HW2 Report

张驰 2022010754 zhang-ch22@mails.tsinghua.edu.cn

1. Training & Testing

Explain how `self.training` work. Why should training and testing be different?

The `self.training` attribute is used to differentiate between the training and evaluation modes of a model. When the model is in training mode, it is set to `True`, otherwise it will be `False`.

The training and testing mode of the model should be different because the layers should be activated differently in the training and testing process. In this model, the dropout layer is used to prevent overfitting, and the batch normalization layer is used to normalize the data. The dropout operations are used to improve the model's generalization and prevent overfitting. However, during evaluation or testing, we want the model to make predictions without any randomness or normalization based on the specific batch of data. Therefore, setting `self.training` to False ensures that the model behaves appropriately during evaluation, leading to more consistent and reliable predictions. The batch normalization layer uses statistical information of the input batch. However, in testing we can only use the statistical information of the whole dataset, so we need to set `self.training` to False.

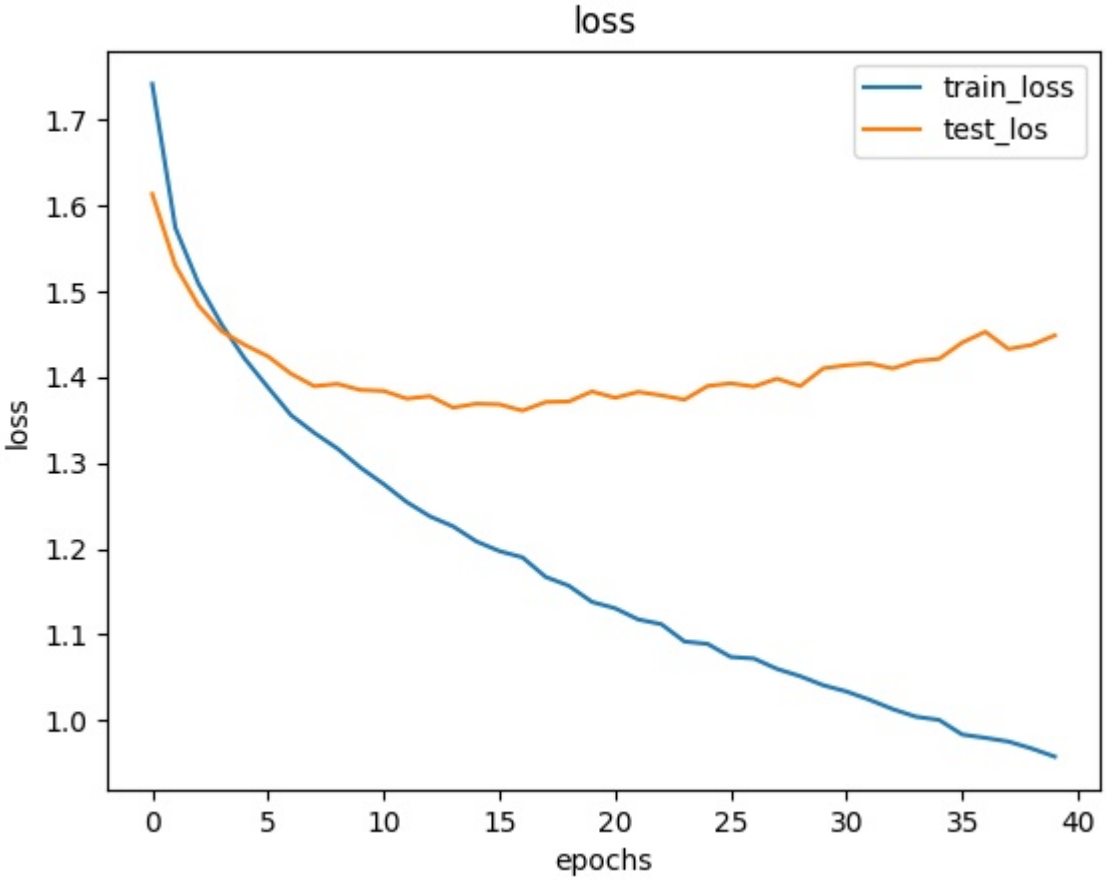
2. MLP & CNN

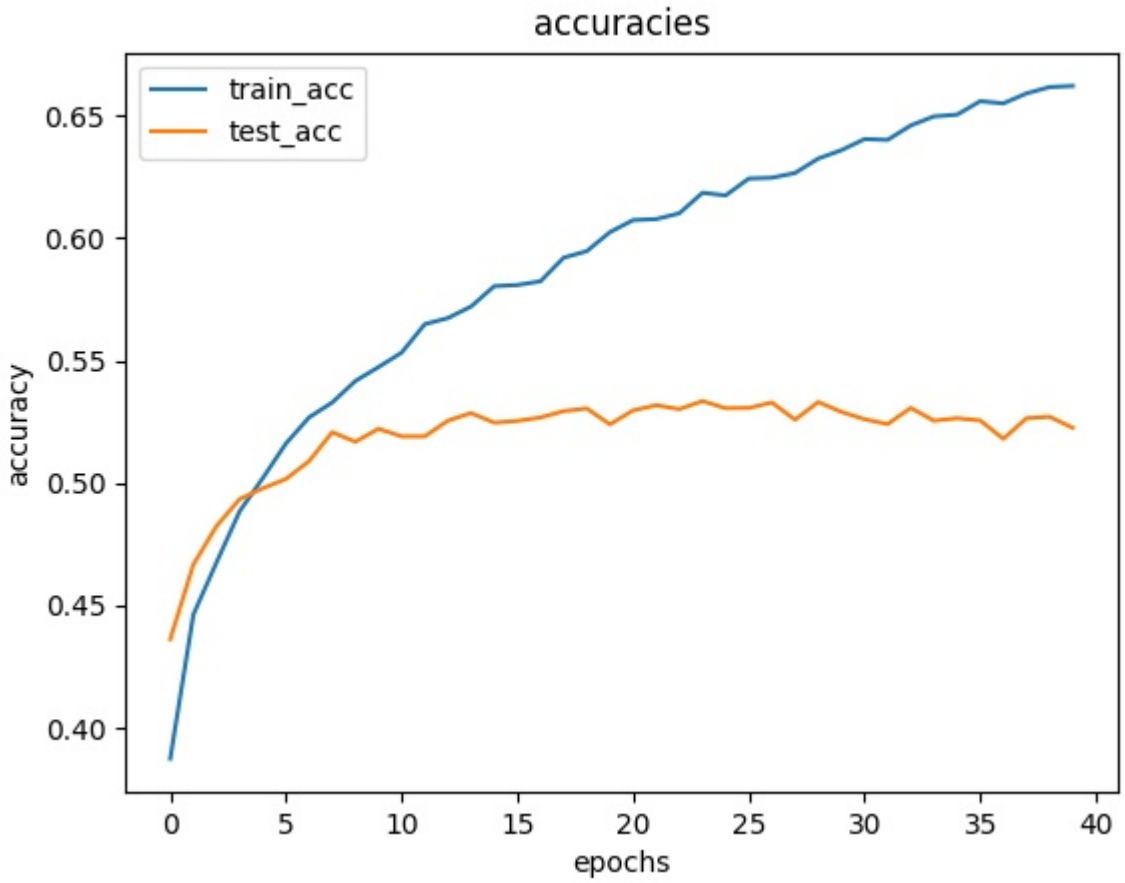
2.1 MLP with BN and Dropout

For the MLP network, I use the following network structure and hyperparameters to gain the best performance (of all the experiments I have done):

```
hidden-layer-size: 128
learning-rate: 0.001
batch-size: 100
drop-rate: 0.2
```

