10 April 2025

Research Assignment Report

For this assignment, I explored the e-commerce domain by creating a clothing classifier. Such a classifier could be useful to online shopping companies like Amazon or eBay. For example, they could use it to automatically tag their products based on their associated images, which could be used to provide better search results and recommendations. It could also be used to make a tool for customers that would allow them to upload an image and search for similar items.

To accomplish my goal of creating a classifier, I trained a network on the Fashion MNIST dataset. This set contains thousands of images labeled under one of ten different classes. These classes include t-shirts, trousers, pullovers, dresses, coats, sandals, shirts, sneakers, bags, and ankle boots. I chose this domain and dataset because I wanted to create my first multi-classification model. I also wanted to see how well the model would perform given the low resolution of the images, since it loses many important details of the clothing. The dataset itself is loaded through Keras with keras.datasets.fashion_mnist.load_data(). It contains 60,000 training images/labels and 10,000 testing images/labels. The 28x28 grayscale images are stored as 2D arrays of values ranging from 0 to 255. Below is an example of one of the training images:

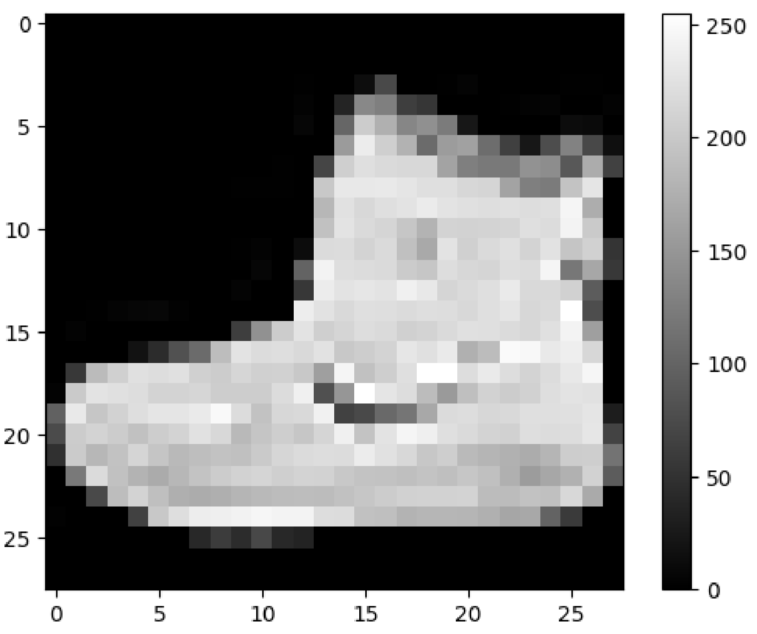
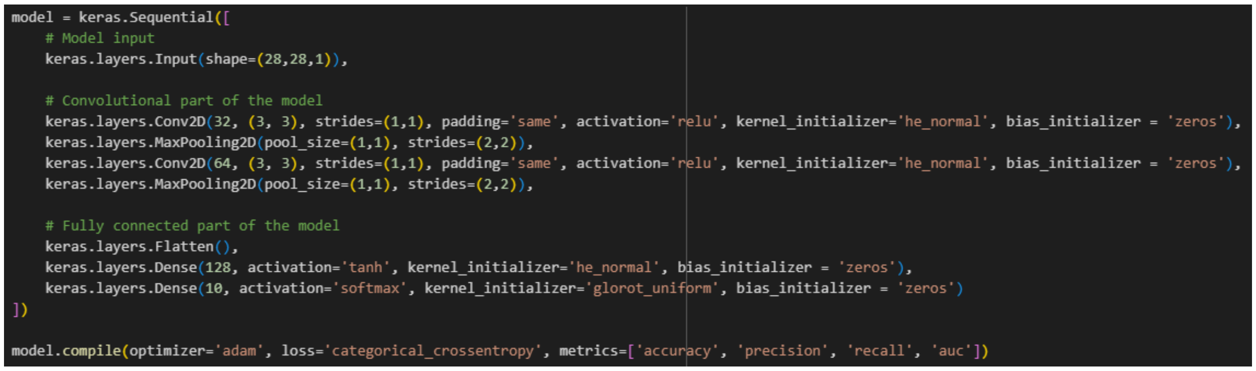


Figure 1 A training image representing an ankle boot.

I also ensured that the dataset was balanced by checking the frequency of each label. Thankfully, each label has exactly 7,000 occurrences, meaning that strategies like stratified sampling were not needed. Finally, the images were standardized before being used with the model to increase its performance in training.

