Classifying Fashion Items using a Convolutional Neural Network

Luis Gilbuena
CSC650: Neural Networks
Soumyashree Sahoo

Quinnipiac University
Hamden, CT

Abstract

This project addresses the task of classifying grayscale images of different clothing pieces, provided by Kaggle, into separate labels using a Convolutional Neural Network. This model was trained on a dataset of 70,000 images of clothing items provided by Kaggle, and evaluated on a separate dataset of 10,000 images. This architecture was built with several convolution layers, with the cross-entropy loss function, and the Adam optimizer over 20-30 epochs testing different hyperparameters to further optimize the results of this project. Throughout the development process, this CNN has achieved accuracies hovering around 92%. Overall, this project can be deemed a success given the final accuracies and a great learning experience on how to build and fine tune neural networks.

Keywords: (Convolutional Neural Network, epochs, optimizers, loss function, dataset)

Chapter 1

Introduction

This project tackles the problem of classifying thousands of images provided by Kaggle using a Convolutional Neural Network developed and fine-tuned from scratch. This project is important because this issue can be used to tackle the same problems on a larger scale. Rather than assigning this classification problem to humans, we can use AI tools to classify more images in a more efficient manner with higher accuracy. This project was developed with the goal of achieving around a final accuracy of 90%. The plan for this project was to develop the base building blocks of a convolution network and test different combinations of hyperparameters to find the best configuration for finding the highest final accuracy.

Chapter 2

Background

Convolutional Neural Networks (CNNs) have become the foundation for modern image classification due to their ability to learn features from raw pixel data. The development of CNNs go back to early work by LeCun et al. in the 90s, specifically the LeNet-5 architecture, showing that convolutional layers with pooling can calculate digit recognition. Since this creation, there have been more advanced/deeper architectures like AlexNet, VGGNet, and ResNet that have proven to be more accurate and efficient across all metrics.

This project specifically focuses on the Fashion-MNIST dataset, consisting of 70,000 greyscale images of size 28x28 of clothing items. This dataset was created to reflect the problem of classifying images with simplicity and size under consideration. As a result, the Fashion-MNIST dataset has become the standard for testing the potency of CNNs and exploration of potential architectures.

The concepts in this project are based off previously established deep learning techniques. The architecture implemented consists of many existing concepts. Convolutional layers in a model will attempt to cover key pattens like edges of an image. The pooling layers will reduce dimensionality while keeping these key features for the purpose of efficiency. Batch normalization will help stabilize and speed up training. Dropout prevents overfitting by disregarding a portion of data during training. When these pieces are put together, they make use for an effective CNN for image classification tasks in the same vein as the Fashion-MNIST dataset.

Though previous works have shown that simple CNN architectures can reach high accuracies using frameworks like Keras or PyTorch, this project builds upon this foundation through thorough experimentation of different architectural choices with vast combinations of hyperparameter values with the goal to improve the final accuracy. Through tuning factors like dropout rate, learning rate, and batch size, this project becomes a strong learning experience of the effects of hyperparameter values on the performance of CNNs.

Overall, this project takes existing concepts from previous CNNs techniques but is more focused on the effects of certain architecture decisions and sets a strong basis of how to fine tune potential networks in the future for tasks more complex than classifying simple greyscale images.

Chapter 3

Methodology/Implementation

This project uses already established CNN methods that are implemented in Python using several machine learning libraries, including NumPy, Pandas, Matplotlib, PyTorch, and Scikit-Learn inside the VSCode development environment. The training and testing datasets were obtained through Kaggle and provided in a CSV format. Each row of the CSV represents grayscale image and its associated class label. The pixel value originally with the value range of 0-255, was further normalized to the range from 0-1 to improve computational efficiency.

After preprocessing, batch sizes were set to 32 for the training set and 64 for the testing set. The normalized images are then passed to the CNN consisting of 4 convolutional blocks with varying kernel sizes, batch normalization, and dropoff. These layers will gather patterns from the images and are then fed into the fully connected classification layer that will output the logits for the ten classes.

