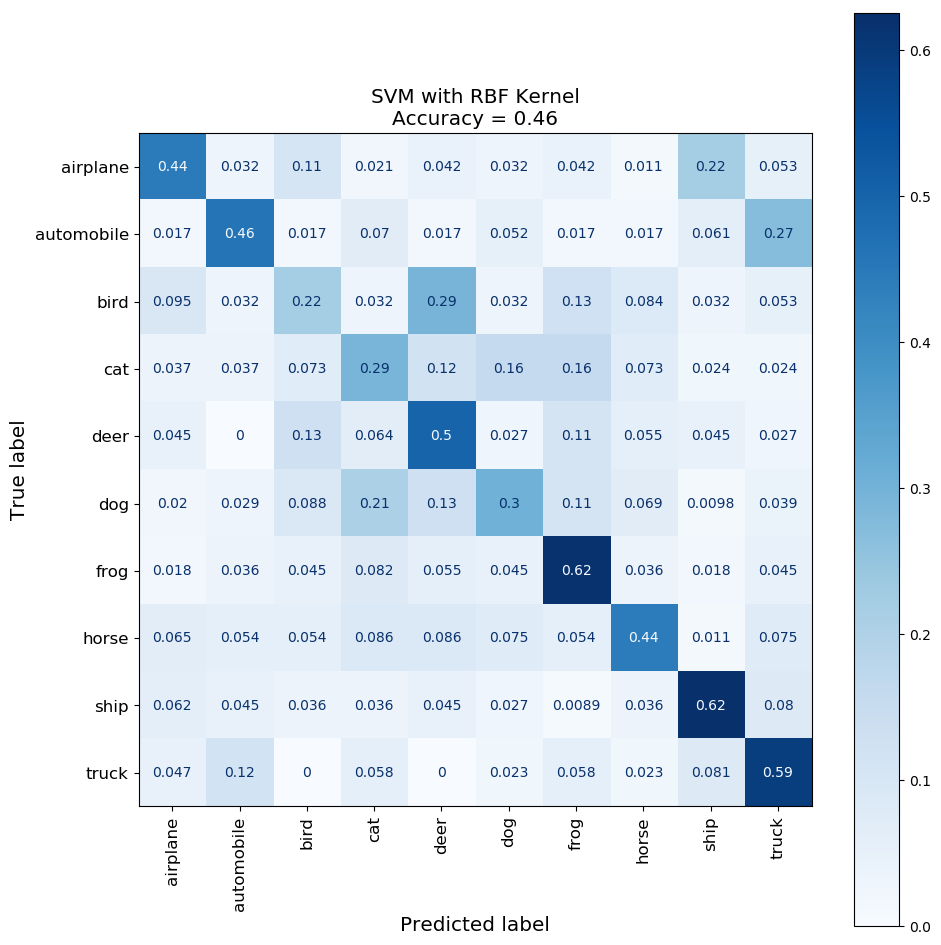
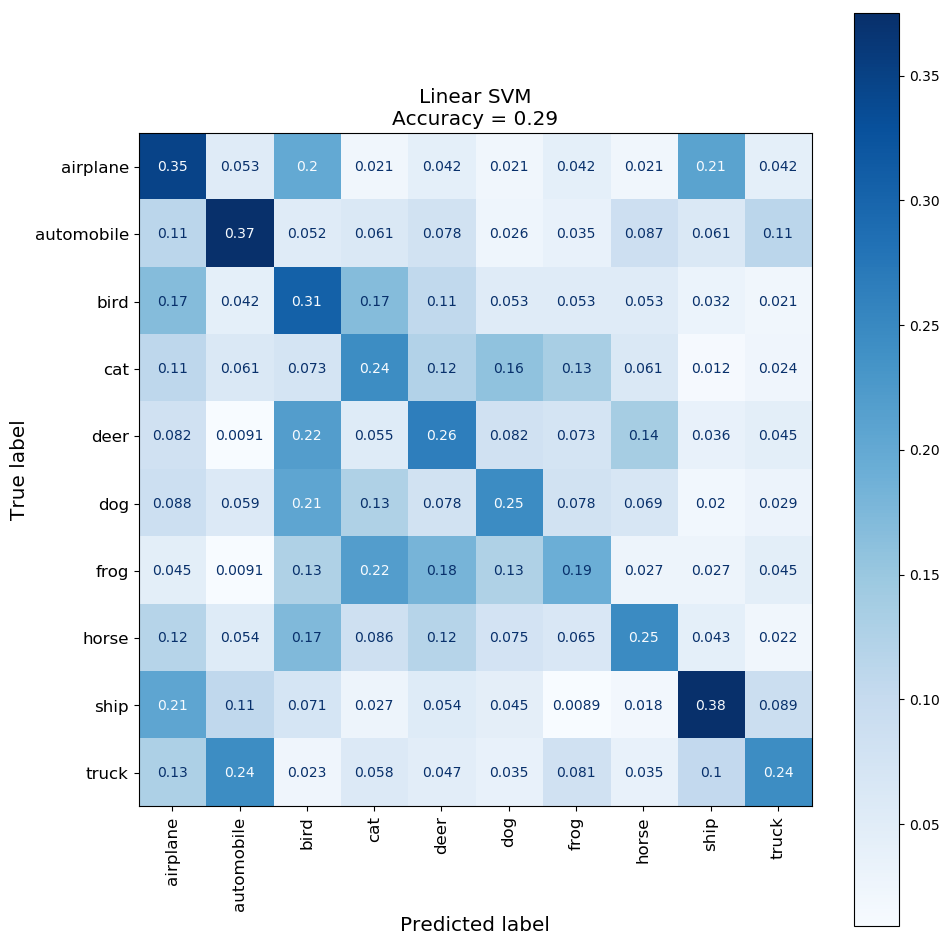
**Assignment 1**