The training and validation curves are as follows:


loss
train_loss
test_los
1.7
1.6
1.5
1.4
1.3
1.2
1.1
1.0
loss
0
5
10
15
20
25
30
35
40
epochs



The testing information are as follows:

```
best epoch:                    24
best validation accuracy:      0.5334999892115593
test loss:                     1.358432720899582
test accuracy:                 0.5269999870657921
```

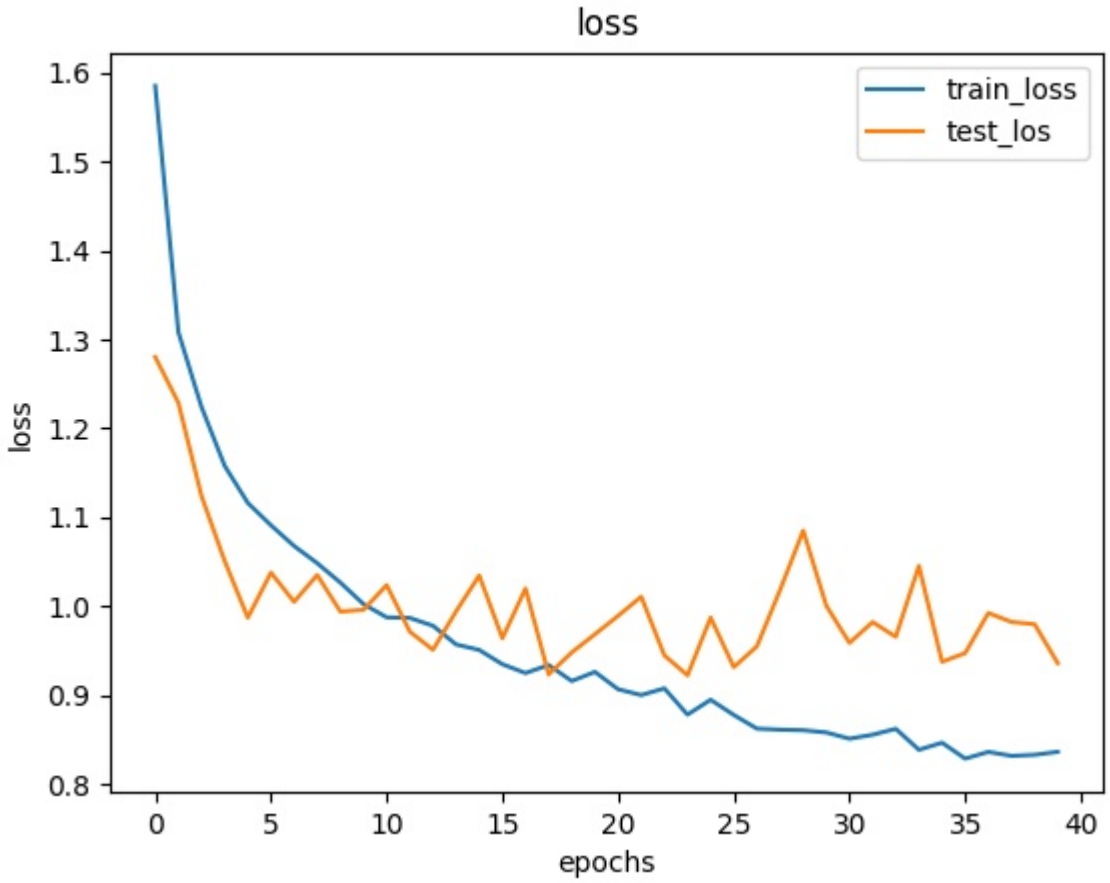
We can see that, though with the dropout layer the model isn't overfitting too severely, but the performance is still not good enough. The final test accuracy is only 0.527.

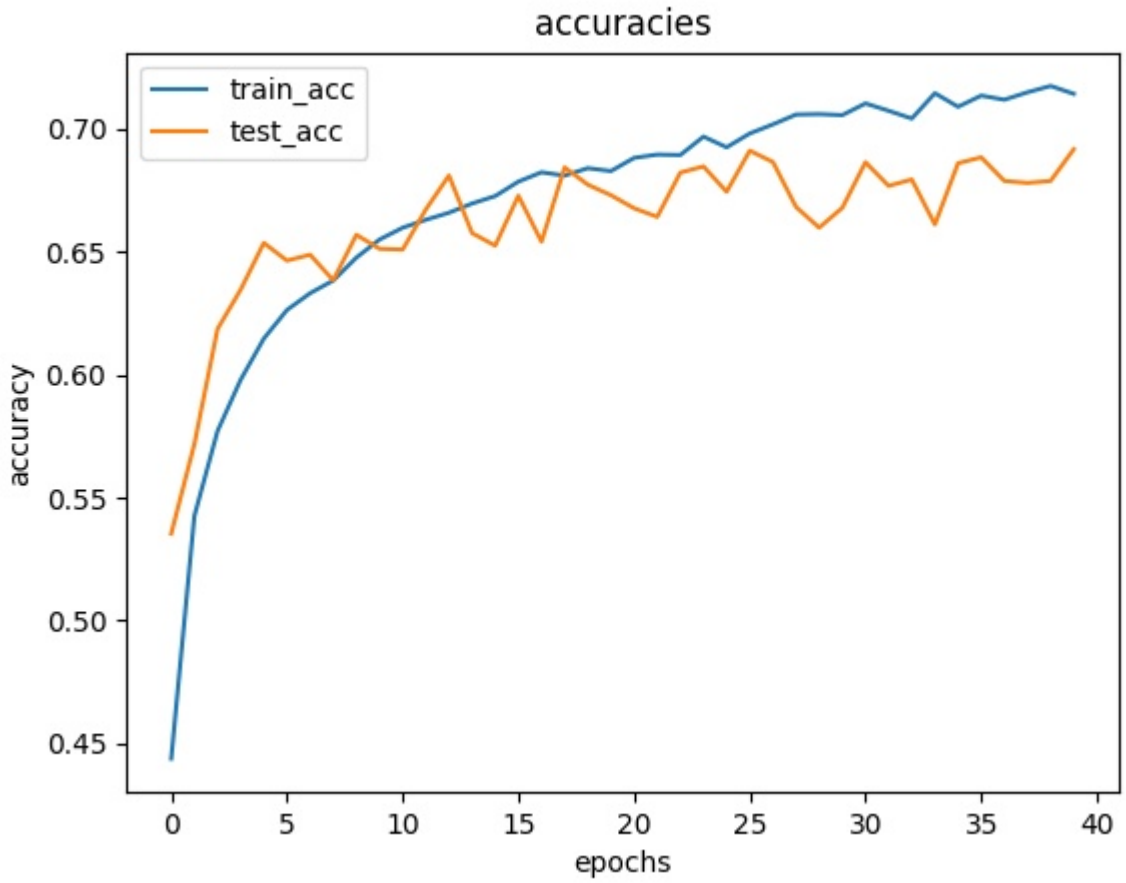
2.2 CNN with BN and Dropout

For the CNN network, I use the following network structure and hyperparameters to gain the best performance (of all the experiments I have done):

```
conv1: 3x3, 32, padding=1
maxpool1: 2x2, stride=2
conv2: 3x3, 64, padding=1
maxpool2: 2x2, stride=2
learning-rate: 0.001
batch-size: 100
drop-rate: 0.2
```

The training and validation curves are as follows:





The testing information are as follows:

```
best epoch:                    36
best validation accuracy:      0.7252999836206436
test loss:                     0.8844859760999679
test accuracy:                 0.7177999836206436
```

We can see that, the CNN model performs much better than the MLP model. The final test accuracy is 0.718, which is much higher than the MLP model.

