ANN:DCNN

Quentin De Haes

March 2021

1 Introduction

In this paper we will show and discuss the results received for the various training variations done for the practical session Convolutional Neural Networks for the course of artificial neural networks.

2 Training a Deep Network from Scratch

2.1 No normalisation or regularization

For the first model we take an untrained model and train it with raw unchanged training data. This has been done multiple times and below are some of the different results received.

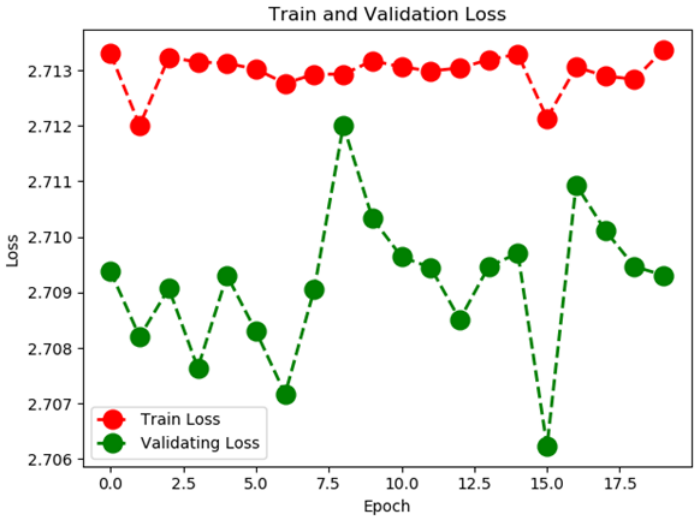

Figure 1: Untrained model Without any data editing 1

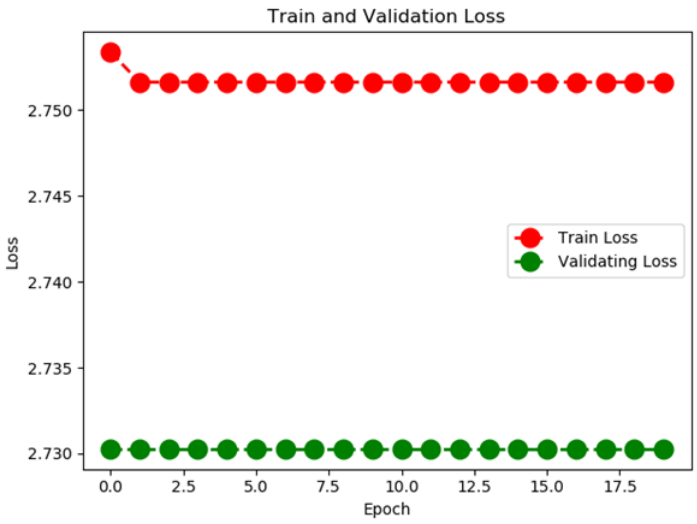

Figure 2: Untrained model Without any data editing 2

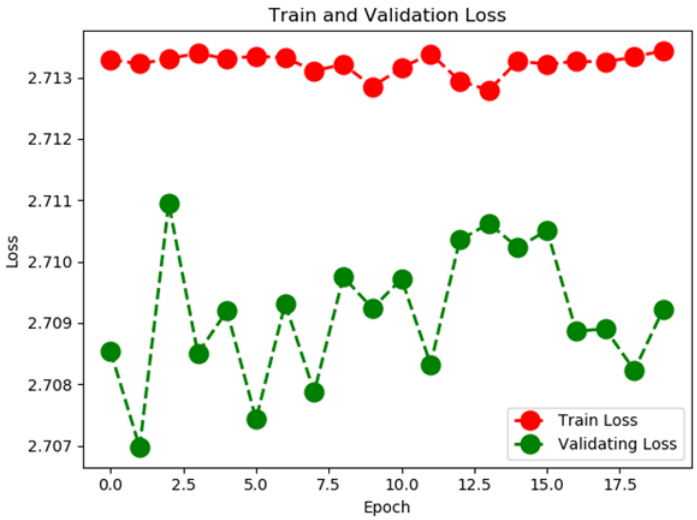

Figure 3: Untrained model Without any data editing 3

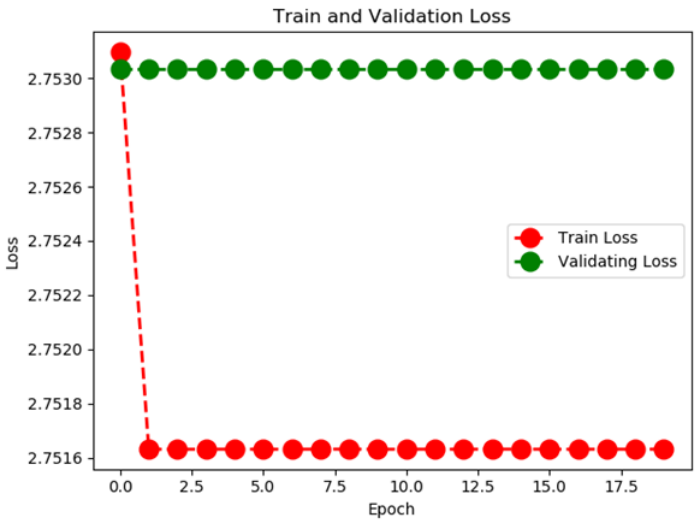

Figure 4: Untrained model Without any data editing 4

As can be seen, the result vary somewhat from run to run, this is because of the random initialisation that is done at the beginning of the model, so this will need to be remembered throughout this comparison, as minimal improvements could simply be because of a lucky initial state.

