Disclaimer/ Notes:

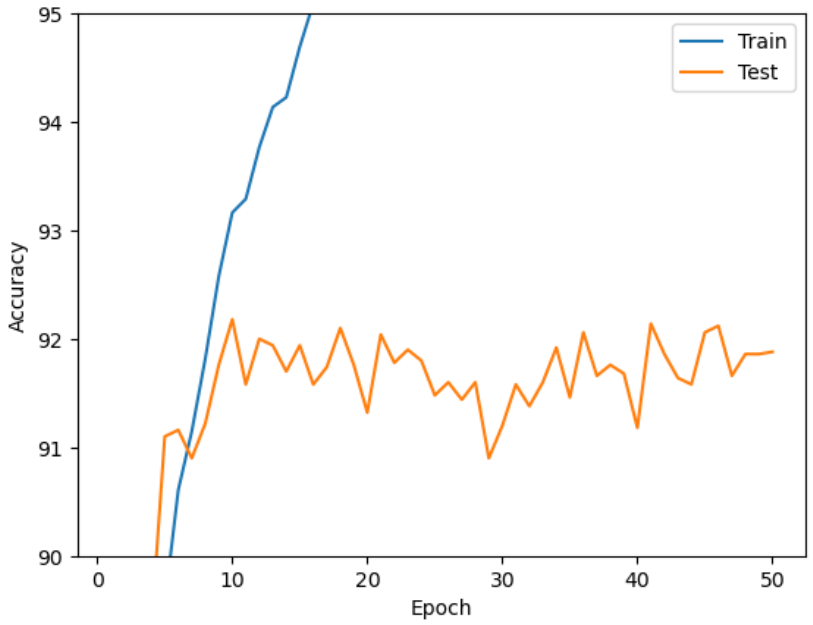
- To save time, all intermediate processes were run in Kaggle, which supports running multiple notebooks at the same time. Its CPU is considerably slower than my PC's CPU, so the runtime of these processes could be much longer than my normal runtime. The final runtime is the only relevant runtime. The data_dir was also different in the intermediates and the final notebook.

- Python version and torch version were set similarly to Kaggle's environment.

- Before running the given templates, the codes for model evaluation have been slightly modified. That is, the code of evaluating test accuracy is put into each epoch loop to have a better view of training versus test accuracy throughout the epochs. Test accuracy of each label, as requested in this assignment, has also been added.

- The results in each section below were also shown in the given zipped codes, with each folder referring to each section and *FINAL.ipynb* being the final model.

Starting by running the given template for many epochs (50 in this run), it could be found that the training accuracy is excellent while the test accuracy only yielded a fair accuracy. This was a case of overfitting.



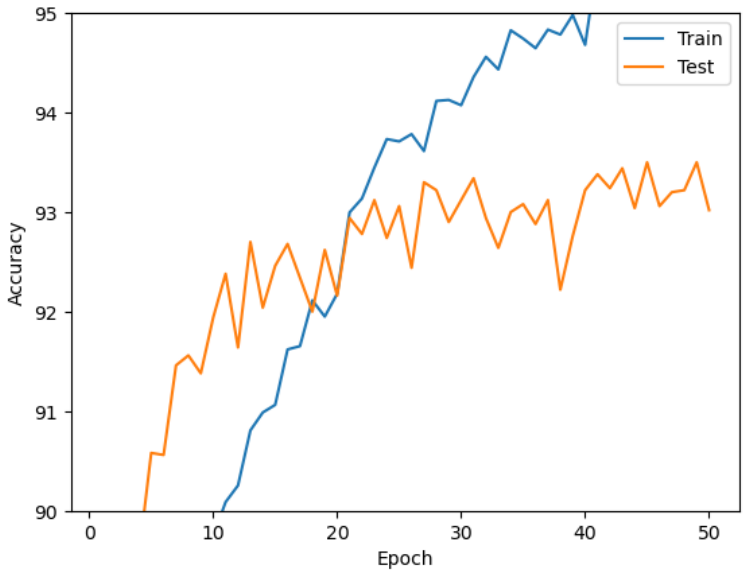
Initial Setting

According to this graph, the overfitting occurred on most epochs. Therefore, the overfitting should be not just due to high epochs. In the following part, data augmentation strategy, optimizer, learning rate and its scheduler were tested and modified to reduce the overfitting, thus possibly increasing the test accuracy.

