Machine Learning Exercise 2

Michail Mitsios cs2200011

June 2021

2.1 Training and Evaluation

NN

For the Simple Neural Network I followed this architecture:

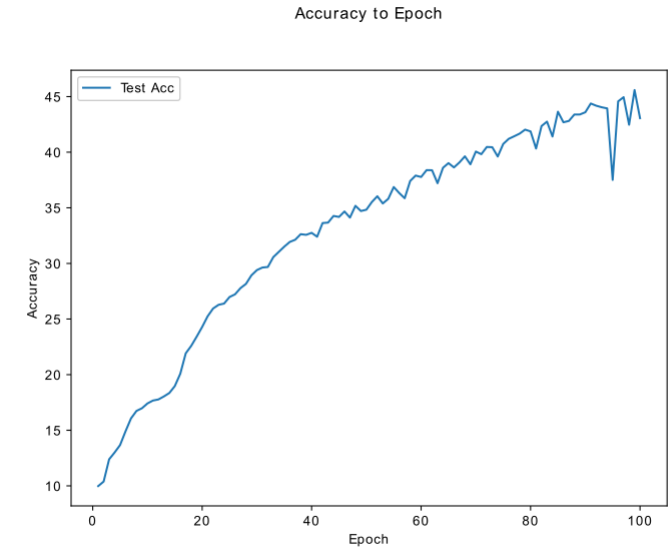
nn.Linear(32*32*3, 512),
nn.ReLU(),
nn.Linear(512, 512),
nn.ReLU(),
nn.Linear(512, 512),
nn.ReLU(),
nn.Linear(512, 512),
nn.ReLU(),
nn.Linear(512, 10)

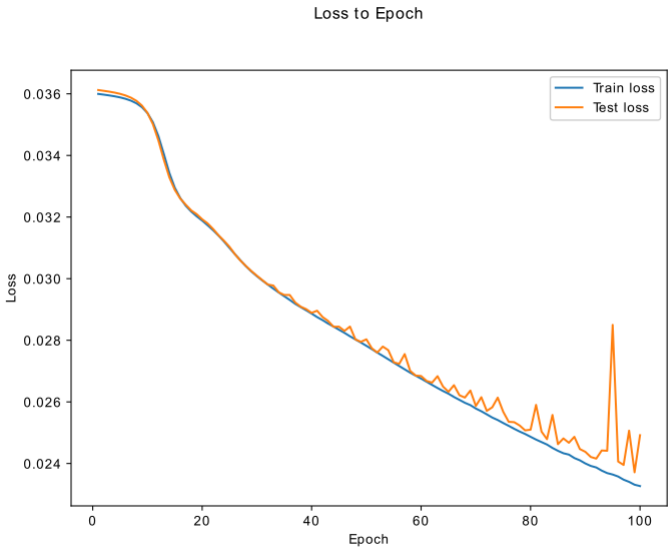
The Linear represents the layers of our NN. The ReLU is a function that normalize the negative values to zeros (and it is called to fix the outputs of each layer). The reason why I used 5 layers was because I thought that we do not take into account the first layer (because it is the input layer).

On the other hand, Pytorch ignores that theoretical rule and starts the calculations from its first layer. So I think that is wrong to have 5 nn.Linear for a 4 Layer NN. I should have 4 nn.Linear. Because I have already done the statistics and plots, I will leave this architecture choice as it is (I noticed it late after I had finished the running and the plots).

As for the optimizer I chose a small **learning rate 0.001**. I used this value in order to make small steps towards the best solution.

Epoch 100





From the sheets we can see that the generalization is good, because the test loss is close to the train loss. Also, the test loss is bigger than the train loss and this is a logical outcome because we use the train set for fitting our model. The NN is trained based on the train set and not the test set, so we don't expect higher loss to the test set than the train set. That's why the closer the test loss to our train loss is the better the results. In this case we can also say that our model has good generalization because test set is an unknown set for the model. The accuracy is not very high and this is why we are not making enough big steps, so we are away from the solution.

CNN

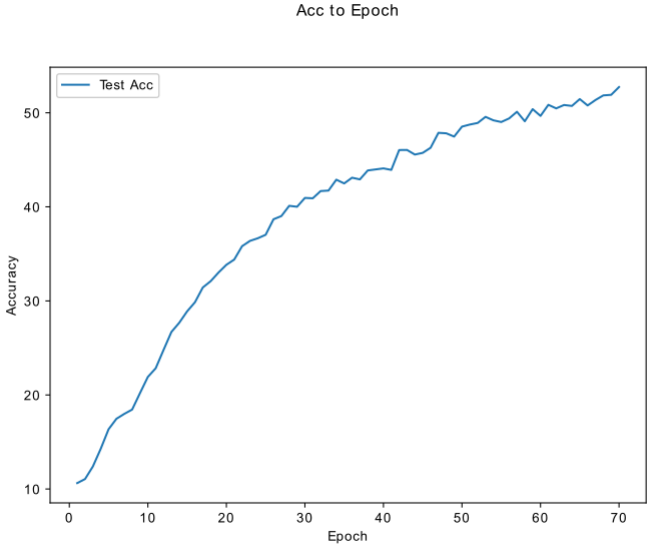
The architecture used in CNN is LeNet-5. The layer composition consists of 2 convolutional layers, 2 subsampling layers and 3 fully connected layers:

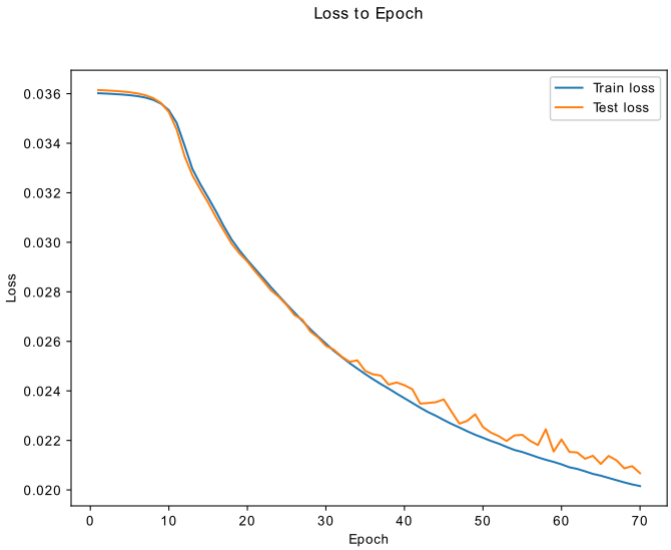
nn.Conv2d(3, 6, kernel_size = 5), #(32x32x3)->(28x28x6)
nn.ReLU(),
nn.MaxPool2d(2, stride = 2), #(30x30x6)->(14x14x6)
nn.Conv2d(6, 16, kernel_size = 5), #(14x14x6)->(10x10x16)
nn.ReLU(),
nn.MaxPool2d(2, stride = 2)) #(10x10x16)->(5x5x16)

---

nn.Linear(5*5*16, 512), nn.ReLU(),
nn.Linear(512, 512), nn.ReLU(),
nn.Linear(512, 10))

learning rate=0.001 and epoch=70





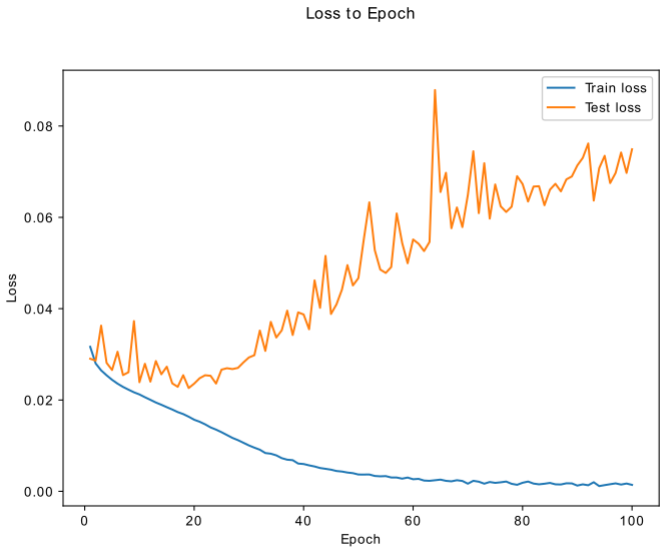
First of all as the simple NN the CNN test loss is greater than the train loss for the same reason.

Furthermore, the CNN is performing better than the simple NN when it comes to image recognition. This happens because the image pass through some filters first before entering the Fc model. In this way he can relate input objects (images) by generating features from them. As a result the CNN not only has the test loss lower but also the accuracy we achieve is higher than the accuracy of the simple NN.

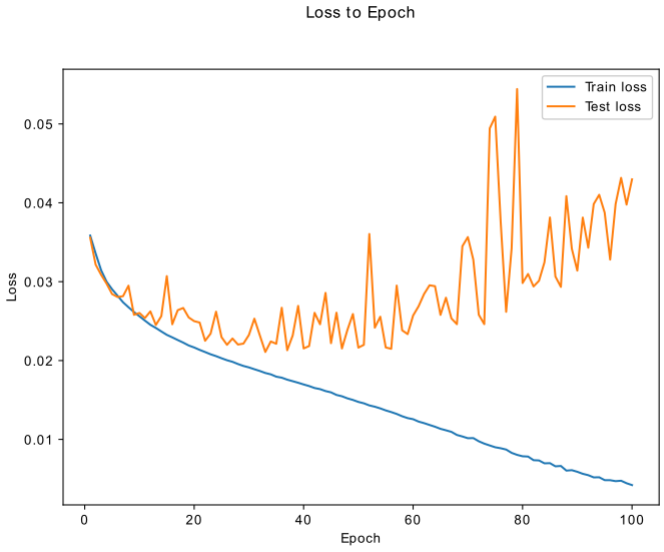
The CNN has better performance than the NN and in even less epoch (70).

