Classifying MNIST-Fashion Clothing Articles Using Convolutional Neural Networks

Dino Anastasopoulos: 1900661
Philani Mpofu: 1848751
Chloë Smith: 1877342

23 June 2021

Contents

1 Introduction **4**

2 Dataset Description **4**

3 Data Exploration and Preprocessing **5**
3.1 Data Exploration 5
3.2 Data Preprocessing 6

4 Baseline Methodology and Implementation **7**

5 Presentation of Results **8**

6 Employment of Optimization Techniques **8**

7 Conclusion **14**

Declaration

We, Dino Anastasopoulos, Philani Mpofu and Chloë Smith, declare that this report is our own, unaided work. It is being submitted for the Adaptive Computation and Machine Learning course for Computer Science Honours 2021. We all contributed equally to all components of this project.

June 23, 2021

June 23, 2021

June 23, 2021

1 Introduction

A Convolutional Neural Network (CNN) is a deep learning neural network architecture used primarily in image classification problems. Essentially, this architecture takes in a set of images as input and assigns weights and biases to different features in the image dataset. What makes it unique from the basic multi-layer perceptron is the component from which it receives its name: the convolutional layer. Convolutional layers are filters/kernels used to combine or "convolve" features in the dataset, sharpen or blur the images, further exploiting invariance in the data per category. Pooling layers are then used to reduce dimensionality, reducing the computation necessary in the hidden layers. After training, the CNN can use the weights and biases it learnt during training to classify new images in the same problem domain as the dataset [Saha 2018]. A CNN is commonly used to recognize sign language or classify handwritten digits. For this project, a CNN is used to classify clothing articles from the MNIST-Fashion dataset. Section 2 describes the dataset used to train and test the CNN. Section 3 provides a brief exploration of the dataset, as well as an explanation of the various methods and techniques used to process the dataset for training and testing. Section 4 details the baseline implementation of the CNN architecture used in this project. Section 5 presents an analysis of the results of the baseline implementation, and section 6 employs various optimization techniques to improve those results.

2 Dataset Description

The following data description is adapted from [Zalando 2017].

The Fashion-MNIST dataset is a collection of 70 000 images of clothing articles belonging to the Zalando catalogue. Every example in the dataset is a 28x28 gray-scale image with an associated label from 10 different classes. Each image consists of 784 pixels, where each pixel is a scalar value in the range from 0 to 255. Essentially, these values indicate the intensity level of a pixel at a point in a given image. Lower values indicate lighter intensities, and higher values indicate darker intensities. Both training and testing datasets have 785 columns. The first column stores the class of clothing in the image, and the remaining columns store the pixel intensity values associated with the image. Every example in the training and testing datasets corresponds to one of the following labels:

0. T-Shirt
1. Pair of Trousers
2. Pullover
3. Dress
4. Coat
5. Pair of Sandals
6. Shirt or Top
7. Pair of Sneakers
8. Bag
9. Pair of Ankle Boots

Figure 1 on the following page displays several samples from every class in the MNIST-Fashion dataset to give the reader a feel for the data.