The initial structure of the model was based on my notes from weeks 12-13 and my work on Assignment 4. They suggested that I build a 4-layer CNN on top of a dense network. The initial model consists of two pairs of convolutional and max pooling layers, followed by a flattening layer, a dense hidden layer, and an output layer. The convolutional layers have 32 and 64 neurons, respectively. Both layers use strides of 1x1 and filter sizes of 3x3. Meanwhile, the pooling layers both use pool sizes of 1x1 and strides of 2x2. In addition, the dense hidden layer has 128 neurons, and the output layer has 10 (one for each class). The convolutional layers use the ReLU activation function, as it was recommended throughout the lectures for these problems. The dense hidden layer uses tanh, and the output layer uses softmax (needed for multiclass classification). All the layers also use He normal as their kernel initializers, except for the output layer, which uses glorot uniform.



```
model = keras.Sequential([
    # Model input
    keras.layers.Input(shape=(28,28,1)),

    # Convolutional part of the model
    keras.layers.Conv2D(32, (3, 3), strides=(1,1), padding='same', activation='relu', kernel_initializer='he_normal', bias_initializer = 'zeros'),
    keras.layers.MaxPooling2D(pool_size=(1,1), strides=(2,2)),
    keras.layers.Conv2D(64, (3, 3), strides=(1,1), padding='same', activation='relu', kernel_initializer='he_normal', bias_initializer = 'zeros'),
    keras.layers.MaxPooling2D(pool_size=(1,1), strides=(2,2)),

    # Fully connected part of the model
    keras.layers.Flatten(),
    keras.layers.Dense(128, activation='tanh', kernel_initializer='he_normal', bias_initializer = 'zeros'),
    keras.layers.Dense(10, activation='softmax', kernel_initializer='glorot_uniform', bias_initializer = 'zeros')
])

model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy', 'precision', 'recall', 'auc'])
```

Figure 2 The code that implements the initial model.

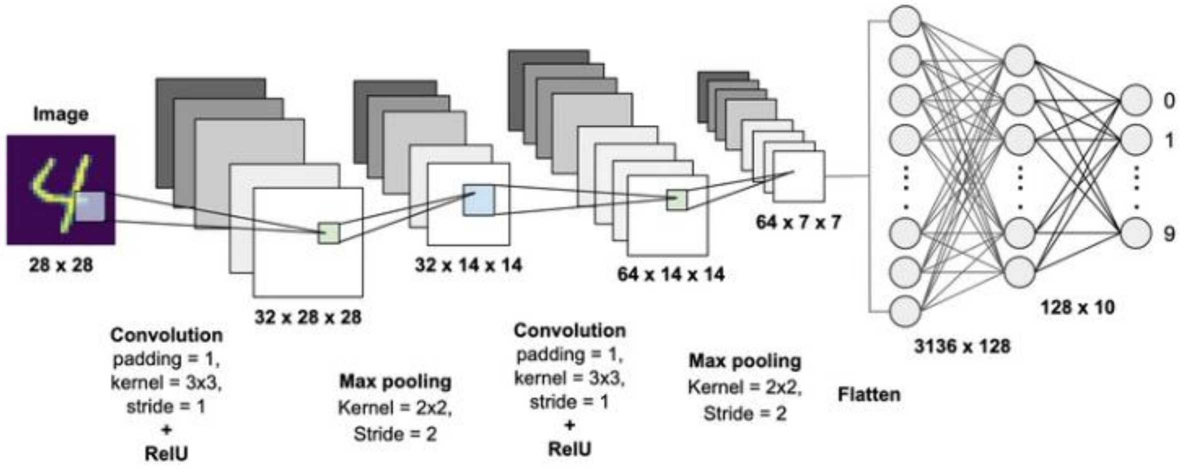


Image
28 x 28
32 x 28 x 28
32 x 14 x 14
64 x 14 x 14
64 x 7 x 7
3136 x 128
128 x 10
0
1
9
Convolution padding = 1, kernel = 3x3, stride = 1 + RelU
Max pooling Kernel = 2x2, Stride = 2
Convolution padding = 1, kernel = 3x3, stride = 1 + RelU
Max pooling Kernel = 2x2, Stride = 2
Flatten

Figure 3 An example CNN from the lectures designed for the MNIST handwritten digits dataset.