2.2 Optimization

The 2 different values for learning rate are: 0.1 and 0.01.



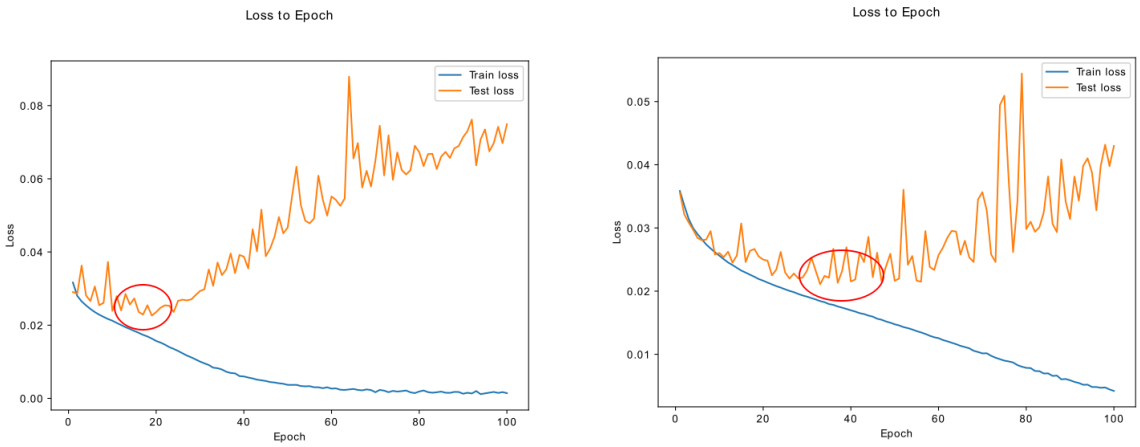
Learning Rate=0.1



Learning Rate=0.01

As we can see from the loss plots for both learning rates the test loss moves away from the train loss. This is happening because we are using a bigger learning rates than before and after an epoch we could say that in a way, we are overfitting our model. Also, we can see that we are going quicker to the convergence of the algorithm. In simple words the NN is looking for the point where the loss is the smallest possible one (a local minimum). The learning rate represents how big or small are the steps towards this optimal solution. If we

are making small steps we may not get enough close to the solution and if the steps are very big we are going quicker to the solution, but we may pass it. In this case we are forced to go back in order to get closer to the solution. **Note:** In my point of view the convergence is the best possible point we can reach (the smaller loss) and after this point we are not having big changes to loss.

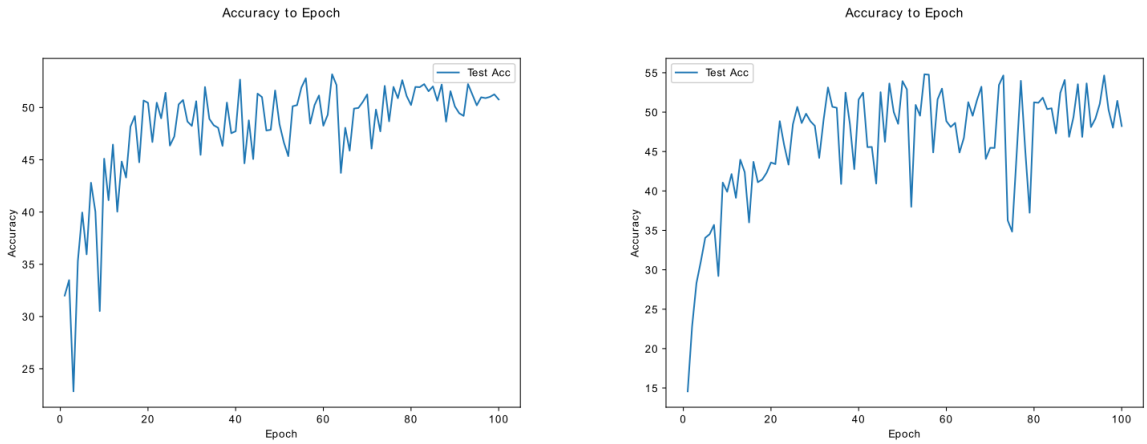


(a) NN Learning Rate=0.1

(b) NN Learning Rate-0.01

Figure 1: Convergence Points

The circles are the points where the NN is having convergence. We can see that the first diagram with learning rate 0.1 is having quicker convergence (20th epoch) than the second one (40th epoch), which has learning rate 0.01. We could say that the bigger the learning rate the quicker we will arrive to our convergence. Although the quicker convergence is achieved by the model with learning rate 0.1,we can see that the generalization is pretty bad (very high test loss). Even if we are having early convergence the test loss is very big comparing with lr=0.01.



