Pragya Jalan
Classification of the Clothing Items

Classification of Clothing Items

Clothing is a very important part for most people. A lot of people today have turned to online shopping in order to browse or buy clothes. Therefore there is a huge demand to online clothing stores. For online retailers, being able to correctly tag a piece of item with the correct label helps them organise their catalog. This in turn allows customers to filter the website and look at the labels they are interested in, instead of having to scroll the website thereby vastly improving their shopping experience.

In this project, I use image classification to detect images of clothing. Image classification is the process by which we are asked to predict a categories for a set of test images and measure the accuracy of the images, given the a set of images which are labeled with those categories. Instead of specifying what each of the image category looks like in the code, we provide the computer with many images of that type of category and then develop learning algorithms that look at these images and learn about the visual appearances.

I use a Convolutional Neural Network (CNN) in order to train the model to classify the images. In a CNN instead of feeding the image as an array of numbers, the image is broken up into a number of tiles. The machine then tries to predict the image based on the prediction of all the tiles.

For this project, I use the Fashion MNIST dataset. It consists of 70,000, 28x28 pixel grey-scale images derived from the online retailer Zalando. It consists of 10 different labels of the fashion items. Each image is annotated with the label indicating the correct type of clothing item/accessory. This dataset promises to be more diverse so that the model has to be more advanced to correctly classify the images.

Using this modal, I am getting a accuracy of about ~91%. I built a fairly complex modal, with a number of convolutions and feature maps. I fit this model with 50 epoches with a batch size of 32. It was trained on 60,000 samples and validated on 10,000 samples.

Installation

The language used for this project is python. I am using Python 3.6.8 for this project.

I have used Keras to build my model. Keras is just a wrapper around TensorFlow, which abstracts away the complexities in building a complex neural network.

In order to install TensorFlow:

```
$ pip install --upgrade tensorflow
```

Before installing Keras, we need to install a couple of dependencies:

```
$ pip install numpy scipy
$ pip install scikit-learn
$ pip install pillow
$ pip install h5py
```

Followed by installing Keras itself:

```
$ pip install keras
```

In addition to that, we use a number of libraries to plot the graphs and process the images:

```
$ pip install matplotlib
$ pip install numpy
```

I have used Jupyter notebook in order to write and run the model:

```
$ pip install --upgrade pip
```

In order to run/open a notebook:

```
$ jupyter notebook
```

While we are able to run deep learning models on our laptops or personal computers, we find that it is faster to run then on a Graphical Processing Unit (GPU). Because I do not have one on my laptop, I choose to run my model on Google Colabs.

Dataset

I have used the **Fashion MNIST** dataset. This dataset consists of 70,000, 28x28 pixel grey-scale images derived from the online retailer Zalando. The training set consists of a set of 60,000 examples while the test set consists of 10,000 examples. Each pixel has a single pixel value associated with it, indicating the lightness or darkness of the pixel, the higher number meaning it was darker. The pixel value is an integer between 0 and 255.
They contain 10 labels -

Label	**Description**
0	T-shirt/top
1	Trouser
2	Pullover
3	Dress
4	Coat
5	Sandal
6	Shirt
7	Sneaker
8	Bag
9	Ankle boot

According to the authors of the dataset, this dataset was intended to be a direct replacement of the MNIST handwritten dataset. Here are some of the reasons:

- MNIST dataset is too easy - Classic machine learning algorithms can achieve ~97% easily. But the Fashion dataset promises to be more diverse so that the algorithms have to learn more advanced features in order to be able to separate the individual classes reliability.
- MNIST is overused - Google Brain researcher and deep learning expert Ian Goodfellow calls for people to move away from MNIST
- MNIST cannot represent modern CV task

Added to this, the Keras library already includes Fashion-MNIST as a build in dataset, therefore, we don't have to download the database. WE just follow their API and are able to use the dataset.