For training, this model utilized the CrossEntropyLoss function for multi-class classification to measure our loss values as well as the Adam optimizer with varying weight decay and learning rate. During and after training, several metrics were recorded to identify patterns for testing hyper parameters. Training loss, testing loss, and test accuracy over epochs were plotted across line graphs, as well as calculated labels from the fully trained model were placed on a confusion matrix.

Chapter 4

Results

Testing different combinations of hyper parameters yield many different results. The main data piece this project aimed to tune for was the final accuracy after training the model for a set number of epochs and compare models with vs without dropout. Looking at figures 4.1 & 4.2, where we compare the effects of dropout on the performance, we see signs of overfitting early into training when there is no dropout present compared to the plot that contains dropout where there are potential signs of overfitting later on in the training. Another feature to note between the two plots is the final accuracies under dropout seem to trend more smoothly compared to the architecture with no dropout. Looking at the effects of batch size from figures 4.3 & 4.4, we see that under a higher batch size, there aren't signs of overfitting compared to the plot of low batch size which shows signs of overfitting later on during training. Comparing the plots yielded from a high learning rate to the low learning rate show in plots 4.5 & 4.6, an outstanding piece of information to note is the sporadic jumping in both testing loss and test accuracy under high learning rate, showing the inability to stabilize within 20 epochs compared to a low learning rate where the lines for both boss and accuracy were smooth.