In the curves, the training and validation curves are very close, which means that the model is not overfitting.

3. Train & Validation loss

Explain why training loss and validation loss are different. How does the difference help you tuning hyper-parameters?

The training loss and validation loss are different because they are calculated with different part of the dataset. The model was trained on the training set and the training loss is minimized by adjusting parameters. However, the model couldn't see the validation set during training which gives the validation loss. Thus the validation loss measures how well the trained model generalizes to new, unseen data. It provides an estimate of how the model is expected to perform on real-world data. Because the validation loss isn't minimized by the algorithm, it may usually be bigger than the training loss.

The ideal hyperparameters should make the training and validation loss low. The difference of the two losses can help tune the hyperparameters.

1. If there exists a big gap between the two losses, it usually means that the model is overfitted on the training dataset. In this case, we should try to reduce the model complexity, such as reducing the number of layers, reducing the number of hidden units. Or we can add the dropout rate to avoid overfitting.
2. If the two losses are both high, it usually means that the model is underfitted. In this case, we should try to increase the model complexity, such as increasing layers and hidden units, or increasing the learning rate.

4. Final Accuracy

The final accuracies of the two networks are given in part two. Where:

```
MLP: 0.5269999870657921
CNN: 0.7177999836206436
```

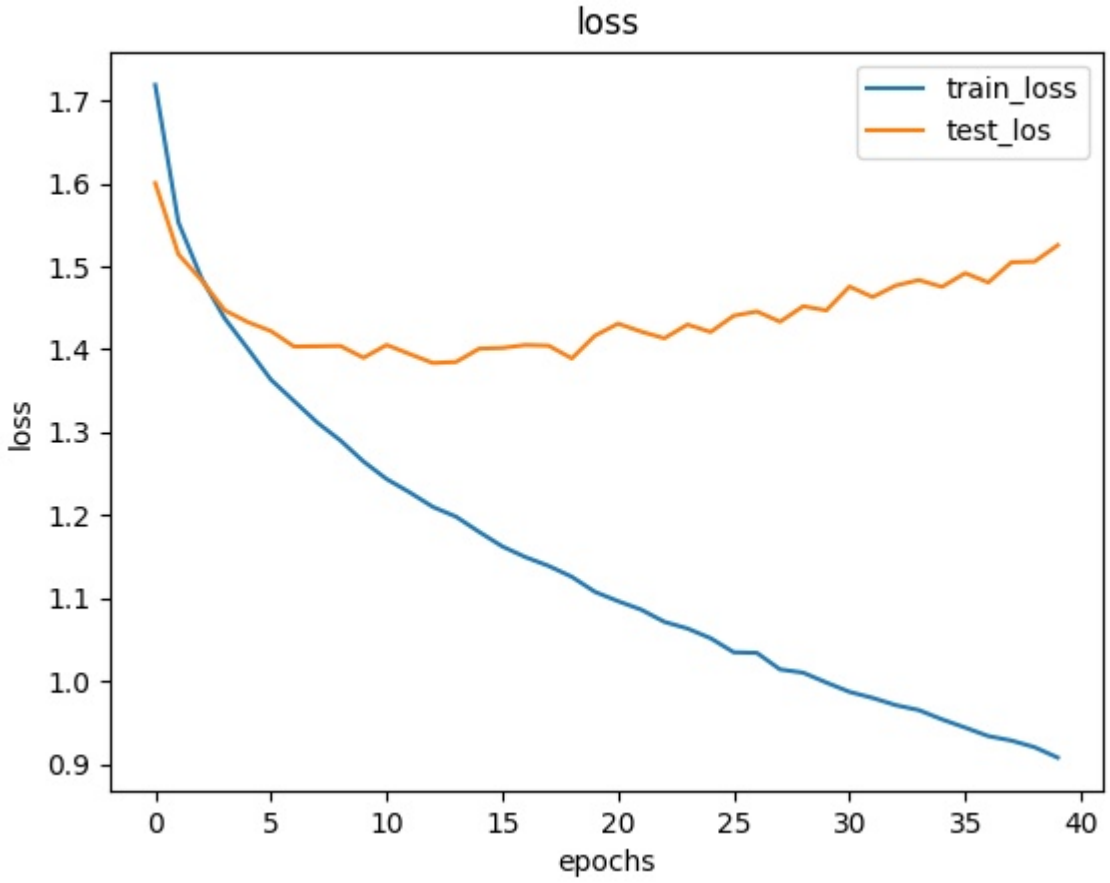
The CNN model performs much better than the MLP model. This may be because that, by taking the convolution product, the CNN focus the local features (the relation of one pixel and its nearest pixels) of the image.To achieve image classification, such local information is more relevant, because different classes of pictures show different local patterns. The MLP model, however, treats the image as a vector, which may not be able to extract the local features of the image.