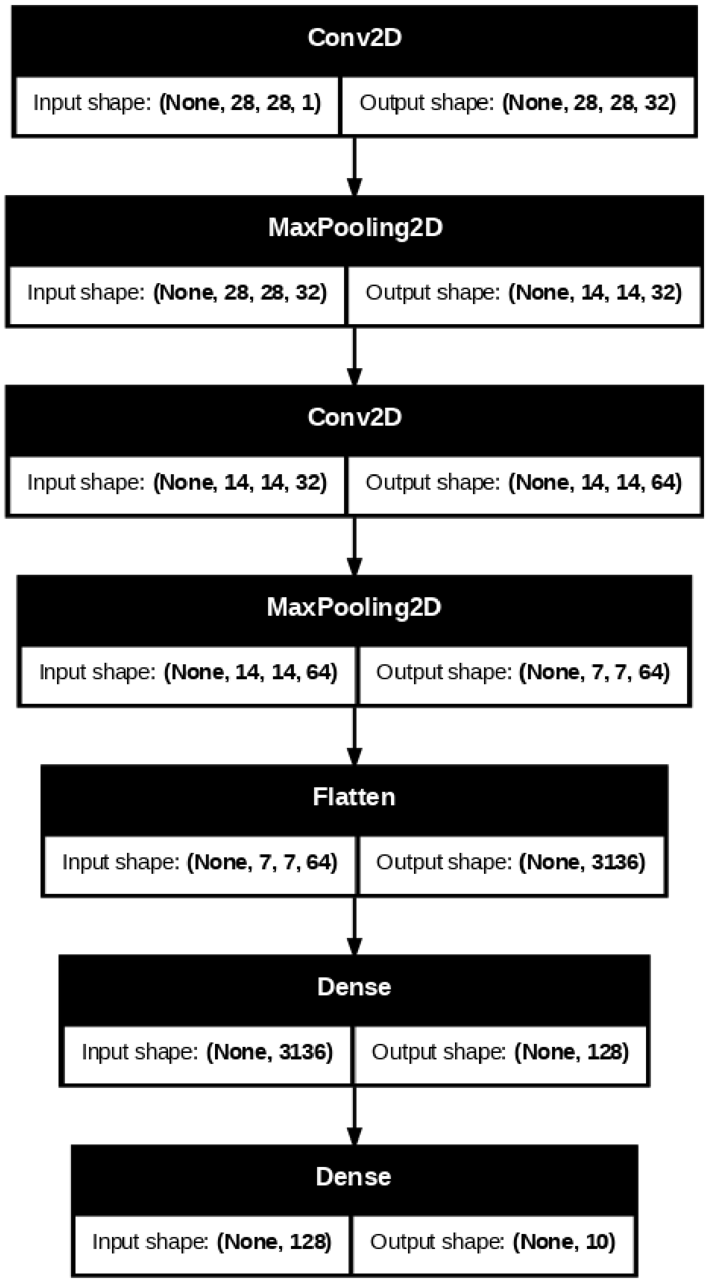
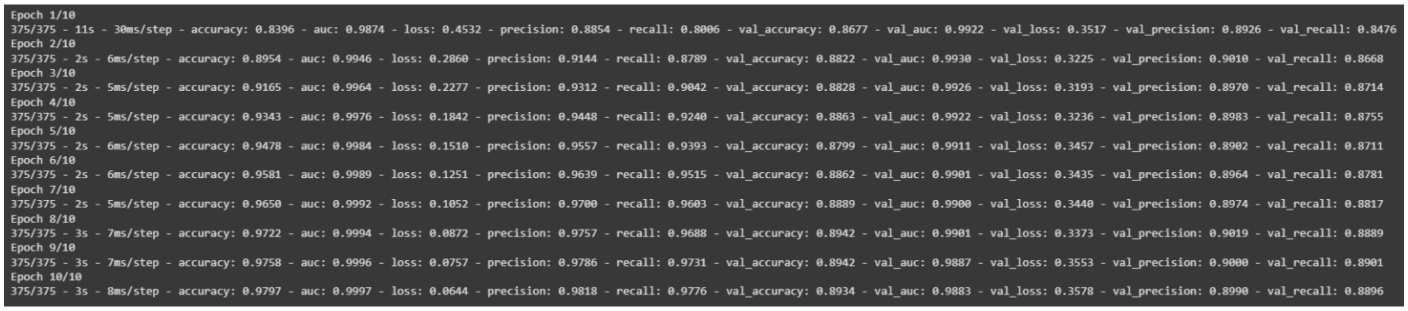

Conv2D
Input shape: (None, 28, 28, 1)
Output shape: (None, 28, 28, 32)
MaxPooling2D
Input shape: (None, 28, 28, 32)
Output shape: (None, 14, 14, 32)
Conv2D
Input shape: (None, 14, 14, 32)
Output shape: (None, 14, 14, 64)
MaxPooling2D
Input shape: (None, 14, 14, 64)
Output shape: (None, 7, 7, 64)
Flatten
Input shape: (None, 7, 7, 64)
Output shape: (None, 3136)
Dense
Input shape: (None, 3136)
Output shape: (None, 128)
Dense
Input shape: (None, 128)
Output shape: (None, 10)

Figure 4 A diagram of the initial model.