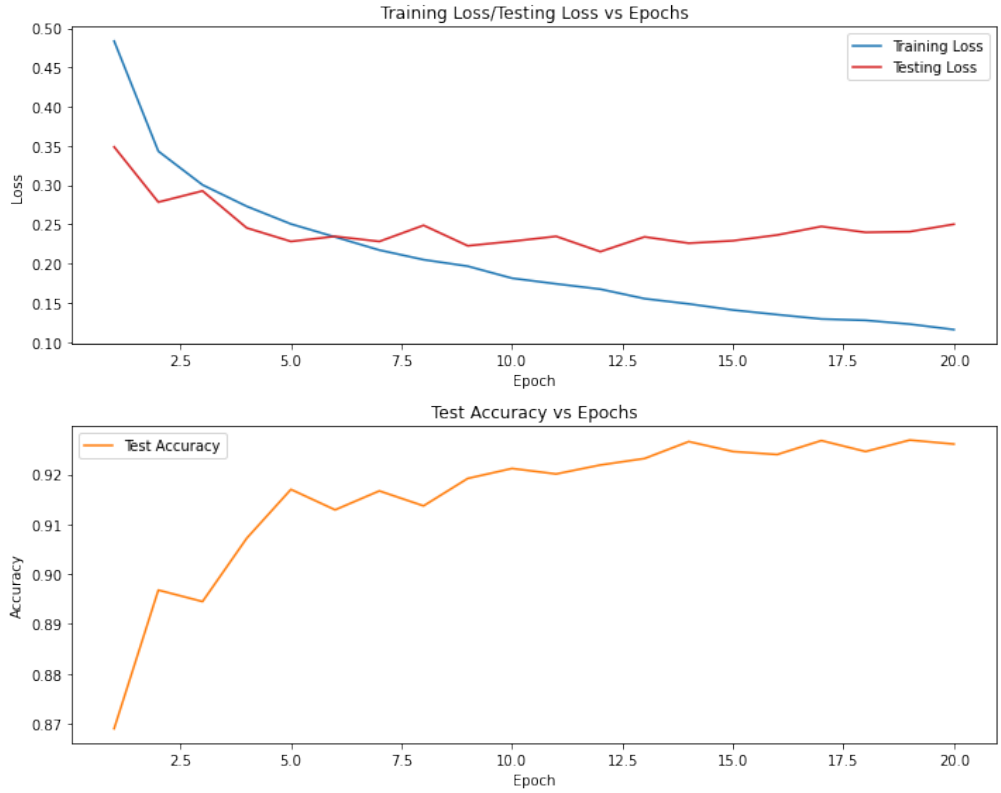

Figure 4.1: Loss & Accuracy vs Epochs w/Dropout

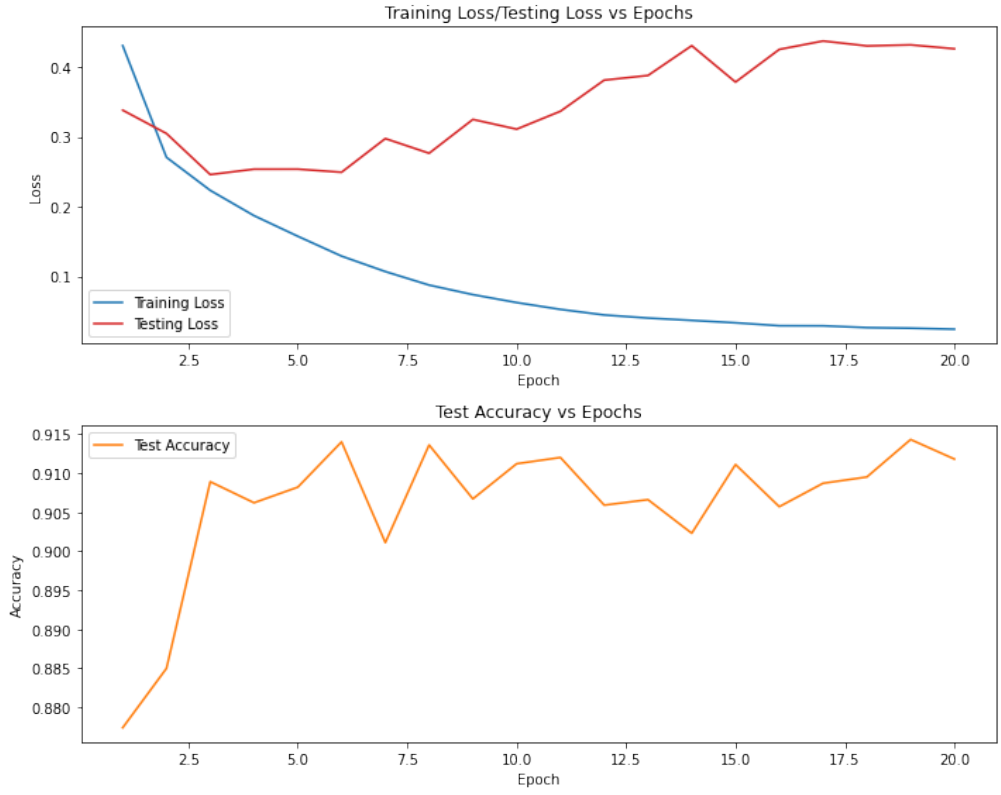

Figure 4.2: Loss & Accuracy vs Epochs without/Dropout