(a) NN Learning Rate=0.1

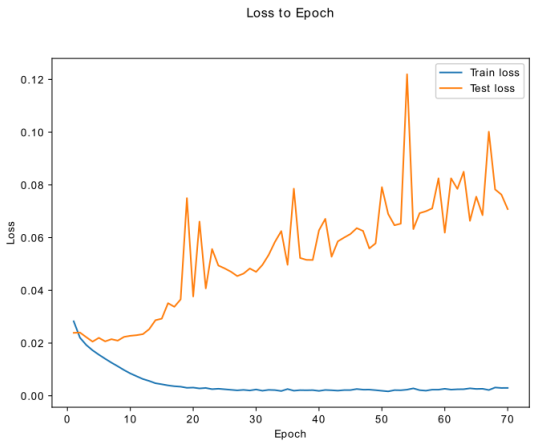
(b) NN Learning Rate=0.01

Figure 2: Test Accuracy

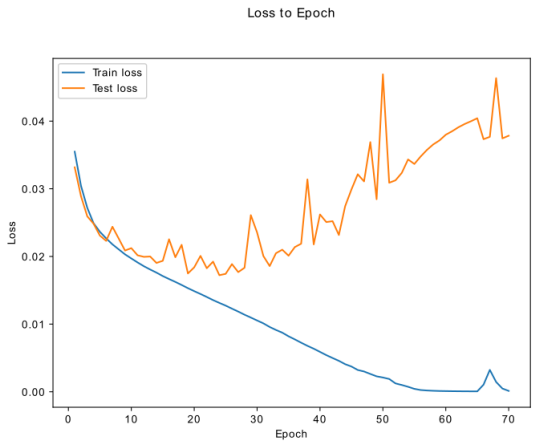
In addition, the accuracy of the 0.1 is more chaotic (ups and downs) than the one of 0.01. There are two values where the loss is very high but rather than that the accuracy and the loss is more consistent. We need more time to achieve our best accuracies but the results are better when the time comes and more stable. In conclusion, I choose as the "best" learning rate the 0.01 because it has better results and generalization.

The same experiment was conducted on CNN and the results were similar.

The test loss moves away from the train loss more when we are having bigger learning rate.

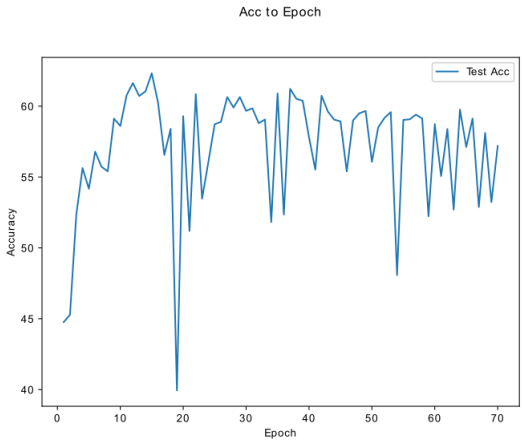


(a) Learning Rate=0.1

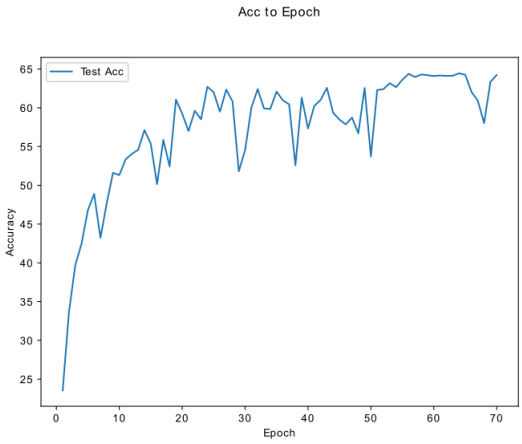


(b) Learning Rate=0.01

Figure 3: CNN loss



(a) Learning Rate=0.1



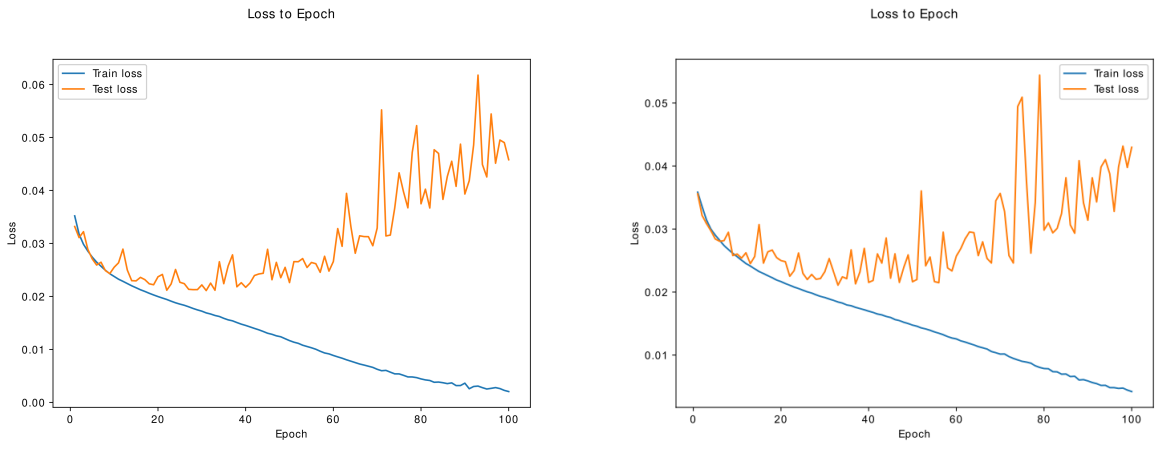
(b) Learning Rate=0.01

Figure 4: CNN Accuracy

Here it is more clear when it comes to the accuracy. The CNN with bigger learning rate has bigger steps and is more chaotic and inconsistent.