After I created the initial version of the model, I proceeded to train it for 10 epochs with batch sizes of 128 and a training/validation set split of 20%. The results from the validation sets showed that it performed quite well, achieving accuracy, precision, and recall scores of 89.34%, 89.90%, and 88.96%. However, assessing it with the testing set and producing a confusion matrix revealed that the model often mixed up shirts and t-shirts/tops, in addition to pullovers and coats (although to a lesser extent). This makes sense since these items can look quite similar to one another.



```
Epoch 1/10
375/375 - 11s - 30ms/step - accuracy: 0.8396 - auc: 0.9874 - loss: 0.4532 - precision: 0.8854 - recall: 0.8006 - val_accuracy: 0.8677 - val_auc: 0.9922 - val_loss: 0.3517 - val_precision: 0.8926 - val_recall: 0.8476
Epoch 2/10
375/375 - 2s - 6ms/step - accuracy: 0.8954 - auc: 0.9946 - loss: 0.2860 - precision: 0.9144 - recall: 0.8789 - val_accuracy: 0.8822 - val_auc: 0.9930 - val_loss: 0.3225 - val_precision: 0.9010 - val_recall: 0.8668
Epoch 3/10
375/375 - 2s - 5ms/step - accuracy: 0.9165 - auc: 0.9964 - loss: 0.2277 - precision: 0.9312 - recall: 0.9042 - val_accuracy: 0.8828 - val_auc: 0.9926 - val_loss: 0.3193 - val_precision: 0.8970 - val_recall: 0.8714
Epoch 4/10
375/375 - 2s - 5ms/step - accuracy: 0.9343 - auc: 0.9976 - loss: 0.1842 - precision: 0.9448 - recall: 0.9240 - val_accuracy: 0.8863 - val_auc: 0.9922 - val_loss: 0.3236 - val_precision: 0.8983 - val_recall: 0.8755
Epoch 5/10
375/375 - 2s - 6ms/step - accuracy: 0.9478 - auc: 0.9984 - loss: 0.1510 - precision: 0.9557 - recall: 0.9393 - val_accuracy: 0.8799 - val_auc: 0.9911 - val_loss: 0.3457 - val_precision: 0.8902 - val_recall: 0.8711
Epoch 6/10
375/375 - 2s - 6ms/step - accuracy: 0.9581 - auc: 0.9989 - loss: 0.1251 - precision: 0.9639 - recall: 0.9515 - val_accuracy: 0.8862 - val_auc: 0.9901 - val_loss: 0.3435 - val_precision: 0.8964 - val_recall: 0.8781
Epoch 7/10
375/375 - 2s - 5ms/step - accuracy: 0.9650 - auc: 0.9992 - loss: 0.1052 - precision: 0.9700 - recall: 0.9603 - val_accuracy: 0.8889 - val_auc: 0.9900 - val_loss: 0.3440 - val_precision: 0.8974 - val_recall: 0.8817
Epoch 8/10
375/375 - 3s - 7ms/step - accuracy: 0.9722 - auc: 0.9994 - loss: 0.0872 - precision: 0.9757 - recall: 0.9688 - val_accuracy: 0.8942 - val_auc: 0.9901 - val_loss: 0.3373 - val_precision: 0.9019 - val_recall: 0.8889
Epoch 9/10
375/375 - 3s - 7ms/step - accuracy: 0.9758 - auc: 0.9996 - loss: 0.0757 - precision: 0.9786 - recall: 0.9731 - val_accuracy: 0.8942 - val_auc: 0.9887 - val_loss: 0.3553 - val_precision: 0.9000 - val_recall: 0.8901
Epoch 10/10
375/375 - 3s - 8ms/step - accuracy: 0.9797 - auc: 0.9997 - loss: 0.0644 - precision: 0.9818 - recall: 0.9776 - val_accuracy: 0.8934 - val_auc: 0.9883 - val_loss: 0.3578 - val_precision: 0.8990 - val_recall: 0.8896
```

Figure 5 The results of training the initial model after 10 epochs.