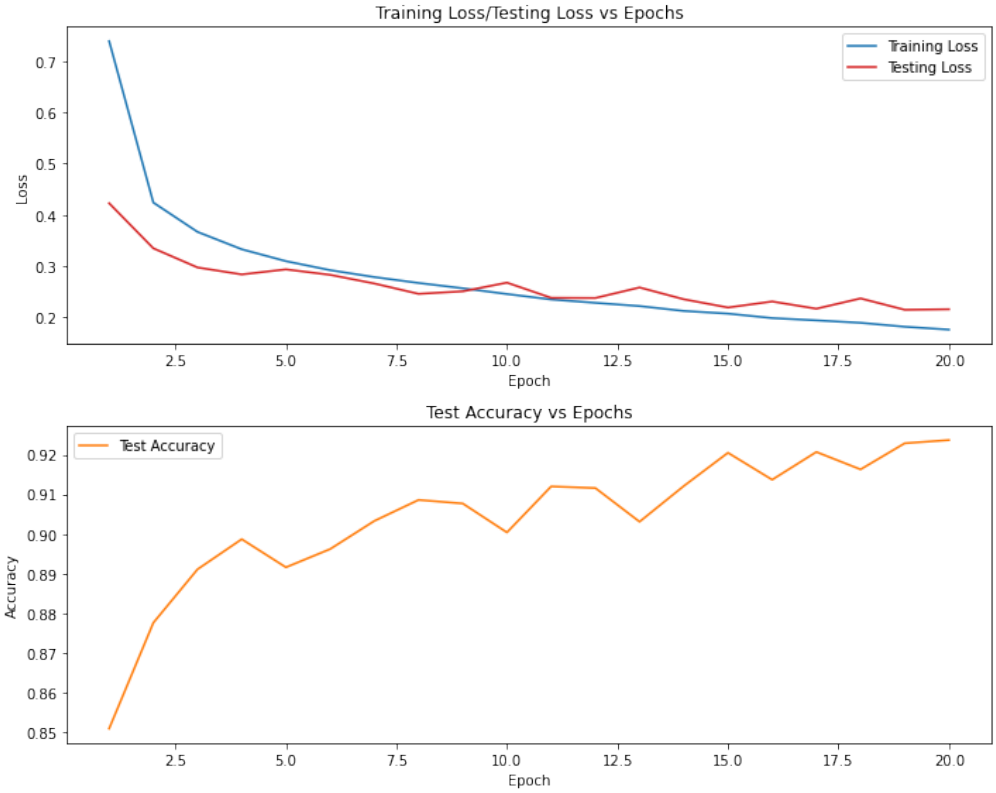

Figure 4.3: Loss & Accuracy vs Epochs high batch size

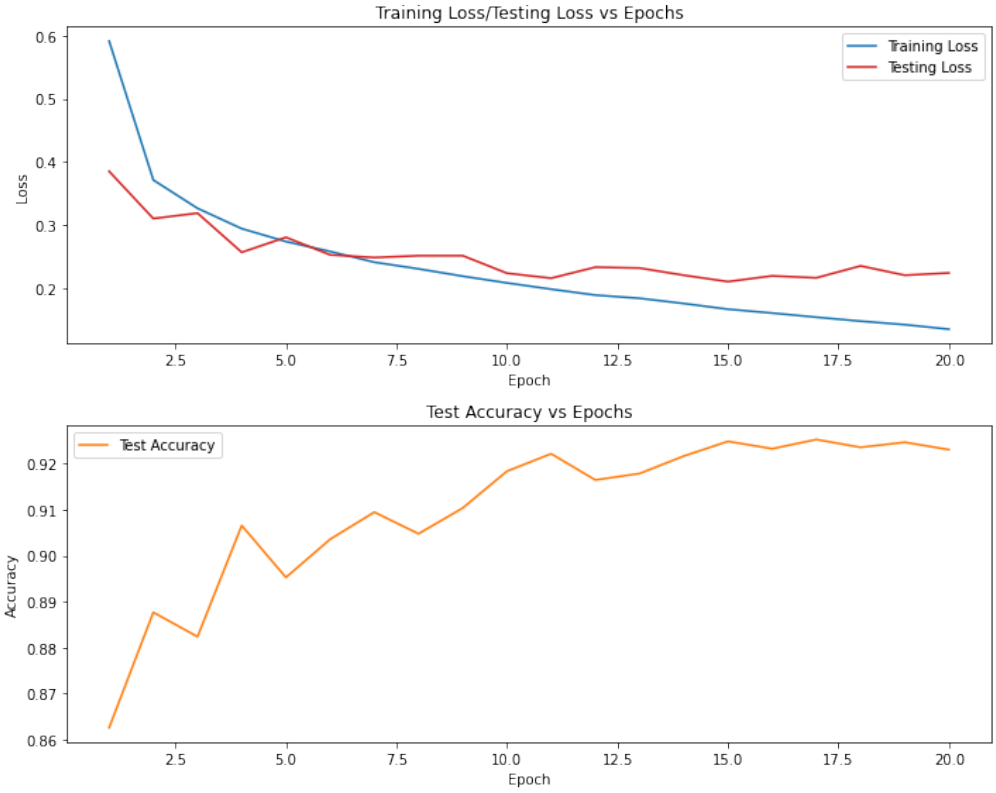

