DISCLAIMER: The report is longer than requested due to the high amount of plots and images. The control over the images' dimensions and location is very limited in LyX.

1 Multi-Layer Perceptrons

1.1 Optimization of an MLP

1.1.1 Question 1

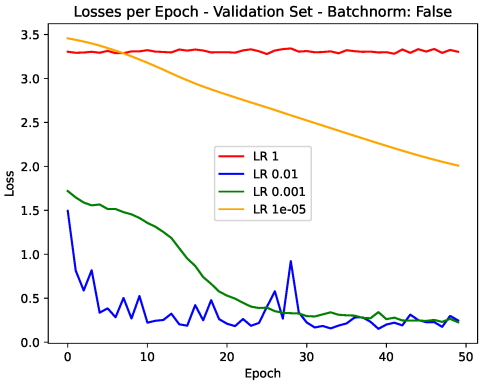


Figure 1: Losses per Epoch on different learning rates, for Validation Set

Analysis As we can see in Figure 1, the highest learning rate (LR=1), marked in red, is somewhat stable (in a bad way), this is the effect of the GD being ineffective with that rate, and the steps are almost meaningless. On the other hand, we can see with the orange plot that the lowest LR (0.0001) is indeed descending, though very slowly, and that is the result of having too small of steps - The contrary of the phenomenon we've seen with the highest LR. So the learning of the model with the lowest LR is possibly effective but not optimal, as we're going to need many more epochs to descent somewhere near LR 0.001 for example. The blue and green LRs in the figure are most optimal, and we're seeing the favorable effect of having a large descent early on (i.e. with less epochs) in the model's training.

1.1.2 Question 2

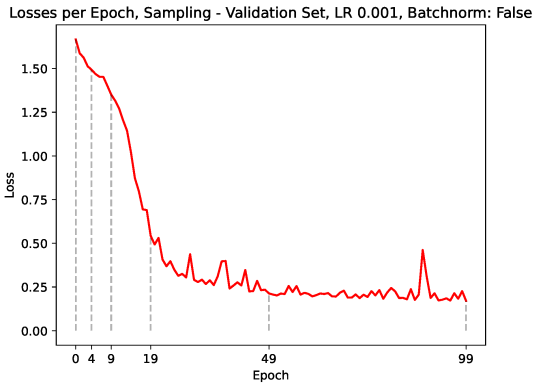
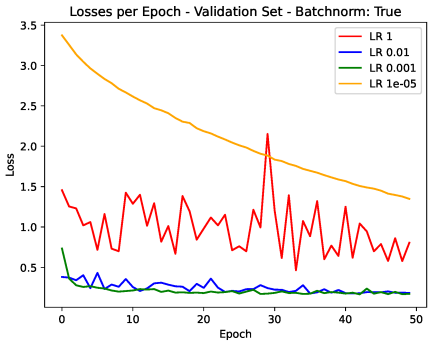


Figure 2: Losses per Epoch for 100 epochs, on Validation Set, LR 0.001

Analysis The most dramatic change in the losses happens up until the ~20th epoch. From there, there isn't a dramatic change and sometimes there is a peaking loss. The peaking loss is possibly the result of parameter miscalculation by the Torch library for one step, considering there are lots of parameters in play with the Neural Network, and that the resulting loss functions are not convex, thus finding the minimum is harder, resulting in miscalculations like we've just seen. The same can be told on the first epochs, since that there are many parameters then the NN requires more iterations to perfect the parameters.

1.1.3 Question 3



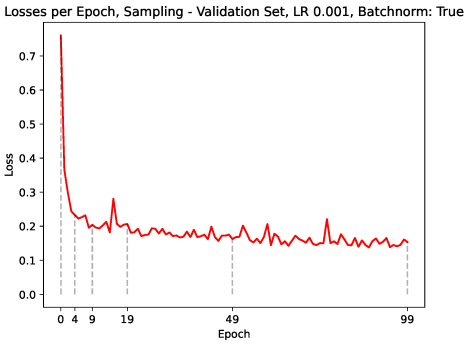


Figure 3 (Left): Losses per Epoch on different learning rates, for Validation Set with BatchNorm

Figure 4 (Right): Losses per Epoch for 100 epochs, on Validation Set, LR 0.001 with BatchNorm

Analysis Comparing the results from Figure 3 to Figure 1, we can see that both LRs are stabilizing faster (with less epochs) and that there aren't as many "peaks" in Figure 3 compared to 1, especially with the 0.01 LR - so in conclusion the loss stabilizes faster and remains stable throughout the training. It is also visible in Figure 4 at a much more precise manner, with the peaks being less dramatic. Comparing the lowest LR (0.0001) for both experiments (with and without the batch normalization), then we can see that the lowest LR also learns faster. And with the highest LR, the losses are unstable, compared to being stable (but bad) in the previous experiment, so the highest LR is still ineffective.