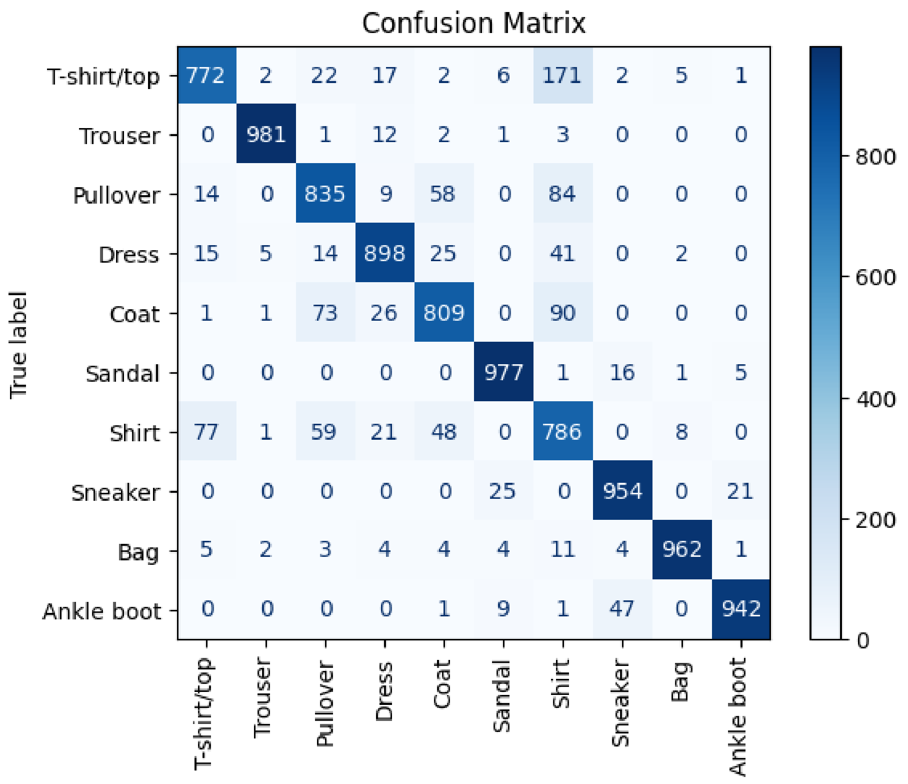
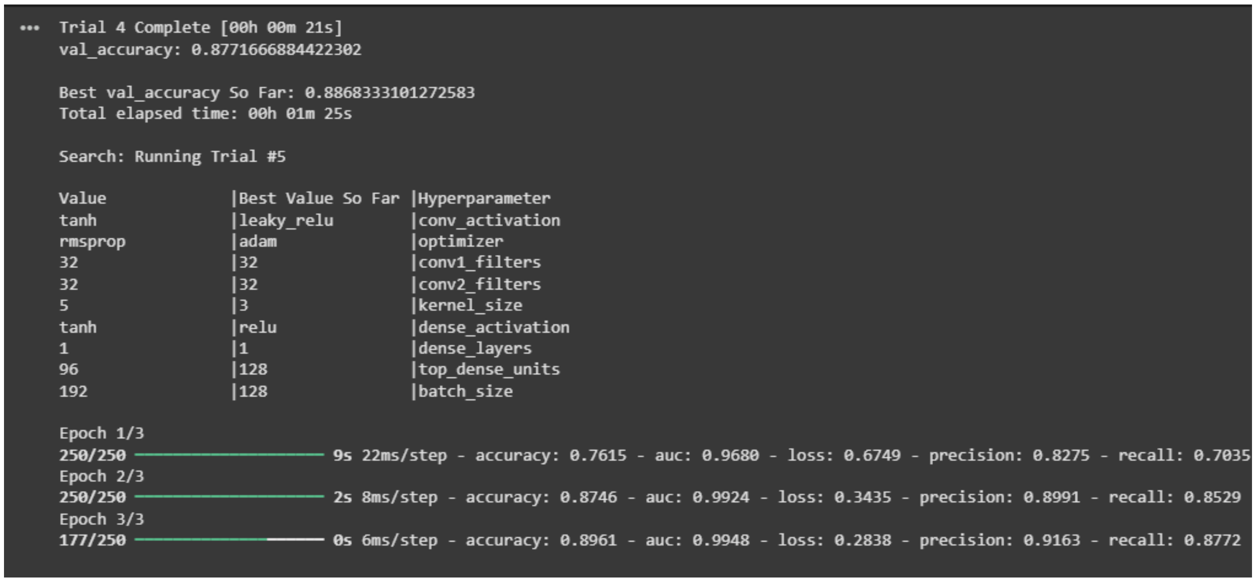


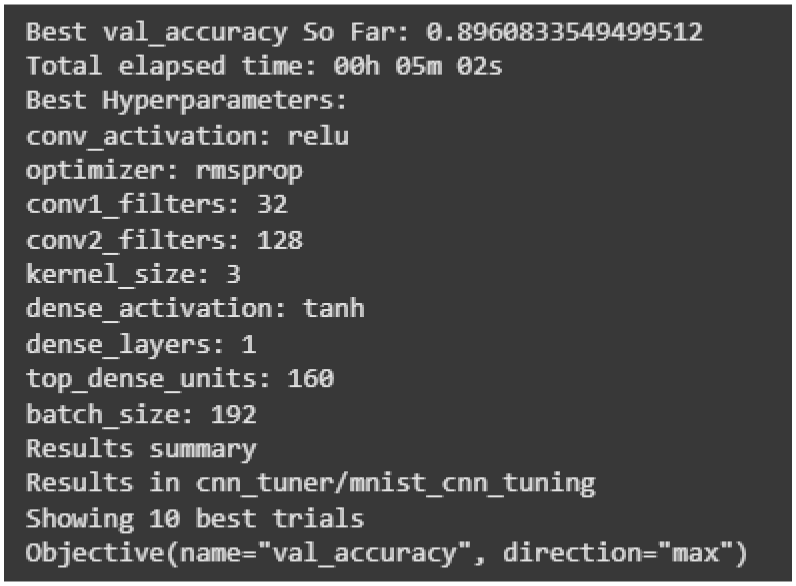
Figure 6 The confusion matrix for the initial version of the model.

