Fashionable Learning

Sondre Kongsgaard
Morten Lie
ECE 283
UCSB Spring 2018

1 Introduction

1.1 Problem Formulation

The goal of this project was to implement several networks for a given data set in order to enhance insight into the properties of optimal versus non-optimal networks. Further analysis on tuning strategies was done as well as a comprehensive comparison between the networks performance.

1.2 Dataset

The data set chosen for this project was Zalando Research's Fashion-MNIST[1]. This dataset was chosen because of its simple structure, yet high complexity that would make it easier to distinguish networks with good performance from the others. A subset of the data set is displayed in fig. 1, which contains a total of 10 classes of clothing types. Furthermore the data set is composed of 60,000 training samples, 5,000 validation samples and 10,000 test samples. A data set of this size would give us the opportunity to train various deep networks.

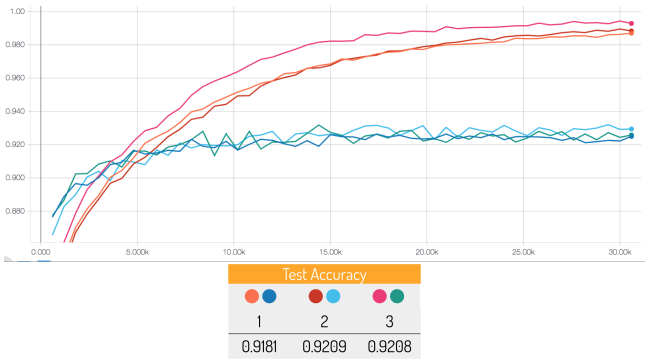
Figure 1: *The dataset used: Fashion-MNIST. Simple and effective for our purpose.*

1.3 Models

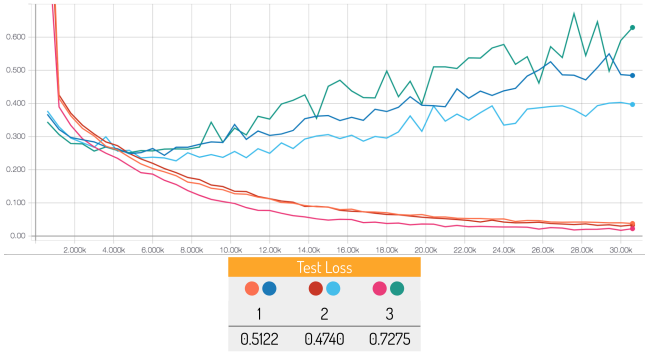
The models that were considered for this project were neural networks with varying complexity. This would give us insight to how complexity is correlated with performance for this particular dataset. From the shallow neural network side, a simple logistic regression model was implemented. As progression into deeper networks were made three artificial neural networks were implemented with one, two and three layers. Each model ending in a readout layer. Further, three convolutional neural networks were implemented, also with one, two and three convolutional layers. Each of these models had a fully connected layer followed up by a readout layer at the end. Lastly, a VGG16 model was implemented which is composed of 13 convolutional layers with ReLU activation, 5 pooling layers and 3 fully connected layers.

1.4 Training

To ensure convergence of the training accuracy, the training was done with 50 epochs. Each epoch had 600 iterations with a batch size of 100 samples in order to train over the entire training set. TensorFlow's visualization tool TensorBoard was used for analysis of the results, which will be discussed in the next section.



(a) *Accuracy*



(b) *Loss*

Figure 2: *Plots of accuracy and loss for the CNNs with one, two or three convolutional layers. Training sets are plotted in red, and validation sets are plotted in blue.*

2 Analysis and results

2.1 Performance of the Implemented Models

The deeper and more complex networks were expected to give better results, and a comparison among them proved this assumption to be correct. However, the increase in accuracy was nearly negligible from a simple CNN implementation to VGG16. On the other hand, the training time increased from about 5 minutes

to as much as 11 hours. CapsuleNet was also implemented, but this network was never trained due to its extremely high memory usage and training time. Training of the Logistic regression model resulted in unsatisfactory performance. Neither the accuracies nor the losses stabilized no matter how many epochs that were used. This indicated that the dataset had a higher complexity than what the network could handle.

After training several models, it became apparent that tuning hyperparameters had a greater effect on a networks performance, rather than increasing its complexity. This is possibly a result that is linked to the complexity of the dataset.

Further analysis is thus based on the CNNs with up to three convolutional layers.

2.2 Analysis of CNNs

An insight from this project is that neural networks do not simply give better results if more layers are introduced. Even though the implemented CNNs had different numbers of convolutional layers, their test accuracies were almost identical. This result can be seen in fig. 2. A closer look at the model neurons (weights, biases) in the different layers is needed to understand why more layers did not give better results.