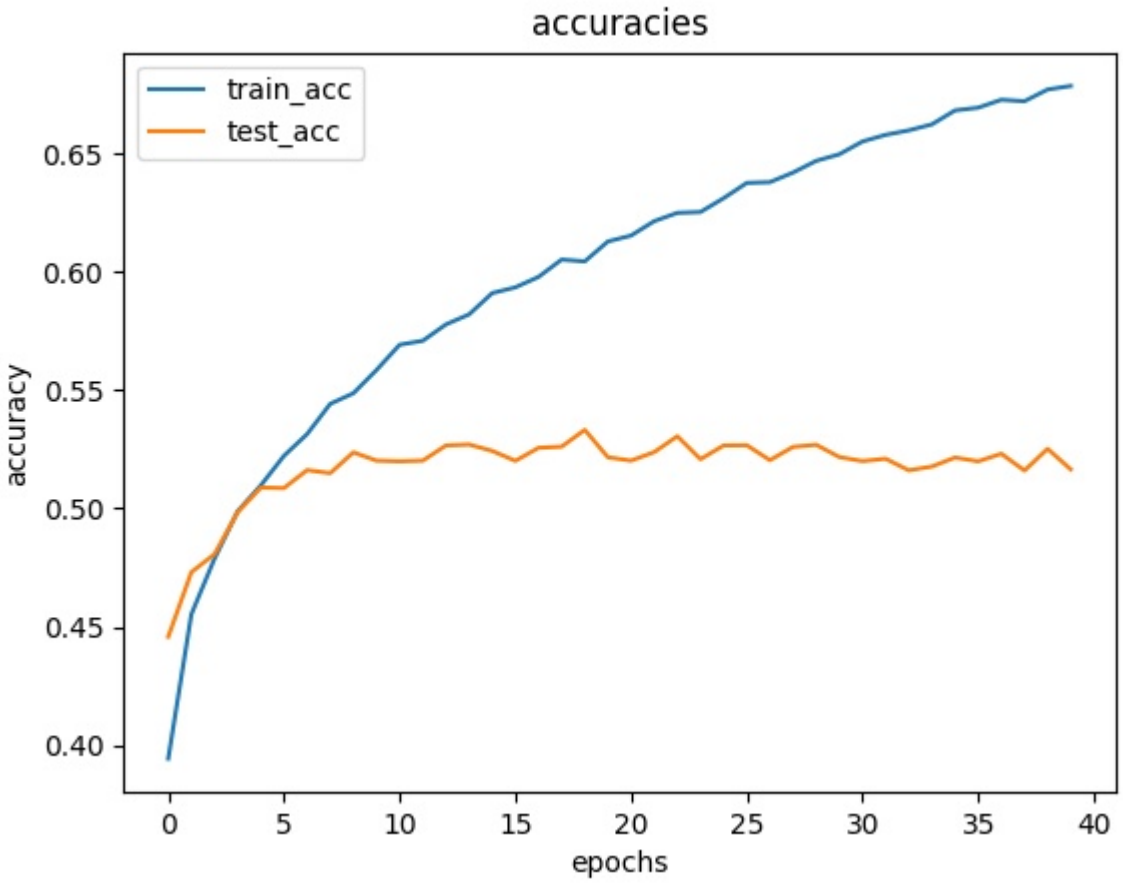
5. Batch Normalization

I tried to skip the batch normalization layer in the MLP and CNN networks respectively. The results are as follows:

> The hyperparameters are the same as the best ones in part two.

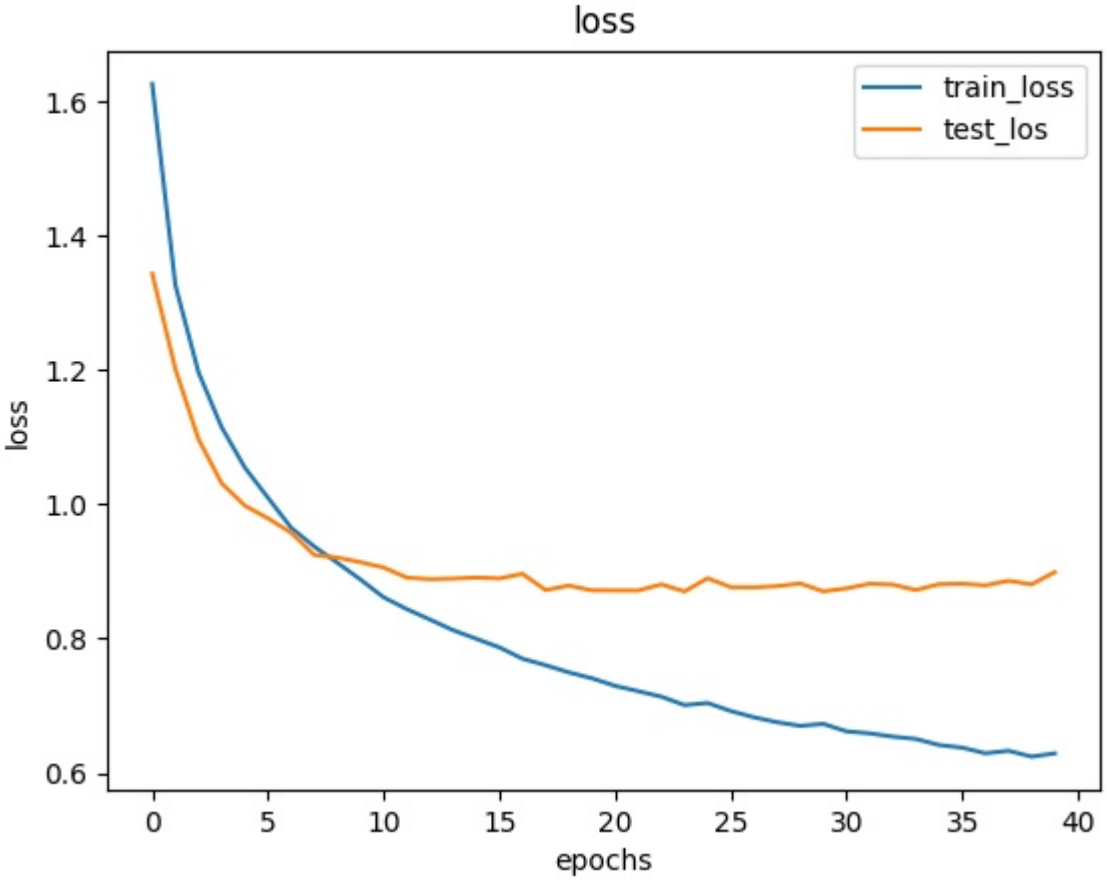
MLP:

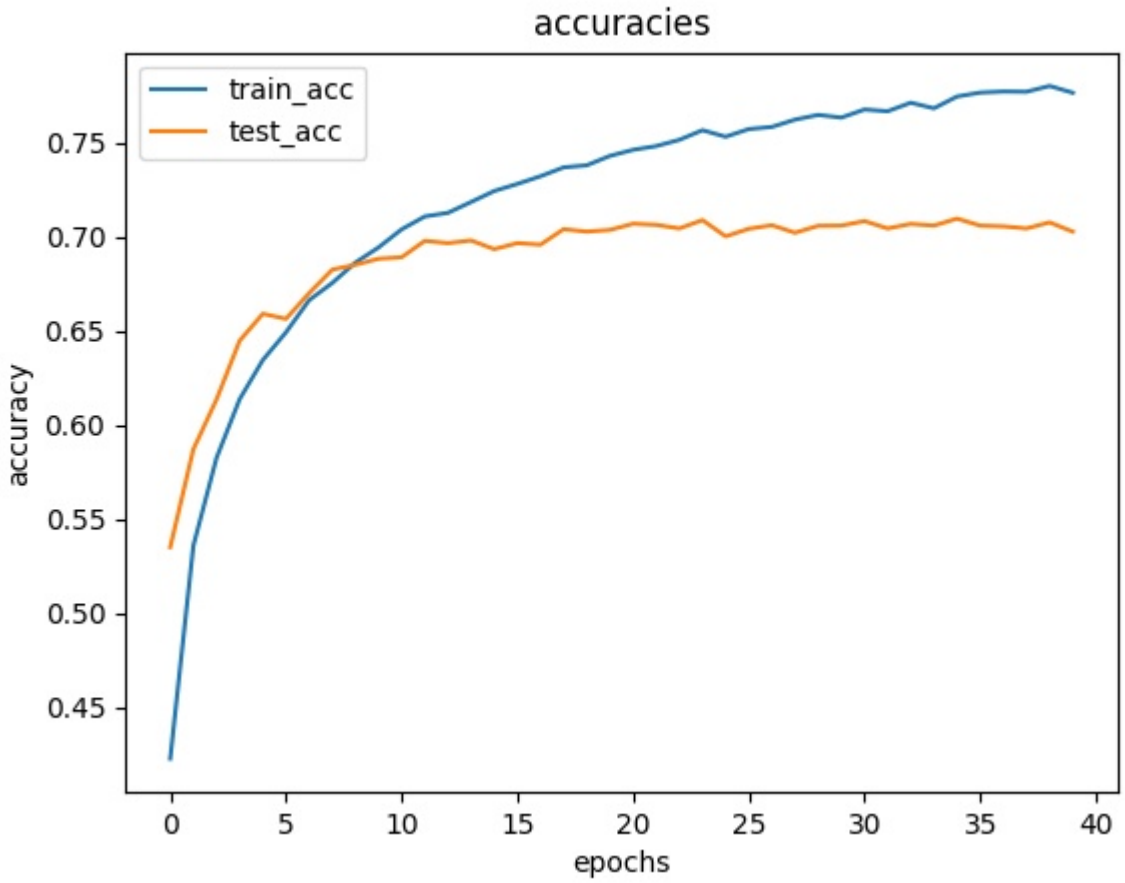

loss
train_loss
test_los
1.7
1.6
1.5
1.4
1.3
1.2
1.1
1.0
0.9
loss
0
5
10
15
20
25
30
35
40
epochs




```
best epoch:                    19
best validation accuracy:      0.5331999865174294
test loss:                     1.383483909368515
test accuracy:                 0.5202999883890151
```