1.1.4 Question 4

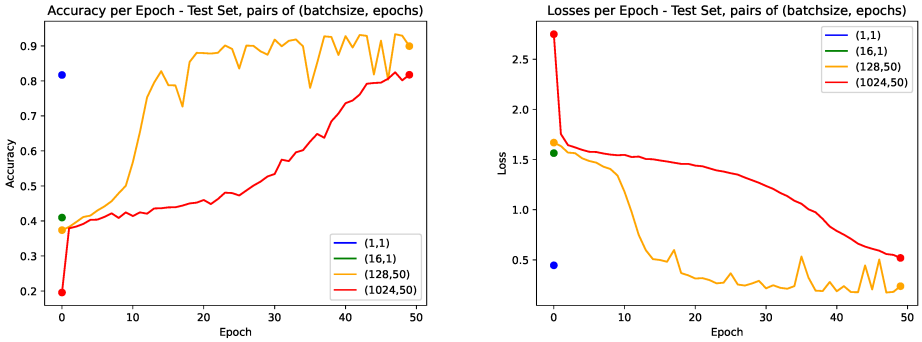


Figure 5 (Left): Accuracy per Epoch on Test Set, batch size and epoch pairs
Figure 6 (Right): Losses per Epoch on Test Set, batch size and epoch pairs

Analysis

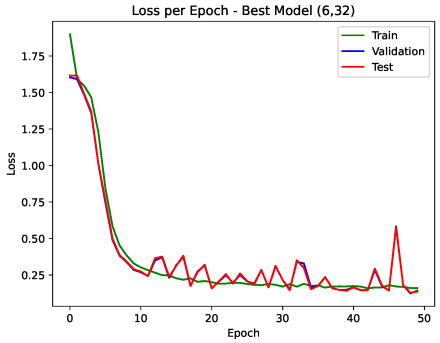
1. Accuracy - with Fig. 5 we can conclude that the **initial** accuracy of the model is lower as the batch size increases, regardless of epochs, with the losses being on the opposite (higher batch size = higher **initial** loss). First, for the hyperparameter pairs of epochs=1 we see that the $(1, 1)$ pair achieves good accuracy (about 0.81), thats possibly because for that one epoch we iterate and update the weights for every single sample, so the algorithm will eventually correct itself for every single sample, that is unlike $(16, 1)$ pair, which achieves (much) worse result than the $(1, 1)$ pair. Second, for the hyperpameter pairs of epochs=50 we see that over time the 128 batch size achieves better result - It is possibly a sweet-spot, we both go over enough samples and updates for the weights through the epochs, and not go through every single sample surgically.

2. Speed - Although not plotted, the training speed was faster with higher batch sizes. From my understanding there are 2 primary reasons for it - The more inherent one is that there are simply SGD is built with batch training in mind, and that there's generally less weight updates and loss calculations. A more indirect reason for this is that the GPU is utilized more effectively with higher batch sizes.

3. Stability - Considering the $(1, 1)$ and $(16, 1)$ pairs are only for one epoch, we cannot tell a lot about its stability over time. On the two other pairs, we can see that $(128, 50)$ is stabilizing faster, but there's some noise (seen as the peaks for the oragnge plot in Fig. 6). The $(1024, 50)$ on the other hand doesn't stabilize as much as the previous pair, and it would possibly require more epochs to reach stabilization, but unlike the previous pair it is not noisy, but it might get noisy as we increase the epochs and reach a more stabilized weights (so every miscalculation means a more dramatic change in the loss).

1.2 Evaluating MLPs Performance

1.2.1 Question 1



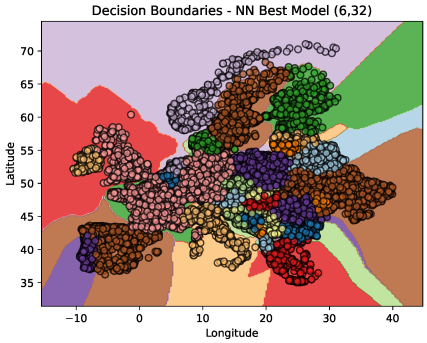
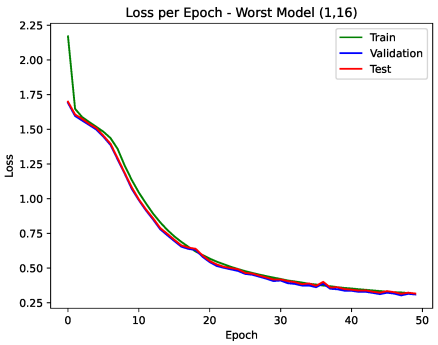


