Neural Networks Final Project

Objective:

Our business' web application involves users uploading a picture of various items of clothing so that our application we can determine if we have similar clothing in stock to provide for the user as options to purchase. To identify the correct clothing type included in the content of the image, we need a Machine Learning model that accurately detects the clothing type in an image. A highly accurate identification result will provide end users with added confidence of the application that uses the inference from the model. The enhanced user experience will increase customer interaction with the application as well as customer sales and churn.

Data:

The data we are using to train the model comes from the Fashion MNIST dataset (https://keras.io/api/datasets/fashion_mnist/) with over 60,000 grayscale images of 10 fashion categories as part of the training set and 10,000 grayscale images used for the testing set.

```
# Load the Fashion MNIST data
(X_train, y_train), (X_test, y_test) = fashion_mnist.load_data()
```

```
print('X_train shape:', X_train.shape)
print(X_train.shape[0], 'train samples')
print(X_test.shape[0], 'test samples')
```

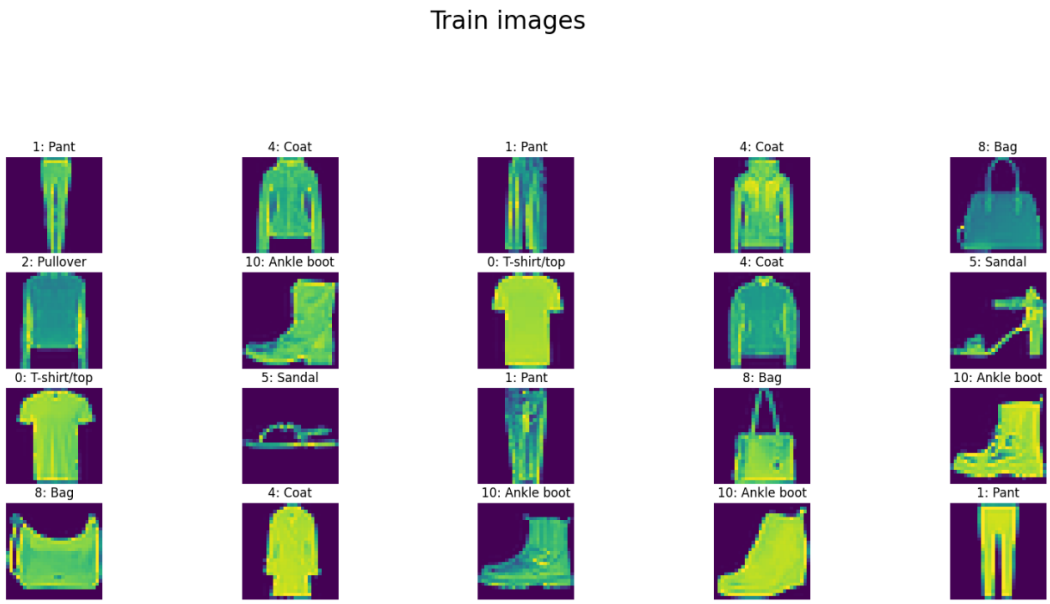
```
X_train shape: (60000, 28, 28)
60000 train samples
10000 test samples
```

```
# Look at the distribution of the labels in the training dataset
y_train_value_counts = Counter(y_train)
print(y_train_value_counts)
```

```
Counter({9: 6000, 0: 6000, 3: 6000, 2: 6000, 7: 6000, 5: 6000, 1: 6000, 6: 6000, 4: 6000, 8: 6000})
```

Looks like the values are relative uniform

Below are twenty random grayscale images selected from the training set with an associated text label to match the numeric label.



Model:

I used a Convoluted Neural Network model for the supervised classification of the content of the images. I chose a model with two convolutional hidden layers, a flatten layer and two dense layers.

In order to determine which parameter values to use for the convolutional and dense layers, I tuned the number of filters (from 32 to 128) and kernel size (from 3 to 5) of the layers. I also tuned the learning rate (0.01, 0.001, 0.0001) for the model. Below are the results of my tuning.

