Foundations of Deep Learning HW1

Yonatan Ariel Slutzky 318965506   Eyal Grinberg 207129792

2 April 2023

1 Part 1

When training the linear SVM classifier, the train accuracy we reached was 0.99, and the test accuracy we reached was 0.32.
When training the RBF SVM classifier (with the default sklearn gamma hyperparameter), the train accuracy we reached was 0.73, and the test accuracy we reached was 0.45.

2 Part 2

2.1 Section 1

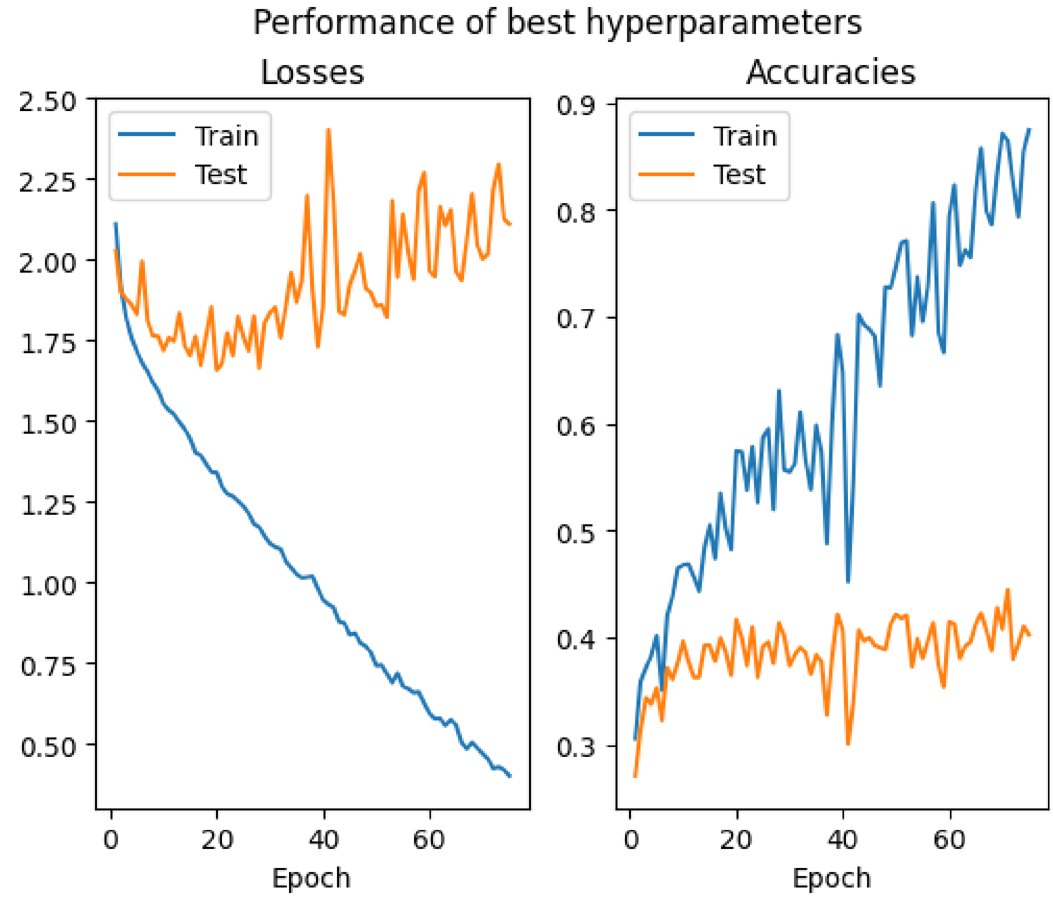
We performed a grid search over the following ranges:

$$\sigma \in [\frac{1}{16}, \frac{1}{8}, \frac{1}{4}, \frac{1}{2}, 1, 2], lr \in \{0.1, 0.01, 0.001\}, momentum \in \{0.95, 0.85, 0.75, 0.65\}$$

We trained the nets for 25 epochs, and evaluated them according to their test accuracies.
The best hyperparameters were

$$\sigma = \frac{1}{16}, lr = 0.01, momentum = 0.75$$

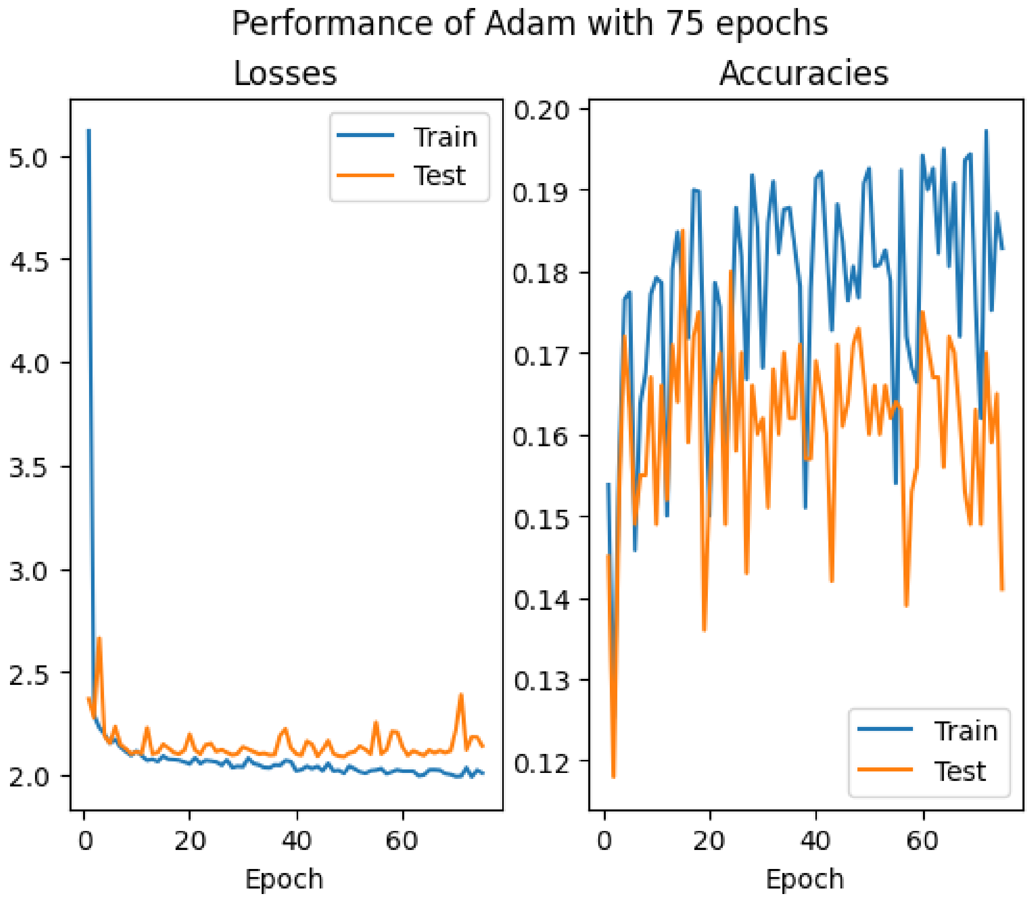
The following are the results when using them over 75 epochs:



The final train loss was 0.4, the final test loss was 2.11, the final train accuracy was 0.87, and the final test accuracy was 0.4.