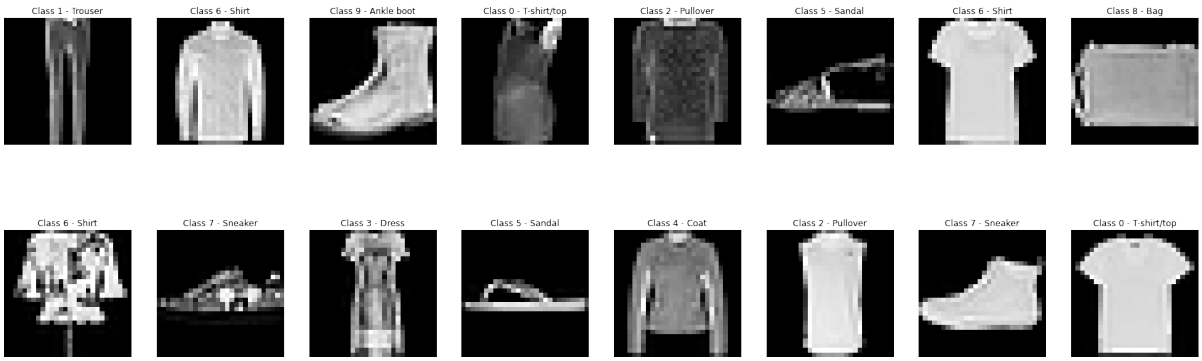

Figure 1: 16 Samples from the MNIST-Fashion Dataset

3 Data Exploration and Preprocessing

This section is divided into two subsections: Data Exploration and Data Preprocessing. The Data Exploration subsection investigates the inherent relationships and trends in the dataset, and the Data Preprocessing subsection details all the techniques implemented to convert the data to a more appropriate form for adequate training.

3.1 Data Exploration

The original MNIST-Fashion dataset used in the Keras source code [Chollet and others 2015] uses 86% of the data for training purposes and 14% of the data for testing purposes. For this project, 69% of the original dataset is used for training, 17% is used for validation and 14% is used for testing. This was done to ensure that the CNN model's hyper-parameters could be fine-tuned after training for improved learning and higher classification accuracy. Figure 2 explores the distribution of classes in the original training dataset and figure 3 explores the distribution of classes in the original testing dataset. There is an even or rectangular distribution of classes in both cases. This means that each class is equally likely to occur during training and testing. Therefore, the CNN model would not need to take into account class weights during training.