Data Augmentation

Two additional lines for data augmentation were appended. The first line randomly made the training images padded, blurred or twisted, making it harder to train the model. For example,

by padding the images, the model might be encouraged to focus on the center pixels. The second line turned images into grayscale with 15% chance, thus making the model learn in a different color scheme.

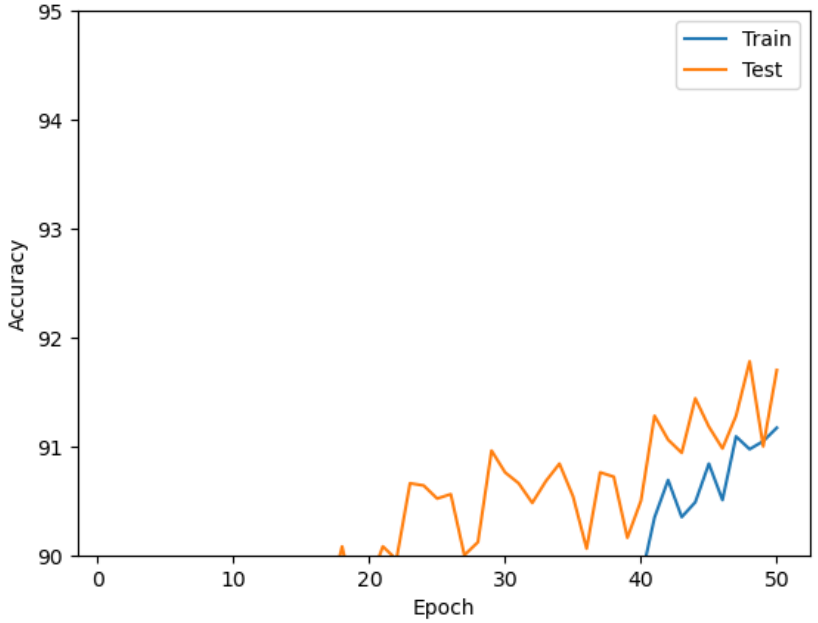


new data augmentation

From the screenshot, it could be seen that the overfitting had been reduced.

Learning Rate

Since the learning rate could be decayed with a scheduler, a high learning rate should be set initially in my opinion. When choosing an appropriate learning rate, it should be high to make the models learn faster, but not too high to avoid underfitting as a result in unsuccessful gradient descent.