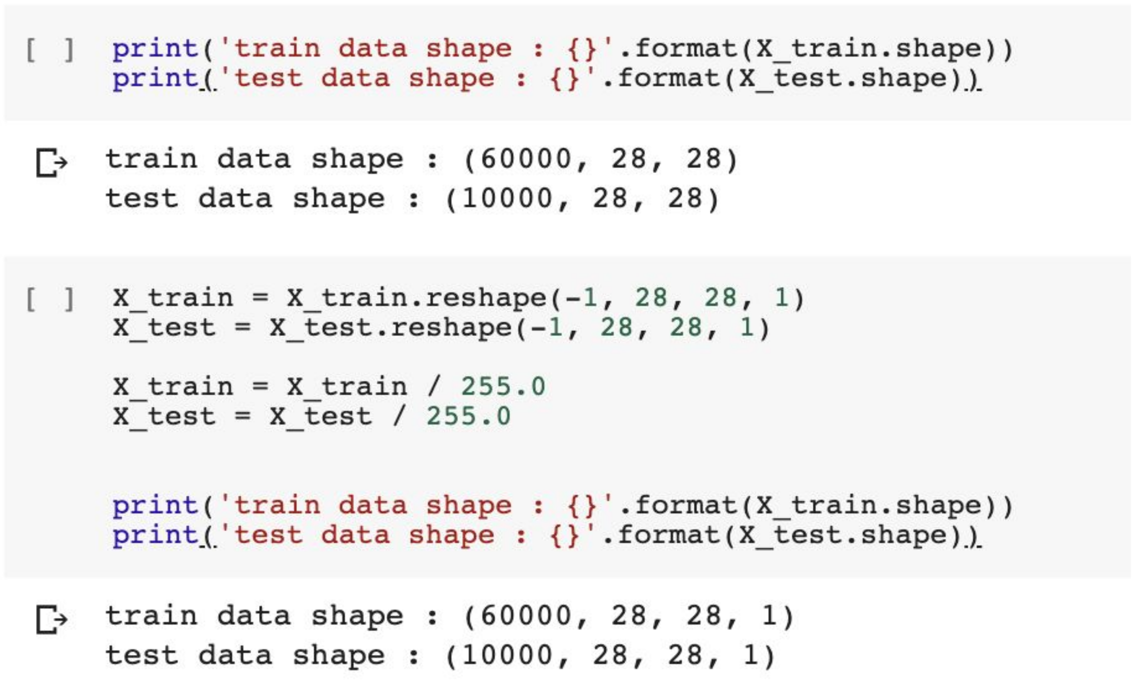
```
from keras.datasets import fashion_mnist

(x_train, y_train), (x_test, y_test) = fashion_mnist.load_data()
```

After reading the data, we convert the data into array and split the pixel data and labels by slicing the arrays and rescaling the pixel data between 0 and 1.

Image Reshaping: After importing the dataset our image arrays are of shape (60000,785). We reshape the array into original image size of (60000, 28, 28, 1)

Pragya Jalan
Classification of the Clothing Items



```
[ ]  print('train data shape : {}'.format(X_train.shape))
     print('test data shape : {}'.format(X_test.shape))
```

```
train data shape : (60000, 28, 28)
test data shape : (10000, 28, 28)
```

```
[ ]  X_train = X_train.reshape(-1, 28, 28, 1)
     X_test = X_test.reshape(-1, 28, 28, 1)

     X_train = X_train / 255.0
     X_test = X_test / 255.0


     print('train data shape : {}'.format(X_train.shape))
     print('test data shape : {}'.format(X_test.shape))
```

```
train data shape : (60000, 28, 28, 1)
test data shape : (10000, 28, 28, 1)
```

<u>One Hot Encoding</u>: We apply one-hot encoding to our class variables to convert then imto a format that works better with the algorithm

```
y_train = np_utils.to_categorical(y_train)
y_test = np_utils.to_categorical(y_test)
num_classes = y_test.shape[1]
```

Model

Building the model requires configuring the layers and then compiling the model. The layers extract the data that is fed into them. Most of the deep learning models consists of chaining together simple layers to build complex models.

I have a used a pattern of Convolutional, Dropout, Convolutional and Max Pooling layers. This pattern will be repeated 3 times with 32, 64, and 128 feature maps. The model will have an increasing number of feature maps with a smaller and smaller size given the max pooling layers. Finally an additional and larger Dense layer will be used at the output end of the network in an attempt to better translate the large number feature maps to class values.