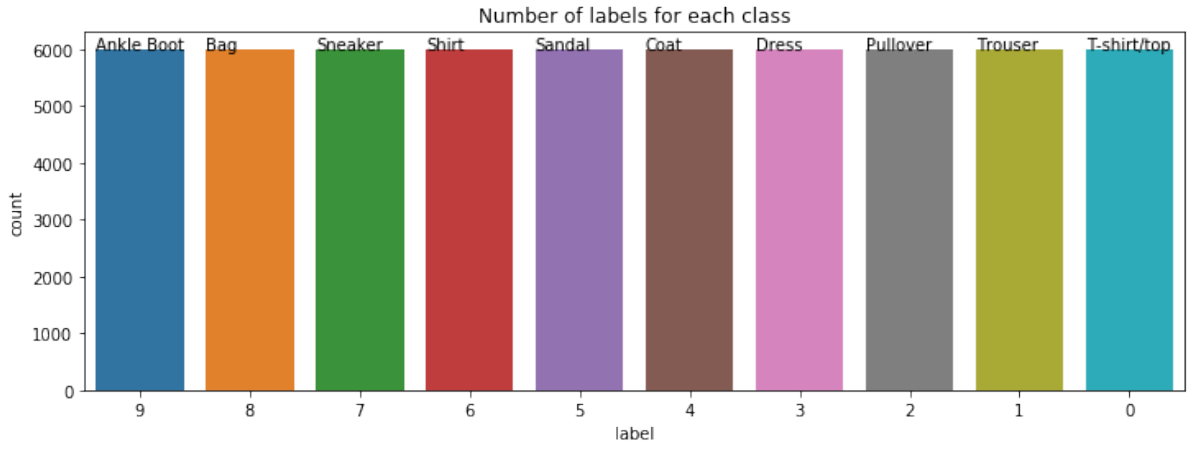

Figure 2: Distribution of Classes in Training Dataset

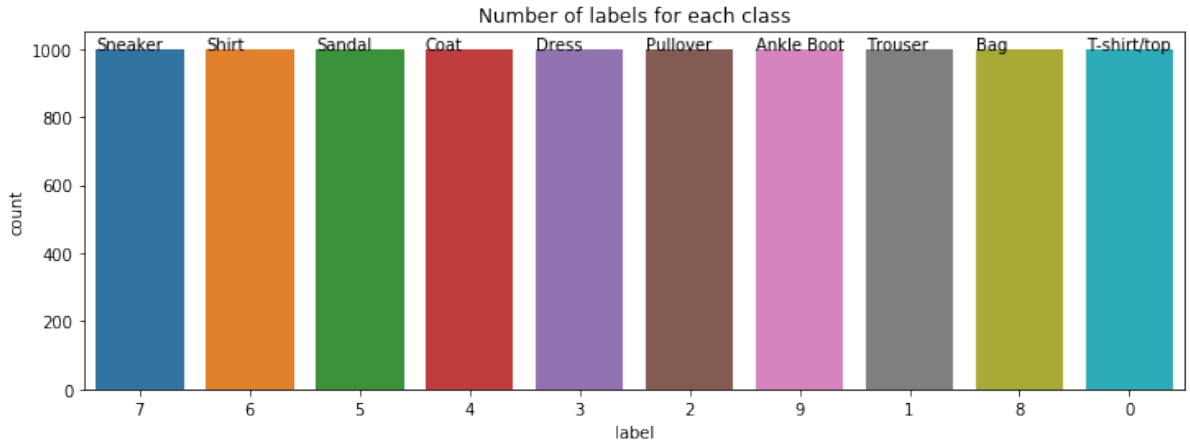

Figure 3: Distribution of Classes in Testing Dataset

3.2 Data Preprocessing

The dimensions of each dataset are three dimensional. The first dimension is the number of data points in that dataset, and the remaining dimensions are the dimensions of all the images in the dataset (28x28). After splitting the dataset in the manner described in section 2 the dimensions for the training, validation and testing datasets have the following shapes:

1. Training Data Feature Matrix: (48000, 28, 28)
2. Training Data Target Vector: (48000,)
3. Validation Data Feature Matrix: (12000, 28, 28)
4. Validation Data Target Vector: (12000,)
5. Testing Data Feature Matrix: (10000, 28, 28)
6. Testing Data Target Vector: (10000,)

However, most of these shapes are inappropriate for the Keras package. This package needs four-dimensional input to fit CNN models. Thus, the feature matrices are reshaped to a four-dimensional space using a dummy dimension. The feature matrices now all have the following shapes:

1. Training Data Feature Matrix: (48000, 28, 28, 1)
2. Validation Data Feature Matrix: (12000, 28, 28, 1)
3. Testing Data Feature Matrix: (10000, 28, 28, 1)

In addition, the feature values span a wide range. Each pixel value in every dataset is normalized to ensure they remain within a small range from 0 to 1.

4 Baseline Methodology and Implementation

The baseline implementation of the CNN algorithm is presented in the Fashion-MNIST.ipynb Jupyter Notebook, written in Python using Tensorflow and Keras. The architecture for the baseline model is illustrated below:

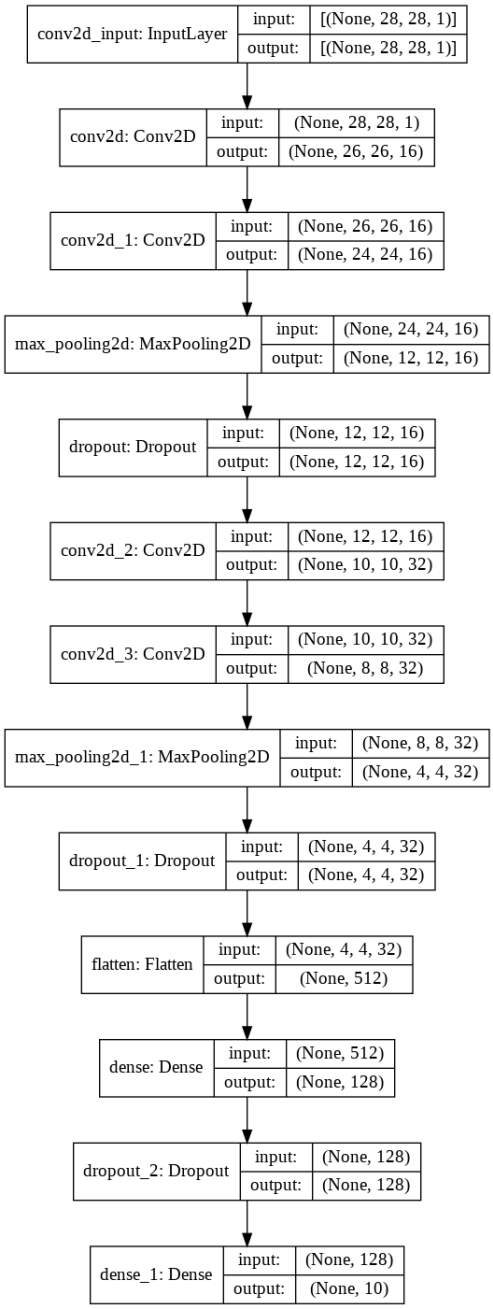

Figure 4: Architecture of Baseline CNN Model

We have 2-dimensional data in the form of each image, so we use 2D convolutional layers, the first of which accepts input of dimension 28x28x1, corresponding to the two dimensions of height and width, and the number of channels per pixel which is 1 since we are using gray-scale images. This layer is followed by two more 2D convolutional layers, which each have outputs of decreasing dimension, followed by a 2D max pooling layer to reduce dimensionality. A dropout layer is added for regularisation purposes (avoiding overfitting by randomly excluding nodes from the forward pass, and the weights of excluded nodes are not updated during backpropagation).

This baseline model has a total of 83 322 trainable parameters. We chose to keep this overall architecture and tune the hyperparameters, rather than add any more layers.

5 Presentation of Results

The results of training the initial baseline model are presented in Figures 5 and 6.

Epoch	Loss	Accuracy
1	0.67	0.75
2	0.45	0.84
3	0.4	0.86
4	0.37	0.86
5	0.35	0.87
6	0.34	0.87
7	0.33	0.88
8	0.32	0.88
9	0.32	0.88
10	0.31	0.89
11	0.31	0.89
12	0.31	0.89
13	0.3	0.89
14	0.3	0.89
15	0.3	0.89
16	0.3	0.89
17	0.29	0.9
18	0.3	0.89
19	0.29	0.89
20	0.29	0.9

Figure 5: Table showing Loss and Accuracy per Epoch for Baseline Model

The validation set is also fed through the model, obtaining the results illustrated in Figures 7 and 8.

We see a final classification accuracy of 89.5 percent, and 90.6 percent on the validation set. Good, but there is room for improvement. In the next section, we aim to improve the accuracy of our model through optimising the hyperparameters of the model.

6 Employment of Optimization Techniques

CNN architectures are becoming larger and larger, with trainable parameters easily reaching the hundreds of millions. This comes as a result of ever deeper model designs, which come with an increasing

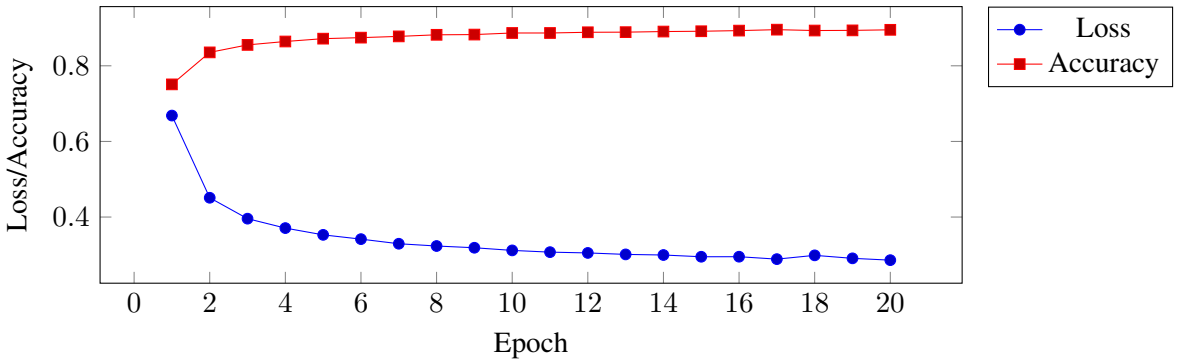

Figure 6: Plot of Loss/Accuracy per Epoch for Baseline Model

number of hyperparameters to tune for. This problem is becoming very tedious and time-consuming, but is now easily automated using Keras. After designing a model, we can now fit a second model which optimises the hyperparameters to find the best fit. Of course, this process itself has hyperparameters such as the hyperparameters we choose to optimise, the ranges we allow them to fall into, the number of trials we do and the number of epochs per trial, but as of yet there is no complete automation of the model-building and fitting process.