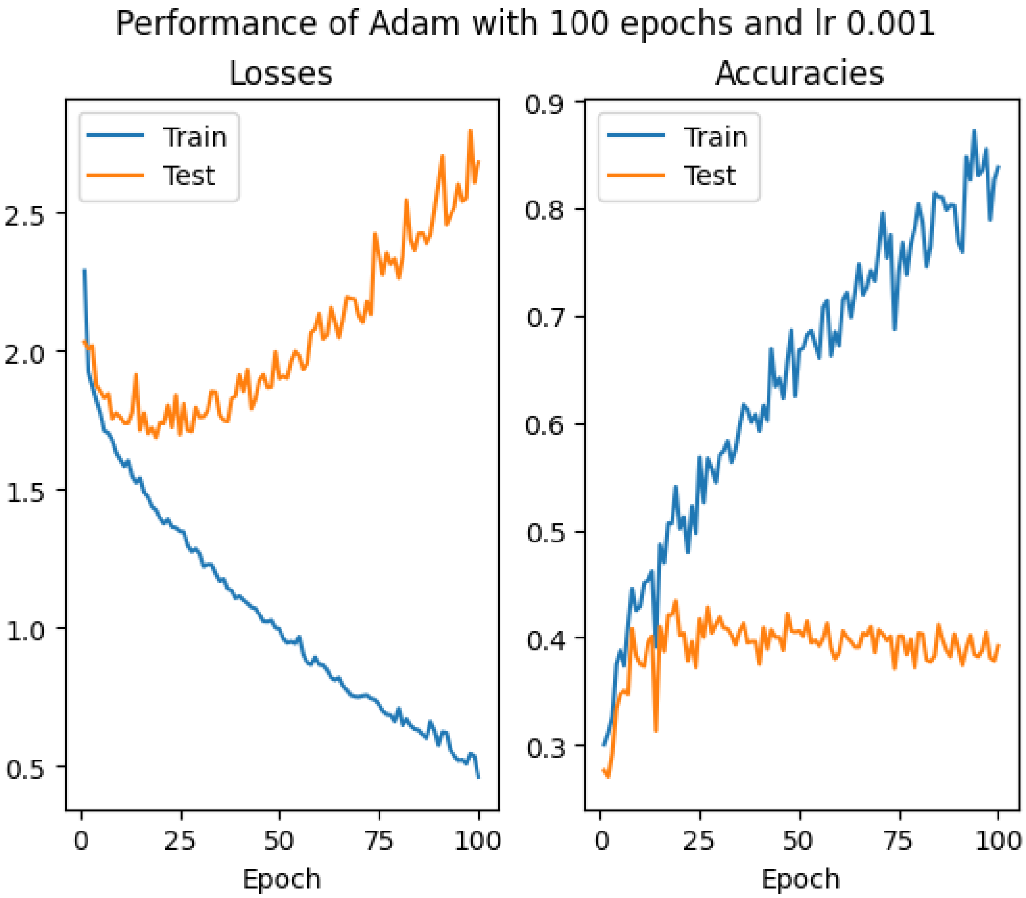
2.2 Section 2

We used Adam with the best learning rate and σ from the baseline.
The following are the results when using them over 75 epochs:



As seen above, Adam converges much quicker than SGD, however, it doesn't converge to the global minima, and seems to be stuck at a local minima. The final train loss was 2, the final test loss was 2.14, the final train accuracy was 0.18 and the final test accuracy was 0.14.

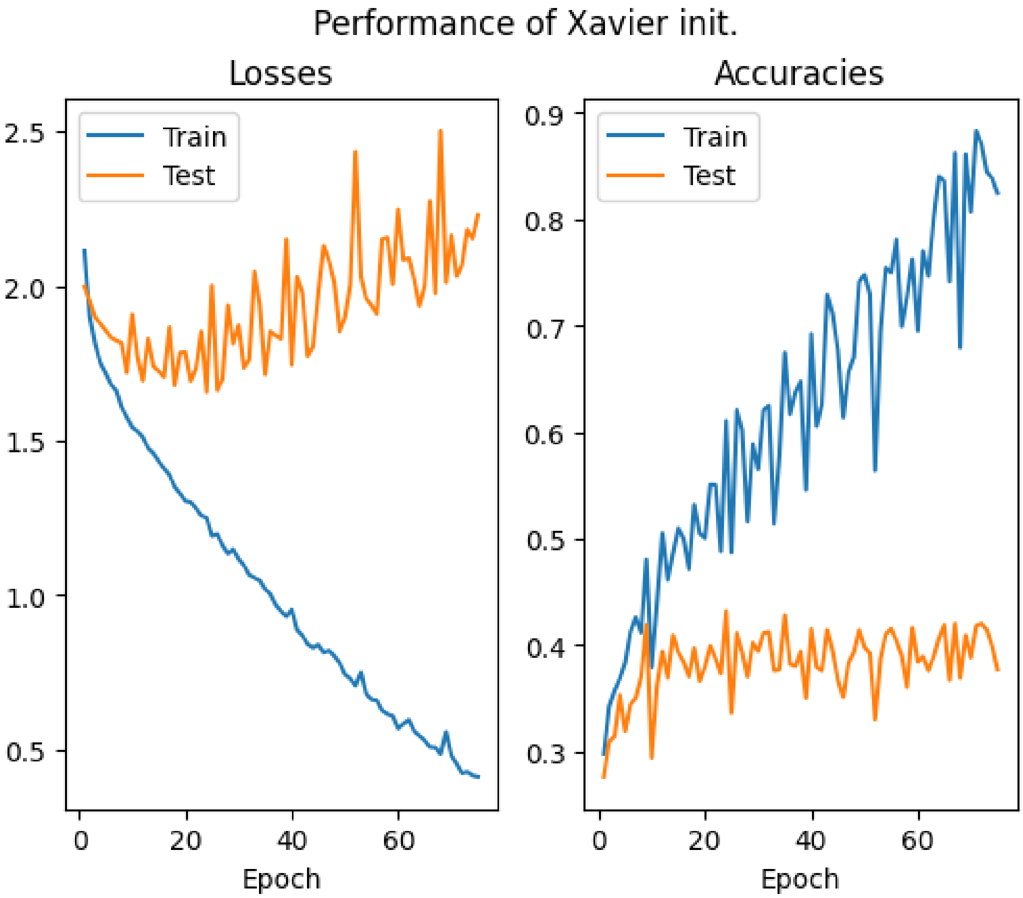
However, when setting a finer learning rate of 0.001, and letting the optimizer run for 100 epochs, the result improve dramatically:



Now, the final train loss was 0.46, the final test loss was 2.68, the final train accuracy was 0.84 and the final test accuracy was 0.39.
These results are much similar to that of SGD, but the convergence times seem to be the same (with regards to the test) - around 20 epochs.
An important observation we see in both cases, is that the learning curve is much less ”bumpy” than when using SGD.

2.3 Section 3

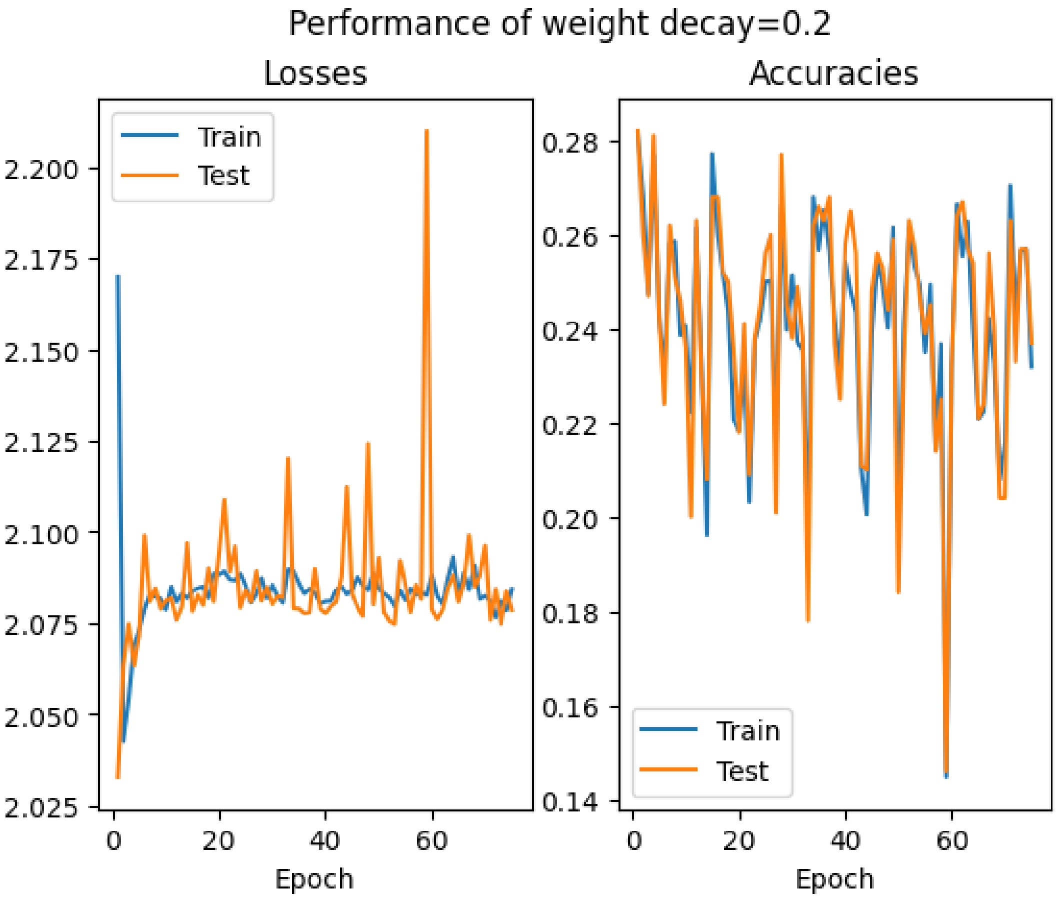
Using Xavier init. and the best hyperparameters from the baseline, we reached the following results:



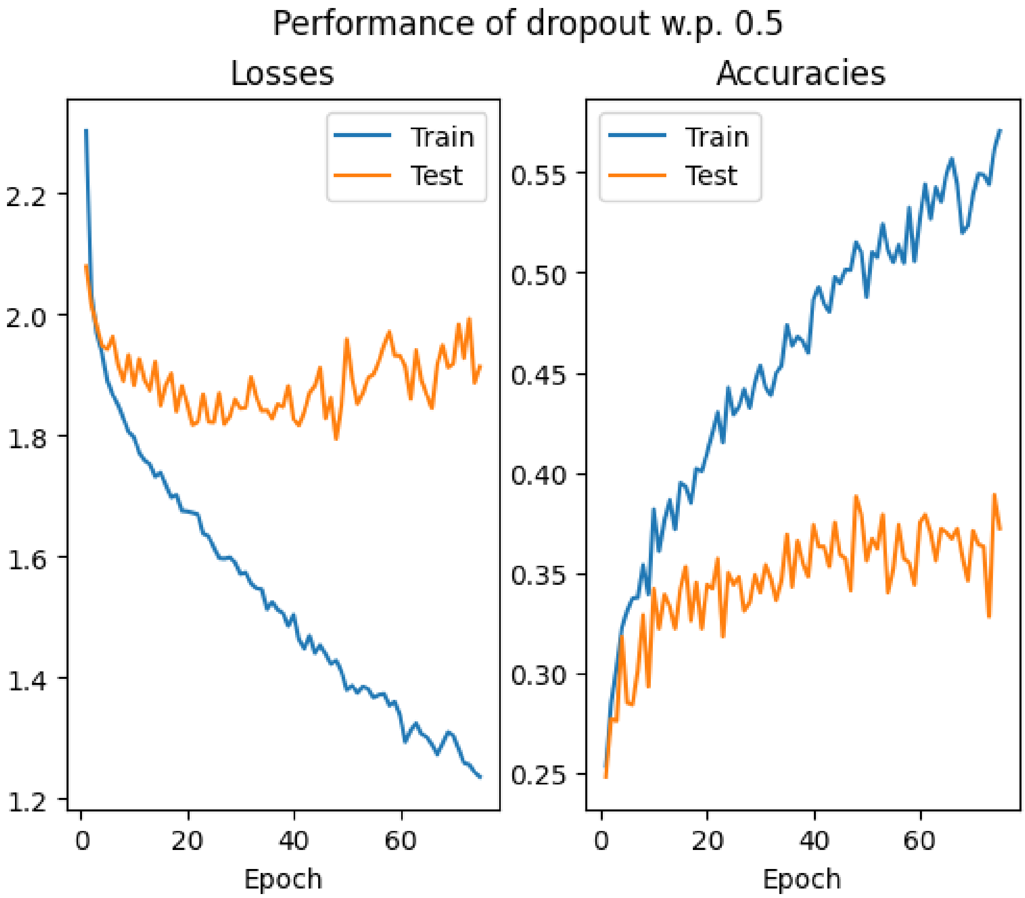
The final train loss was 0.41, the final test loss was 2.22, the final train accuracy was 0.82 and the final test accuracy was 0.38.
In terms of convergence time, the convergence (with regards to the test) happens a little sooner than when normally initializing (at around 15 epochs).

2.4 Section 4

Using weight decay of 0.2, and the best hyperparameters from the baseline, from the baseline, we reached the following results:



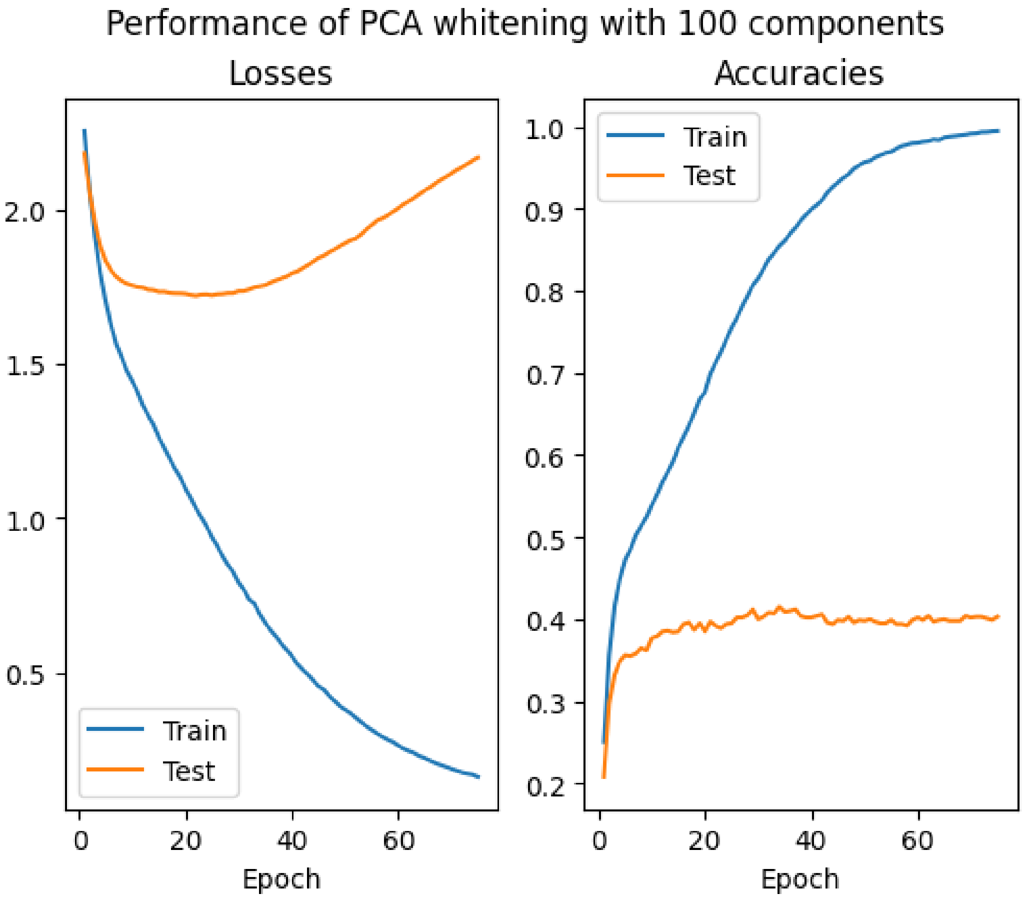
The final train loss was 2.08, the final test loss was 2.07, the final train accuracy was 0.23 and the final test accuracy was 0.23.
As we can see, the learning curve is very ”bumpy”, the results are worse, and another observation is that when using a slightly higher weight decay, the training time increased significantly (with the same training settings besides that hyperparameter). We assume this is due to the difficult optimization problem the optimizer attempts to solve.
Using a dropout layer with probability 0.5, and the best hyperparameters from the baseline, from the baseline, we reached the following results:



The final train loss was 1.23, the final test loss was 1.91, the final train accuracy was 0.57 and the final test accuracy was 0.37.
We can see the test results are pretty similar to that of the baseline, while the results on the training are worse. The overfitting to the data seems to be less harsh than the baseline.