To achieve higher model performance, I tried using a Keras Bayesian optimization tuner to test different hyperparameter configurations. The idea was based on a previous project of mine where we used a tuner to optimize a binary classifier. The tuner includes options for the activation functions used by the convolutional and dense layers (ReLU, tanh, and leaky ReLU), the optimizer (SGD, Adam, or RMSprop), the number of filters for each convolutional layer, the number of dense layers, the number of neurons in each dense layer, and the batch size. To speed up training, I also reduced the number of epochs from 18 to 3.



```
Trial 4 Complete [00h 00m 21s]
val_accuracy: 0.8771666884422302

Best val_accuracy So Far: 0.8868333101272583
Total elapsed time: 00h 01m 25s

Search: Running Trial #5

Value             |Best Value So Far |Hyperparameter
tanh              |leaky_relu        |conv_activation
rmsprop           |adam              |optimizer
32                |32                |conv1_filters
32                |32                |conv2_filters
5                 |3                 |kernel_size
tanh              |relu              |dense_activation
1                 |1                 |dense_layers
96                |128               |top_dense_units
192               |128               |batch_size

Epoch 1/3
250/250 ━━━━━━━━━━━━━━━━━━━━ 9s 22ms/step - accuracy: 0.7615 - auc: 0.9680 - loss: 0.6749 - precision: 0.8275 - recall: 0.7035
Epoch 2/3
250/250 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - accuracy: 0.8746 - auc: 0.9924 - loss: 0.3435 - precision: 0.8991 - recall: 0.8529
Epoch 3/3
177/250 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.8961 - auc: 0.9948 - loss: 0.2838 - precision: 0.9163 - recall: 0.8772
```

Figure 7 A screenshot of the tuner running.



```
Best val_accuracy So Far: 0.8960833549499512
Total elapsed time: 00h 05m 02s
Best Hyperparameters:
conv_activation: relu
optimizer: rmsprop
conv1_filters: 32
conv2_filters: 128
kernel_size: 3
dense_activation: tanh
dense_layers: 1
top_dense_units: 160
batch_size: 192
Results summary
Results in cnn_tuner/mnist_cnn_tuning
Showing 10 best trials
Objective(name="val_accuracy", direction="max")
```

Figure 8 The results of the hyperparameter tuning.

Once the tuning was complete, I trained a new model with the new hyperparameters. The tuning gave the dense layer 160 neurons and the second convolutional layer 128 neurons. It also changed the optimizer to RMSprop and increased the batch size to 192.

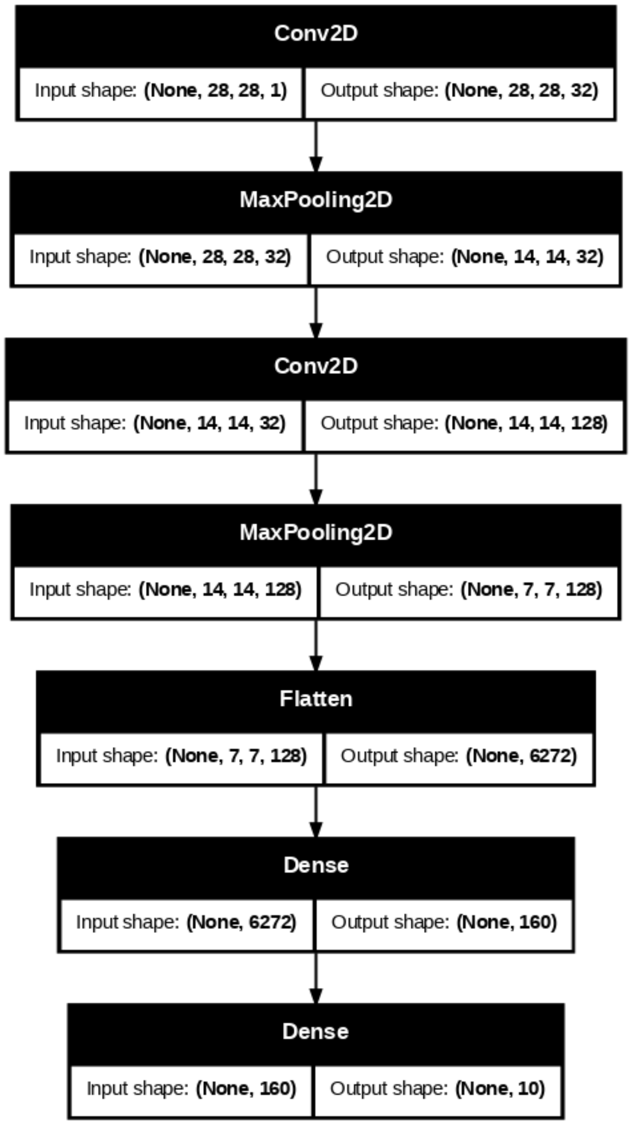
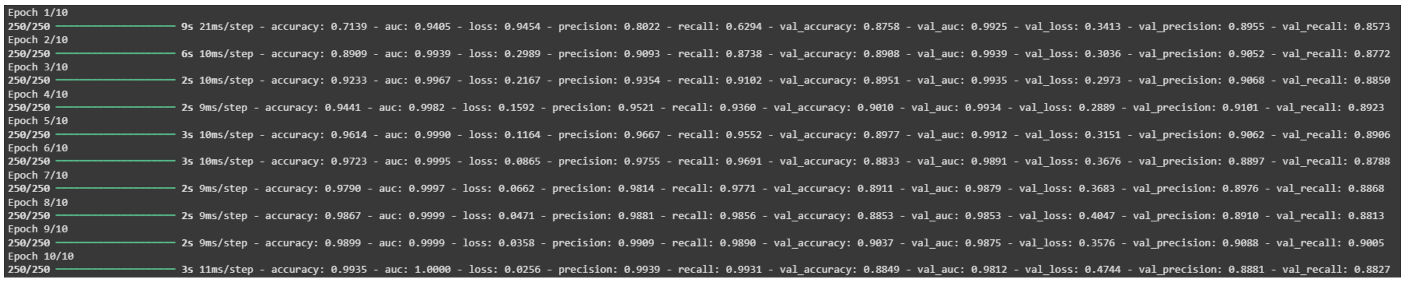