Momentum

The 2 different values I chose for momentum was **0.3** and **0.7**. The momentum is like an acceleration that is applied to our model. The faster our model goes the quicker we are expecting to reach its convergence point. This metric is used to avoid some points that are local minimum in order to find the global one. The more momentum the more the acceleration of our model.



(a) Momentum=0.3

(b) Momentum=0

Figure 5: NN Learning Rate=0.01

Another thing that we notice after applying the momentum to our model is that it smoothens our results, both accuracy and loss. This effect is more straightforward when we see an even bigger value.

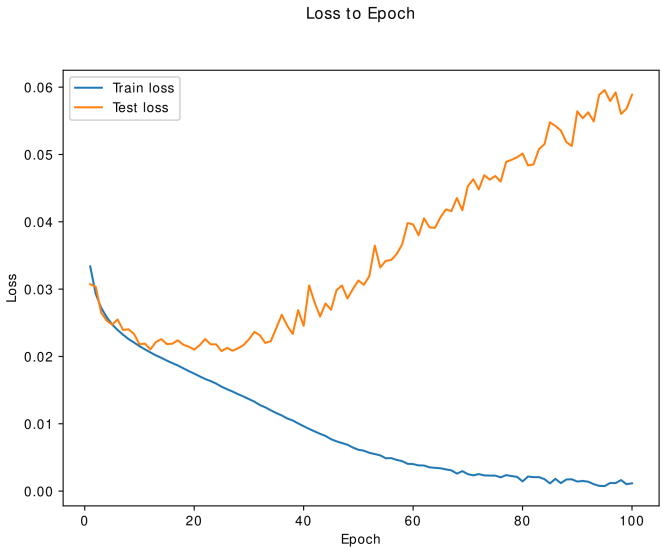
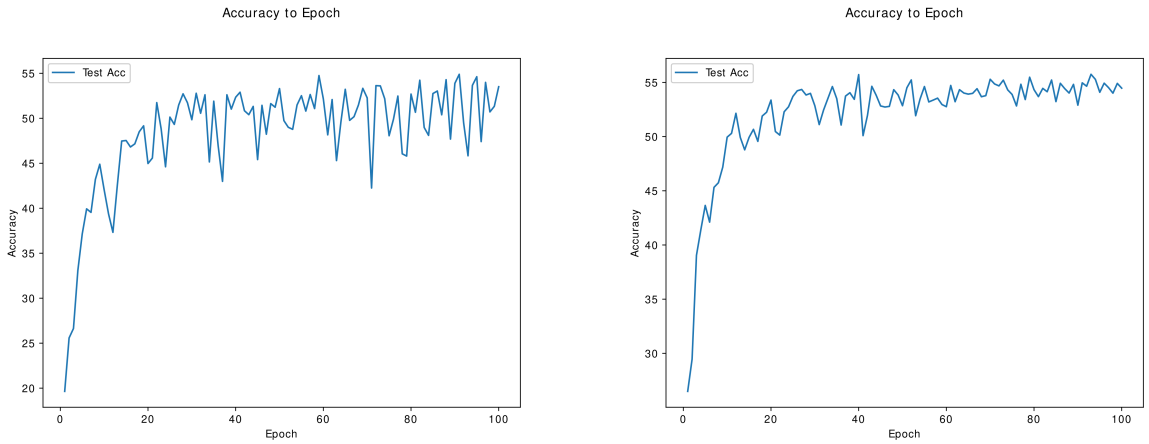


Figure 6: NN Learning rate=0.01 momentum=0.7

The bigger momentum also makes our model to reach its convergence much faster than the previous versions.

The behavior of accuracy is corresponding to the behavior of the loss. In a few words earlier convergence and smoother results.



(a) Momentum=0.3

(b) Momentum=0.7

Figure 7: NN Accuracy /Learning Rate=0.01

A last observation I want to make is that when we have reached the epoch where the test error is going upwards, the momentum (because it adds an acceleration to our model) it will affect negatively the test loss and will make it bigger in less epoch.

Conclusions: Having said that in order to choose the best possible parameter for a NN I would follow this steps:

1. Define a small enough learning rate and print the plots of loss to see that the test loss is close to the train loss (have a good generalization).
2. Then make the learning rate bigger by small steps each time. For every new lr I will check the results to avoid any divergent behavior.
3. After have found the best possible lr I will apply test different values for the momentum and see the corresponding accuracy results.