2.5 Section 5

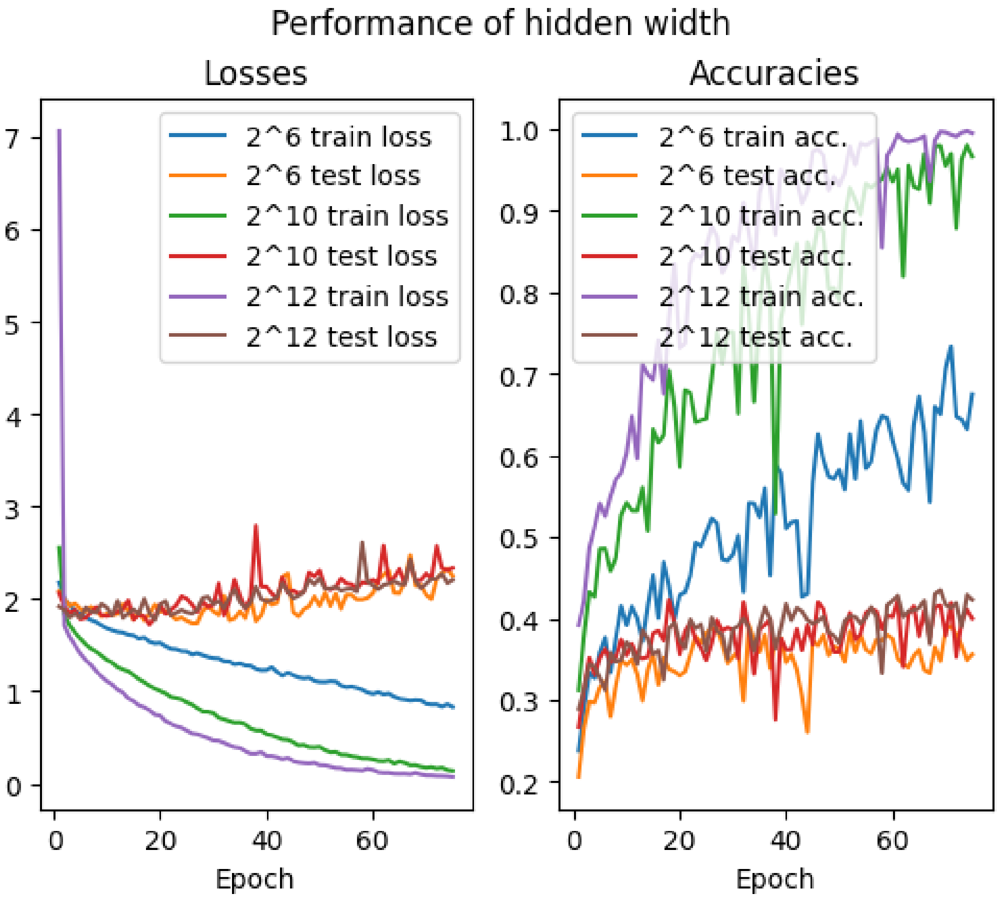
Using PCA with 100 components, and the best hyperparameters from the baseline, from the baseline, we reached the following results:



The final train loss was 0.16, the final test loss was 2.17, the final train accuracy was 0.99 and the final test accuracy was 0.4.
We can see the results (on the test set) do not improve w.r.t the baseline. However, training time was significantly faster (since the input size is a fraction of the original data), and the learning curve is very smooth.

2.6 Section 6

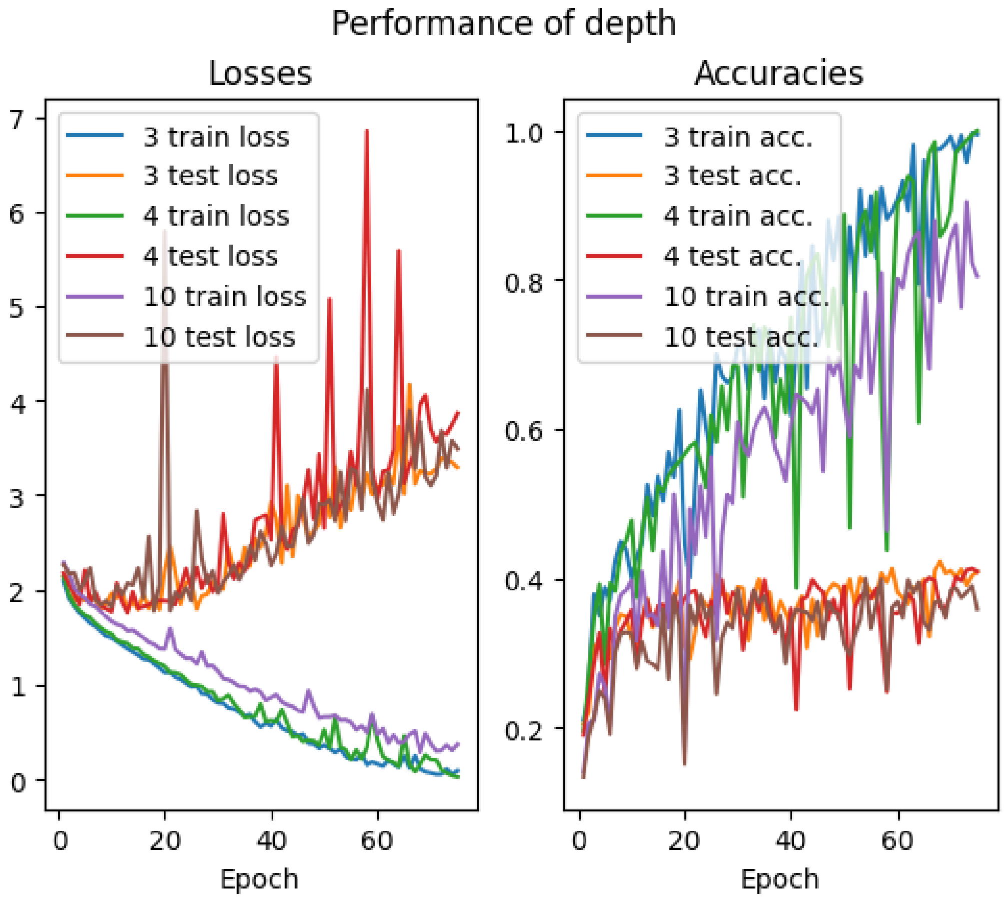
Using various hidden widths, and the best hyperparameters from the baseline, we reached the following results:



For hidden width 2^6, the final train loss was 0.83, the final test loss was 2.25, the final train accuracy was 0.67 and the final test accuracy was 0.35.
For hidden width 2^{10}, the final train loss was 0.14, the final test loss was 2.33, the final train accuracy was 0.966 and the final test accuracy was 0.4.
For hidden width 2^{12}, the final train loss was 0.07, the final test loss was 2.2, the final train accuracy was 0.99 and the final test accuracy was 0.42.
We can see that as the hidden width grows, the test performance seems to improve, but the learning curves become ”bumpier”, probably as a result of overfitting to the data.