Conv2D
Input shape: (None, 28, 28, 1)
Output shape: (None, 28, 28, 32)
MaxPooling2D
Input shape: (None, 28, 28, 32)
Output shape: (None, 14, 14, 32)
Conv2D
Input shape: (None, 14, 14, 32)
Output shape: (None, 14, 14, 128)
MaxPooling2D
Input shape: (None, 14, 14, 128)
Output shape: (None, 7, 7, 128)
Flatten
Input shape: (None, 7, 7, 128)
Output shape: (None, 6272)
Dense
Input shape: (None, 6272)
Output shape: (None, 160)
Dense
Input shape: (None, 160)
Output shape: (None, 10)

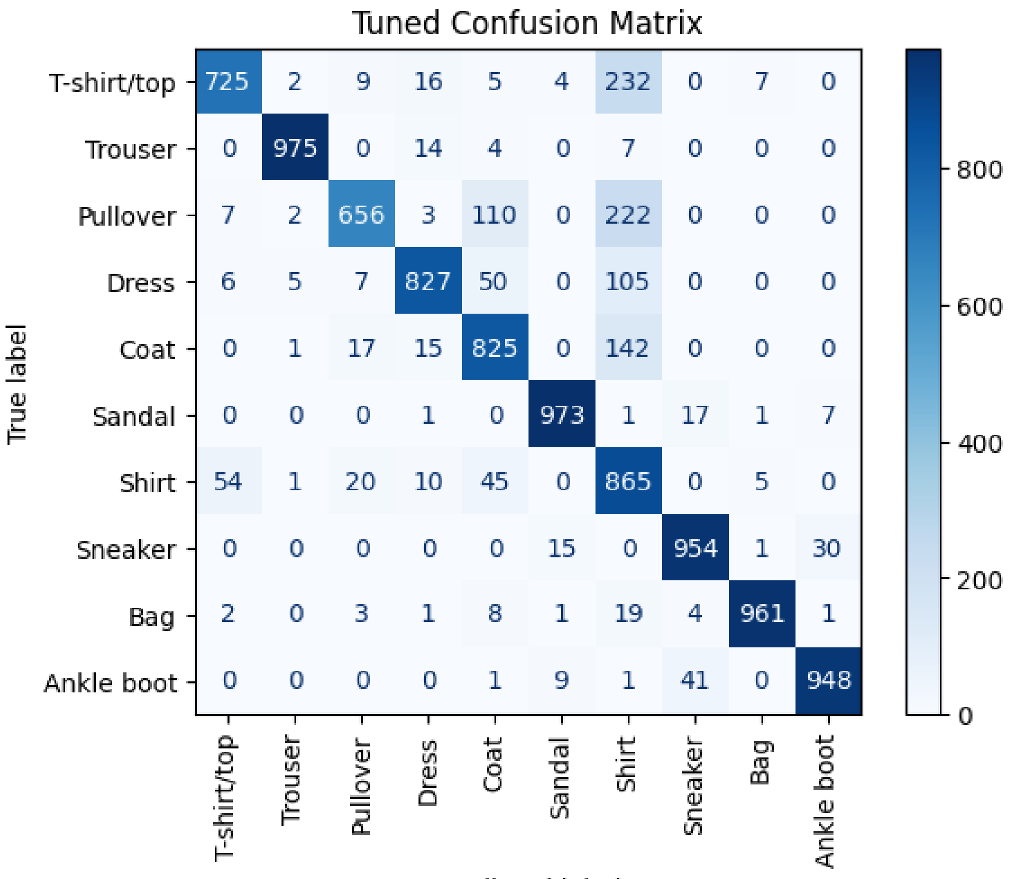
Figure 9 A diagram of the tuned model.