Figure 4.4: Loss & Accuracy vs Epochs low batch size

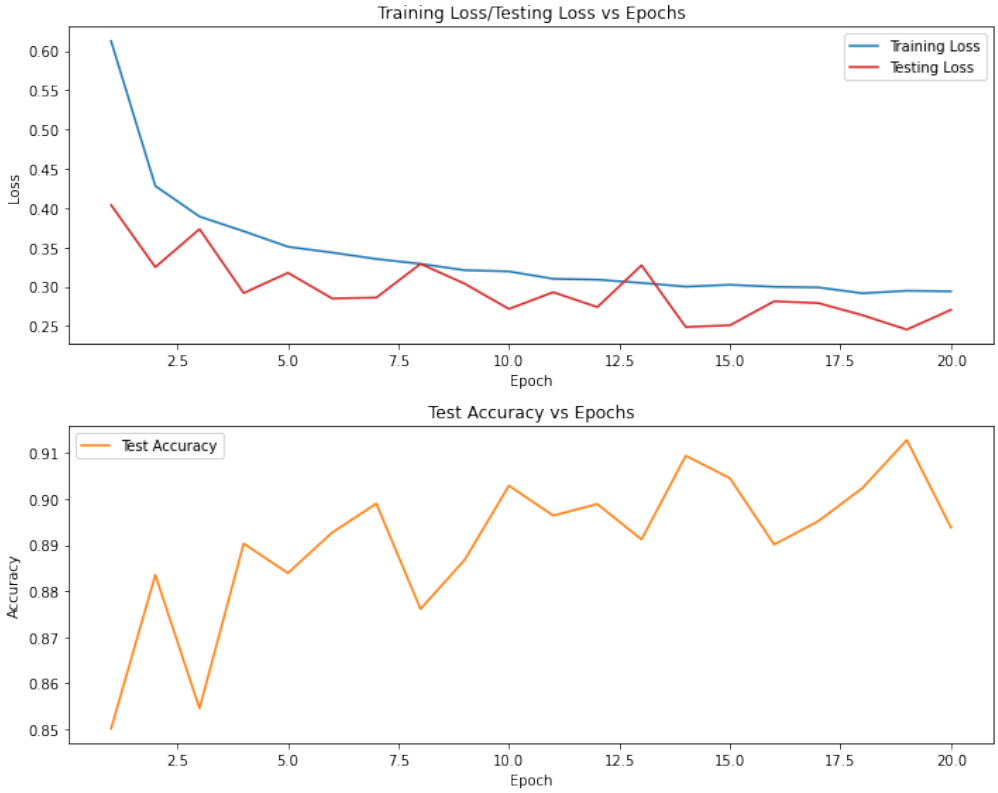

Figure 4.5: Loss & Accuracy vs Epochs high learning rate

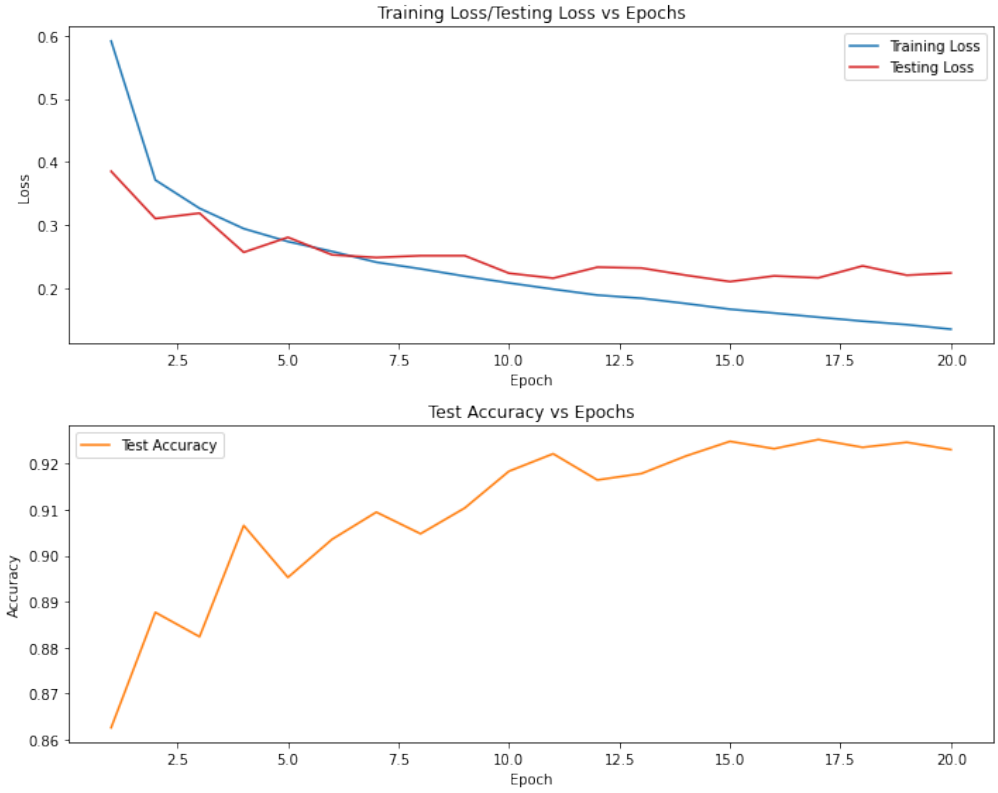