2.7 Section 7

Using various hidden depths, and the best hyperparameters from the baseline, we reached the following results:



For depth size 3, the final train loss was 0.09, the final test loss was 3.29, the final train accuracy was 0.99 and the final test accuracy was 0.41.
For depth size 4, the final train loss was 0.02, the final test loss was 3.86, the final train accuracy was ~ 1 and the final test accuracy was 0.41.
For depth size 10, the final train loss was 0.37, the final test loss was 3.49, the final train accuracy was 0.8 and the final test accuracy was 0.36.
We can see that as the depth size grows, the learning curves become ”bumpier”, probably as a result of overfitting to the data.

3 Part 3

3.1 Section 1

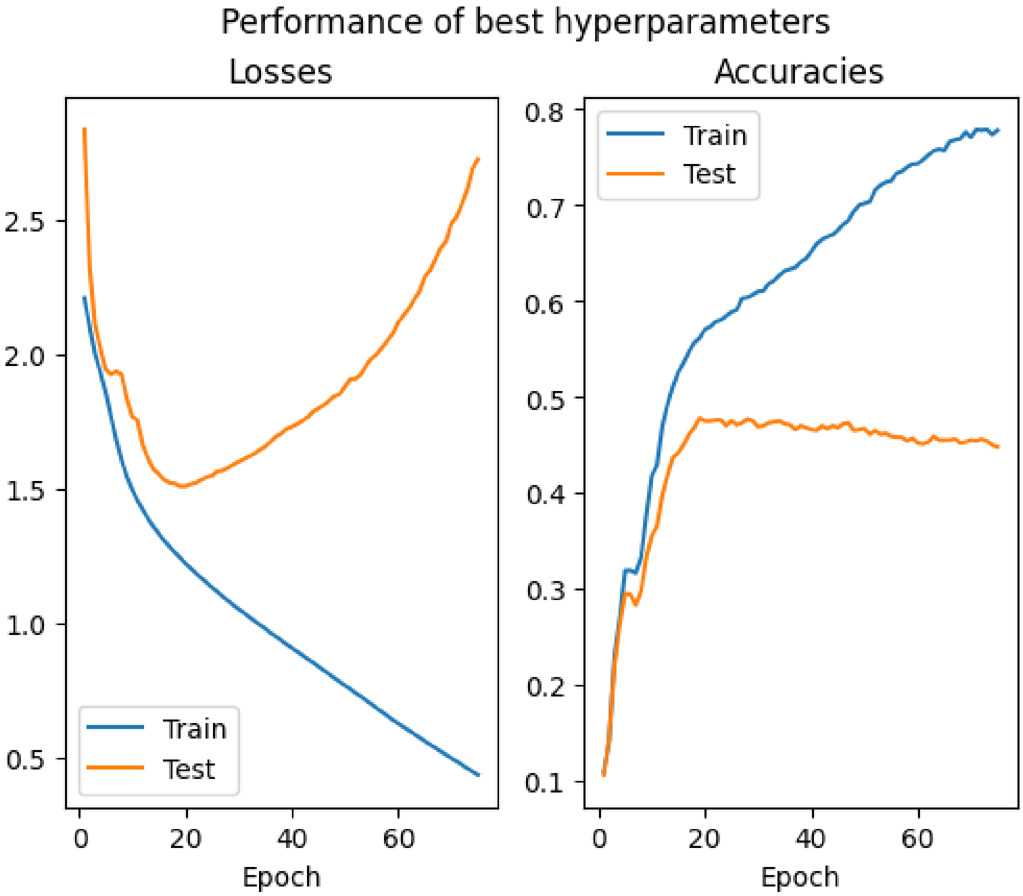
We performed a grid search over the following ranges:

$$\sigma \in [\frac{1}{16}, \frac{1}{8}, \frac{1}{4}, \frac{1}{2}], lr \in \{0.1, 0.01, 0.001\}, momentum \in \{0.85, 0.75, 0.65\}$$

We trained the nets for 25 epochs, and evaluated them according to their test accuracies.
The best hyperparameters were

$$\sigma = \frac{1}{16}, lr = 0.01, momentum = 0.65$$

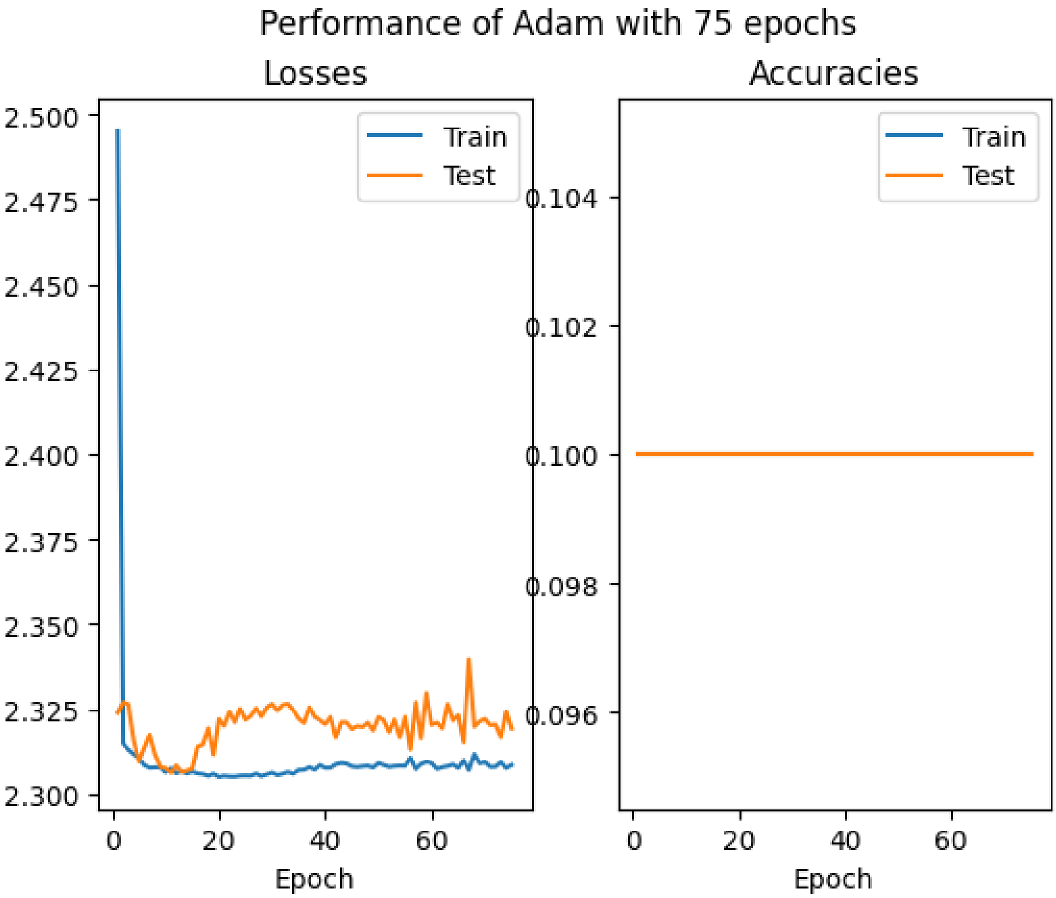
The following are the results when using them over 75 epochs:



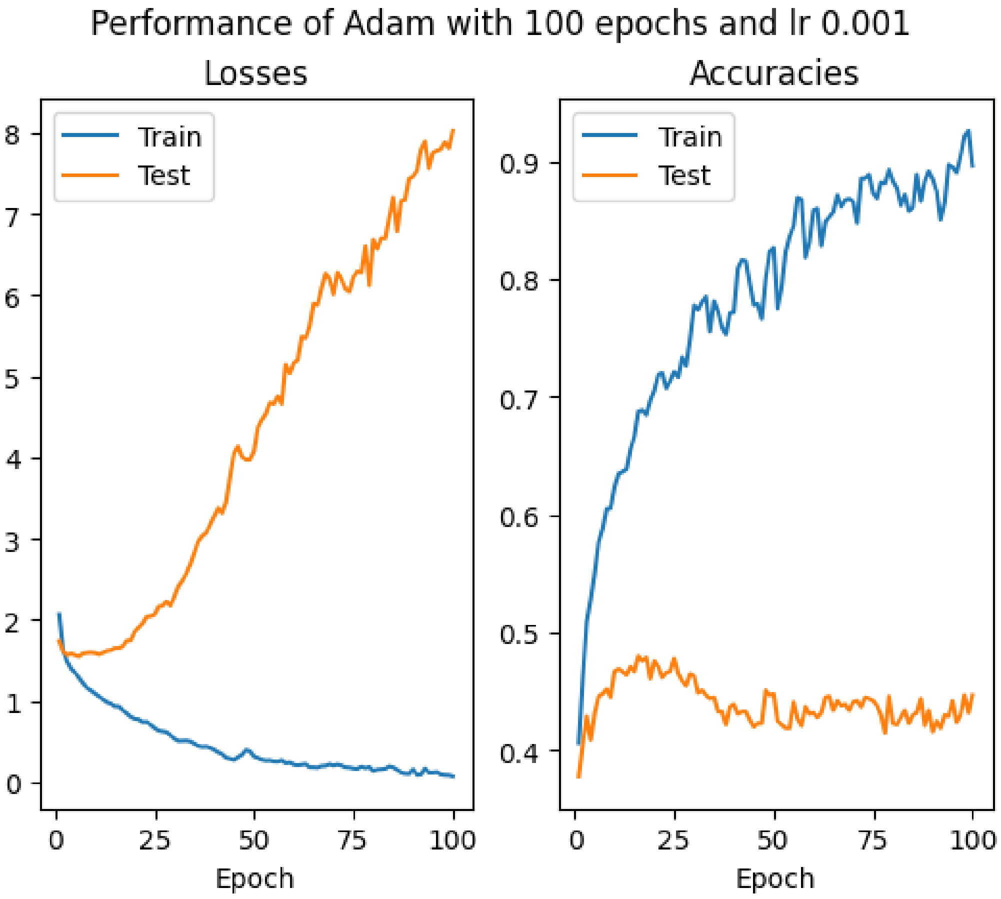
The final train loss was 0.43, the final test loss was 2.72, the final train accuracy was 0.77, and the final test accuracy was 0.45.

3.2 Section 2

We used Adam with the best learning rate and σ from the baseline.
The following are the results when using them over 75 epochs:



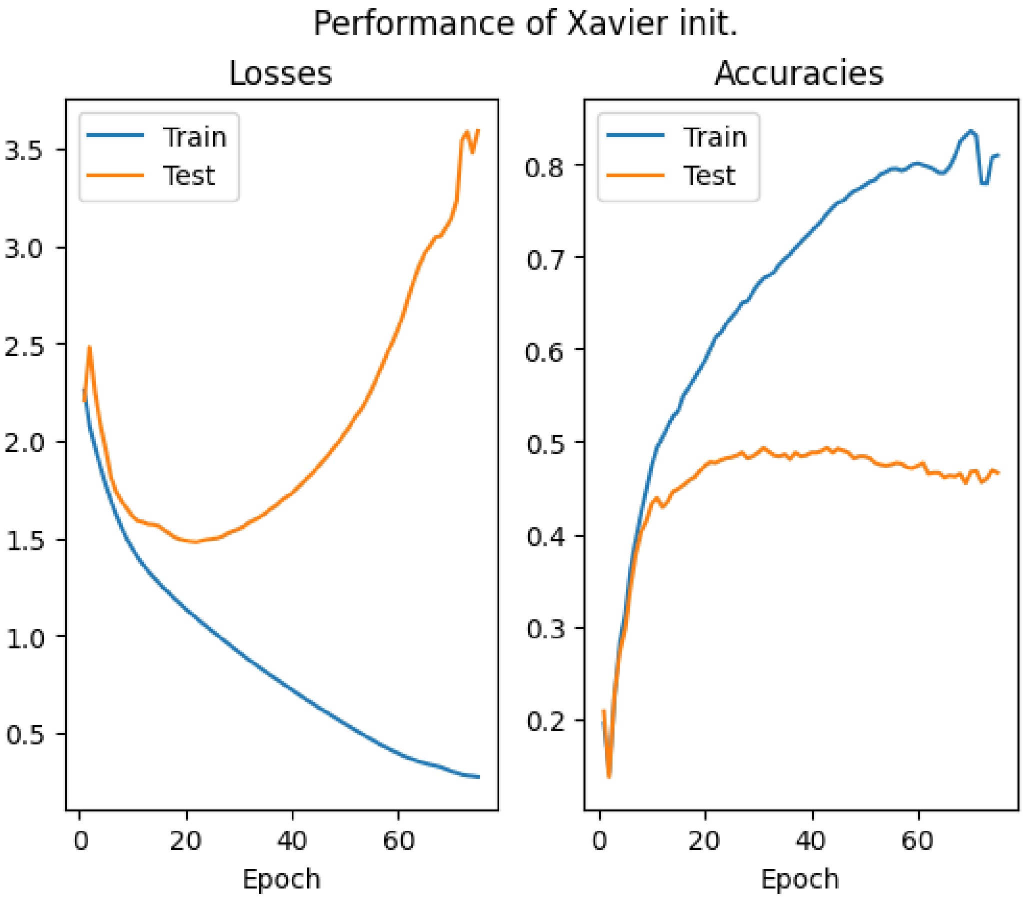
As seen above, Adam converges much quicker than SGD, however, it doesn't converge to the global minima, and seems to be stuck at a local minima. The final train loss was 2.3, the final test loss was 2.32, the final train accuracy was 0.1 and the final test accuracy was 0.1.
However, when setting a finer learning rate of 0.001, and letting the optimizer run for 100 epochs, the result improve dramatically:



Now, the final train loss was 0.07, the final test loss was 8.02, the final train accuracy was 0.89 and the final test accuracy was 0.44.
These results are much similar to that of SGD, but the convergence times seem to be the same (with regards to the test) - around 20 epochs.

3.3 Section 3

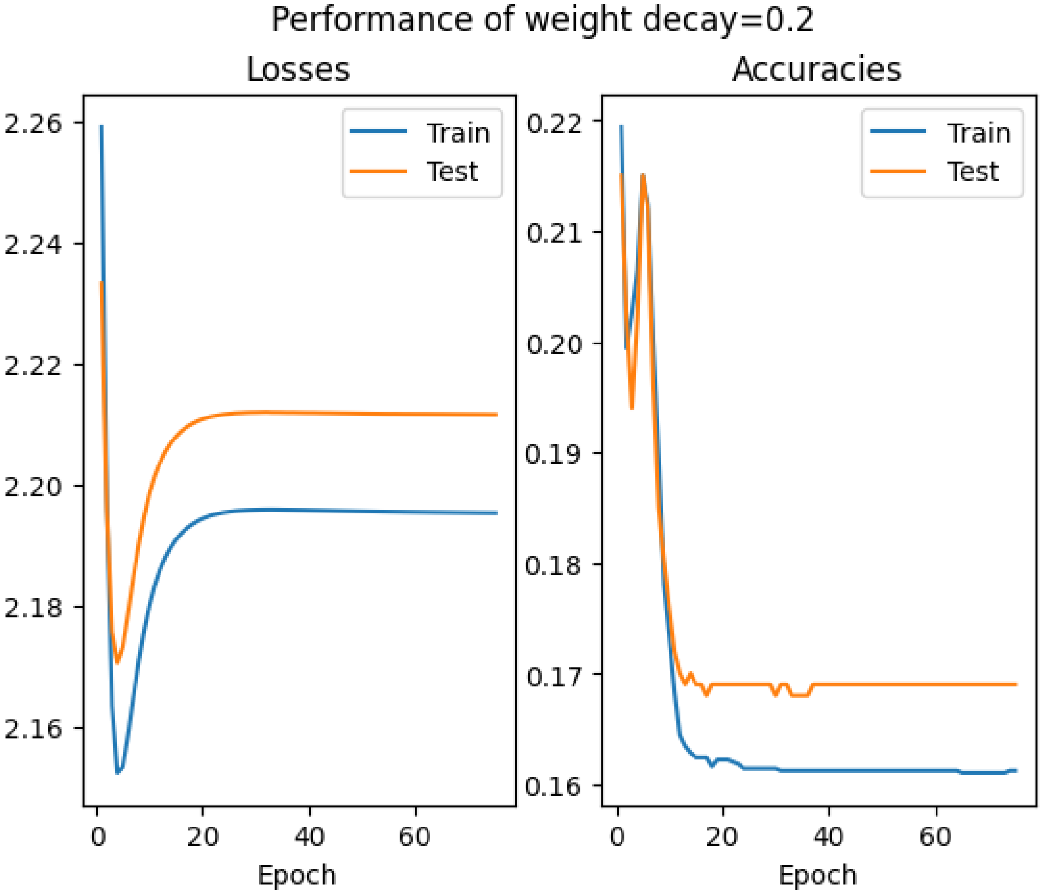
Using Xavier init. and the best hyperparameters from the baseline, we reached the following results:



The final train loss was 0.27, the final test loss was 3.59, the final train accuracy was 0.81 and the final test accuracy was 0.46.
In terms of convergence time, the convergence (with regards to the test) happens a little sooner than when normally initializing (at around 15 epochs).

3.4 Section 4

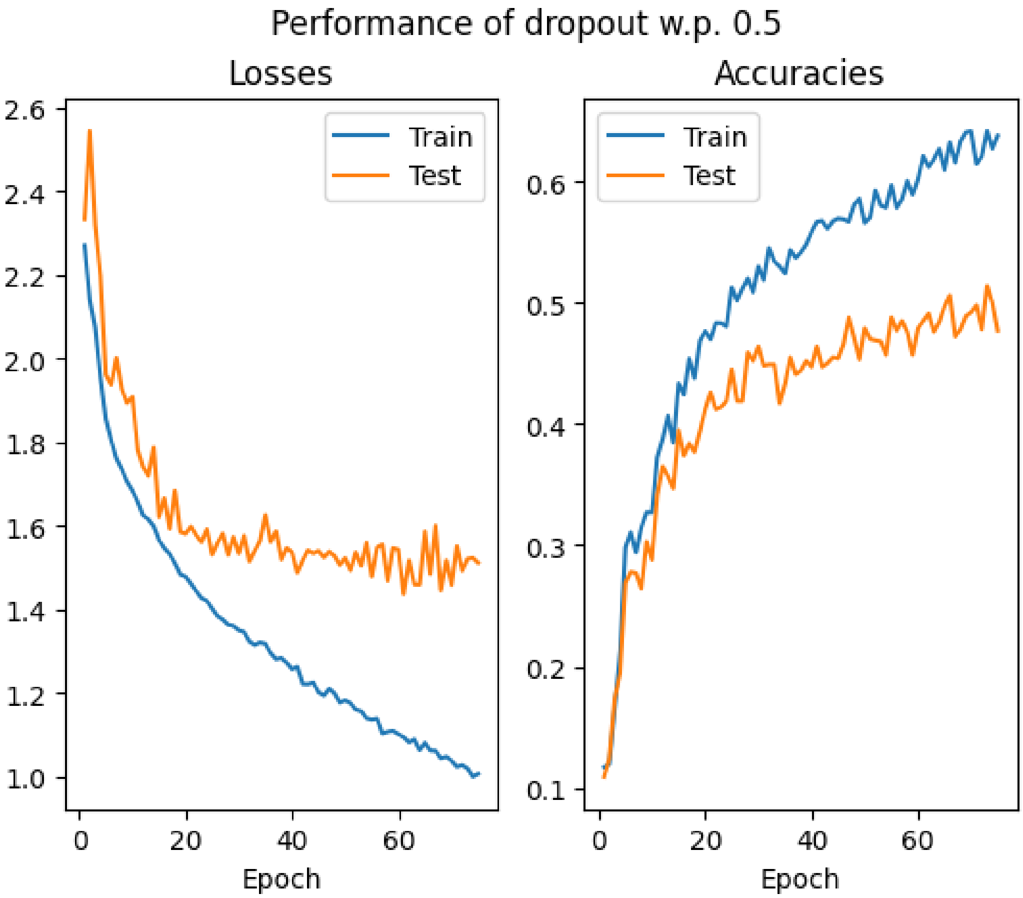
Using weight decay of 0.2, and the best hyperparameters from the baseline, from the baseline, we reached the following results:



The final train loss was 2.19, the final test loss was 2.21, the final train accuracy was 0.16 and the final test accuracy was 0.16.

As we can see, the learning curve is very ”bumpy”, the results are worse, and another observation is that when using a slightly higher weight decay, the training time increased significantly (with the same training settings besides that hyperparameter). We assume this is due to the difficult optimization problem the optimizer attempts to solve.

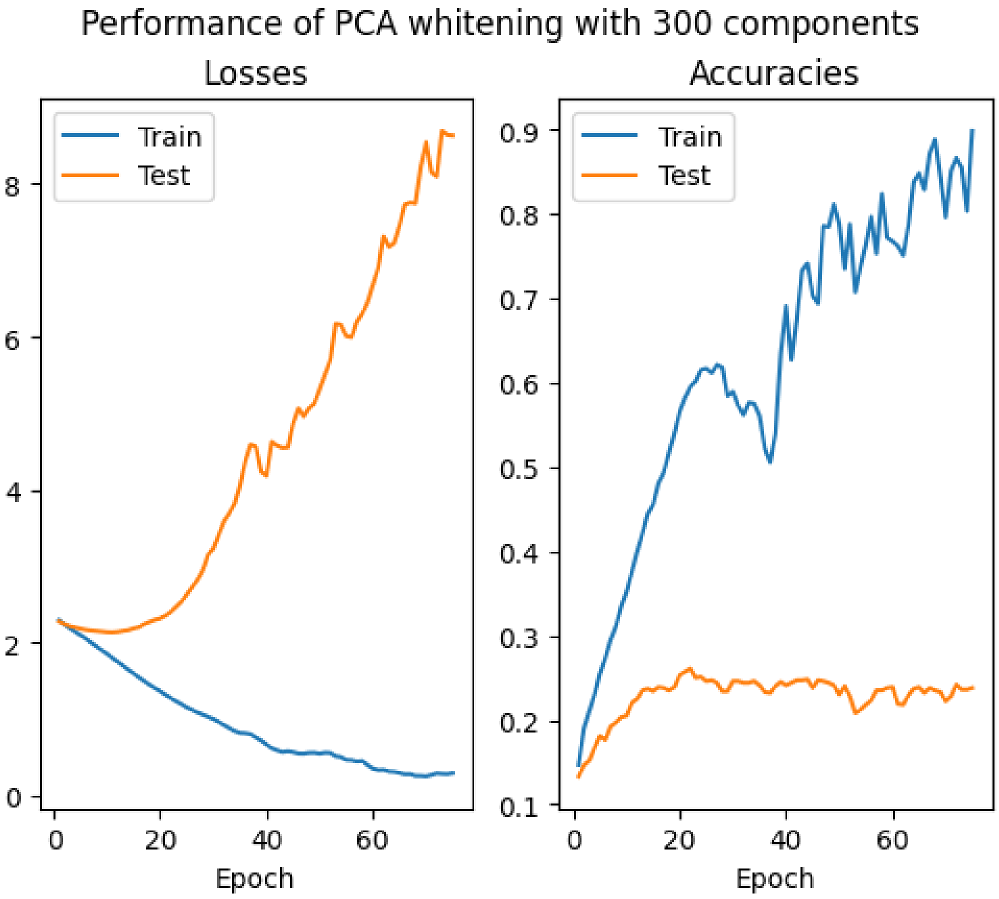
Using a dropout layer with probability 0.5, and the best hyperparameters from the baseline, from the baseline, we reached the following results:



The final train loss was 1, the final test loss was 1.51, the final train accuracy was 0.64 and the final test accuracy was 0.47.
We can see the test results are pretty similar to that of the baseline, while the results on the training are worse.