CNN:


loss
train_loss
test_los
1.6
1.4
1.2
1.0
0.8
0.6
loss
0
5
10
15
20
25
30
35
40
epochs




```
best epoch:                     35
best validation accuracy:       0.7095999830961227
test loss:                      0.886548308134079
test accuracy:                  0.7000999832153321
```

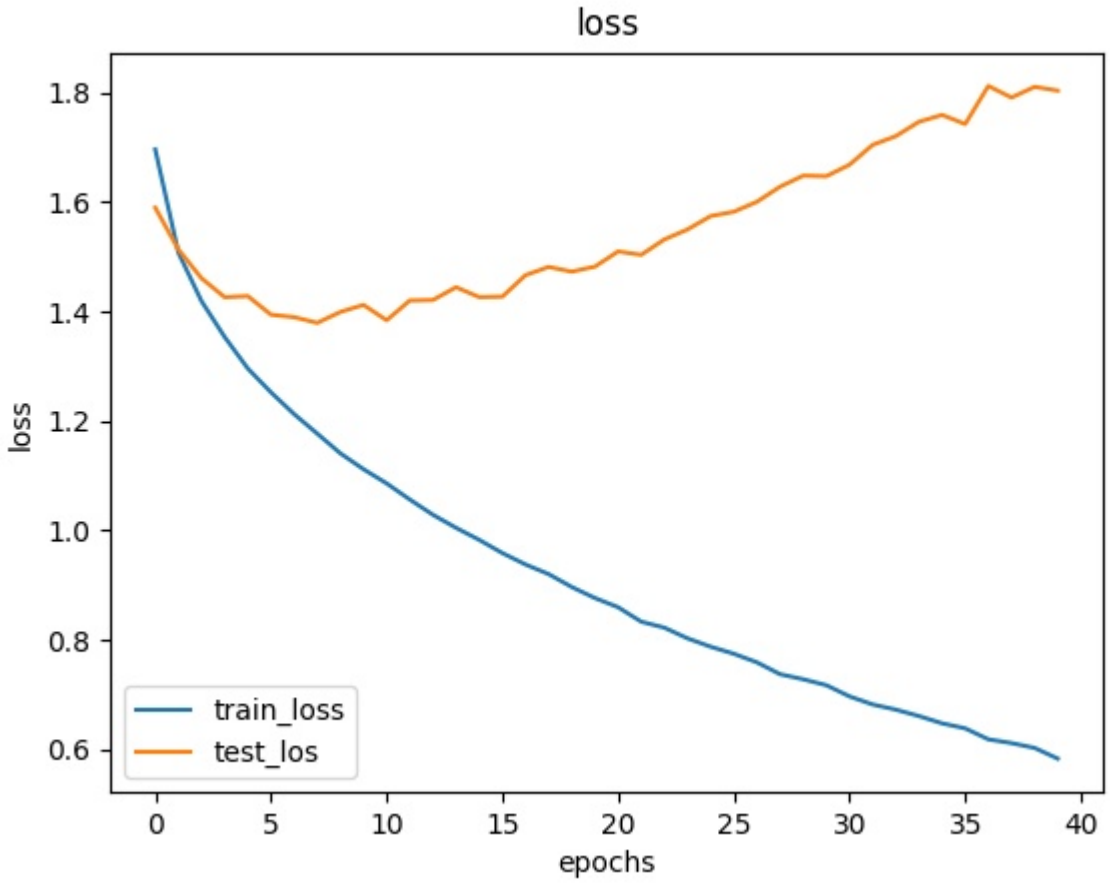
In both networks, skipping the batch-normalization layer makes a slight decrease in the final test accuracy. This may be because that the batch-normalization layer can normalize the data to a standard distribution, making the training process more stable and faster, and giving the model a better generalization ability.

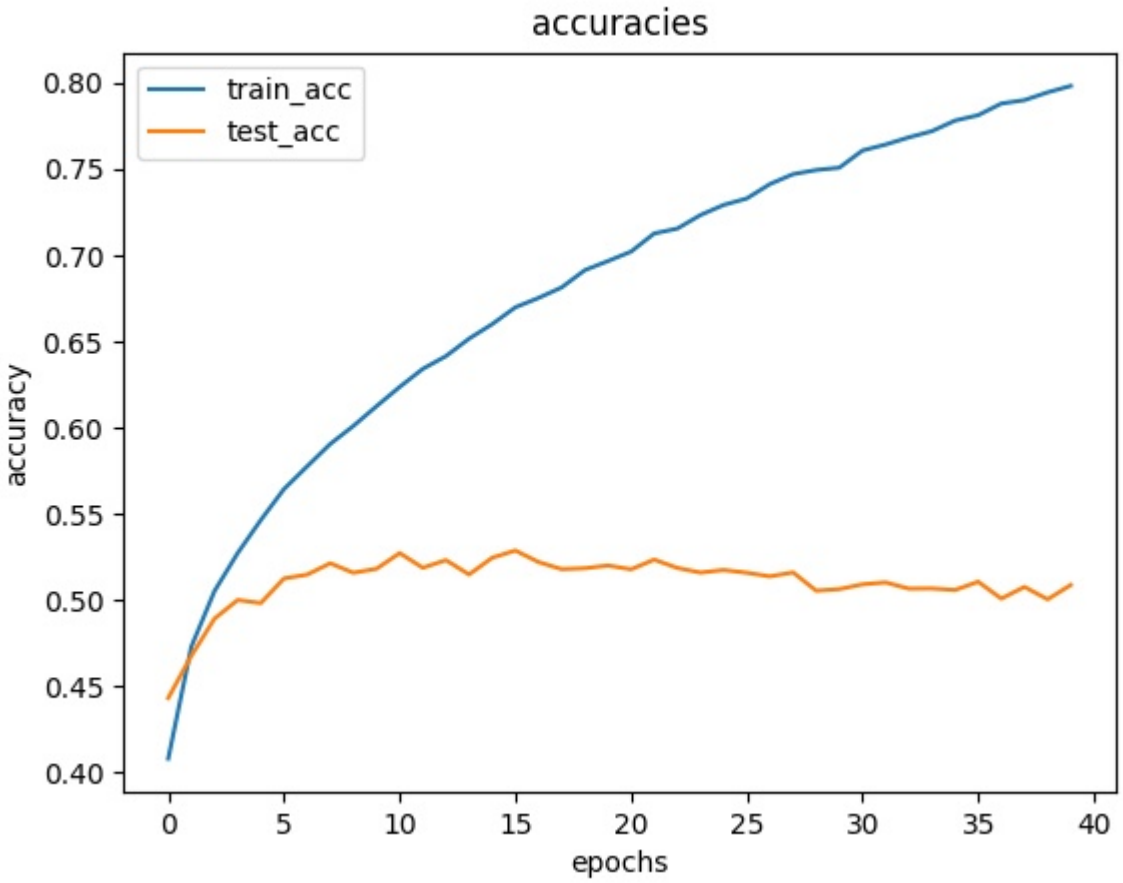
We can see that in the CNN network, when we make batch normalization, the curves of the validation loss and accuracy becomes unsmooth, which means that the validation process is not stable. This may be because that, in the validation process, the "mean" used in normalization is calculated with the running mean during training. This value may be different from the mean of the current validation data, which lead to a slight shift in the data input. Theoretically, as the training process goes on, the running mean will be closer to the mean of the whole dataset, and the shift will be smaller.