Figure 4.6: Loss & Accuracy vs Epochs low learning rate

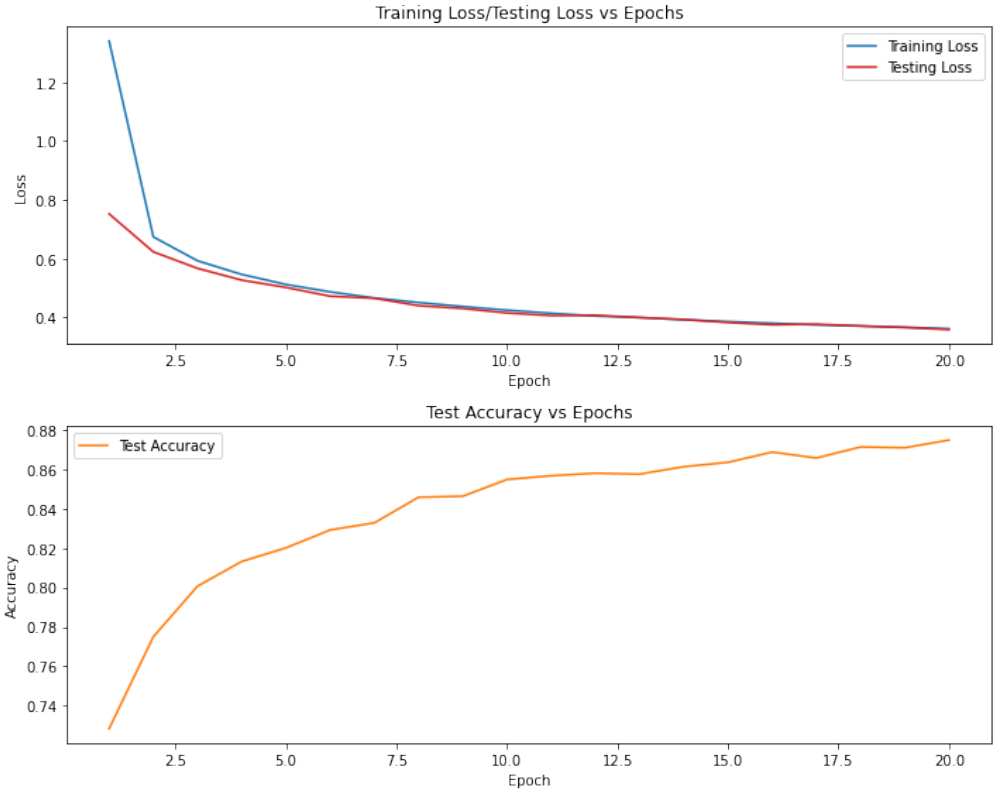

Figure 4.7: Loss & Accuracy vs Epochs (Simple CNN)