2.2 Transformations

We train the untrained network on data that has been transformed by us, but the transformations are done without much reasoning behind why these specific transformations should be done in preference over others.

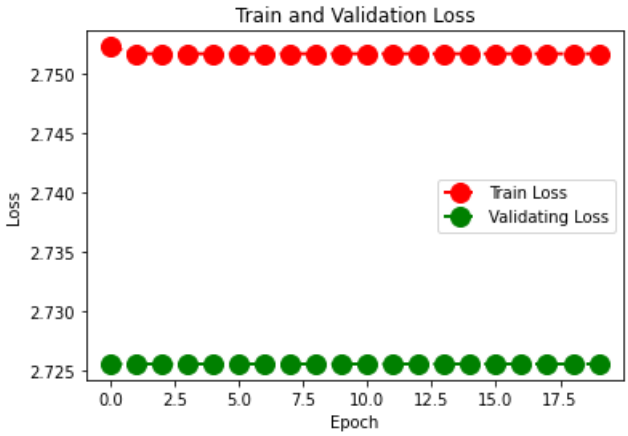

Figure 5: Untrained model With data transformation

by these results we can conclude that the transformations done have little effect, as this graph is near identical to one of the graphs received without the usage of any transformations

2.3 Normalisation

We train the network on data that has been normalised to ensure all data given to the network is relatively similar to one another.

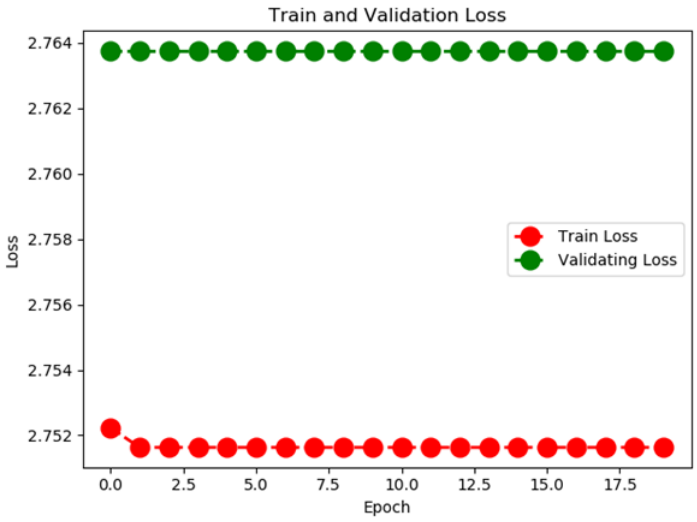

Figure 6: Untrained model With data normalisation

This actually seems to have slightly worse results than not doing anything to the data.

2.4 Using Dropout

Afterward, we try to add dropout to the model to reduce overfitting.

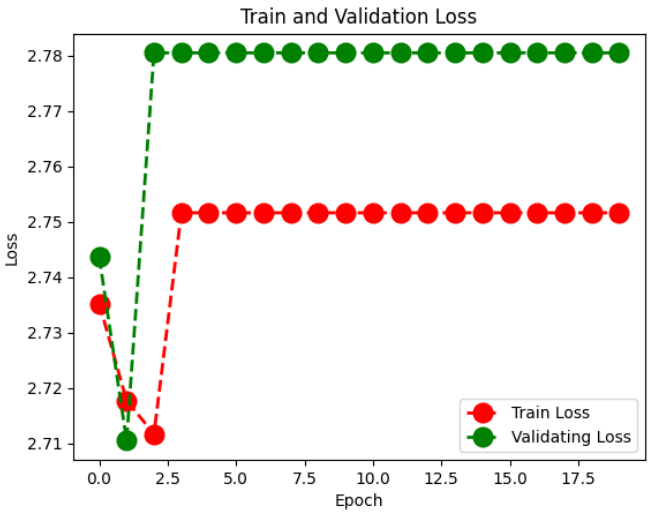

Figure 7: Untrained model With dropout

this change also seems to make no impact on improving the learning rate

3 Transfer learning

Next we try to adapt the pre-trained model VGG16 to give proper answers for our training and validation data. We freeze various layers and see whether the training of the remaining layers creates good results.