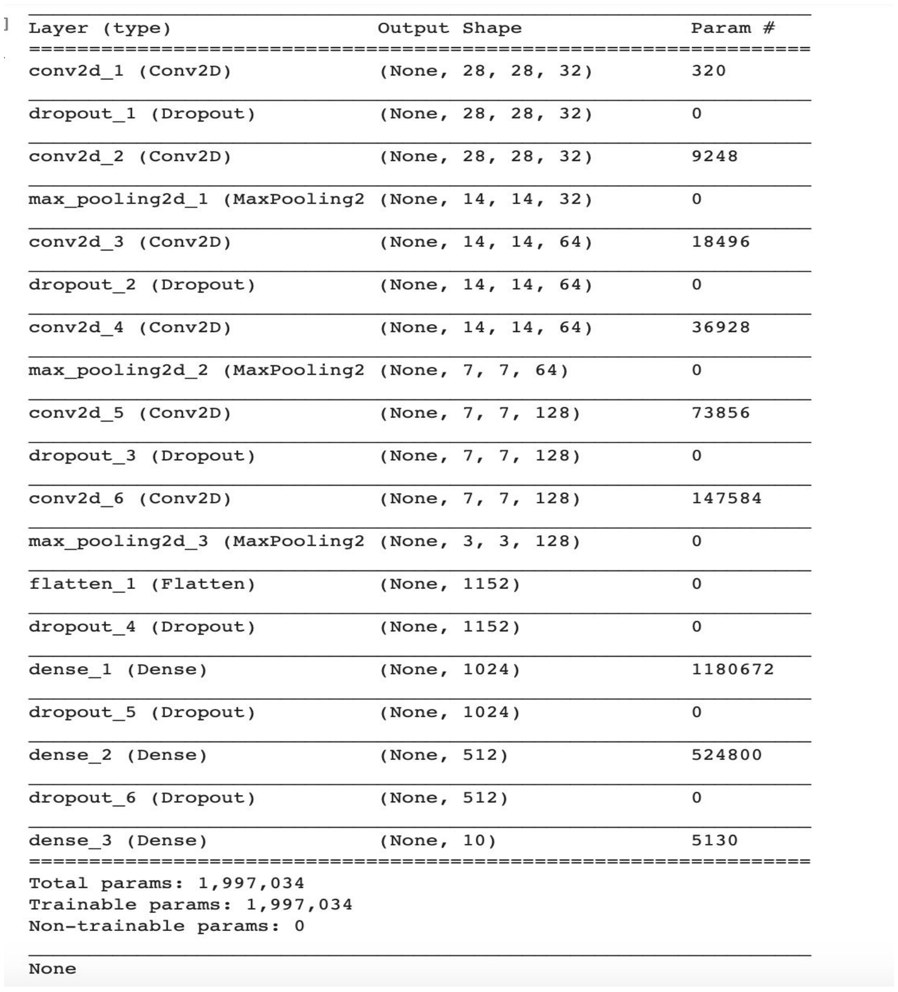
We built a Sequential model. This is a linear stack of layers. Once the model is defined, we can add the different layers to it :

1. Convolutional input layer, 32 feature maps with a size of 3 3 and a rectifier activation function.
2. Dropout layer at 20%.
3. Convolutional layer, 32 feature maps with a size of 3 3 and a rectifier activation function.
4. Max Pool layer with size 2 2.
5. Convolutional layer, 64 feature maps with a size of 3 3 and a rectifier activation function.
6. Dropout layer at 20%.
7. Convolutional layer, 64 feature maps with a size of 3 3 and a rectifier activation function.
8. Max Pool layer with size 2 2.
9. Convolutional layer, 128 feature maps with a size of 3 3 and a rectifier activation function.
10. Dropout layer at 20%.
11. Convolutional layer, 128 feature maps with a size of 3 3 and a rectifier activation function.
12. Max Pool layer with size 2 2.
13. Flatten layer.
14. Dropout layer at 20%.
15. Fully connected layer with 1,024 units and a rectifier activation function.
16. Dropout layer at 20%.
17. Fully connected layer with 512 units and a rectifier activation function.
18. Dropout layer at 20%.
19. Fully connected output layer with 10 units and a softmax activation function.

```
# Create the model
model = Sequential()
```

```
model.add(Conv2D(32, (3, 3), input_shape=(28, 28, 1), activation='relu',
padding='same'))
model.add(Dropout(0.2))
model.add(Conv2D(32, (3, 3), activation='relu', padding='same'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu', padding='same'))
model.add(Dropout(0.2))
model.add(Conv2D(64, (3, 3), activation='relu', padding='same'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(128, (3, 3), activation='relu', padding='same'))
model.add(Dropout(0.2))
model.add(Conv2D(128, (3, 3), activation='relu', padding='same'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dropout(0.2))
model.add(Dense(1024, activation='relu', kernel_constraint=maxnorm(3)))
model.add(Dropout(0.2))
model.add(Dense(512, activation='relu', kernel_constraint=maxnorm(3)))
model.add(Dropout(0.2))
model.add(Dense(num_classes, activation='softmax'))

print(model.summary())
```



```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 28, 28, 32)        320
_________________________________________________________________
dropout_1 (Dropout)          (None, 28, 28, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 28, 28, 32)        9248
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 14, 14, 32)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 14, 14, 64)        18496
_________________________________________________________________
dropout_2 (Dropout)          (None, 14, 14, 64)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 14, 14, 64)        36928
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 7, 7, 64)          0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 7, 7, 128)         73856
_________________________________________________________________
dropout_3 (Dropout)          (None, 7, 7, 128)         0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 7, 7, 128)         147584
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 3, 3, 128)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 1152)              0
_________________________________________________________________
dropout_4 (Dropout)          (None, 1152)              0
_________________________________________________________________
dense_1 (Dense)              (None, 1024)              1180672
_________________________________________________________________
dropout_5 (Dropout)          (None, 1024)              0
_________________________________________________________________
dense_2 (Dense)              (None, 512)               524800
_________________________________________________________________
dropout_6 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_3 (Dense)              (None, 10)                5130
=================================================================
Total params: 1,997,034
Trainable params: 1,997,034
Non-trainable params: 0
_________________________________________________________________
None
```