6. Dropout

I also tried to skip the dropout layer in both the MLP and CNN networks. The results are as follows (still the same hyperparameter):

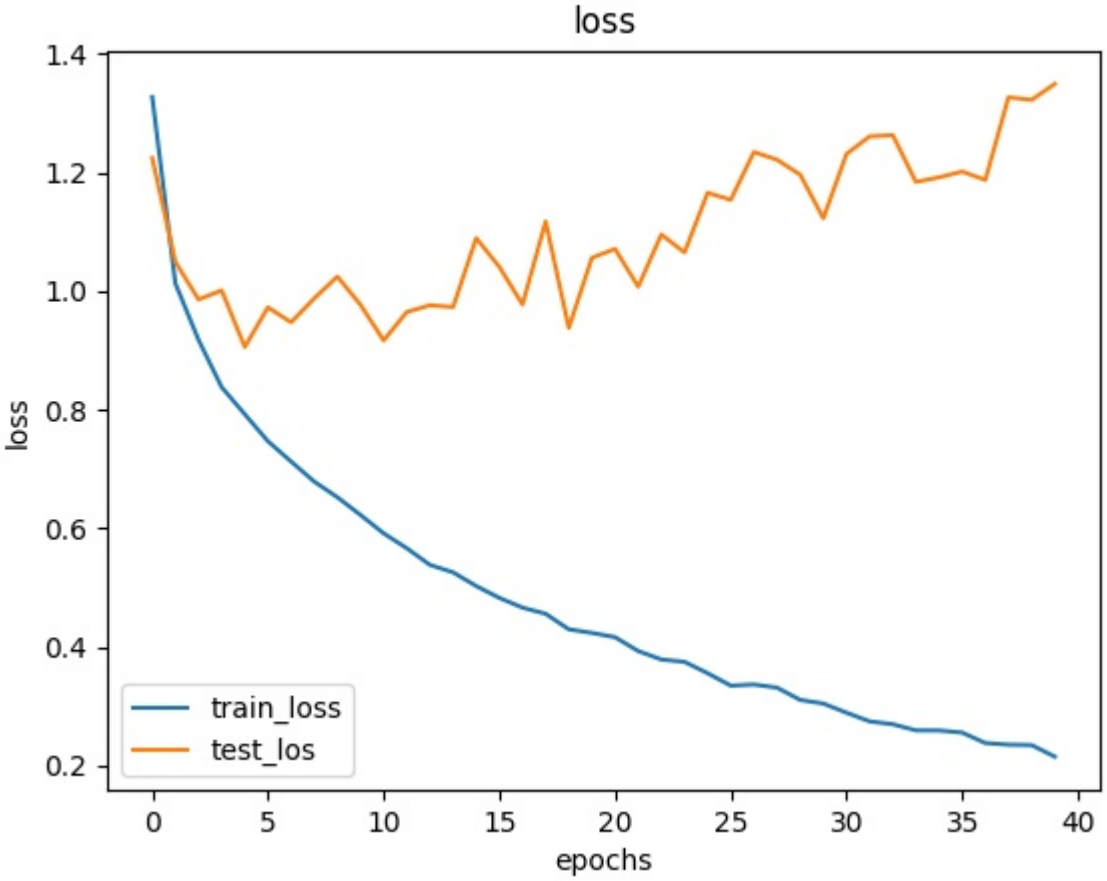
MLP

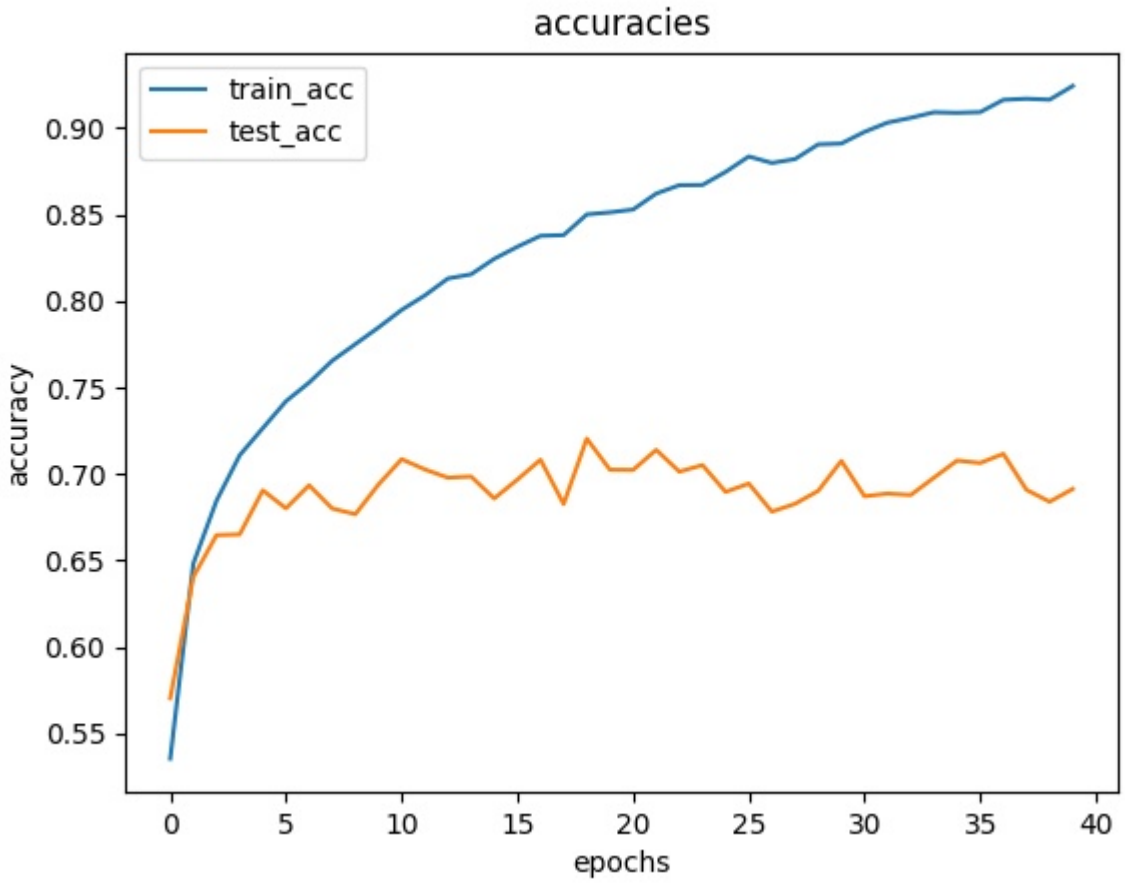





```
best epoch:                     16
best validation accuracy:       0.5288999885320663
test loss:                      1.4180605864524842
test accuracy:                  0.5208999860286713
```

CNN


loss
1.4
1.2
1.0
0.8
0.6
0.4
0.2
loss
0
5
10
15
20
25
30
35
40
epochs
train_loss
test_los