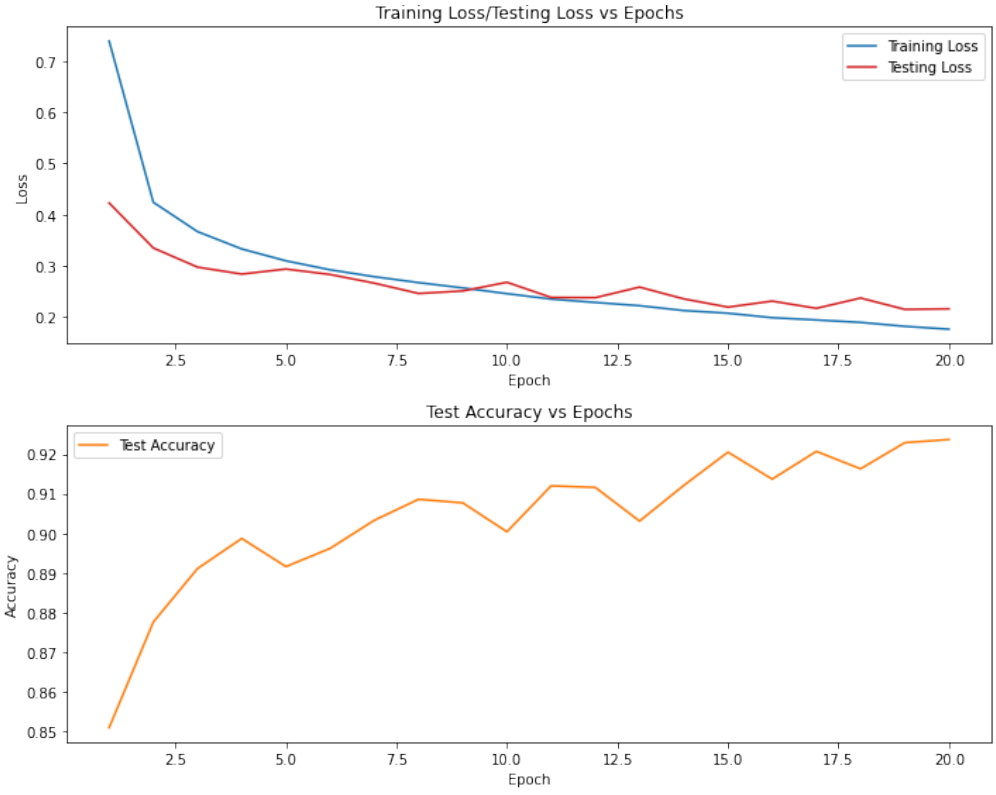

Figure 4.8: Loss & Accuracy vs Epochs (Optimized CNN)

Chapter 5

Analysis and Discussion

This section will analyze the behavior of the CNN across different architectures to evaluate how the configuration of the CNN should be handled as well as the values of the hyperparameter. This discussion will be based on the findings presented from figures 4.1 through 4.6

5.1 Dropout

The comparison between figures 4.1 and 4.2 shows the significant impact of dropout on the model's ability to classify. When applied, dropout causes both the training and testing loss curves to follow identical directions converging to low loss values. As a result, the final test accuracy remains high hovering between 92% and 93% indicating the model is learning representations well beyond the training distribution.

In contrast, the model trained without dropout shows clear signs of overfitting. Though training loss is decreasing in a smooth manner over training, testing loss diverges in the early training phase (around epoch 5). This split between training and testing loss shows that the network is memorizing details specific to the training data rather than patterns found between both datasets.

5.2 Impact of Batch Size

Figures 4.3 and 4.4 show the effects of batch size on the performance of our model. With a high batch size, the training and testing loss curves are smooth and converge towards the lowest observed testing loss from plots all around (approximately .15). The accuracy is similarly stables and achieves a relatively high final accuracy compared to all tested configurations. This could be due to the idea that larger batches provide more accurate gradient estimates.

Compared to high batch sizes, using a low batch seems to add noise into the optimization process with the loss and accuracy curves becoming jagged and irregular. This could be due to the high varied gradient estimates at each step. Though these gradients can help, the training behavior can be difficult to control thus being less efficient. The model's accuracy with low batch size shows that

the optimizer is being affected heavily by unstable gradient directions.

Overall, the comparison between larger and smaller batches show that picking a larger batch size for this case proves to be better for this architecture in terms of final accuracy.