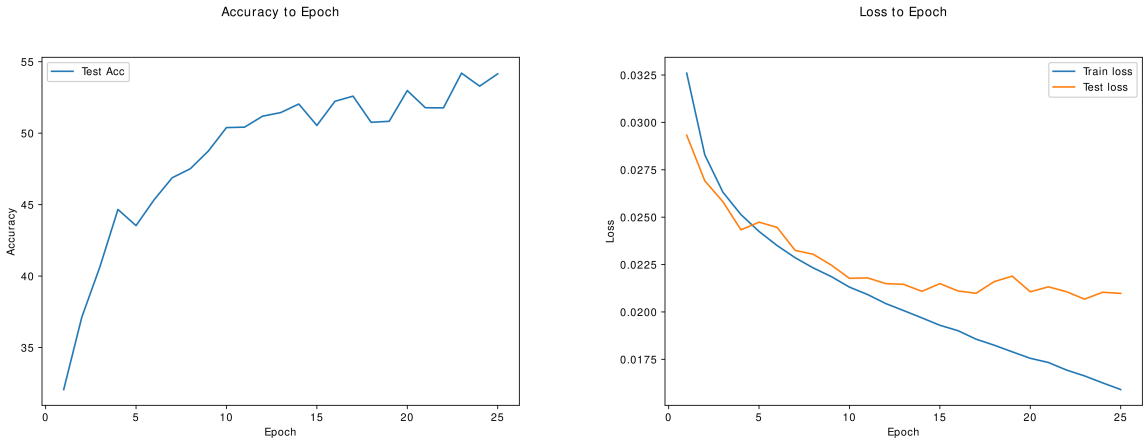
The above steps could be done with the help of cross-validation.

2.3 Model Architecture

NN 1

Number of hidden units	learning rate	epoch	max Accuracy
256	0.005	25	54.4%

I had to lower the epoch to the point where I first met divergence. This is why I have 20 epoch for this model.



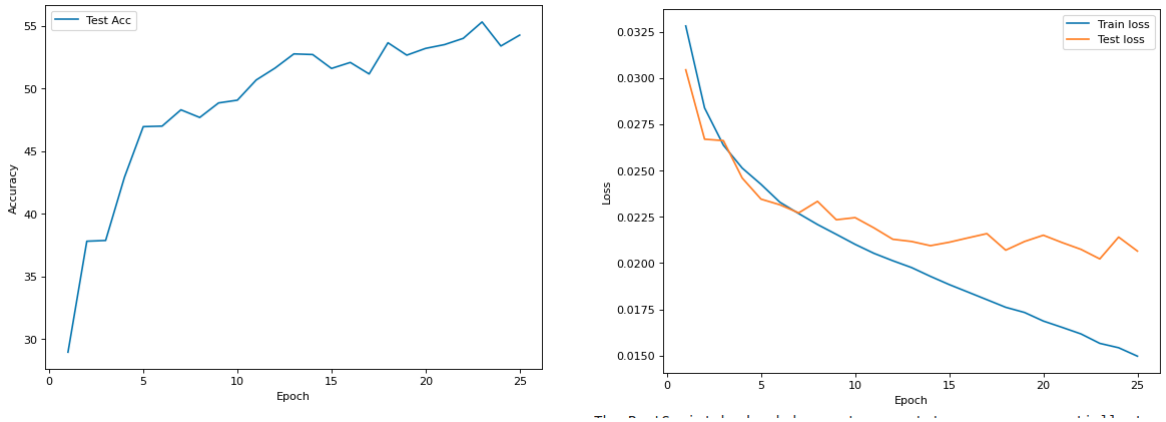
(a) Accuracy

(b) Loss

Figure 8: NN1

NN 2

Number of hidden units	learning rate	epoch	max Accuracy
400	0.005	25	55%



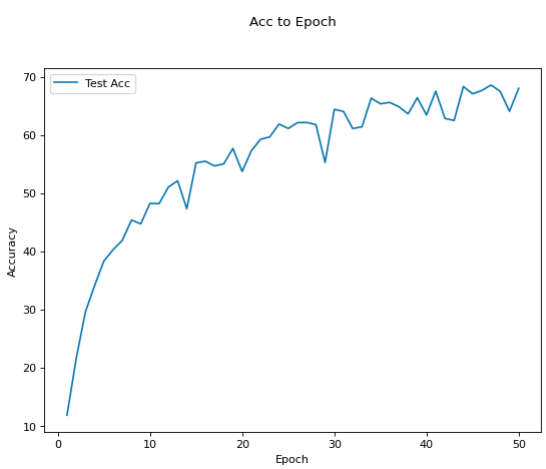
(a) Accuracy (b) Loss

Figure 9: NN2

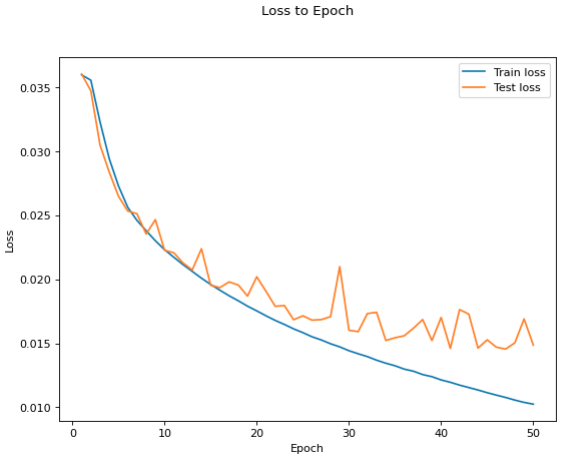
Taking a closer look at the plots we can clearly see that the convergence of the NN2 comes earlier than the one of NN2. Also, the NN2 achieves slightly better accuracy(55%) than the NN1(less than 55%). This means that the generalization of the network also becomes better, which makes sense because we are having a more flexible and complicated model.