3.1 Freeze first 15 feature layers

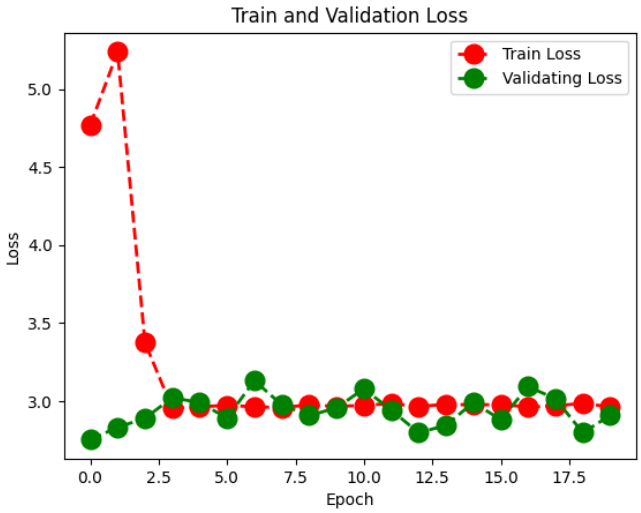

Figure 8: VGG16 with first 15 feature layers frozen

This seems to give worse results than the untrained model.

3.2 Freeze second 15 feature layers

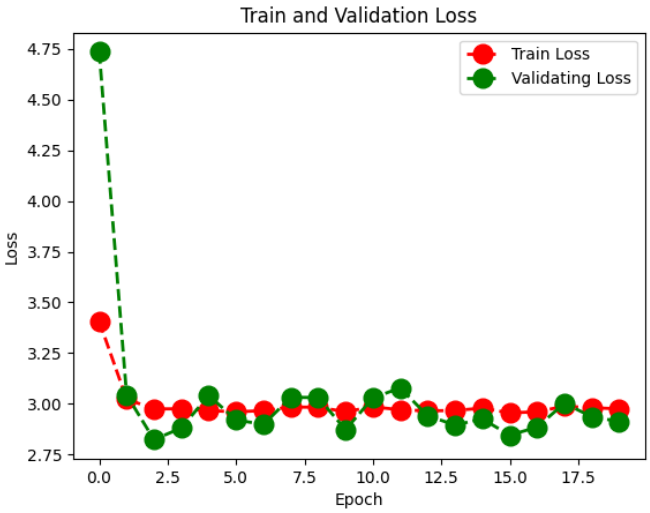

Figure 9: VGG16 with second 15 feature layers frozen

This has similar results as when freezing the first 15 feature layers.

3.3 Freeze all feature layers

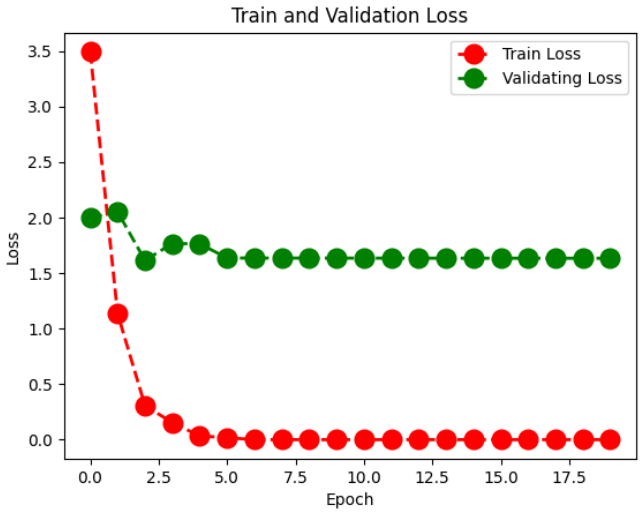

Figure 10: VGG16 with all 30 feature layers frozen

While this is the best model we have currently. found, it seems that this model suffers from overfitting, since the trainloss goes to 0, while the validation loss does not improve after the fifth epoch.

3.4 Freeze all classifier layers

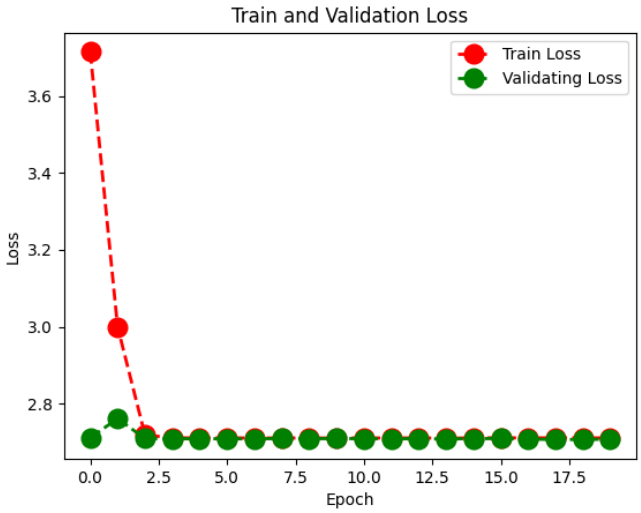

Figure 11: VGG16 with all classifier layers frozen

This quickly reaches results similar to when using an untrained model and then remains on that line similar to many others.