Figure 7 (Left): Losses per Epoch for the Best Model, chosen according to Validation Accuracy

Figure 8 (Right): Decision Boundaries for the Best Model, chosen according to Validation Accuracy

Analysis The model generalized well, we can see that the losses of the Test and Validation converge quite near to the Train losses, although there is some noise in some epochs, but the calculations eventually converge approx. the same. Moreover, we can see from the decision boundaries that the classification is very accurate, even in the tightest spots (take Albania or Sardinia for example).

1.2.2 Question 2



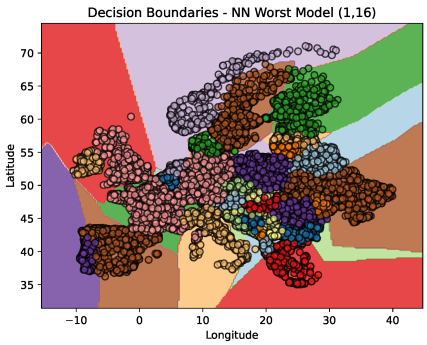


Figure 9 (Left): Losses per Epoch for the **Worst** Model, chosen according to Validation Accuracy

Figure 10 (Right): Decision Boundaries for the **Worst** Model, chosen according to Validation Accuracy

Analysis The model did not seem to overfit, considering the losses on the Test and Validation sets converge closely to the Training set's losses. Moreover, the decision boundaries are looking good, not too inaccurate, especially when compared to the best model from Q1. So with that, it looks like the model is underfitted, and it may possibly reached the best result it could considering that it's the number of hidden layers (only 1) could have possibly reached its limit.

1.2.3 Question 3 - Depth of Network

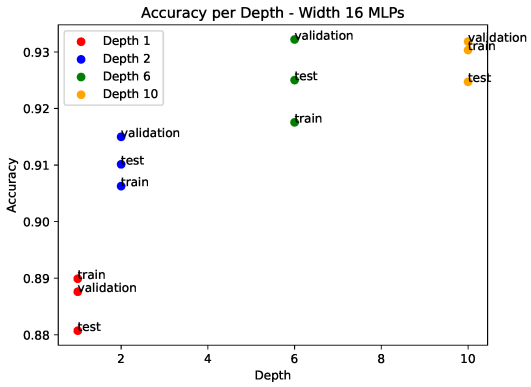


Figure 11: Accuracy for Train, Test and Validation sets on MLPs with 16 neurons in each Hidden Layer

Analysis Looking at Figure 11, we can see that the accuracies of the lowest depths (specifically 1 and 2, marked in blue and red), are lower than that of the higher depths. The advantages are that a deeper NN gives us better results for this dataset, but we should not overexaggerate - since looking at Depth 10 we can see that there might be slight decay in accuracy, though we'd have to experiment more on deeper networks for this dataset to conclude this hypothesis.

1.2.4 Question 4 - Width of Network

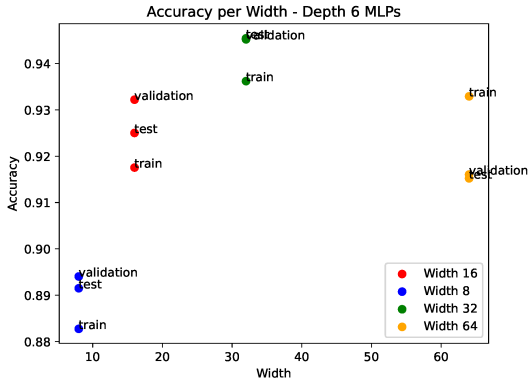


Figure 12: Accuracy for Train, Test and Validation sets on MLPs with 6 Hidden Layer each

Analysis We can see a sweet-spot hanging over the Width of 32, with it being with best overall results for all the datasets. Looking at Width 64, we might have a case of overfitting - The train accuracy is higher than the test and validation accuracies, which indicates that the NN has been optimized better for the training data, and has worse generalization for the rest of the data. This result makes sense, since Width 64 means we'd have lots of neurons to fit weights to, hence having more "decision stumps", thus creating the effect of misclassifying samples which are "on the edge".