After 10 epochs of training, the resulting model achieved accuracy, precision, and recall scores of 88.49%, 88.81%, and 88.27%. The performance of the model on the validation set turned out barely worse than the initial model, which suggests that either the hyperparameter search configuration is weak or that more tuning is needed. However, due to processing time restrictions, I decided to leave it at that. The confusion matrix indicates that the model struggles more with identifying pullovers than the original but is slightly better at identifying shirts.



```
Epoch 1/10
250/250 ━━━━━━━━━━━━━━━━━━━━ 9s 21ms/step - accuracy: 0.7139 - auc: 0.9405 - loss: 0.9454 - precision: 0.8022 - recall: 0.6294 - val_accuracy: 0.8758 - val_auc: 0.9925 - val_loss: 0.3413 - val_precision: 0.8955 - val_recall: 0.8573
Epoch 2/10
250/250 ━━━━━━━━━━━━━━━━━━━━ 6s 10ms/step - accuracy: 0.8909 - auc: 0.9939 - loss: 0.2989 - precision: 0.9093 - recall: 0.8738 - val_accuracy: 0.8908 - val_auc: 0.9939 - val_loss: 0.3036 - val_precision: 0.9052 - val_recall: 0.8772
Epoch 3/10
250/250 ━━━━━━━━━━━━━━━━━━━━ 2s 10ms/step - accuracy: 0.9233 - auc: 0.9967 - loss: 0.2167 - precision: 0.9354 - recall: 0.9102 - val_accuracy: 0.8951 - val_auc: 0.9935 - val_loss: 0.2973 - val_precision: 0.9068 - val_recall: 0.8850
Epoch 4/10
250/250 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step - accuracy: 0.9441 - auc: 0.9982 - loss: 0.1592 - precision: 0.9521 - recall: 0.9360 - val_accuracy: 0.9010 - val_auc: 0.9934 - val_loss: 0.2889 - val_precision: 0.9101 - val_recall: 0.8923
Epoch 5/10
250/250 ━━━━━━━━━━━━━━━━━━━━ 3s 10ms/step - accuracy: 0.9614 - auc: 0.9990 - loss: 0.1164 - precision: 0.9667 - recall: 0.9552 - val_accuracy: 0.8977 - val_auc: 0.9912 - val_loss: 0.3151 - val_precision: 0.9062 - val_recall: 0.8906
Epoch 6/10
250/250 ━━━━━━━━━━━━━━━━━━━━ 3s 10ms/step - accuracy: 0.9723 - auc: 0.9995 - loss: 0.0865 - precision: 0.9755 - recall: 0.9691 - val_accuracy: 0.8833 - val_auc: 0.9891 - val_loss: 0.3676 - val_precision: 0.8897 - val_recall: 0.8788
Epoch 7/10
250/250 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step - accuracy: 0.9790 - auc: 0.9997 - loss: 0.0662 - precision: 0.9814 - recall: 0.9771 - val_accuracy: 0.8911 - val_auc: 0.9879 - val_loss: 0.3683 - val_precision: 0.8976 - val_recall: 0.8868
Epoch 8/10
250/250 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step - accuracy: 0.9867 - auc: 0.9999 - loss: 0.0471 - precision: 0.9881 - recall: 0.9856 - val_accuracy: 0.8853 - val_auc: 0.9853 - val_loss: 0.4047 - val_precision: 0.8910 - val_recall: 0.8813
Epoch 9/10
250/250 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step - accuracy: 0.9899 - auc: 0.9999 - loss: 0.0358 - precision: 0.9909 - recall: 0.9890 - val_accuracy: 0.9037 - val_auc: 0.9875 - val_loss: 0.3576 - val_precision: 0.9088 - val_recall: 0.9005
Epoch 10/10
250/250 ━━━━━━━━━━━━━━━━━━━━ 3s 11ms/step - accuracy: 0.9935 - auc: 1.0000 - loss: 0.0256 - precision: 0.9939 - recall: 0.9931 - val_accuracy: 0.8849 - val_auc: 0.9812 - val_loss: 0.4744 - val_precision: 0.8881 - val_recall: 0.8827
```

Figure 5 A screenshot of the tuned model's training results.



	T-shirt/top	Trouser	Pullover	Dress	Coat	Sandal	Shirt	Sneaker	Bag	Ankle boot
T-shirt/top	725	2	9	16	5	4	232	0	7	0
Trouser	0	975	0	14	4	0	7	0	0	0
Pullover	7	2	656	3	110	0	222	0	0	0
Dress	6	5	7	827	50	0	105	0	0	0
Coat	0	1	17	15	825	0	142	0	0	0
Sandal	0	0	0	1	0	973	1	17	1	7
Shirt	54	1	20	10	45	0	865	0	5	0
Sneaker	0	0	0	0	0	15	0	954	1	30
Bag	2	0	3	1	8	1	19	4	961	1
Ankle boot	0	0	0	0	1	9	1	41	0	948

Figure 10 The confusion matrix of the tuned model.

Overall, it seems that the models achieved good results on the testing and validation sets, despite some weakness in identifying a few item categories. The results indicate that the structure of the model is okay, and I believe that performance could be improved further with more training epochs and tuning. However, I lack the time and hardware needed for that. The code for this project can be found in the accompanying document alongside its outputs.