3.5 Train entire model

This time we don't freeze any layers at all

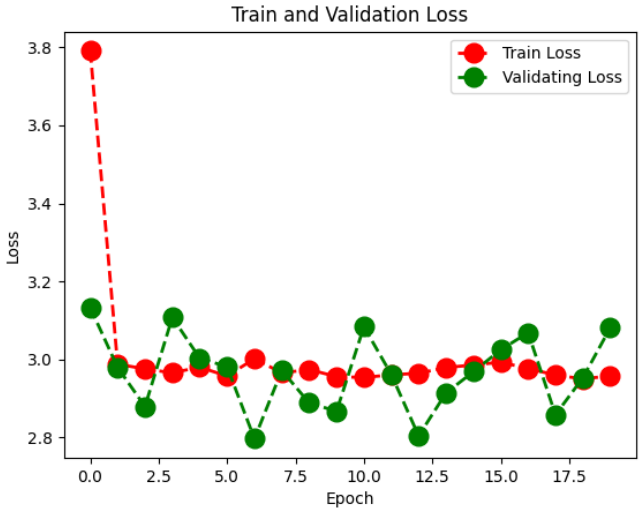

Figure 12: VGG16 with no layers frozen

The results here are similar in value as to when 15 feature layers have been frozen.

3.6 various batch sizes

When having a smaller training dataset, it tends to be a good idea to use larger batches to train your data. So below we'll try a number of different batch sizes, due to the time taken to run these model trainings, it will only be done for the model where no layers are frozen, and we'll assume extrapolation to the other variation is possible.

3.6.1 batchsize 2

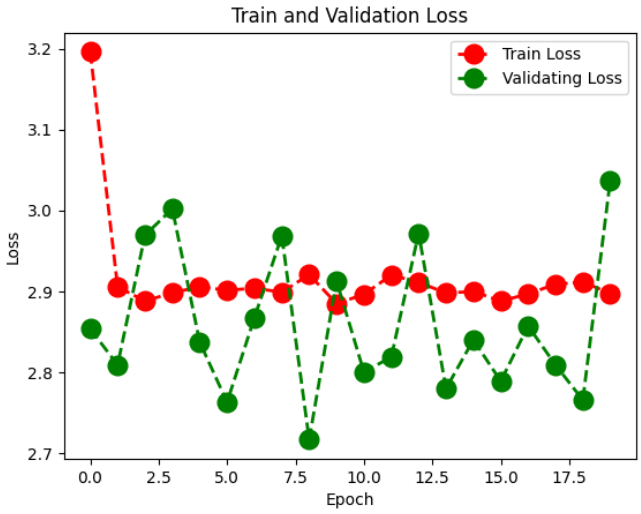

Figure 13: VGG16 with no layers frozen and batchsize 2

This is slightly better than using a batchsize of 1, the improvement is not significantly large however

3.6.2 batchsize 6

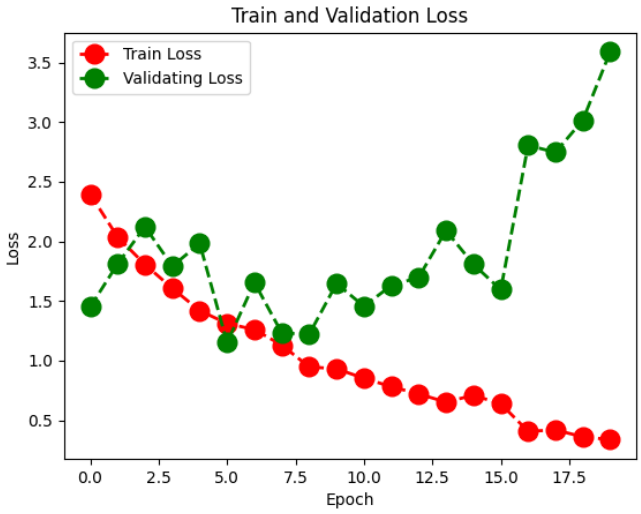

Figure 14: VGG16 with no layers frozen and batchsize 6

Here the training loss decreases to near 0 but the validation loss actually increases over the epochs, so this clearly overfits the model.

3.6.3 batchsize 10

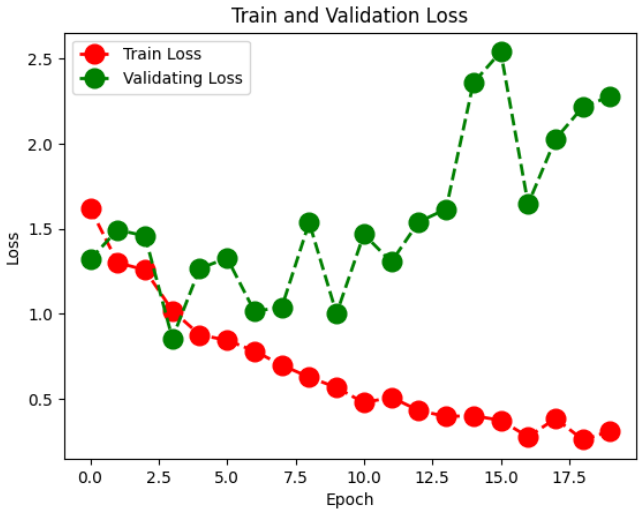

Figure 15: VGG16 with no layers frozen and batchsize 10

The results are very similar to when the batchsize was 6, not much has changed.