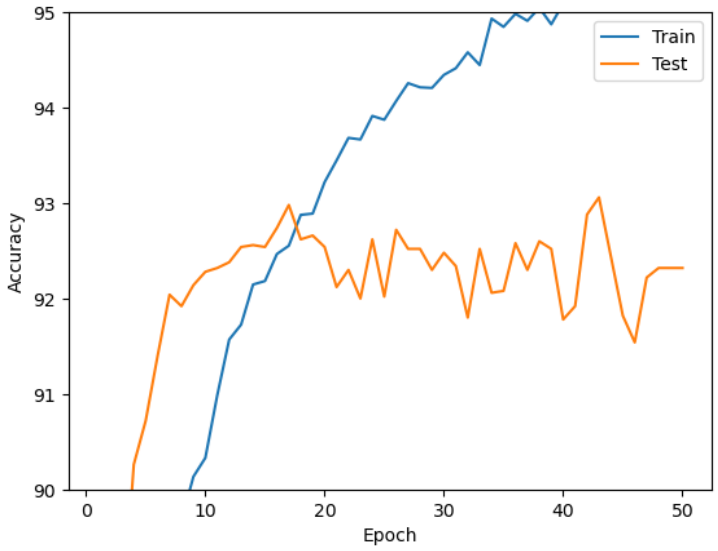


0.002 learning rate

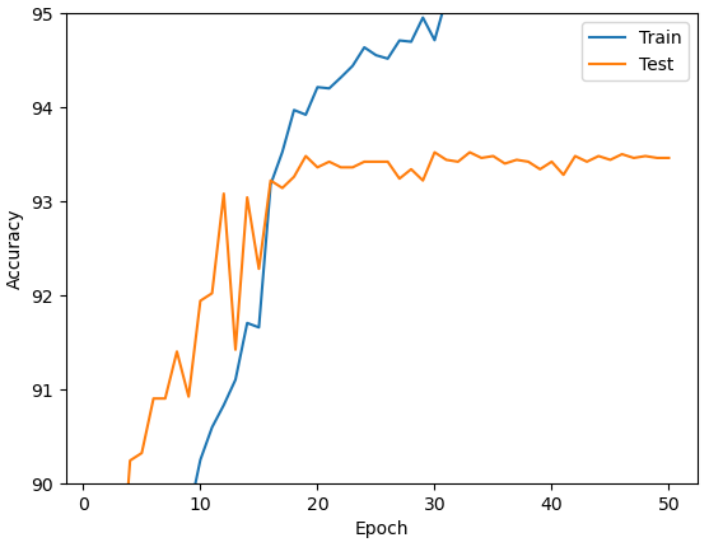
0.002 was tested to suffer from underfitting, so 0.001 remained unchanged as the learning rate.

Learning Rate Scheduler

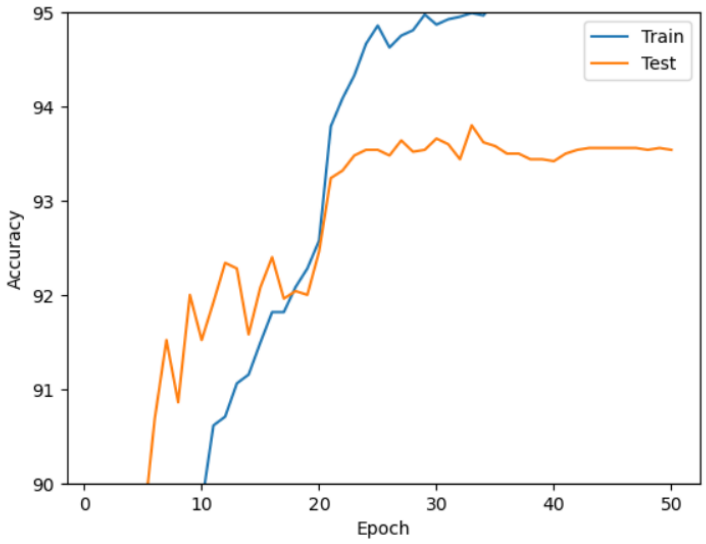
When the training loss remains still for several consecutive epochs, a learning rate decay may make the model “dig deeper”. Since the test accuracy reached a plateau in epoch 15-20, decaying the learning rate at epoch 15/20 seemed to be a good idea. Different gamma for the scheduler was tested too.