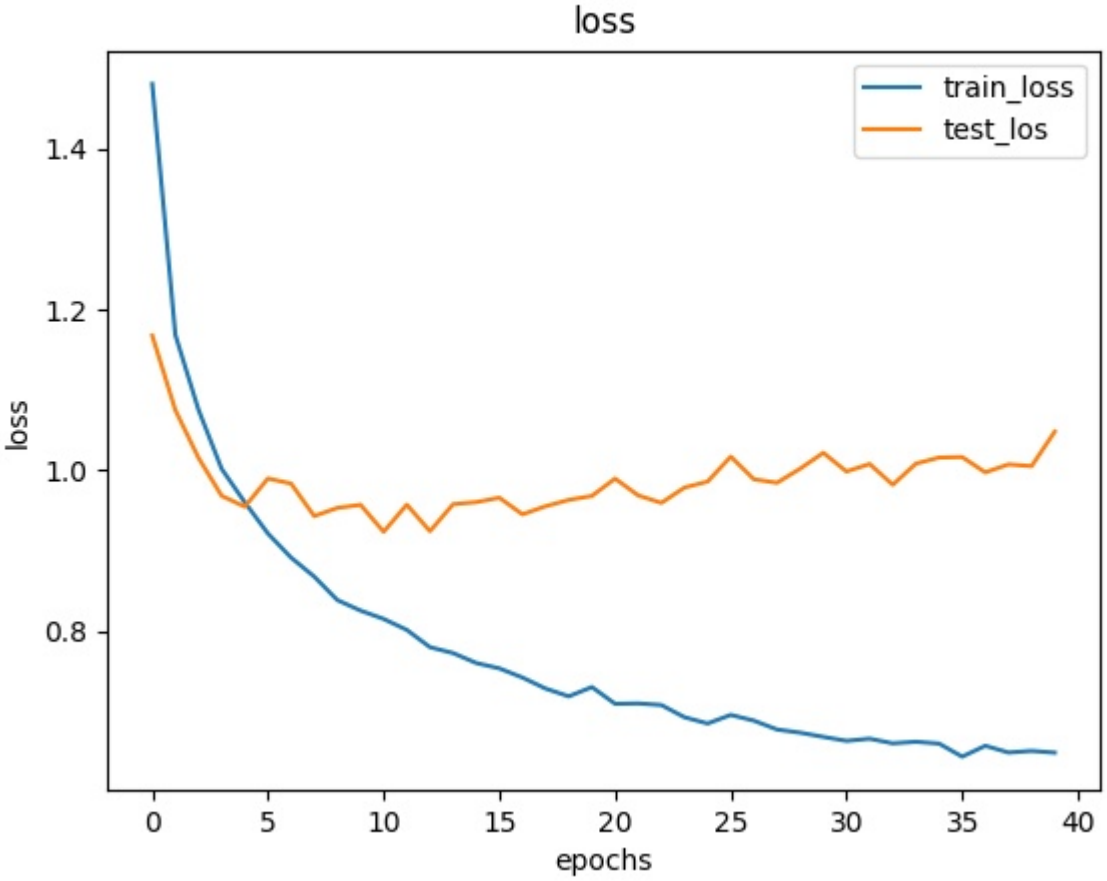


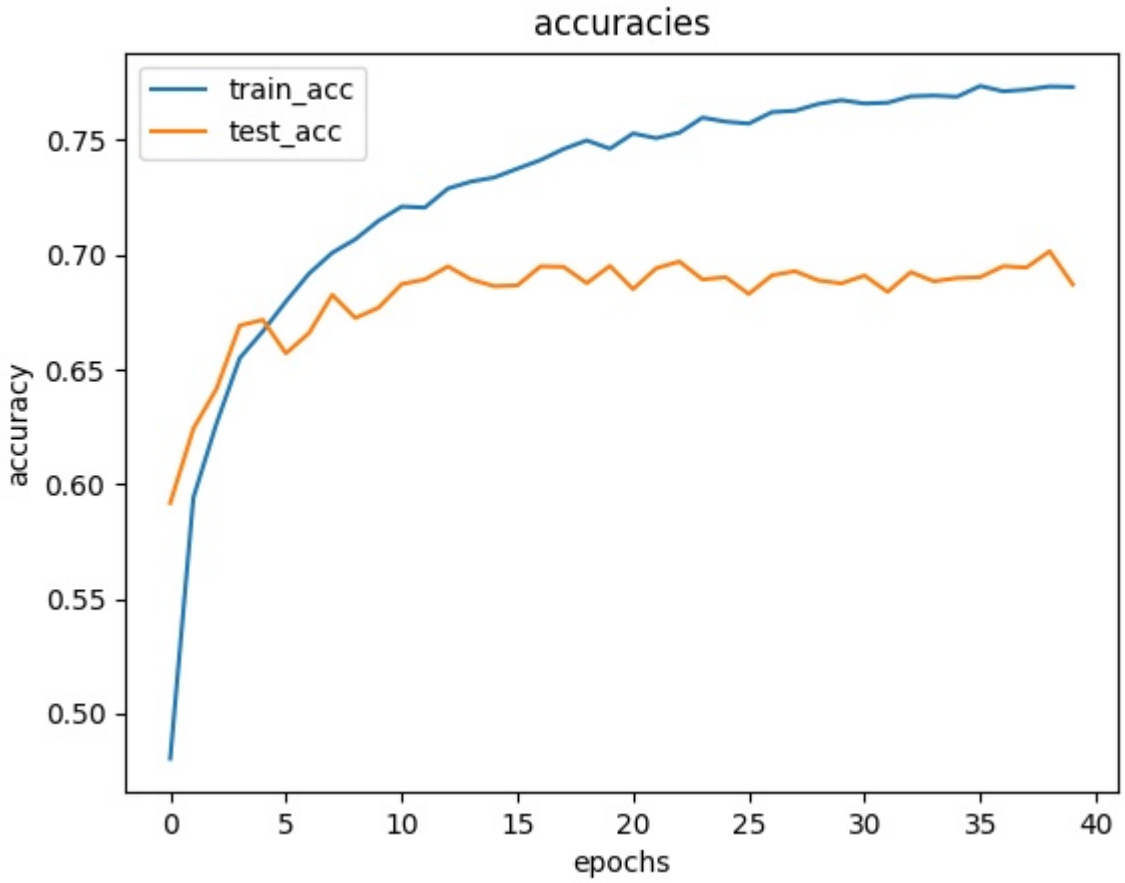

```
best epoch:                    19
best validation accuracy:      0.7204999852180481
test loss:                     0.9698055040836334
test accuracy:                 0.7148999828100204
```

In both networks, skipping the dropout layer causes severe overfitting. The validation loss begins to rise as the training iterations gets big enough. Meanwhile the validation accuracy reached its best before 20 epochs, which also indicates overfitting. Thus we can conclude that the dropout layer has a significant effect on preventing overfitting.