4 Conclusion

In most if not all cases, the model seems to get stuck in some local optimum after some of the first epochs. The used learning rate of 0.2 seems to not allow the model to escape these local optima in preference of a better set of optima. So below we will rerun all models but with a learning rate of 0.002

5 Training a Deep Network from Scratch with a learning rate of 0.002

5.1 No normalisation or regularization

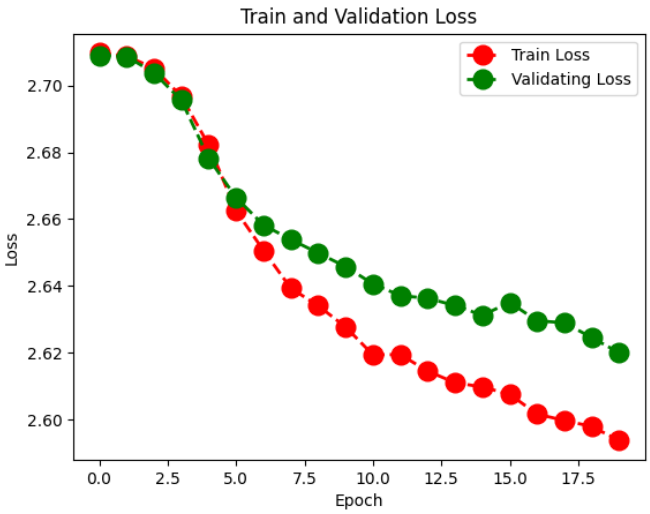

Figure 16: Untrained model Without any data editing LR=0.002

This lower learning rate is already superior even with an untrained model and no data editing. Since the loss for both training and validation has a downward trend continuing over the 20 epochs, unlike previously, where the loss remained nearly identical throughout most of the epochs.

5.2 Transformations

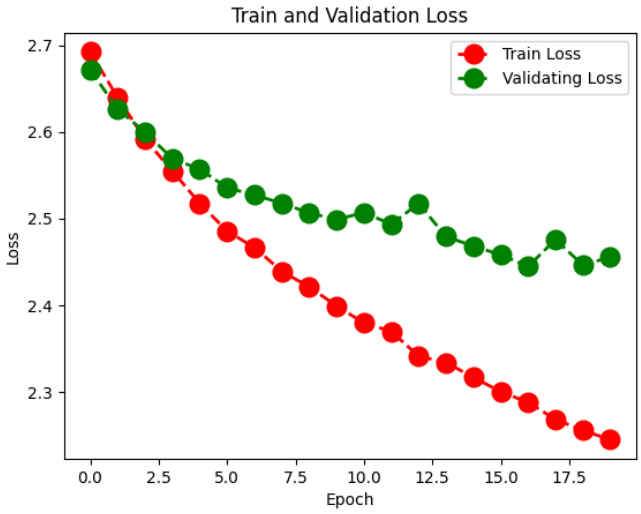

Figure 17: Untrained model With data transformation LR=0.002

These results seem better than when no editing was done to the input data as the loss during the final epoch is less than before, the current trend does make it seem as if the validation loss will soon reach a plateau however.

5.3 Normalisation

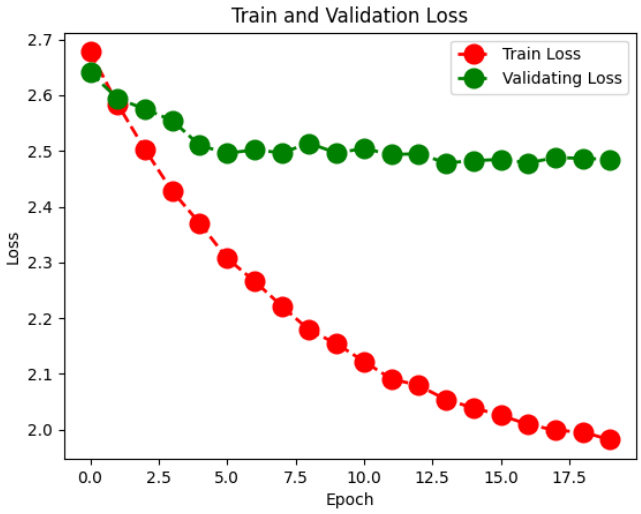

Figure 18: Untrained model With data normalisation LR=0.002

Where the validation loss seems to plateau to around 2.5 very quickly, getting results similar as to when transformations were done to the data, the train loss seems to be going down even quicker, getting us a very overfitted model.