Histograms is a useful tool to see how the model neurons change during training. In fig. 3 the distribution of weights and biases for each layer is plotted. The histograms are read in the following way: The height of each point on the histogram is the number of neurons that have the specific value as given by the horizontal axis. The darker to brighter histograms show how the distribution change as the training progresses. In fig. 3 a slice is added at the end of each epoch during training.

For the first layer in fig. 3, it can be seen that the distribution of both biases and weights are changing over time. The layer is learning and changing the value of the neurons to better adapt the training set. However, this is not the case for the next three layers. For them, only the bias change noteworthy, while the weights stay as bell curves from start to finish. Furthermore, this bell curve comes from the initialization of the weights (a normal distribution), which indicates that for their corresponding layers next to no learning is occurring. The limited increase in accuracy for this network compared to the CNN with only one convolutional layer is likely to only come from the added biases.

In the next section the efforts to improve the performance of a model by tuning its hyperparameters is presented.

2.3 Tuning strategies

In order to maximize the test accuracy and minimize the test loss, multiple tuning strategies were tried out on the convolutional neural network with one layer. Before tuning the networks we noticed a significant amount of overfitting. As one may observe in fig. 2 the validation loss starts to increase after the validation accuracy had plateaued. Even though the validation accuracy stoped growing, the training accuracy kept converging to identity. This is a classical sign of overfitting, which should be reduced by tuning the model's hyperparameters.

Batch normalization

Batch normalization is done in order to minimize the variance in the hidden layers. This speeds up learning and helps to minimize the overfitting discussed earlier. After implementing batch normalization between the activation and max pooling in the

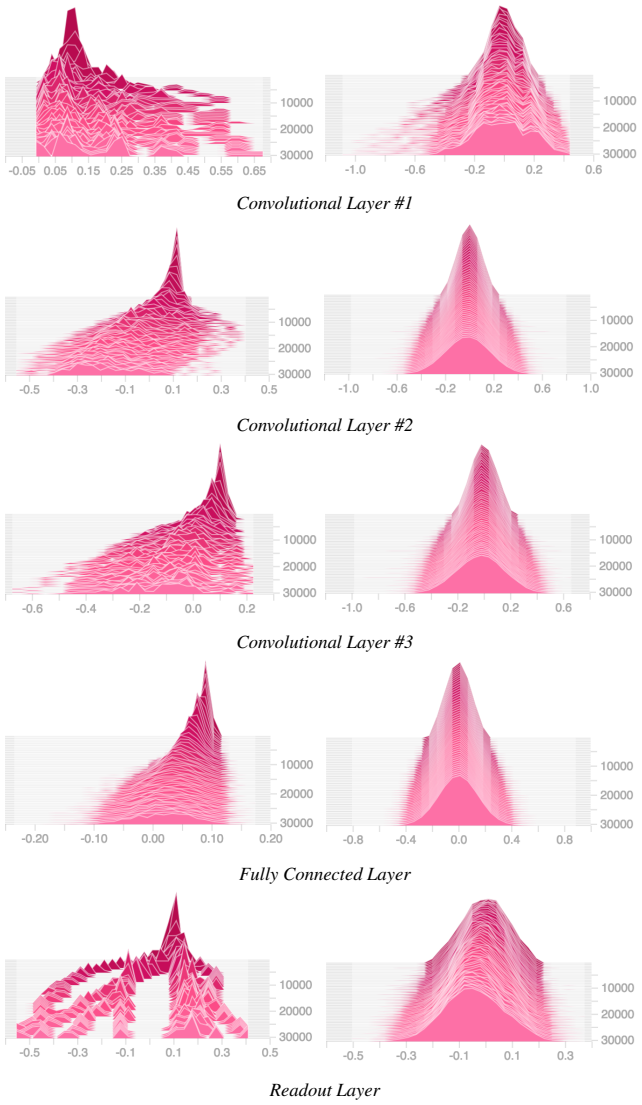


Figure 3: *Histogram distribution of biases (left) and weights (right) during training for the CNN with three convolutional layers.*

convolutional layer, the training accuracy dropped from 99% to 97% as well as a reduction of 20% in the validation loss. This benefited the model as the validation and test accuracy stayed the same.

Regularization

Experiments with L2 regularizations were conducted, but they did not result in any less overfitting. Rather, with regularization the overfitting behavior was about the same as without regularization, but both the training and validation accuracy were reduced. More epochs during training were also needed to reach high accuracies. Tuning the regularization parameters did not improve the output, so in the end it was removed from the model.

Dropout probability

Dropout probability is the likelihood of an individual neuron being ignored in the forward- and backward propagation, which is done at every iteration in the training phase. Even though this reduces

the number of neurons that are being considered in the training, the network will still be able to learn the same robust features as before. In fact, tuning the dropout probability right will increase the test accuracy and reduce the amount of overfitting. As shown in fig. 4, different values of dropout probability was tried out in order to maximize the test accuracy. The best dropout probability for this model turned out to be 0.7, which gave us a test accuracy of 92.22%.