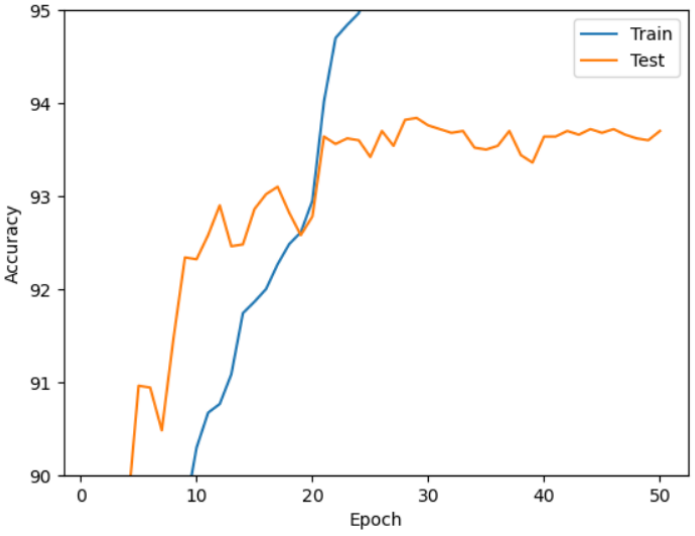
no decay



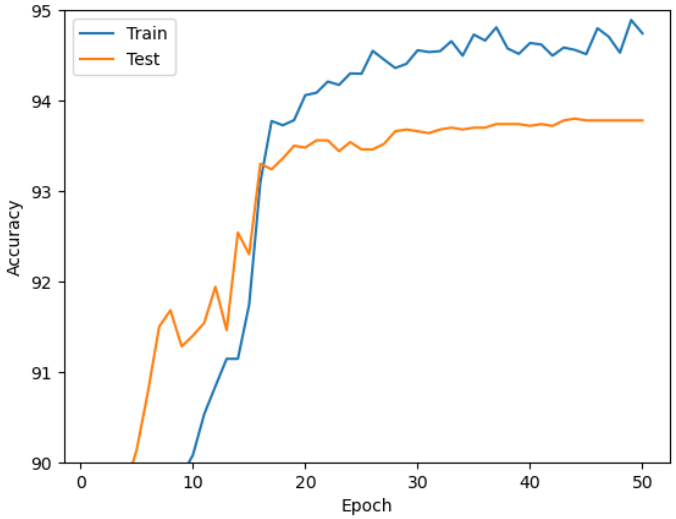
gamma = 0.1, step_size = 15



gamma = 0.05, step_size = 20



gamma = 0.1, step_size = 20

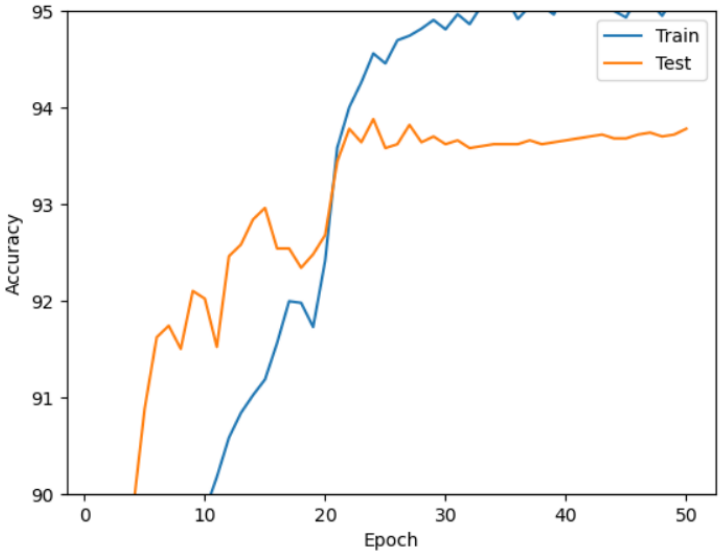


gamma = 0.05, step_size = 15

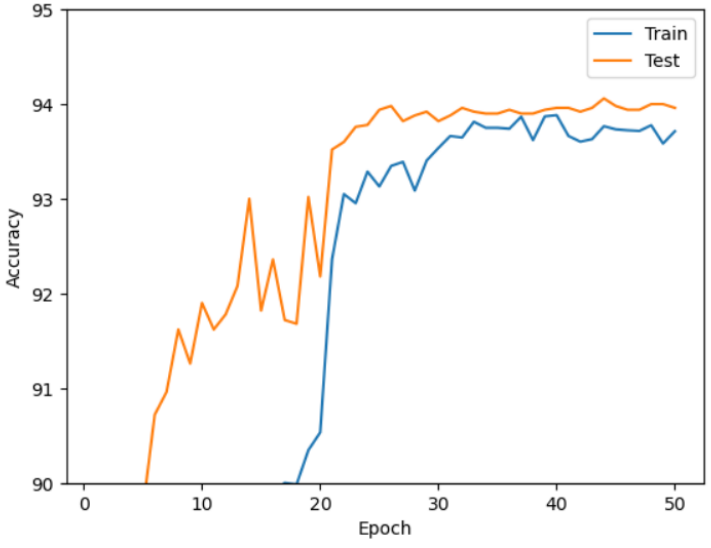
Actually, most of these trials with scheduler seemed to give similar test accuracy. After many trials, the learning rate scheduler was set with gamma=0.05 and steps=[20,30]. This was because the optimal first step seemed to be at epoch 20, and the second step was set to epoch 30 as opposed to 40 since the latter was too large to be effective.

Optimizer

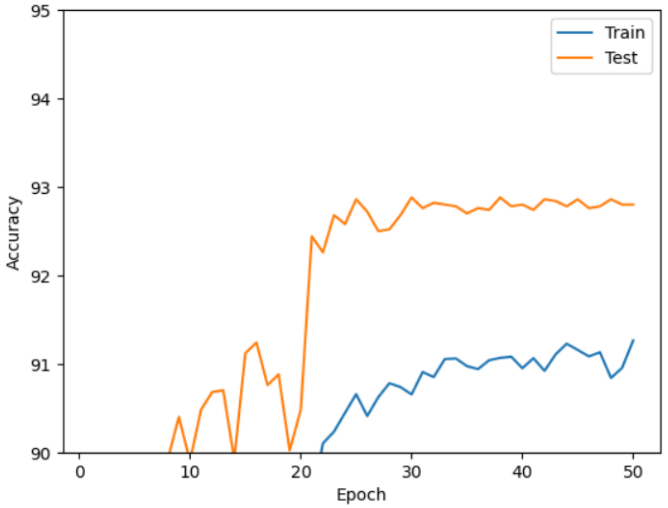
The given template used Adam as the optimizer. However, some online articles and forums claimed that Adam does not work well with weight decay/regularization. Therefore, AdamW was used for weight decay. Values of 1, 0.5, 0.1 were tested, and it seemed 0.5 worked the best.



weight_decay = 0.1



weight_decay = 0.5



weight_decay = 1

Conclusion

```
epoch = 40

if __name__ == '__main__':
    end = time.time()
    model_ft = Net().to(device) # Model initialization
    print(model_ft.network)