5.4 Using Dropout

To reduce the overfitting, we introduce dropout to the model.

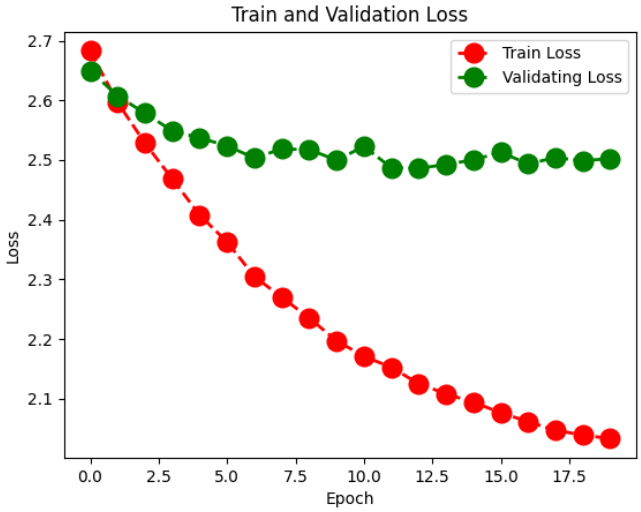

Figure 19: Untrained model With dropout LR=0.002

Since the results are nearly identical to when no dropout was used, we can assume this introduction had little to no effect.

6 Transfer learning

Next we try to adapt the pre-trained model VGG16 but with a Learning rate of 0.002 instead of 0.2

6.1 Freeze first 15 feature layers

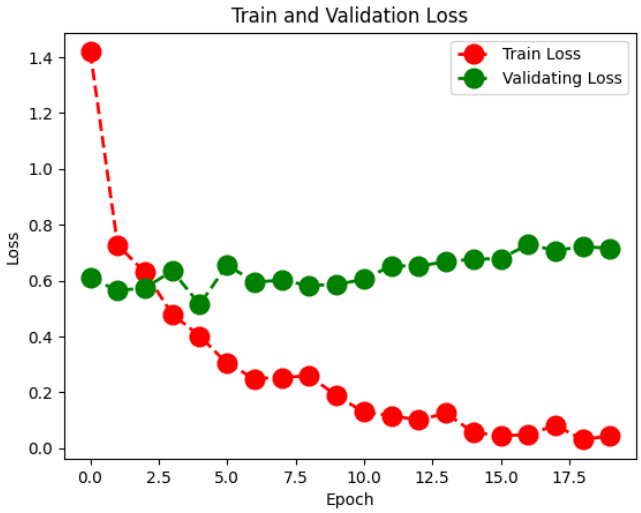

Figure 20: VGG16 with first 15 feature layers frozen LR=0.002

Where the training loss quickly dives to approximate 0 in the later epochs, the validation loss ends up slightly increasing over the varying epochs, however, the loss value of 0.6 is already a significant improvement over the losses we acquired in the untrained model.

6.2 Freeze second 15 feature layers

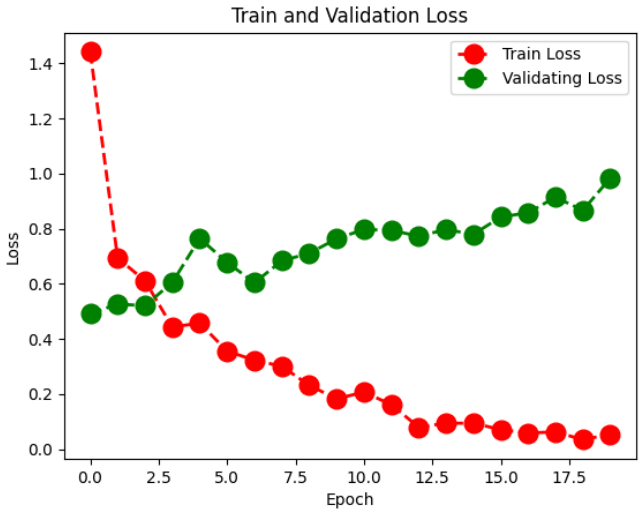

Figure 21: VGG16 with second 15 feature layers frozen LR=0.002

Whereas the train loss is almost identical to the train loss received when freezing the first 15 feature layers. The validation loss has a larger rising incline reaching a loss of around 0.8 near the the final epochs. This is still better than the validation loss achieved with an untrained model.

6.3 Freeze all feature layers

freeze all 30 feature layers.