3.5 Section 5

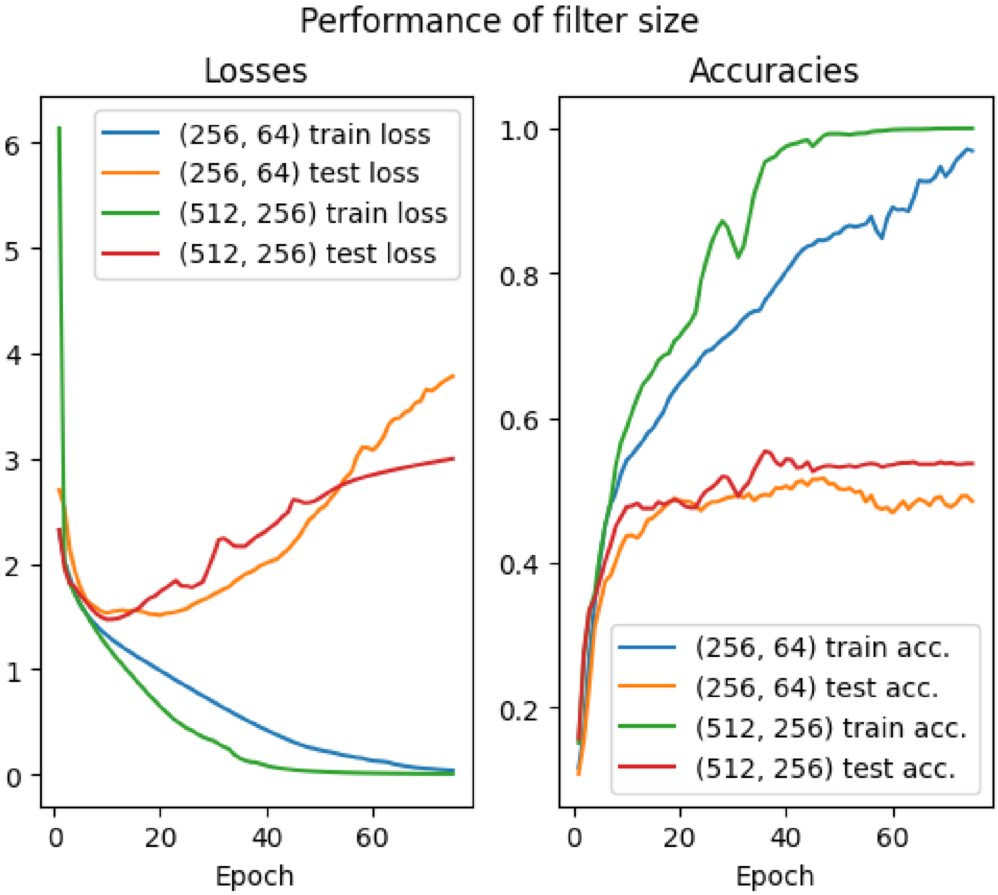
Using PCA with 300 components, and the best hyperparameters from the baseline, from the baseline, we reached the following results:



The final train loss was 0.3, the final test loss was 8.63, the final train accuracy was 0.9 and the final test accuracy was 0.24.
We can see that the transformation results in worse performance when comparing to the baseline. This seems to resonate with the fact that a CNN ”captures relations” between the features (using the various filters), and thus transforming and discarding some of the features hinders this ability.

3.6 Section 6

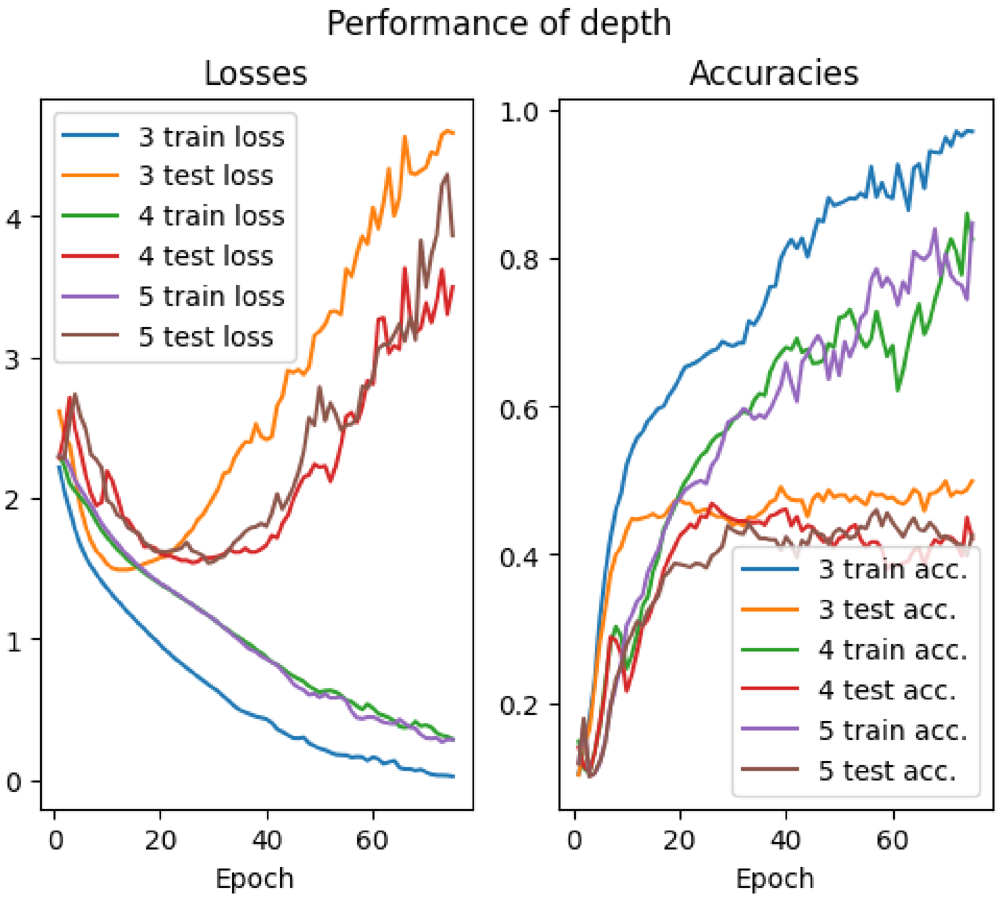
Using various filter size sets, and the best hyperparameters from the baseline, we reached the following results:



For filter sizes $256, 64$, the final train loss was 0.04, the final test loss was 3.78, the final train accuracy was 0.97 and the final test accuracy was 0.48.
For hidden width $512, 256$, the final train loss was 0.006, the final test loss was 3, the final train accuracy was 1 and the final test accuracy was 0.53.
We can see the greater filter sizes do improve the test performance (over the baseline), but not in a significant manner.
The training performance however improves significantly (with virtually perfect accuracy). This result obviously comes from overfitting to the training data.

3.7 Section 7

Using various amounts of convolutional layers, with alternating filter sizes (between 64 and 16), and the best hyperparameters from the baseline, we reached the following results:



For 3 layers, the final train loss was 0.03, the final test loss was 4.59, the final train accuracy was 0.97 and the final test accuracy was 0.5.
For 4 layers, the final train loss was 0.29, the final test loss was 3.5, the final train accuracy was 0.82 and the final test accuracy was 0.42.
For 5 layers, the final train loss was 0.29, the final test loss was 3.86, the final train accuracy was 0.84 and the final test accuracy was 0.42.
We can see the greater amount of convolutional layers don't make much impact over the baseline (in terms of test performance).
The training performance however improves significantly. This result obviously comes from overfitting to the training data.