Tomer Ronen 308492909 & Dana Cohen 314865973

**Part 1 - Setup and Baseline**



**Part 2 - Feed Forward Neural Network**

1. **Baseline**

We tried 3 different values for the learning rate, SGD momentum and the standard deviation of the Gaussian weight initialization.

Our choice: learning\_rate=0.001, sgd\_momentum=0.9, init\_gaussian\_std=0.01

We chose this configuration since it had a high maximal test accuracy, stable convergence, and also because its different components (lr, momentum and std) were involved in many successful models.

In retrospect, we ran many of our experiments (subparts 2-7) again with another high-scoring configuration (lr=0.01, momentum=0.5, std=0.01) but we decided to stick with our original choice since the second configuration was less stable, probably due to the higher learning rate and lower momentum.

Interesting observations:

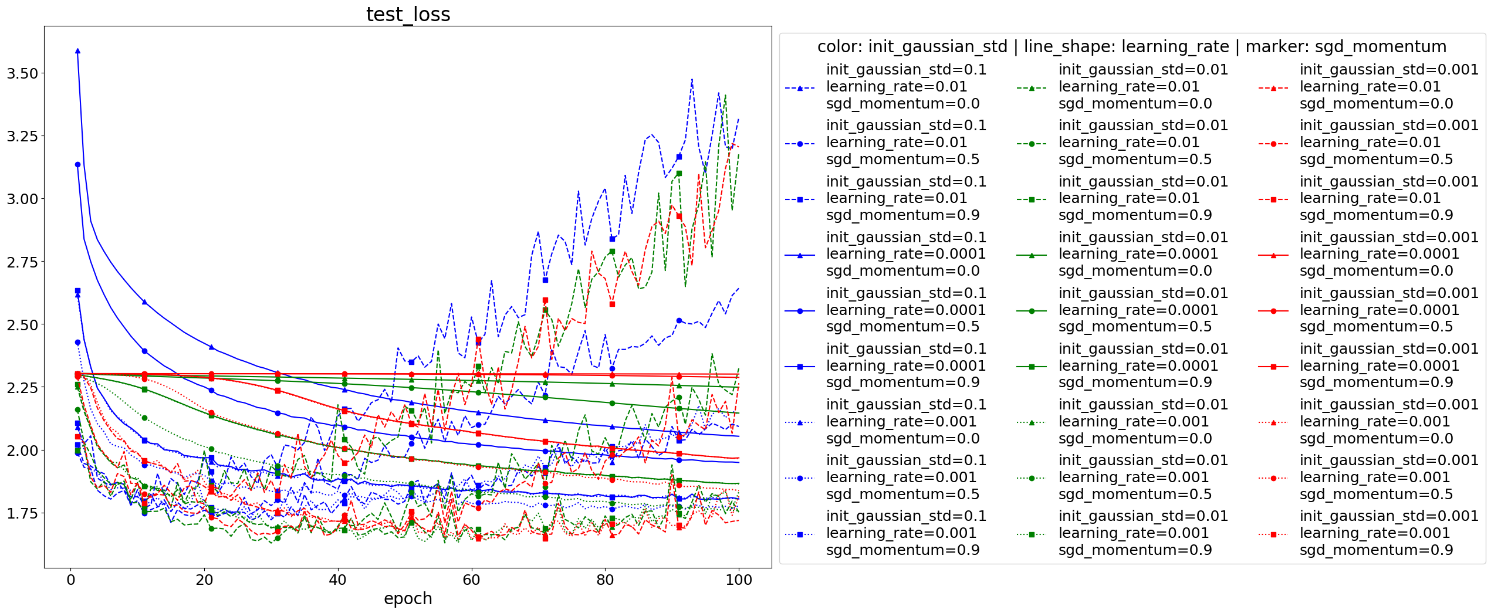
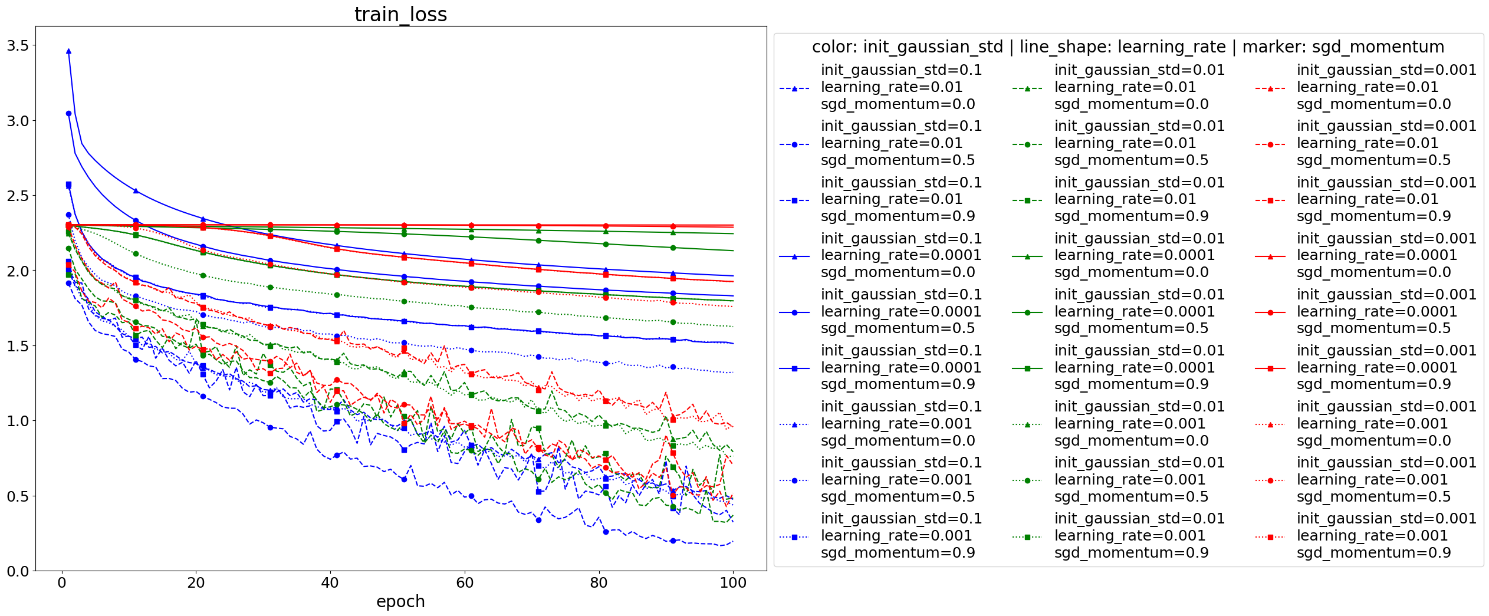
* As we can see from the loss plots, the models most prone to overfitting were the ones with high learning rate and high momentum. (dashed squares)
* High test loss doesn't necessarily result in low test accuracy (most notably, dashed squares). This phenomenon repeats itself in many of our experiments. This is indeed mathematically feasible, as the cross entropy loss can get worse or better without actually changing the predictions of the model, as long and the argmax stays the same.
* Low learning rate resulted in very slow (if any) convergence, especially when paired with a low Gaussian STD. This is probably due to very small gradients, as can be seen from the plot of the gradient's L2 norm (notice the solid lines). Also notice the drastic effect of the Gaussian STD on the gradients (line color).