Layers in the CNN:

- <u>Convolution Layer</u>: The primary purpose of the convolution layer is to extract features from the input image
- <u>Pooling Layer</u>: This layer reduces the dimensionality of each feature map but retains most of the important information
- <u>Dense Layer/Fully Connected Layer</u>: This implies that every neuron in the previous layer is connected to every neuron in the next layer.

The <u>Dropout Layer</u> is a process to overcome overfitting - where the model learns well from the training data but it does not generalise well, so the performance is poor for testing data. This is a technique where random neurons are dropped or ignored during training, therefore other neurons have to step in and handle the representation required to make predictions. This helps the network better generalise the data.

The model is then compiled with a few more settings:

- Loss function - this measures how accurate the model is during training
- Optimizer - This is how the model is updated based on the data it sees and its loss function
- Metrics - Used to monitor the training and the testing steps

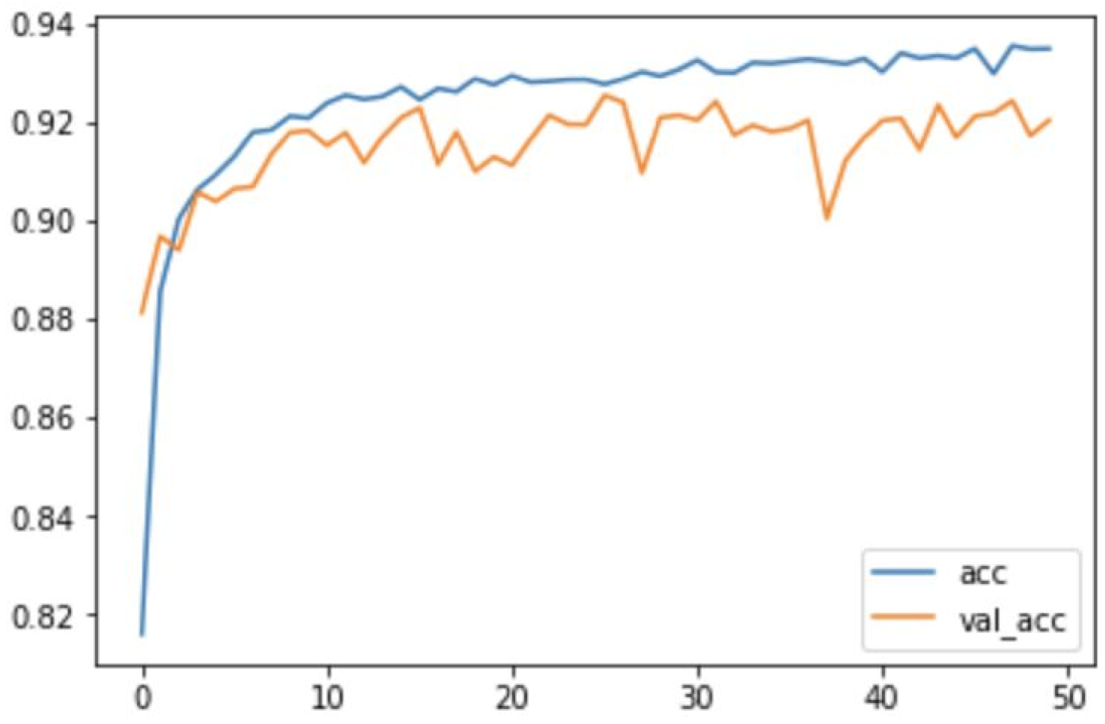
```
model.compile(loss='categorical_crossentropy',
              optimizer=keras.optimizers.Adam(),
              metrics=['accuracy'])
```