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 26, 26, 32)        320

 conv2d_1 (Conv2D)           (None, 24, 24, 32)        9248

 flatten (Flatten)           (None, 18432)             0

 dense (Dense)               (None, 32)                589856

 dense_1 (Dense)             (None, 10)                330

=================================================================
Total params: 599754 (2.29 MB)
Trainable params: 599754 (2.29 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
For the first convolutional layer, the optimal number of filters is 32 and the optimal kernal size is 3.
For the second convolutional layer, the optimal number of filters is 32 and the optimal kernal size is 3.
For the first Dense layer, the optimal number of units is 32.
The optimal learning rate for the optimizer is 0.001.
```

With the tuned parameters, I reached a training set accuracy of 98%, a validation accuracy of 91% and a testing accuracy of 90% (achieved on the 9th of 10 epochs) to obtain the optimal parameters for the number of filters and kernel size for the respective layers and an optimal learning rate.

```
[26]: optimal_model.fit(X_train, y_train, epochs=epochs, validation_split=0.2)

      Epoch 1/10
      1500/1500 [==============================] - 78s 50ms/step - loss: 0.4268 - accuracy: 0.8500 - val_loss: 0.3206 - val_accuracy: 0.8866
      Epoch 2/10
      1500/1500 [==============================] - 74s 49ms/step - loss: 0.2783 - accuracy: 0.9000 - val_loss: 0.2822 - val_accuracy: 0.8965
      Epoch 3/10
      1500/1500 [==============================] - 73s 49ms/step - loss: 0.2235 - accuracy: 0.9177 - val_loss: 0.2751 - val_accuracy: 0.9021
      Epoch 4/10
      1500/1500 [==============================] - 73s 49ms/step - loss: 0.1825 - accuracy: 0.9329 - val_loss: 0.2651 - val_accuracy: 0.9068
      Epoch 5/10
      1500/1500 [==============================] - 75s 50ms/step - loss: 0.1444 - accuracy: 0.9475 - val_loss: 0.2581 - val_accuracy: 0.9110
      Epoch 6/10
      1500/1500 [==============================] - 74s 49ms/step - loss: 0.1118 - accuracy: 0.9595 - val_loss: 0.2703 - val_accuracy: 0.9154
      Epoch 7/10
      1500/1500 [==============================] - 73s 49ms/step - loss: 0.0841 - accuracy: 0.9697 - val_loss: 0.3034 - val_accuracy: 0.9130
      Epoch 8/10
      1500/1500 [==============================] - 73s 48ms/step - loss: 0.0633 - accuracy: 0.9764 - val_loss: 0.3538 - val_accuracy: 0.9113
      Epoch 9/10
      1500/1500 [==============================] - 74s 49ms/step - loss: 0.0478 - accuracy: 0.9832 - val_loss: 0.3677 - val_accuracy: 0.9126
      Epoch 10/10
      1500/1500 [==============================] - 74s 49ms/step - loss: 0.0403 - accuracy: 0.9852 - val_loss: 0.4581 - val_accuracy: 0.9044
[26]: <keras.src.callbacks.History at 0x7f1cbd5c7640>

[27]: optimal_eval_dict = optimal_model.evaluate(X_test, y_test, return_dict=False)

      313/313 [==============================] - 4s 12ms/step - loss: 0.4899 - accuracy: 0.8997
```

Key Findings and Insights:

I am pleased that our deeply tuned Convolutional Neural Network led to a testing accuracy of 89.97% along with a training set accuracy of 98.32%. These accuracy results indicates that in production, the model will return extremely accurate results. This level of accuracy will help ensure customers are able to retrieve consistently accurate labeling of fashion items in images during their interaction with the application.

Flaws and Next Steps:

Initially, I considered making the neural network deeply layered to ensure more complexity, but eventually I settled on only using two convolutional layers and two dense layers. Using a high number of epochs (10) allowed me to circumvent around any issues using a less deep neural network may raise. I was concerned that with a deeper neural network, I would run into overfitting that would lead to poor testing accuracy. The flaw in this approach led to a high training accuracy but not as equally high testing accuracy. Next steps involve tuning the number of layers I use in my model to determine if I can achieve higher testing accuracy with a higher number of layers and a lower number of epochs to use as a crutch.