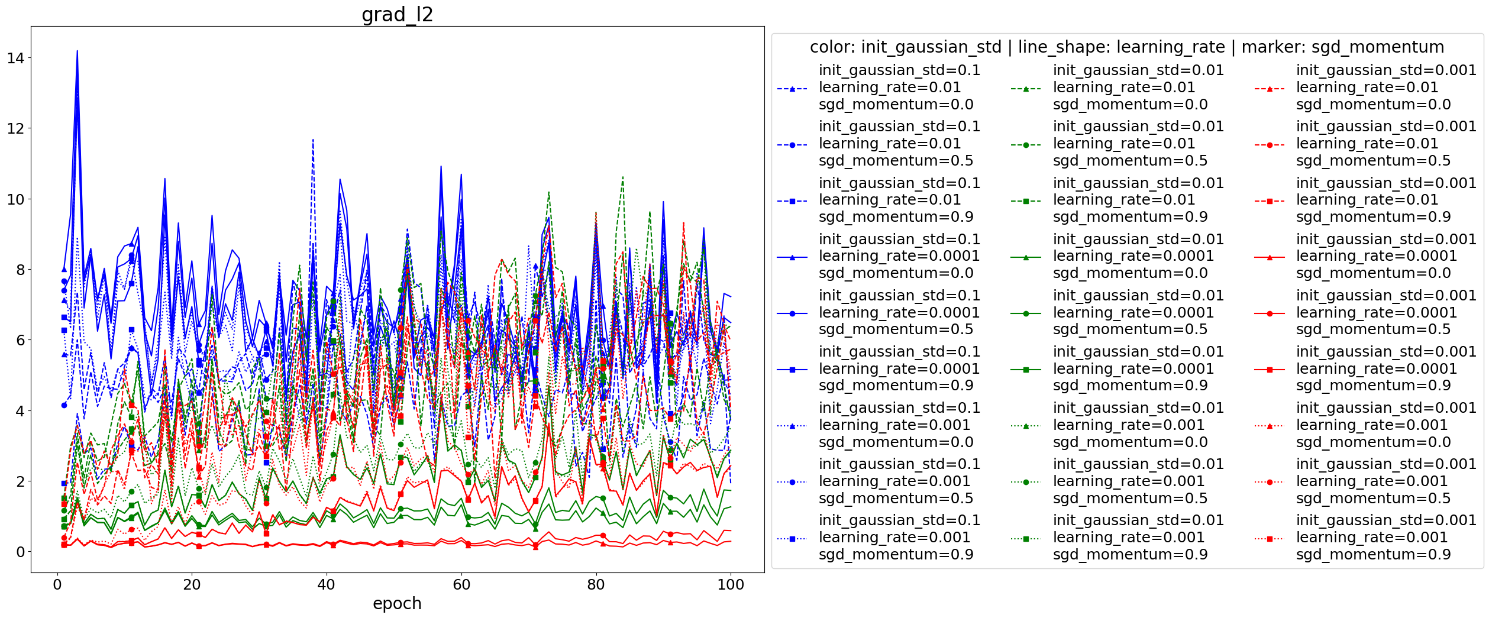
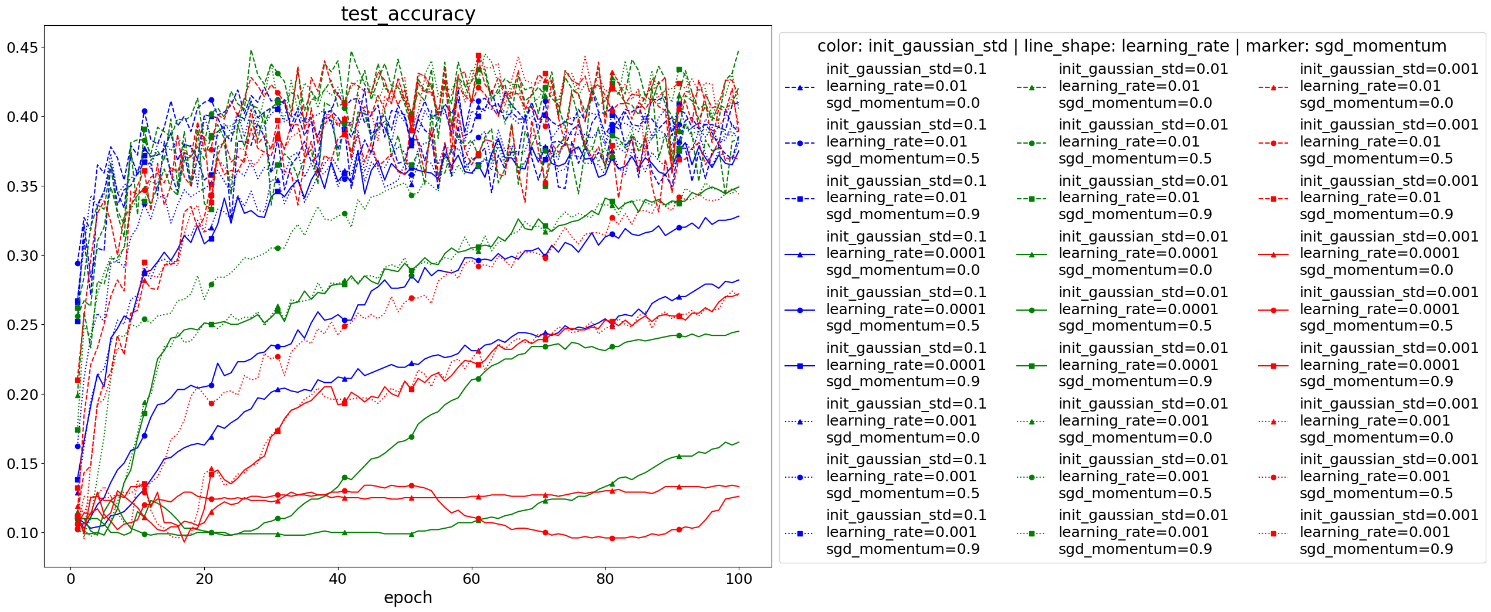
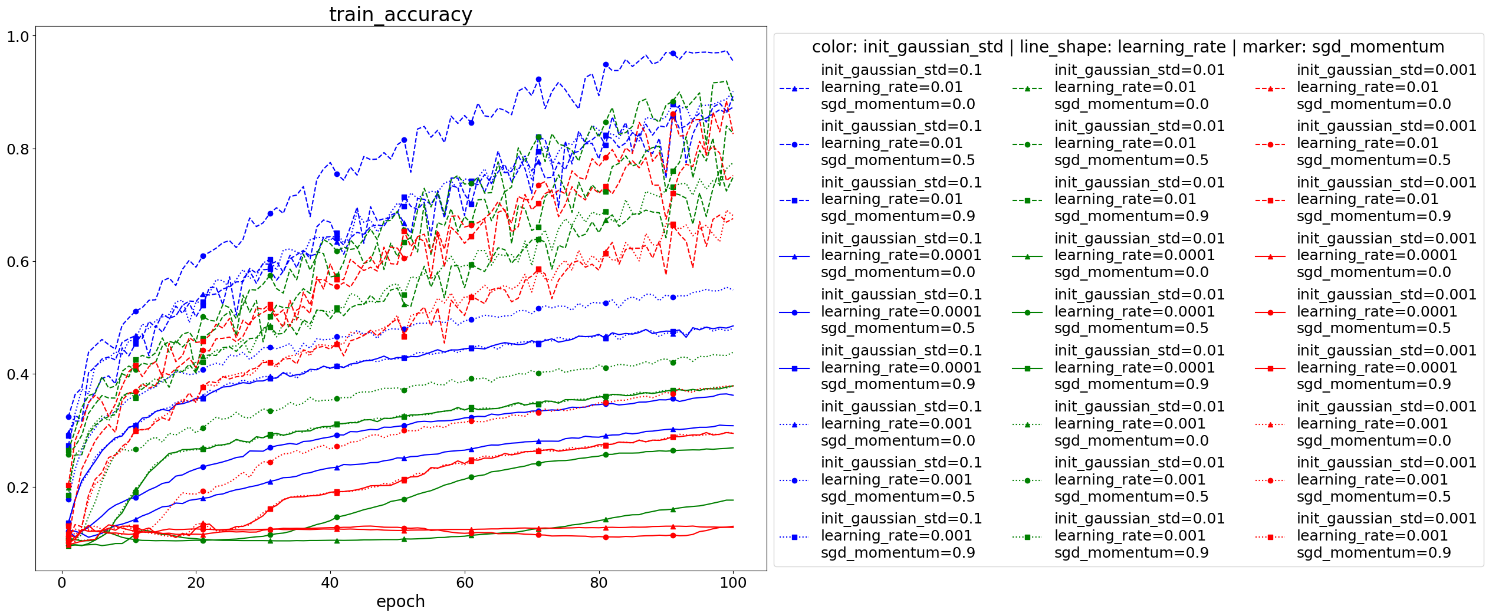
Best Models:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **init\_gaussian\_std** | **learning\_rate** | **sgd\_momentum** | **test\_accuracy** | **train\_accuracy** |
| **1** | 0.01 | 0.01 | 0.5 | 0.45 | 0.52 |
| **2** | 0.01 | 0.01 | 0 | 0.45 | 0.75 |
| **3** | 0.01 | 0.001 | 0.9 | 0.44 | 0.62 |
| **4** | 0.001 | 0.001 | 0.9 | 0.44 | 0.54 |
| **5** | 0.001 | 0.01 | 0 | 0.44 | 0.54 |
| **6** | 0.001 | 0.01 | 0.5 | 0.44 | 0.57 |
| **7** | 0.01 | 0.01 | 0.9 | 0.42 | 0.79 |
| **8** | 0.1 | 0.01 | 0.5 | 0.42 | 0.66 |
| **9** | 0.1 | 0.01 | 0 | 0.42 | 0.65 |
| **10** | 0.1 | 0.001 | 0.9 | 0.41 | 0.88 |