In the Fashion-MNIST.ipynb notebook, we implement this tuning using the Keras Random Search to fit the best set of hyperparameters, using the validation accuracy as our target to optimise. Referring to the model architecture illustrated in Figure 4, we choose the following hyperparameters per layer to tune, and set their bounds:

- **conv2d** and **conv2d_1**: We allow the number of filters to take on one of the values 16, 32, 64.
- **dropout, dropout_1 and dropout_2**: We allow the dropout rate to fall within the interval [0, 0.5], with a default value of 0.25 and a step size of 0.05.
- **conv2d_2** and **conv2d_3**: We allow the number of filters to take on one of the values 32, 64, 128, as well as the activation functions to be either sigmoid or ReLU, with a default of ReLU activations.
- **dense**: We let the number of activation units in this dense layer take on a value in the interval [16, 128], with a default value of 64 and a step size of 16.
- **Keras Adam Optimiser**: Finally, we set the learning rate of the optimiser itself as a hyperparameter for tuning, to fall within the interval $[1e^{-4}, 1e^{-2}]$ with logarithmic sampling, and a default value of $1e^{-3}$.

The hyperparameter tuner takes in a few arguments itself, which we summarise in the table below:

Argument	Value
Maximum number of trials	20
Executions per trial	2
Number of epochs	10

A seed value is also used, to ensure repeatable results. The optimum set of hyperparameters is chosen by running a number of trials using different values for the hyperparameters, within the constraints defined above, after which the Keras tuner chooses the best values by choosing the combination that had the best performance on the validation set. Due to the nature of the optimiser used, there is also a need to find the ”best epoch”, i.e. the epoch which has the best accuracy, after which the performance of the model deteriorates instead of improves. This is done by running the model with the tuned parameters, and the

Epoch	Loss	Accuracy
1	0.42	0.85
2	0.34	0.88
3	0.33	0.88
4	0.3	0.89
5	0.28	0.9
6	0.28	0.9
7	0.26	0.9
8	0.26	0.9
9	0.26	0.9
10	0.26	0.91
11	0.25	0.91
12	0.25	0.91
13	0.25	0.91
14	0.24	0.91
15	0.24	0.91
16	0.24	0.91
17	0.24	0.91
18	0.23	0.91
19	0.25	0.91
20	0.26	0.91

Figure 7: Table showing Validation Loss and Accuracy per Epoch for Baseline Model

best number of epochs is then the number of that ”best epoch”.

The tuned set of parameters was found to have a 91.6 percent accuracy on the validation set, compared to 89.5 percent before tuning; a mild improvement. A summary of the best 5 trials is shown in Figure 9, with the learning rates and validation accuracy scores for each of the trials shown in Figures 10 and 11, respectively.