5.3 Learning rate

The testing of the effects of learning rates in figures 4.5 and 4.6 show how the step size affects convergence in both loss and testing accuracy. With a high learning rate, the model makes very optimistic updates every epoch causing the loss and accuracy to spike up and down dramatically. With 20 epochs, it's hard to make anything truly meaningful off this graph which leaves consideration for potentially training the model for more epochs than wanted.

In contrast, a low learning rate creates slow improvements in performance with both training and testing loss decreasing smoothly with minimal fluctuation. Though convergence rate is slower, the optimizer stays well-aligned with the gradient direction, proving to be more reliable and generalizable compared to using a high learning rate.

These results are no different from their expected results in theory: higher learning rates tend to risk instability while lower rates are more dependable.

5.4 Simple vs Optimized

Putting all of our chosen optics of hyperparameters, we can see how much of a improvement these make when compared to a simple CNN made of 2 convolutional layers, no batch normalization, and a small fully connected layer (Figure 4.7 & 4.8). Though the gap between testing and training in loss is not visible in 4.7 when compared to 4.8, looking at the final accuracy of the two, we can see how the optimized CNN was able to generate a much higher final accuracy compared to the simple CNN thus showing how much of an improving tweaking hyperparameters can do for our accuracies.

Chapter 6

Conclusion

This proejct shows both the effectiveness and limitations of applying CNNs to the Fashion-MNIST dataset. Across all experiments, it was found that the best configuration included dropout, a low learning rate, and a high batch size, consistently achieving accuracies between 92% and 93%. This project can be deemed successful in terms of model performance and in the depth of insight learned through thorough experimentation with different combinations of hyperparameters.

The most influential hyperparameter observed was dropout, showing the most improvement between figures 4.1 and 4.2 which prevented the model from overfitting to the training set. Models trained with dropout maintained low training and testing loss with stable accuracies during training. On the other hand, models that trained without dropout had very early signs of overfitting, with the testing loss diverging from training loss very early into training. This key observation had heavy consideration upon deciding whether to include or not include dropout.

The experiments on learning rates emphasized the importance of convergence. A high learning rate caused the model to overshoot repeatedly resulting in big jumps in the both the loss and accuracy (Figure 4.5). Compared to its opposite, a lower learning rate showed best signs of convergence with a smoother trend during training allowing the model to settle in the given amount of epochs.

The difference in batch sizes held a significant weight to the choice of model configurations as larger batches (Figure 4.3) kept the distance between the training and testing loss a minimum during training despite test accuracy being more jumpy. In contrast, if low batch size was used for our model, though the test accuracy was more smooth (Figure 4.4) in comparison to the plot from 4.4, there were late signs of overfitting from judging the gap between testing and training loss.

In summary, the findings of this project show that dropout, batch size, and learning rate have significant influence over the training dynamics and performance of a model. Dropout is important for generalization, learning rate is key for convergence, and batch sizing is key for stability throughout training. Together, these experiments show the pros and cons of using CNNs for simple datasets like Fashion-MNIST.

Chapter 7

Future Work

Future work on this project can focus more on expanding the limitations of our dataset. As previously stated, one big limit of this dataset can be attributed to both its size and color scale which provides opportunities to explore datasets more modern/reflective compared to the Fashion-MNIST dataset. Images that capture real-world variance like lighting, background noise, and fabric texture would prove to be more relevant to the task of classifying images of clothing as well as further highlight the importance of CNN architectures.

Expanding the dataset and model to a more modern approach can also expand the task of simple classification to a more complex task like multi-label recognition, where images can contain multiple attributes/categories. In this context, imagine a model that can identify the type of clothing shown in the image as well as the type of fabric used on the clothing. Exploring more complex topics like this have the potential to provide a greater understanding of modern Convolutional Neural Networks/ modern computer vision when there is more variance than 28x28 images of grayscale clothing.

Real world applications, like online retail, can benefit from these advances. Rather than having sellers manually tag clothing features, this task can be delegated to a multi-label CNN to be able to create the product descriptions, making this process more efficient and accurate. In practice, creating a model that can process user-uploaded images with the goal of describing what's being shown without human interaction can be very meaningful for future research and companies.