1.2.5 Question 5 - Gradient Magnitudes

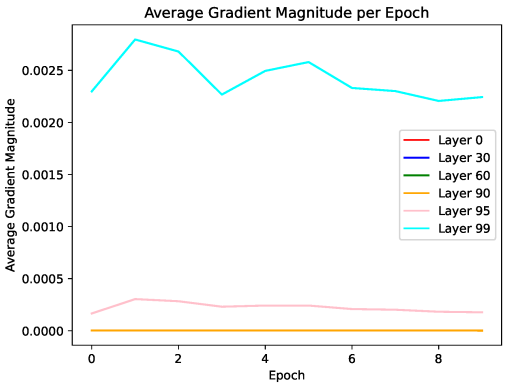


Figure 13: Averge Gradient Magnitude per Epoch for model with 100 hidden layers

Analysis In Figure 13 we can see a vanishing gradient, with all the layers before the 95th having their gradient converging to zero (as seen by the apparent $y = 0$ function). The reason we see it in a backwards manner (that the deeper layers still have some non-zero gradient) is because of the backpropagation algorithm applied when updating the weights. What we can possibly do to tackle this issue, is what we were taught in the lecture for deeper networks - Make our network a ResNet (i.e. with residual layers) - This method has been experimented and shows better results for deeper networks.

2 Convolutional Neural Networks

2.1 Question 1

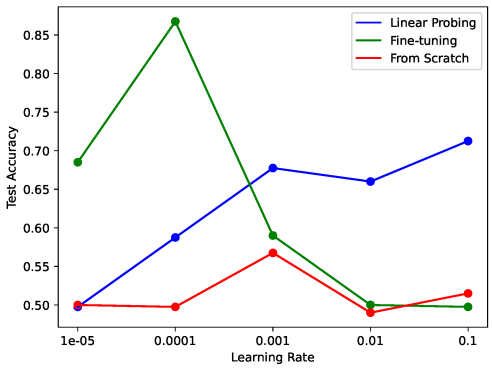


Figure 14: Accuracy on Test Set for each LR on the different CNN baselines

Analysis Below is a table for the best and worst models, in pairs of Learning Rate and Test Accuracy, for each baseline (with the XGBoost having a single best, as we are using only the default parameters):

$Baseline/Result$	$Best\ (1st)$	$Best\ (2nd)$	$Worst$
$XGBoost$	$LR = Default$ $Acc = 0.73$	/ /	/ /
$From\ Scratch$	$LR = 0.001$ $Acc = 0.5675$	$LR = 0.1$ $Acc = 0.515$	$LR = 0.01$ $Acc = 0.49$
$Linear\ Probing$	$LR = 0.1$ $Acc = 0.7125$	$LR = 0.001$ $Acc = 0.6775$	$LR = 0.00001$ $Acc = 0.4975$
$Fine-Tuning$	$LR = 0.0001$ $Acc = 0.8675$	$LR = 0.00001$ $Acc = 0.685$	$LR = 0.1$ $Acc = 0.4975$

As we can see in Fig. 14, I used five different LRs for each model (except XGBoost as mentioned earlier). The best performing model is with the Fine-Tuning baseline, and it should not come as a surprise, considering that in this baseline we've gotten an already trained ResNet for the task of Image Processing and continued to train it on the specific task of classifying between real and faked pictures of faces. It is safe to assume that trained ResNet in the Fine-Tuning baseline is much stronger and accurate than a single-epoch trained model we've built from scratch, thus giving us better results. The Linear Probing model also performing better than the Training from Scratch is also not a surprise, considering that the Linear Probing model also contains the pretrained model, although it's not updated during training phase. The most interesting comparison is between the Linear Probing and the Fine-Tuning models. We can see that the Linear Probing performs better than the Fine-Tuning when both use higher LRs, that can be the result of the higher LRs modifying the pretrained ResNet in the Fine-Tuning model, to an undesirable degree, while in the Linear Probing model only the Regression layer would be updated, and not the rest of the net, thus a single-layer optimization is more effective with the higher LRs (being convex), compared to the many-layer optimization in the Fine-Tuning model, which gets highly inaccurate for higher LRs.

2.2 Question 2

Below are the misclassified pictures, with the best baseline being the Fine-Tuning with LR 0.0001, and the worst baseline being the From-Scratch with LR 0.001.
(The way I chose the best and worst is by taking the best from each baseline and then picking the best/worst, so the worst-performing baseline is not necessarily the worst performing model).
Additionally, to watch the pictures at full resolution, the names of the pictures are as follows: image-2019-02-17_024628, image-2019-02-17_024606, image-2019-02-17_024558, image-2019-02-17_024536, image-2019-02-17_024331