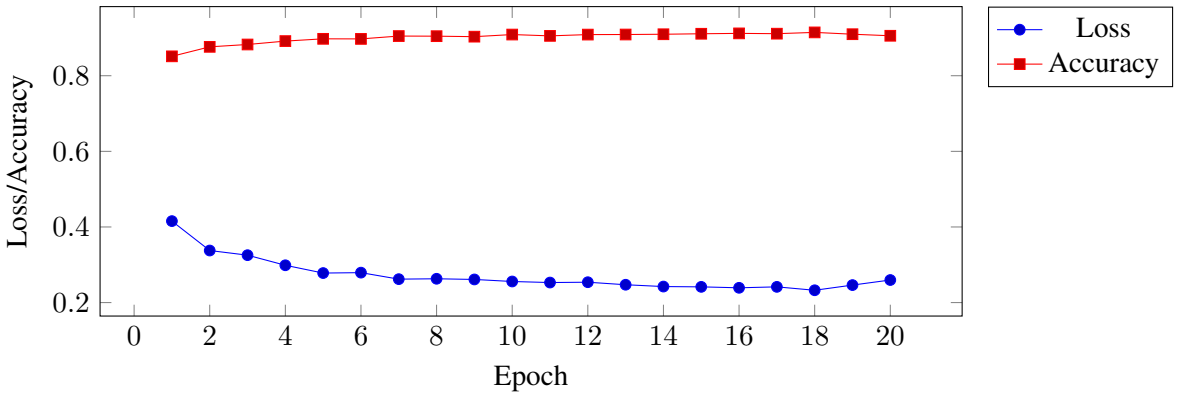

Figure 8: Plot of Validation Loss/Accuracy per Epoch for Baseline Model

Layer	Hyperparameter	Before Tuning	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5
conv2d	Number of filters	16	64	64	32	64	64
conv2d_1	Number of filters	16	64	64	16	32	16
	Activation function	ReLU	ReLU	ReLU	sigmoid	sigmoid	sigmoid
dropout	Dropout Rate	0.5	0.25	0.25	0.30	0.2	0.45
conv2d_2	Number of filters	32	128	32	32	32	128
	Activation Function	ReLU	ReLU	sigmoid	ReLU	ReLU	ReLU
conv2d_3	Number of filters	32	32	32	128	128	32
	Activation Function	ReLU	ReLU	ReLU	ReLU	ReLU	sigmoid
dropout_1	Dropout rate	0.3	0.25	0.4	0.30	0.45	0.25
dense	Number of nodes	128	80	32	48	48	48
dropout_2	Dropout rate	0.2	0.15	0.2	0.15	0.05	0.2

Figure 9: Layer-specific hyperparameter values for the best 5 trials

Looking more closely at the accuracy per class (Figure 13), we see that we achieve very high accuracy on some classes, nearly 100 percent on a few classes, while the model clearly gets confused with other classes, specifically shirts (74.02 percent), coats (89.05 percent) and pullovers (87.85 percent). From the confusion matrix, it's clear that these classes were often confused for each other, with coats being misclassified as shirts 5.06 percent of the time, and shirts confused for coats 7.61 percent of the time. Coats were also often misclassified as pullovers (3.20 percent), further evidence that the model has trouble distinguishing between very similar classes. The classes with nearly 100 percent classification accuracy, sandals, sneakers, ankle boots and trousers, are not visually similar to any other classes in the dataset; the other footwear classes are sneakers and ankle boots, which are quite visually different and also had high accuracy scores of 97.25 and 96.55 percent respectively. There are also no other pants in the dataset, making it easy to correctly classify trousers (98.20 percent). In Figure 14 we can see the actual class of a clothing item versus the predicted class, as well as the probability of that prediction.

Trial	Learning Rate
Before Tuning	0.001
1	0.002312113349326375
2	0.0007393219895594406
3	0.0009749332269823054
4	0.00036276965095376624
5	0.0009719978423652108

Figure 10: Table of learning rate values for best 5 trials

Trial	Score
Before Tuning	0.8994
1	0.9162916839122772
2	0.9050416648387909
3	0.9045416414737701
4	0.9035833477973938
5	0.901666671037674