    criterion = nn.CrossEntropyLoss() # Loss function initialization

    # TODO: Adjust the following hyper-parameters: initial learning rate, decay strategy of the learning rate,
    #       number of training epochs
    optimizer_ft = optim.AdamW(model_ft.parameters(), lr=0.001, weight_decay=0.5) # The initial learning rate is 1e-3

    exp_lr_scheduler = lr_scheduler.MultiStepLR(optimizer_ft, milestones=[20,30], gamma=0.05)

    history, accuracy = train_test(model_ft, criterion, optimizer_ft, exp_lr_scheduler,
                num_epochs=epoch)

    print("time required %.2fs" %(time.time() - end))
```

After a few runs of the final model shown above, a maximum of 94.08% test accuracy was achieved.

```
Epoch with max test accuracy: 27
```

```
Epoch:[27]-Iteration:[100], training loss: 0.199
Epoch:[27]-Iteration:[200], training loss: 0.206
Time cost of one epoch: [48]s
Epoch:[27], training accuracy: 94.1, training loss: 0.204
Accuracy of the network on test images: 94.08
Test Accuracy of 0: 94.4
Test Accuracy of 1: 94.0
Test Accuracy of 2: 95.0
Test Accuracy of 3: 90.8
Test Accuracy of 4: 96.4
Test Accuracy of 5: 93.6
Test Accuracy of 6: 94.4
Test Accuracy of 7: 95.0
Test Accuracy of 8: 92.8
Test Accuracy of 9: 94.4
```