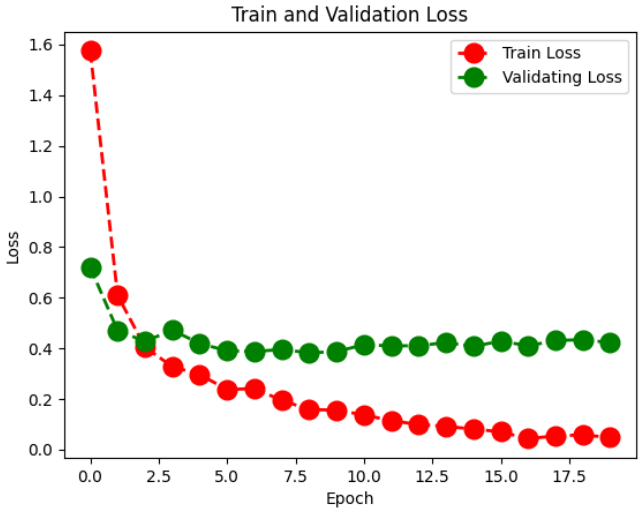

Figure 22: VGG16 with all 30 feature layers frozen LR=0.002

The train loss looks extremely similar to both previous attempts, but the validation loss actually decreases further in the first few epochs to around 0.4 where it remains for the continuing epochs.

6.4 Freeze all classifier layers

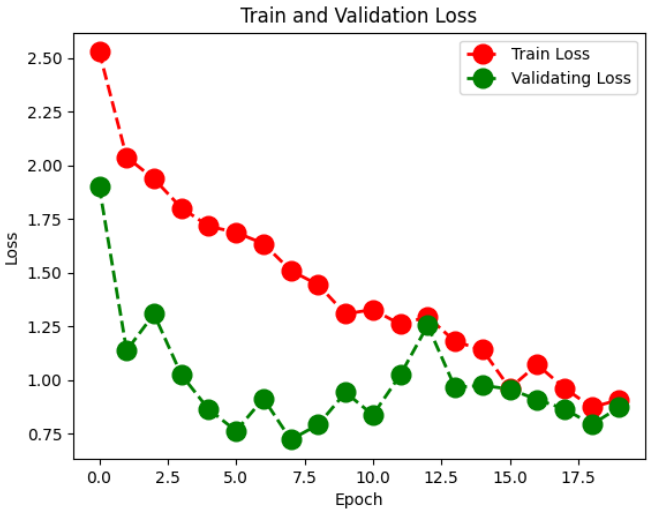

Figure 23: VGG16 with all classifier layers frozen LR=0.002

Both losses decrease quickly at the first few epochs, but whereas the training loss continues its descent, the validation loss quickly reaches a loss of around 1 and remains there somewhat. These are currently the worst results received from a pre-trained model.

6.5 Train entire model

This time we don't freeze any layers at all

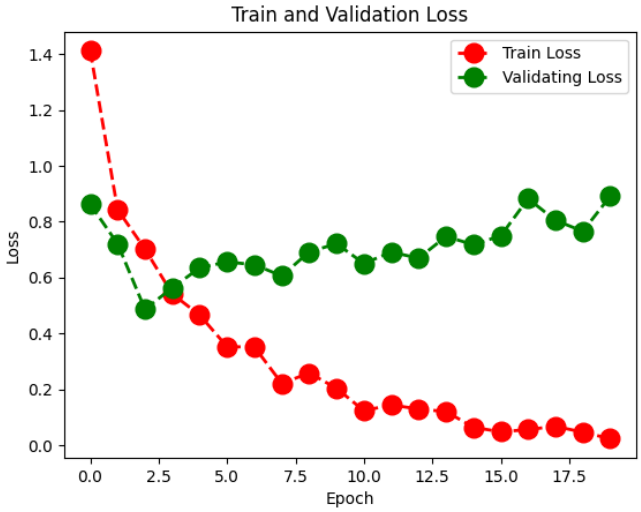

Figure 24: VGG16 with no layers frozen LR=0.002

The training loss is similar to when parts of the feature layer where frozen, starting with a steep descent in the first few epochs to then descent slower and end up close to 0. The validation loss however starts surprisingly low and starts rising slowly over these epochs, this is a clear sign that the model might be overfitting the data.

6.6 various batch sizes

Try to train the model with data in larger batch sizes to try and help overfitting

6.6.1 batchsize 2

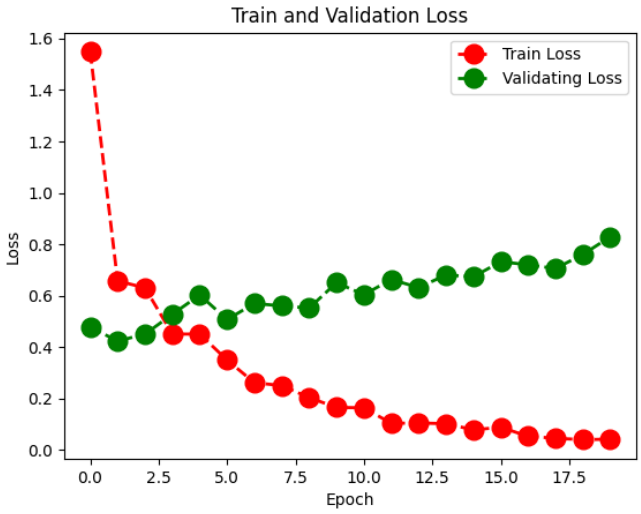

Figure 25: VGG16 with no layers frozen and batchsize 2 LR=0.002

The graph is very similar to the graph we got when using a batchsize of 1, however , while the validation loss still increases at a similar rate, the line is far straighter, meaning the variation between epochs has reduced.