Figure 11: Tuner scores (validation accuracy) for the best 5 trials

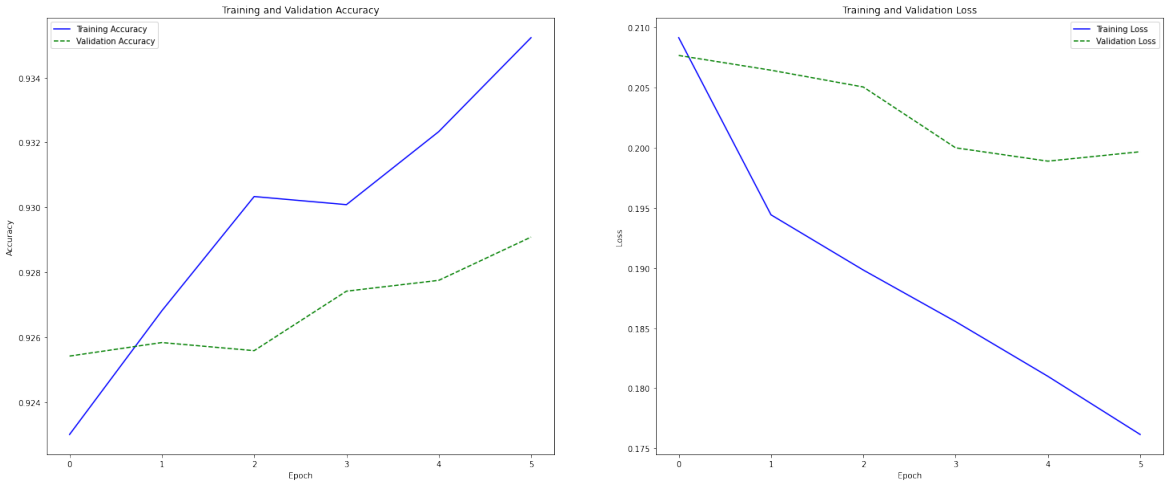

Figure 12: Training and Validation Accuracy/Loss per epoch of best model after random search tuning