Hyper parameter statistics:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | **test\_accuracy** | | | **train\_accuracy** | | |
|  |  | **median** | **min** | **max** | **median** | **min** | **max** |
|  |  |  |  |  |  |  |  |
| **init\_gaussian\_std** | **0.001** | 0.35 | 0.13 | 0.44 | 0.38 | 0.13 | 0.57 |
| **0.01** | 0.40 | 0.17 | 0.45 | 0.43 | 0.17 | 0.79 |
| **0.1** | 0.39 | 0.28 | 0.42 | 0.54 | 0.31 | 0.88 |
| **learning\_rate** | **0.0001** | 0.27 | 0.13 | 0.38 | 0.29 | 0.13 | 0.45 |
| **0.001** | 0.39 | 0.27 | 0.44 | 0.48 | 0.30 | 0.88 |
| **0.01** | 0.42 | 0.40 | 0.45 | 0.65 | 0.50 | 0.79 |
| **sgd\_momentum** | **0** | 0.35 | 0.13 | 0.45 | 0.37 | 0.13 | 0.75 |
| **0.5** | 0.39 | 0.13 | 0.45 | 0.43 | 0.13 | 0.66 |
| **0.9** | 0.41 | 0.27 | 0.44 | 0.54 | 0.29 | 0.88 |