6.6.2 batchsize 6

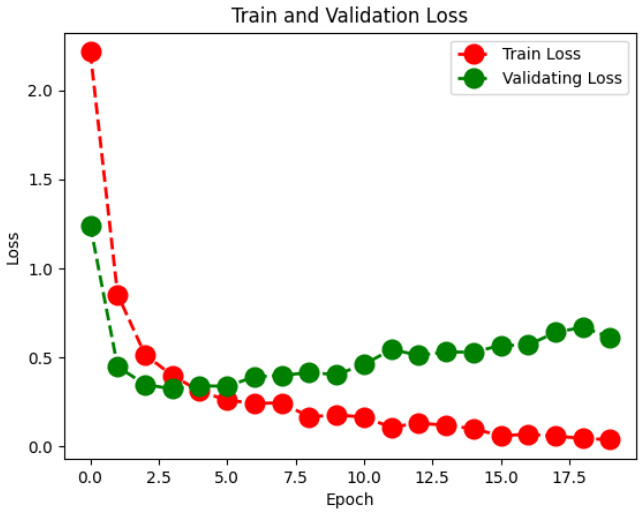

Figure 26: VGG16 with no layers frozen and batchsize 6 LR=0.002

The training loss acts the same as in previous models, the validation loss starts with a fast descent between the first few epochs but then starts rising again. However ,the rise is slower than with the smaller batchsizes, as at epoch 20 we have a loss of about 0.5

6.6.3 batchsize 10

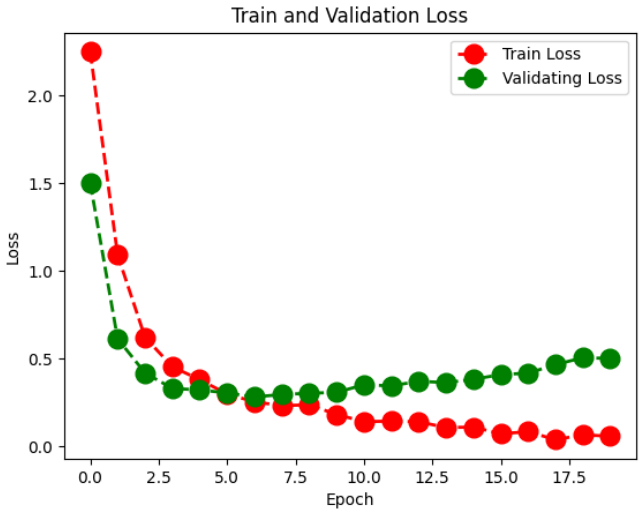

Figure 27: VGG16 with no layers frozen and batchsize 10 LR=0.002

The training loss starts once again with a steep descent in the first few epochs and continues to slowly descent in the continuing epochs. The validation loss follows the trend set by the previous batch sizes where it has a fast decrease in the first few epochs, but start to slowly rise over the subsequent epochs, however once again the incline is less than with the smaller batchsizes.

7 fine-tuning Strategies

In my opinion, fine-tuning the learning rate would be an important strategy for ensuring we do not get stuck at a sub-optimal loss-rate, as well as not learn at a rate that would require a near infinite amount of epochs to actually start learning and reducing the loss-rate.

The worst strategy would, in my opinion be the freezing of random layers in the model, there are simply too many combinations of layers that could be frozen $O(2^n)$ and fine-tuning implies uncertainty of which exact combination would be optimal, thus the necessity of trying them all. So unless you know which layers are already exactly what we wish them to be, in which case it would not be considered a fine tuning strategy. Freezing random layers would be a bad fine-tuning strategy

8 Project Information

This project was implemented both in a jupyter notebook on google collab CLICK HERE and locally on a github repository CLICK HERE, the collaborator on the project was my brother who simply ran the code. In the jupyter all cells state which specific task is researched before, in the repository the files that need to be run are the "partX_mainY.py" files representing the various tasks as X.Y . Both the repository and the notebook have a GPU variable in CONFIG.py or the firstmost cell respectively. setting this variable to true will make the models run on GPU instead of CPU, making it run much faster. Due to space restraints, the training and validation data are not included in the github repository.The data is added to the collab through my personal google drive, so as long as this is accessible through the collab it can be run. Should this not be case, the above link gives editing permissions, so slight changes could be made to allow editing of the data location etc. However, due to the risk of having a link anyone could access with editing permissions, this link CLICK HERE references a copy of the original unaltered code, which none but me can alter, as reference should the original reach a problematic state.