In order to train the model, we need to specify the batch_size and the number of epochs. We use batch size to specify the number of observations after which we want to update the weights. Epoch is the total number of iterations you want the model to run. I trained the model with 50 epochs with a batch size of 32. The model was trained on a dataset of 60000 and validated on a dataset of 10000

Initially I tried using a much simpler model with only one convolutional, Dropout, Convolutional and Max-pooling layer, but that gave me only a ~73% accuracy, which was comparatively low. Therefore I decided to use a more complex model to train this data.

Results

This model gave me an accuracy of ~91%

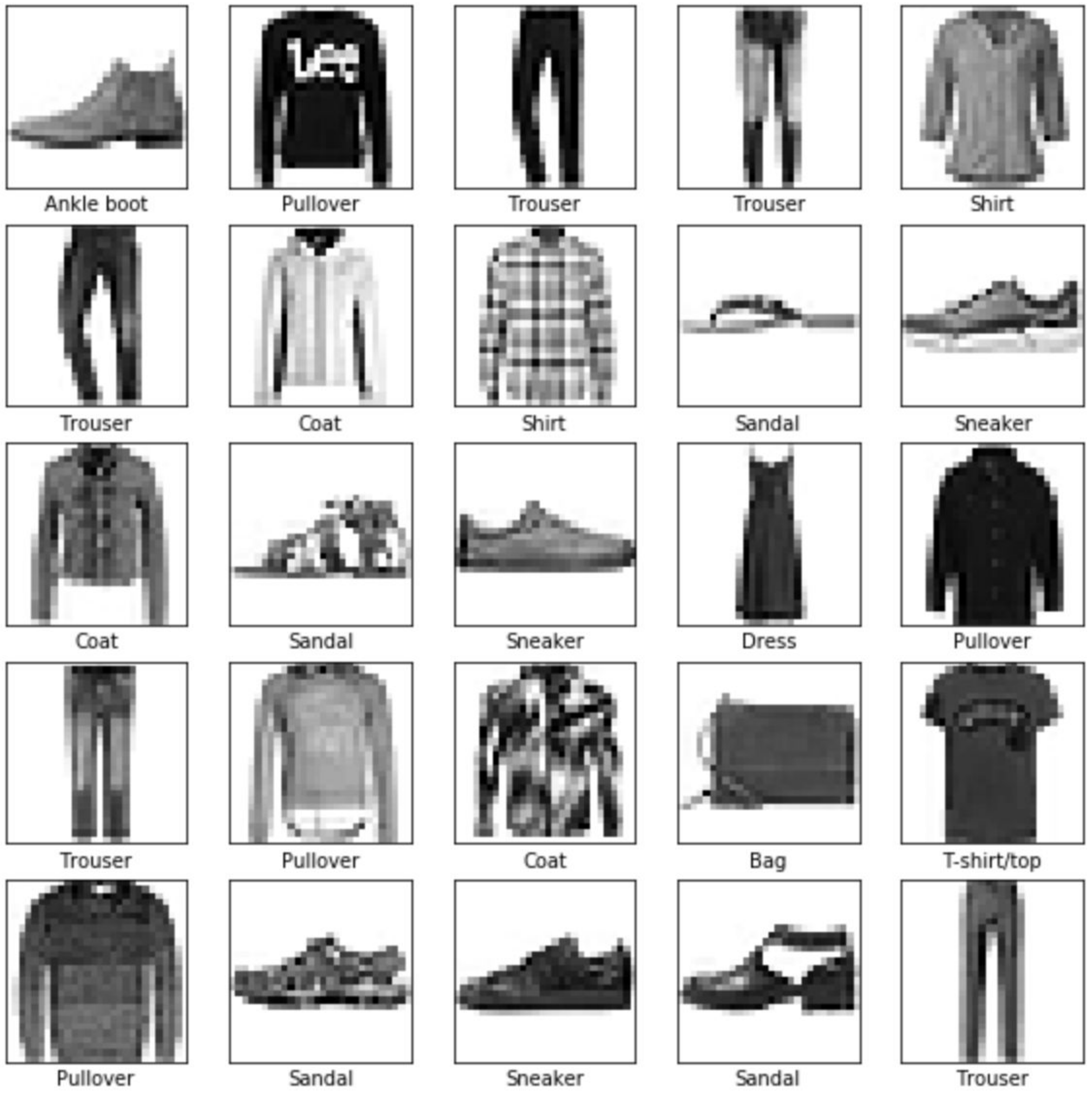


The above graph shows the training accuracy as well as the validation accuracy. We see that both the accuracy and the validation accuracy is improving - although the validation accuracy starts to get lower than the training accuracy which shows a little bit of overfitting.

For future improvements, I would tweak my model by reducing the network's capacity to learn. Added to that, add a Regularization and maybe add another Dropout layer in order to counterbalance the overfitting.

Pragya Jalan
