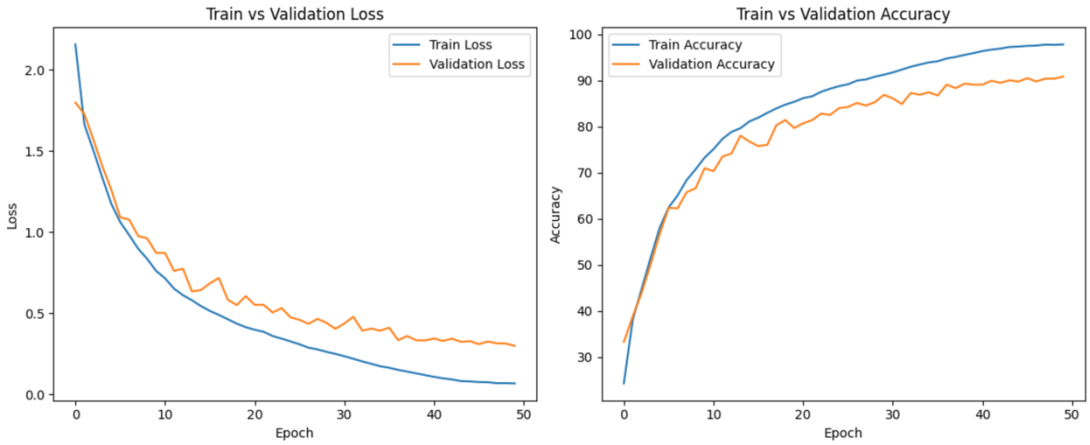
Variations	Learning Rate	Scheduler	Test Accuracy(%)
Baseline	0.01	-	23.07
Normalization	0.01	-	79.73
Augmentation	0.01	-	54.11
Normalization + Augmentation	0.01	-	88.41
Normalization + Augmentatio	0.1	-	87.89
Normalization + Augmentation	0.001	-	83.88
Normalization + Augmentation	0.01	Step Decay	81.32
Normalization + Augmentation	0.01	Cosine Annealing	90.60
Normalization + Augmentation	0.1	Step Decay	65.75



- Learning rate = 0.1 + Cosine Annealing

Finally, I tried another combination and found that it performed very well. When using an initial learning rate of 0.1 with cosine annealing, the model was able to quickly converge in the early stages, and the gradually decreasing learning rate helped the model converge even better. Therefore, I ultimately decided to choose this model as our final model.

Variations	Learning Rate	Scheduler	Test Accuracy(%)
Baseline	0.01	-	23.07
Normalization	0.01	-	79.73
Augmentation	0.01	-	54.11
Normalization + Augmentation	0.01	-	88.41
Normalization + Augmentatio	0.1	-	87.89
Normalization + Augmentation	0.001	-	83.88
Normalization + Augmentation	0.01	Step Decay	81.32
Normalization + Augmentation	0.01	Cosine Annealing	90.60
Normalization + Augmentation	0.1	Step Decay	65.75
Normalization + Augmentation	0.1	Cosine Annealing	92.17



(8) What challenges did you encounter, how did you solve them, and what are your thoughts and suggestions regarding this lab? (15%)

For the first question, it was initially very difficult to achieve a test accuracy above 80%, and even after switching to a more complex model, the results didn't improve much. Fortunately, I had a sudden realization to try lowering the learning rate to 0.001 and stop using the scheduler. I also increased the number of epochs to 100, allowing the model to train more gradually. Finally, this

approach led to a test accuracy exceeding 80%.

For the second question, thanks to the ablation experiments, I was able to find better parameter settings. However, conducting such many experiments was extremely time-consuming, making time and computing power the biggest challenges for this assignment. If I had access to more computing power, I might have been able to conduct the experiments more efficiently, create a more detailed report, and further improve the model's performance.

Regarding these two assignments, I'm truly grateful for the detailed explanations from the teaching assistants and the sample code provided, which helped me get started more quickly. I sincerely thank the professor and teaching assistants for the thoughtful allocation of these assignments, which allowed me to learn a great deal even from just these two tasks. For the second assignment, having to handle two tasks at once was one of the biggest challenges for me, especially since all the code had to be managed within a single Jupyter notebook. This not only required me to write the code in a clean and concise manner, but I also had to be careful not to rerun cells that had already been executed. This was quite difficult for me, and since I could only run one experiment at a time, it also took a considerable amount of time. Perhaps handling one task per code file could be a solution, but learning how to work with this setup is also part of the process for me.