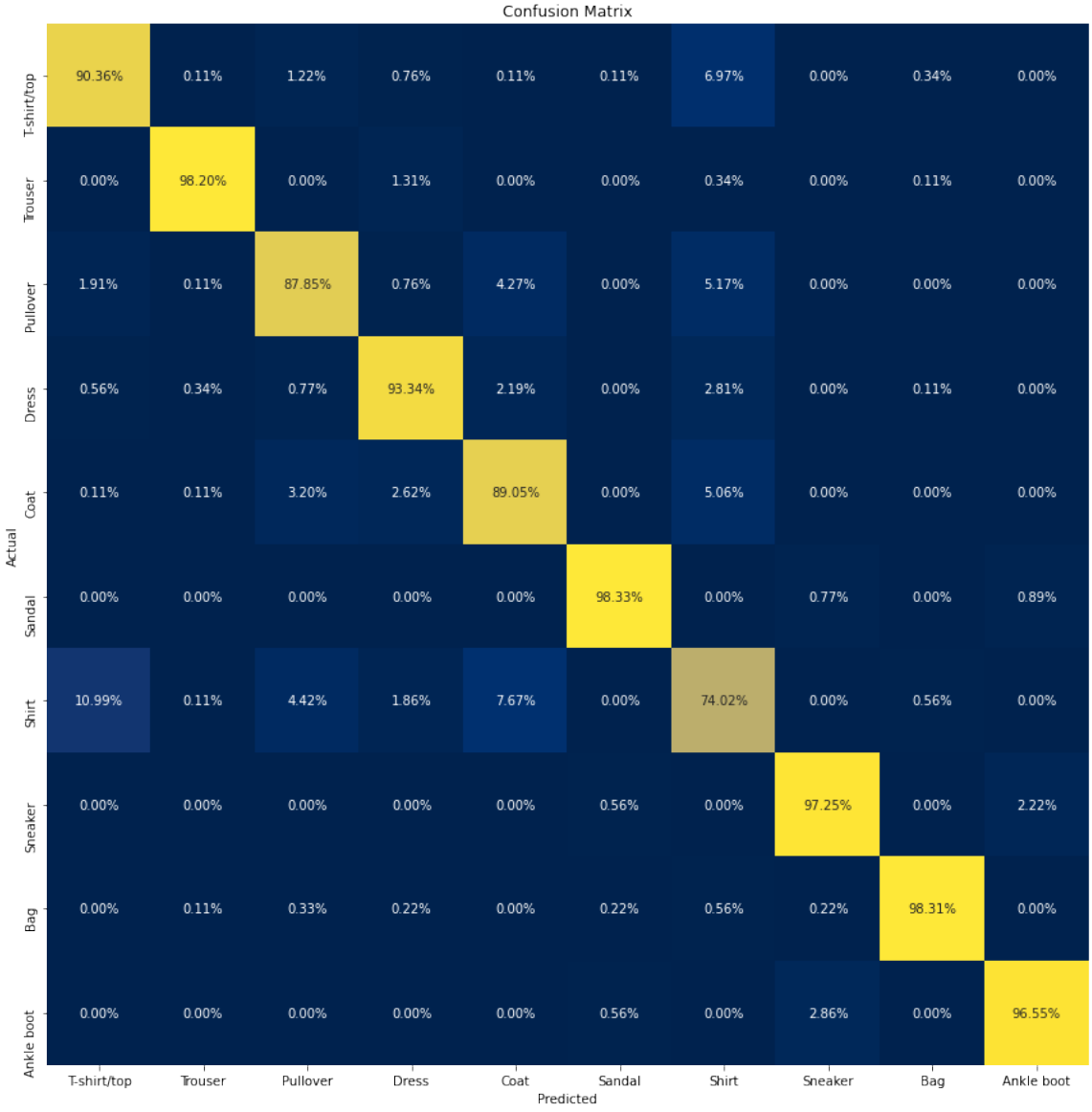

Figure 13: Confusion matrix of tuned model on testing set

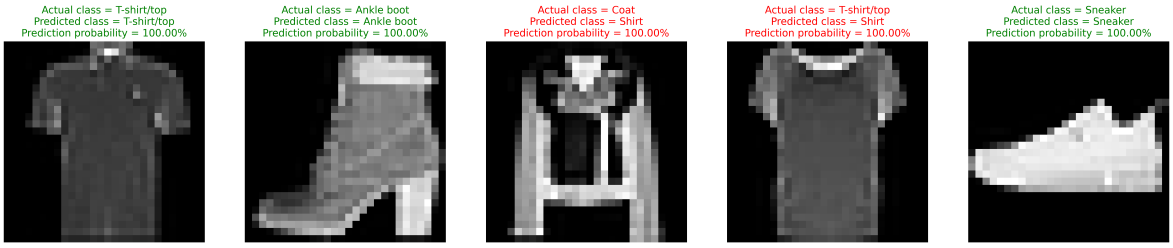

Figure 14: Visualizations of Model Predictions

7 Conclusion

In conclusion, we see that the choice of ReLU activations was the correct choice in the beginning, as the tuned model found that this was the best combination, a more than doubled learning rate was optimal, increasing the number of filters in the convolutional layers and the number of nodes in the dense layer yielded the best results, and keeping the dropout rate around 20-25 percent was optimal. The model was very accurate when dealing with classes that were not similar to other classes in the set, and had trouble distinguishing very visually similar classes. This may be due to these items having very similar shapes, and the low-resolution images not providing much further detail to distinguish them. From this, we can say the CNN was successful in detecting edges and shapes, and could benefit from higher detail in the images to better classify similarly-shaped articles.

Bibliography

[Chollet and others 2015] Francois Chollet et al. *Keras*, 2015. Retrieved 2 June, from `https://github.com/fchollet/keras`

[O'Malley *et al.* 2019] Tom O'Malley, Elie Bursztein, James Long, François Chollet, Haifeng Jin, Luca Invernizzi, et al. *Keras Tuner*. `https://github.com/keras-team/keras-tuner`, 2019.

[Prost 2020] Julie Prost. *How to Perform Hyperparameter Tuning with Keras Tuner*, Nov 2020.

[Saha 2018] Sumit Saha. A comprehensive guide to convolutional neural networks — the eli5 way. *Medium*, Dec 2018.

[Zalando 2017] Research Zalando. *Fashion MNIST*. `https://www.kaggle.com/zalando-research/fashionmnist`, Dec 2017.