Classification of the Clothing Items

The model though correctly identifies the clothing items from the test set.
Above are the first 25 items from the test set with the predicted labels generated from the model.



The model is able to predict the type of clothing from an image which is not in the dataset. This is helpful as once the model is trained you want to use the model for other images instead of just the images in the dataset. This way the model is able to be used for a wide variety of applications.

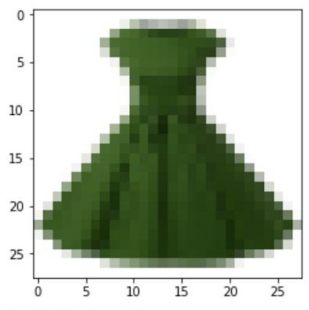
Below an example of a image of a dress - the model is correctly able to classify that as a Dress.

```
--2019-05-13 22:07:48--  https://images-na.ssl-images-amazon.com/images/I/412V6B5Fo4L._SY445_QL70_.jpg
Resolving images-na.ssl-images-amazon.com (images-na.ssl-images-amazon.com)... 52.222.171.116
Connecting to images-na.ssl-images-amazon.com (images-na.ssl-images-amazon.com)|52.222.171.116|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 12011 (12K) [image/jpeg]
Saving to: '412V6B5Fo4L._SY445_QL70_.jpg.3'

412V6B5Fo4L._SY445_ 100%[===================>]  11.73K  --.-KB/s    in 0s

2019-05-13 22:07:48 (323 MB/s) - '412V6B5Fo4L._SY445_QL70_.jpg.3' saved [12011/12011]
```