7. CNN Layer Order

I tried to change the order of the different layer in the CNN network. The current network is "input – Conv – BN – ReLU – Dropout – MaxPool – Conv – BN – ReLU – Dropout – MaxPool – Linear – loss", I exchanged the order of the "Dropout - MaxPool" part and the "BN - ReLU" part, so the network goes "input – Conv – Dropout – MaxPool – BN – ReLU – Conv – Dropout – MaxPool – BN – ReLU – Linear – loss". The result are as follows:


loss
train_loss
test_los
1.4
1.2
1.0
0.8
loss
0
5
10
15
20
25
30
35
40
epochs



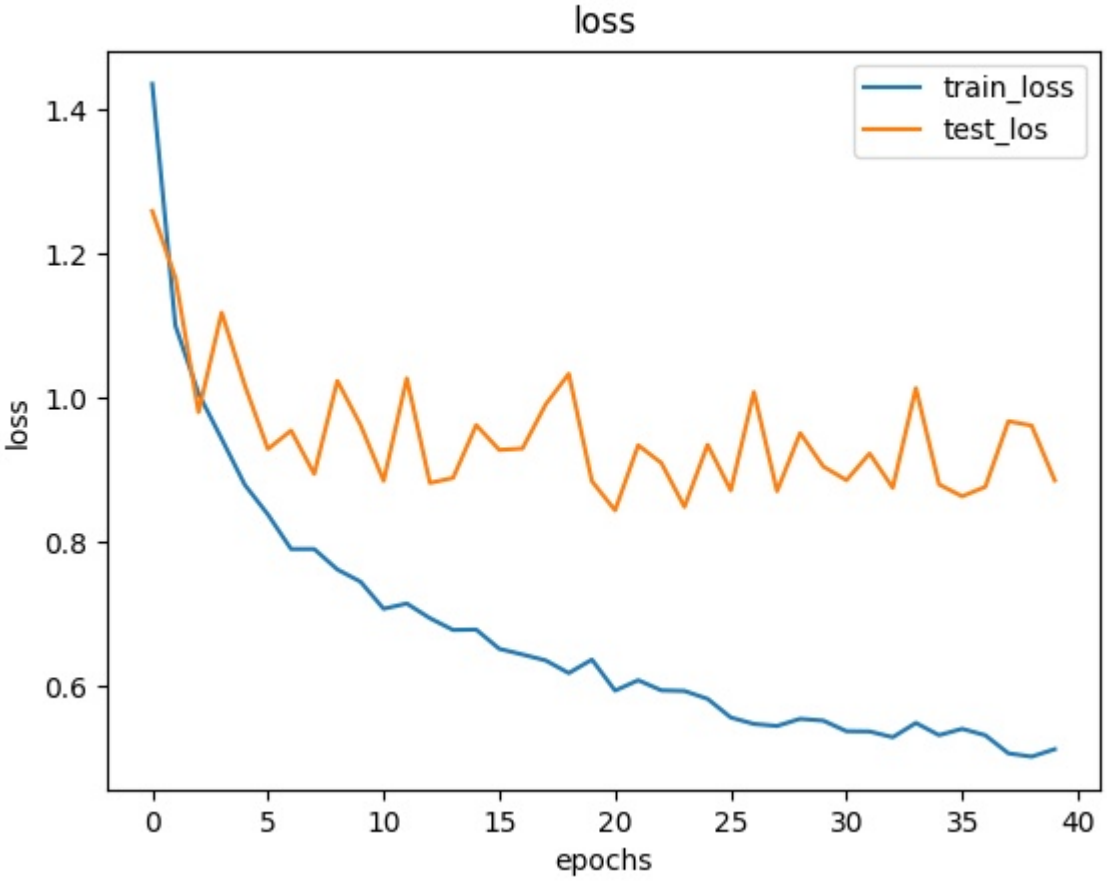

```
best epoch:                     39
best validation accuracy:       0.7014999824762345
test loss:                      1.0175276285409927
test accuracy:                  0.6958999872207642
```

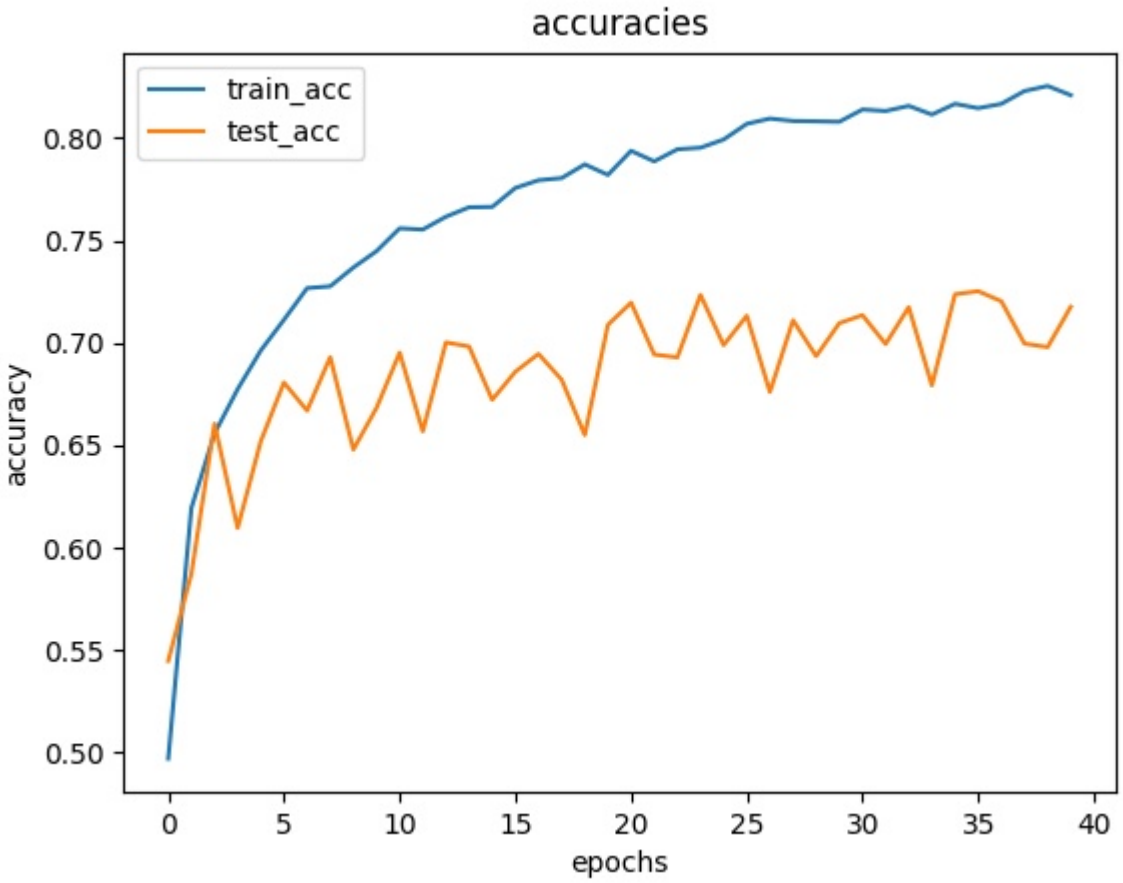
The final loss is higher and the accuracy is lower than the original network, but not too significantly. I guess the reduce in model ability may be due to the fact that the Non-Linear layer (ReLU) is placed after the after the MaxPool layer, which means the non-linear operation is performed on the down-sampled data, causing the representation ablility of the model to decrease.

However, the decrease in performance is not too significant, and by doing the non-linear operation on the down-sampled data, we can reduce the computational complexity of the model, which may be helpful in some cases.

8. Other Experiments

In the dropout layer of the CNN network, the layer is to randomly zero out entire channels rather than neurons. I also tried to to the dropout by randomly zero out neurons. The results are as follows:


loss
train_loss
test_los
1.4
1.2
1.0
0.8
0.6
loss
0
5
10
15
20
25
30
35
40
epochs



In fact, the result is almost the same as the original one, even a bit higher.