CNN1

Number of filters	learning rate	epoch	max Accuracy
Constant 24	0.005	50	68.6%



(a) Accuracy

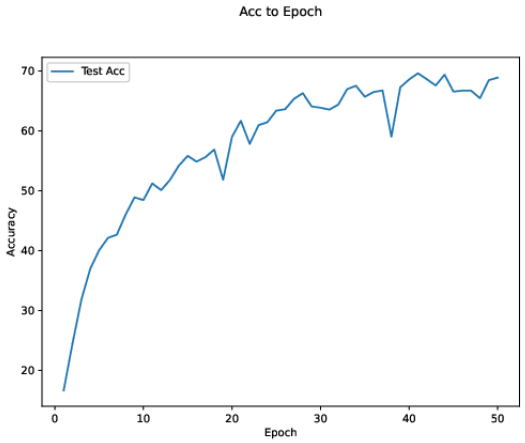


(b) Loss

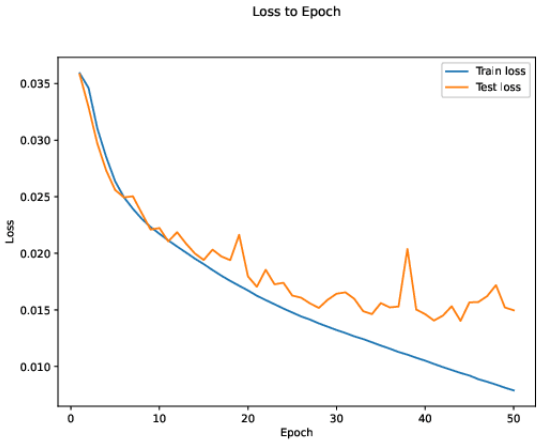
Figure 10: CNN1

CNN2

Number of filters	learning rate	epoch	max Accuracy
3->32,32->64	0.005	50	69.6%



(a) Accuracy



(b) Loss

Figure 11: CNN2

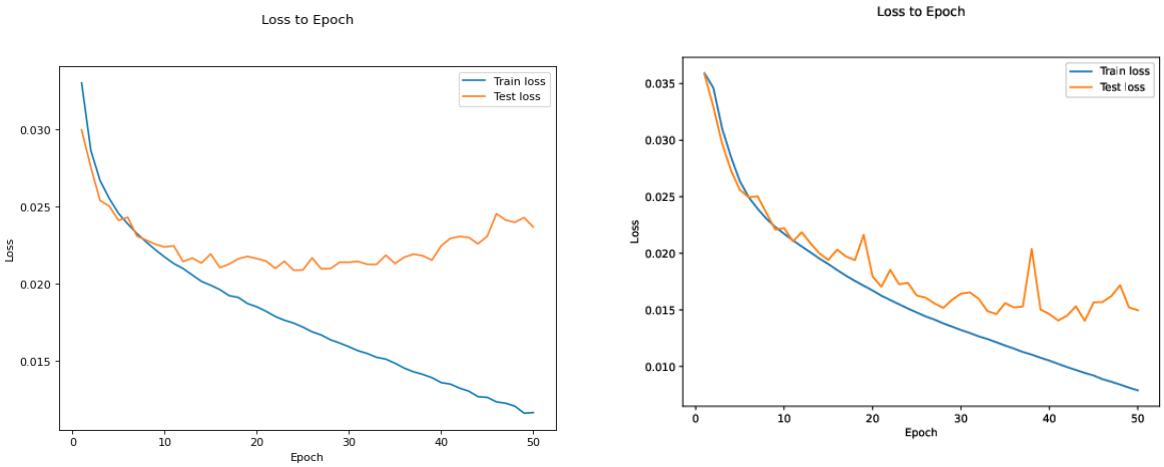
Now from the above results we can conclude that having more filters improves the accuracy. Having more filters means we recognize more features, and

we have more options to find relations between the images. Also I think we are getting to convergence quicker, CNN1 converge around 40 and CNN2 converge around 35. But having more filters made the model to train longer.