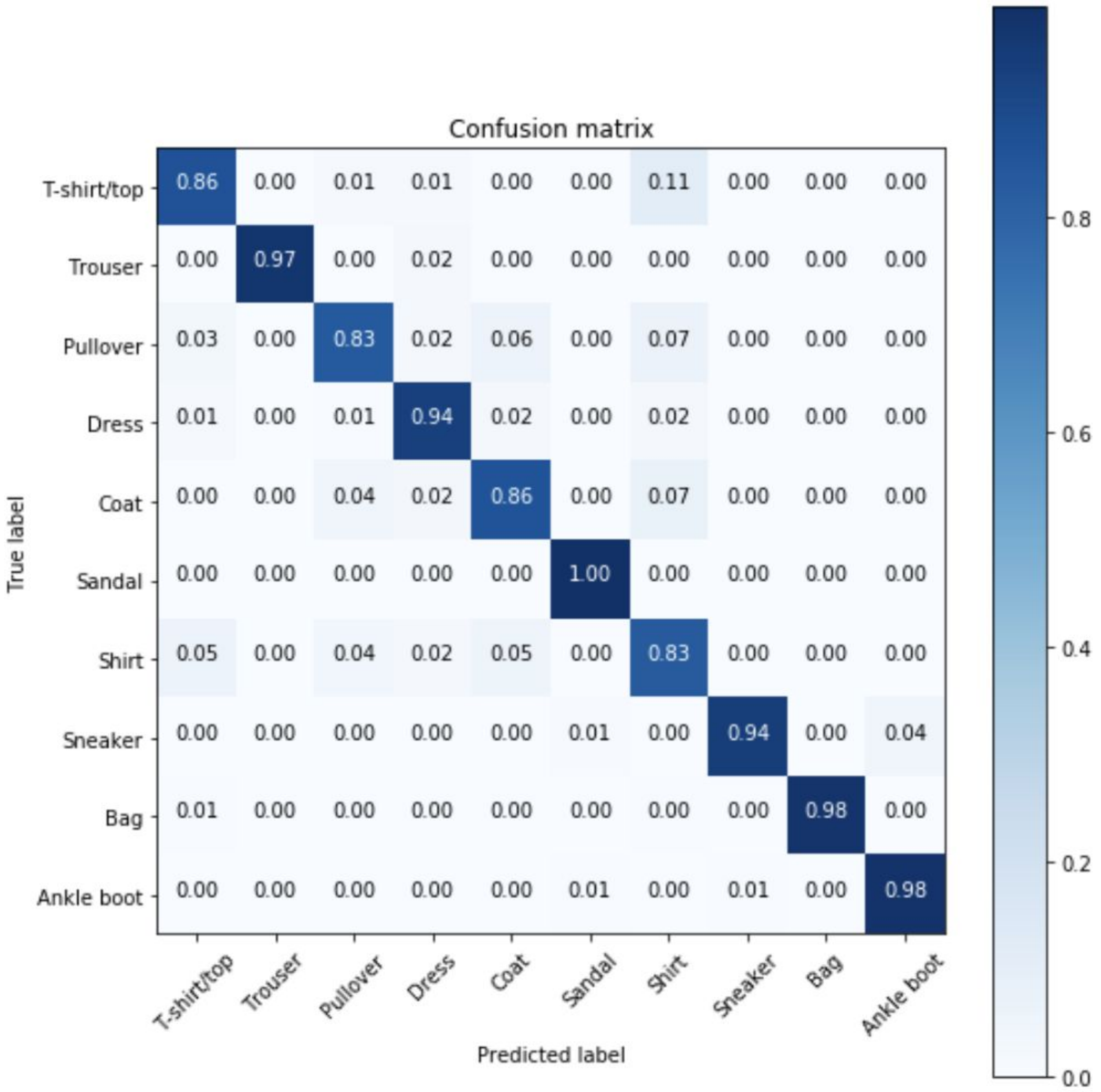
```
(1, 28, 28, 3)
```



```
Predicted as Dress
```

The model is not a 100% accurate. This misclassification can be observed below looking at the confusion matrix. The confusion matrix helps us describe the performance of the classifier visually by plotting the true labels with the predicted labels.

Here we see that most of the misclassifications are happening between the T-shirt/top, Pullover, Coat and Shirt, which is impacting the performance of the classifier.



	T-shirt/top	Trouser	Pullover	Dress	Coat	Sandal	Shirt	Sneaker	Bag	Ankle boot
T-shirt/top	0.86	0.00	0.01	0.01	0.00	0.00	0.11	0.00	0.00	0.00
Trouser	0.00	0.97	0.00	0.02	0.00	0.00	0.00	0.00	0.00	0.00
Pullover	0.03	0.00	0.83	0.02	0.06	0.00	0.07	0.00	0.00	0.00
Dress	0.01	0.00	0.01	0.94	0.02	0.00	0.02	0.00	0.00	0.00
Coat	0.00	0.00	0.04	0.02	0.86	0.00	0.07	0.00	0.00	0.00
Sandal	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00
Shirt	0.05	0.00	0.04	0.02	0.05	0.00	0.83	0.00	0.00	0.00
Sneaker	0.00	0.00	0.00	0.00	0.00	0.01	0.00	0.94	0.00	0.04
Bag	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.98	0.00
Ankle boot	0.00	0.00	0.00	0.00	0.00	0.01	0.00	0.01	0.00	0.98

Conclusion

In conclusion, I feel that my model performed very well giving an accuracy of ~91%. From the above confusion matrix we see that there were some misclassification, most of which was happening between the classes of Shirt, T-shirt/top, Pullover and Coat. This majorly impacted the performance of the classifier.

As a deep learning model needs a large number of data to be trained on to achieve good performance, I would collect more samples and give more images, especially from the above 4 classes to help the classifier better learn the pattern for these classes.

2-minute video youtube link: https://youtu.be/gHL2ggJRzm8

15-minute video youtube link: https://youtu.be/Ii9cFvA-EiQ