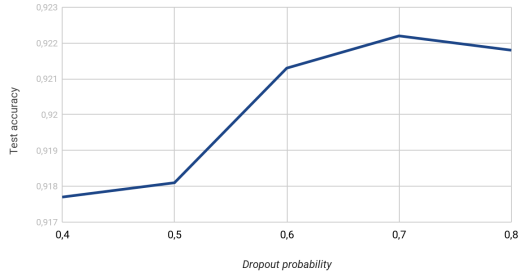


Figure 4: *Resulting test accuracy for varying dropout probabilities. The maximum is for a dropout probability of* 0.7.

Learning rate

After the models were implemented, different learning rates were tried out in order to maximize test accuracy. We noticed that if the learning rate was too high, the model would get stuck with a relatively high loss and yield a poor test accuracy. Furthermore, if the learning rate was set too small, the accuracies would need a long time to stabilize at non-optimal levels. After trial and error, the learning rate value we ended up using was 0.001. This value gave us the best test accuracy, but as one may see in fig. 2, the validation accuracies oscillates after it plateaus. This indicates that the learning rate made the algorithm take too long steps at this point, giving room for improvement in the final epochs. Our solution was to implement an adaptive learning rate, which decreases exponentially with time according to the following equation.

$$L[k] = L[0]d^k \tag{1}$$

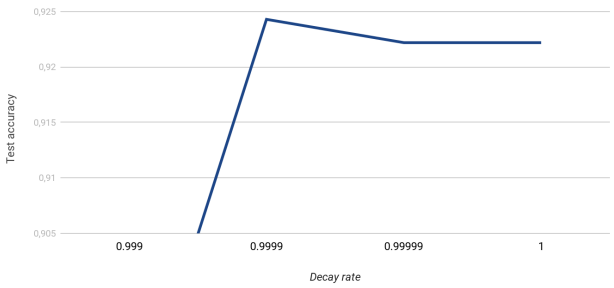
where $L[k]$ is the learning rate at step k and d is the decay rate which had to be tuned. We used the most successful static learning rate as the initial value, and tried out different decay rates. The test accuracy is plotted as a function of the decay rate that was put to the test in fig. 5a. Clearly an optimal value would be $d = 0.9999$, which gave us the exponentially decreasing learning rate displayed in fig. 5b.

After all the tuning strategies were implemented into the model, the training yielded a test accuracy of 92.43% which is at the same level as the state-of-the-art models with similar complexity presented by Zalando Research.

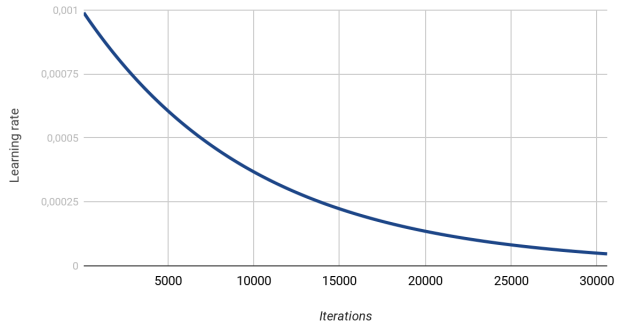
3 Conclusion

We presented neural networks of varying complexity, and an in-depth analysis of the performance of CNNs with up to three convolutional layers. By using visualizations such as filter images and histograms we got insights about how networks evolve and change during their training.

We obtained state-of-the-art results in line with the benchmarks as given by Zalando Research. This project shows that model complexity should be adapted to the complexity of the dataset, and that tuning a model with fewer layers could give better results than choosing a deeper network.



(a) *Test accuracy with different decay rates*



(b) *Adaptive learning rate curve*

Figure 5: *Improvement of the test accuracy was accomplished by introducing an adaptive learning rate with an optimized decay rate*

4 Role played by each group member

For us, the most important factor in this project was that we were interested and wanted to maximize our learning outcome just from the sheer nature of the topics covered. Both of us came with ideas and experiments that have driven the work forward. We did not try to split everything equally, but in the end it seems like the effort shown from each team member is about the same.

During the initial phase, we discussed several datasets and possible problem definitions, and we agreed on choosing a project where we would learn the most about general strategies in deep learning. Sondre set up a project structure for our codebase, in addition to some initial models. Morten implemented the CNNs, including VGG16. Both contributed to the visualization code, and to the tweaking and tuning needed for the models.

The project code can be found at `https://github.com/kongsgard/FashionableLearning`.

The project presentation can be viewed at `http://bit.ly/FashionableLearning`.

References

[1] Han Xiao, Kashif Rasul, and Roland Vollgraf. *Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms*. Aug. 28, 2017. arXiv: `cs.LG/1708.07747 [cs.LG]`.