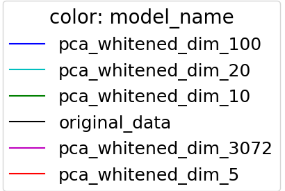


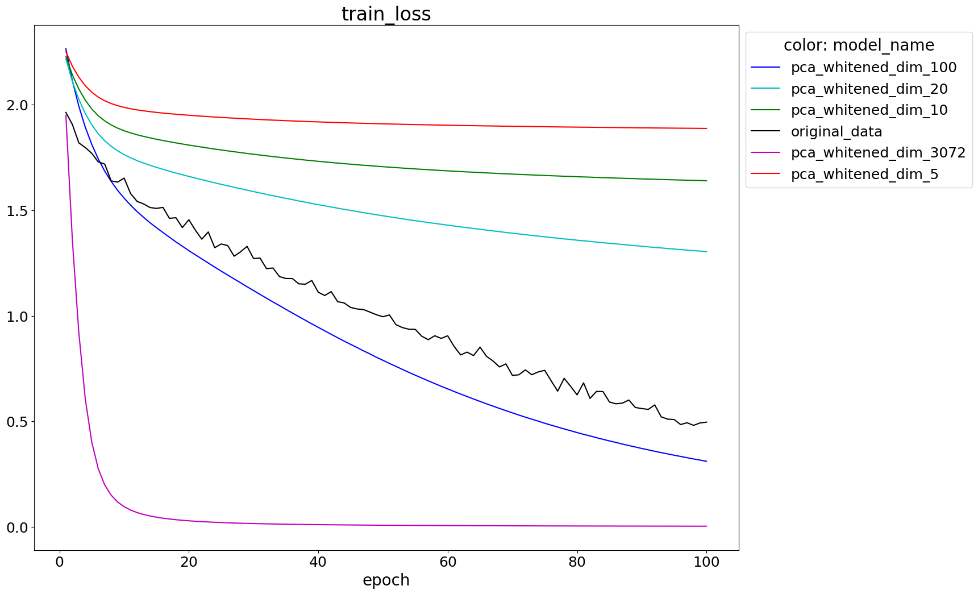
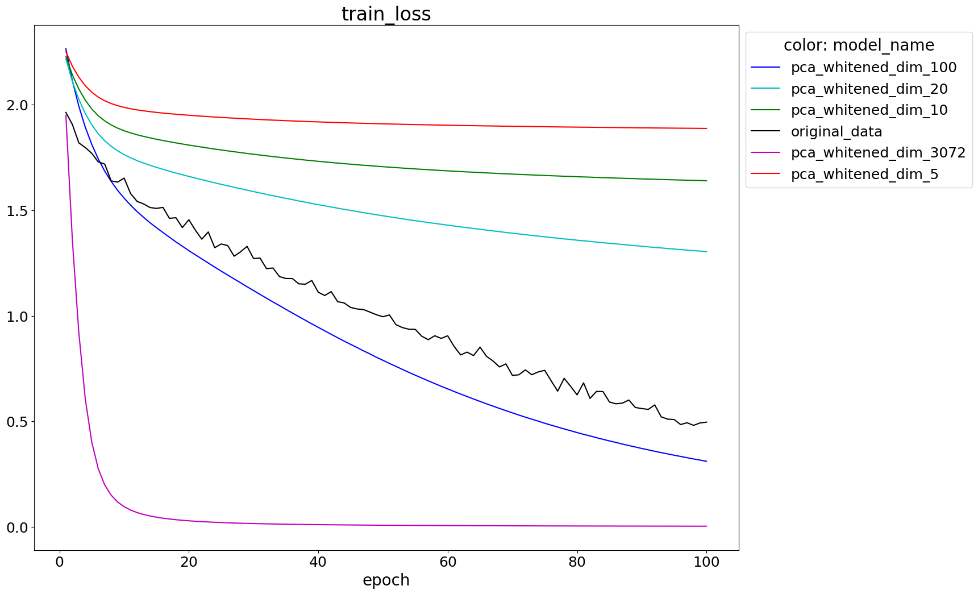
1. **Optimization**