2.4 FC Networks vs CNNs

NN Type	Layers	total parameters
CNN	CNN2	531914
NN	150 constant	530410

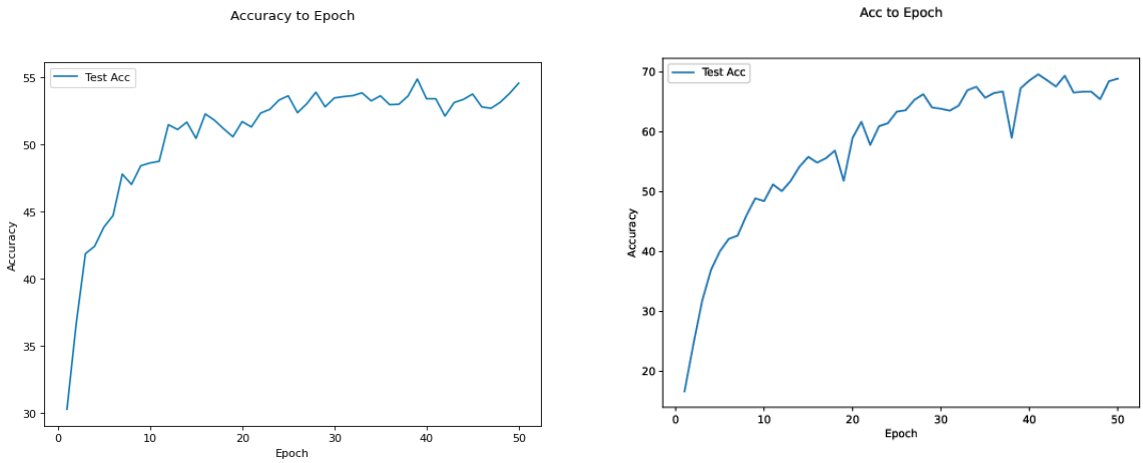
Based on all the above, we expect to see a better generalization on CNN. This happens because the NN has less number of hidden units than the NN1 or NN2 and based on one of our previous conclusions we know that the more the number of hidden units the more general and flexible our model will eventually be. **Note**: The learning rate and momentum are the same (lr=0.005, momentum=0.9).



(a) NN-150 (b) CNN2

Figure 12: Loss-Comparison

The diagrams above confirm our claim. As we have mentioned earlier the closer the test and train loss are the better for our model. If they are close it means that the train and test loss are similar-close. This means that the model which is trained with a set A can predict a set B as good as predicting A. Because B is an unknown set we can say that our model has a good generalization.



(a) NN-150 (b) CNN2

Figure 13: Accuracy-Comrarison

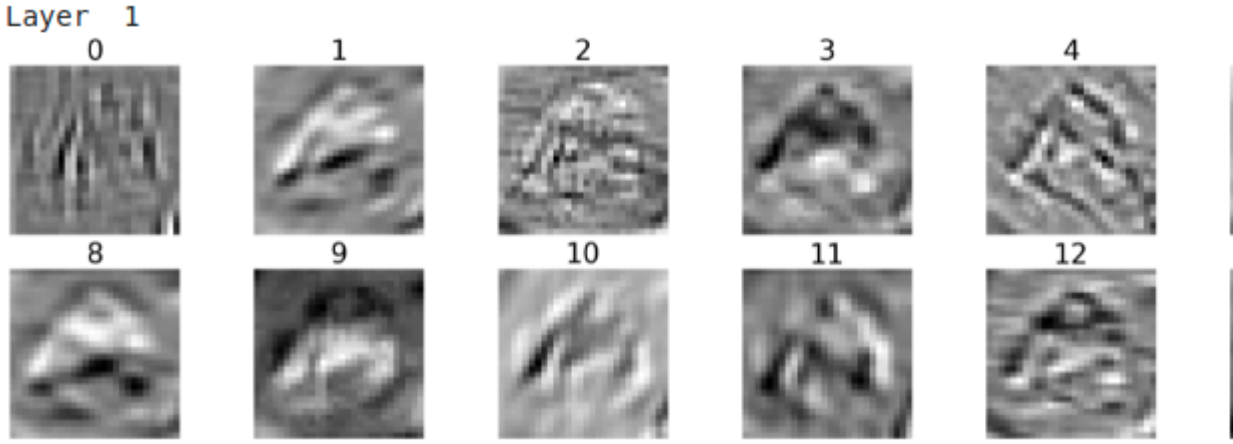
Even though the convergence is slower in CNN2 the results and the generalization is way better.

Plot the first layer output of the CNN2

The **CNN2** used for the accomplishment of this task. For this part I used the first picture of the trainset which is a frog.

The CNN will apply on this image 32 different filters on the first layer. In this way he can identify-produce many features for each image. Later on the process it will be easier to compare the features of the image and classify the input-data.

The output results:



The filters are trying to reveal different characteristics of the image such as shapes, edges, textures etc.

For example Filter#2 is trying to reveal a type of texture for the image. The Filter#5 is trying to form a shape on the image and the Filter#0 is trying to identify some edges on the frog. All these filters combined together will eventually create some good features that later on the NN will use them to classify the images.

Notes

- Before running each CNN or NN we make sure that we have inserted the correct type of data. Also before each run we have to define the train_loop and test_loop which are located inside the **NN** section.
- The diagrams wont passed with the final derivable because they are too many. If something is not very clear on the report please contact with me in order to provide the original Sheet.