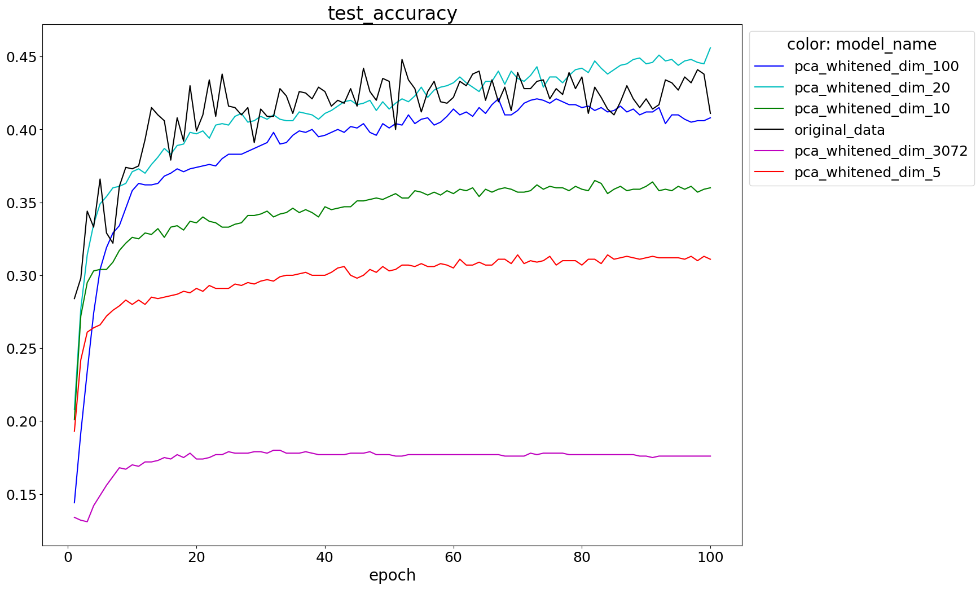
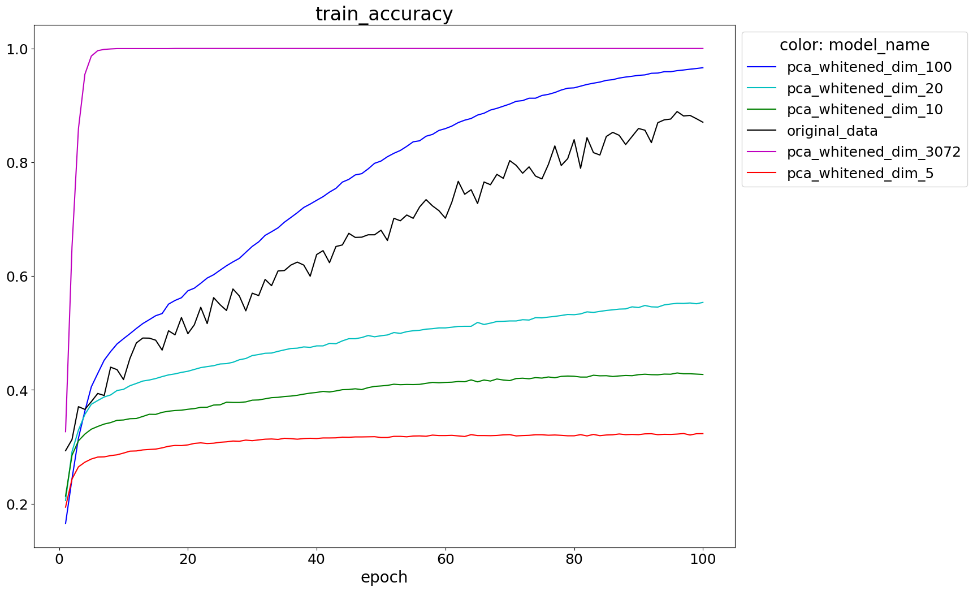
The net trained with the Adam optimizer converges much faster than SGD, but after enough epochs SGD reaches better results, in terms of both loss and accuracy. These results reflect a conventional wisdom in the deep learning community, which leads many practitioners to train their nets using Adam and then switch to SGD for those tricky final percents.

1. **Initialization**
2. **Regularization**
3. **Preprocessing**

We fitted the PCA model on the training data and performed the transformation on both training and test data. When we used all 3072 components, results on the training set (loss and accuracy) were excellent, but results on the test set were horrible. We believe that this effect is caused by the small amount of training data, which results in overfitting of the PCA model. Since there are only 5000 samples for 3072 components, the lower variance components don't represent the data distribution well, but instead capture noise and artifacts in the training data.

****When we used fewer components, results were comparable to the regular results (without preprocessing), but the loss and accuracy graphs were much smoother than our regular graphs. Results for dimensions 10, 20, 100 were pretty good, with 20 being the best, slightly surpassing the original results. 5 components didn't capture enough of the data's variance, and provided poor results.



****

1. **Network Width**
2. **Network Depth**