Step # 1 - Import Libraries

Lets import all the libraries we are going to require for this classification project. It is always good to put all the import statements at the begining of the file.

In [ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sbn
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Dropout, Dense, Flatten
from keras.optimizers import Adam
from keras.callbacks import TensorBoard
from keras.utils import to_categorical
```

Step # 2 - Load Data

Now lets use **pandas** library to read the train and test datasets in the respective csv files. We are going to use the **read_csv** function which reads a csv file and returns a pandas **DataFrame** object.

In [ ]:
```
fashion_train_df = pd.read_csv('fashion-mnist_train.csv')
fashion_test_df = pd.read_csv('fashion-mnist_test.csv')
```

Now that we have loaded the datasets, lets check some parameters about the datasets.

In [ ]:
```
fashion_train_df.shape   # Shape of the dataset
```

Out[ ]:
```
(60000, 785)
```

In [ ]:
```
fashion_train_df.columns   # Name of the columns of the DataSet.
```

Out[ ]:
```
Index(['label', 'pixel1', 'pixel2', 'pixel3', 'pixel4', 'pixel5', 'pixel6',
       'pixel7', 'pixel8', 'pixel9',
       ...
       'pixel775', 'pixel776', 'pixel777', 'pixel778', 'pixel779', 'pixel780',
       'pixel781', 'pixel782', 'pixel783', 'pixel784'],
      dtype='object', length=785)
```

So we can see that the 1st column is the label or target value for each row.

Now Lets find out how many distinct lables we have.

In [ ]:
```
print(set(fashion_train_df['label']))
```

```
{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
```

So we have 10 different lables. from 0 to 9.

Now lets find out what is the min and max of values of in the other columns.

In [ ]:
```
print([fashion_train_df.drop(labels='label', axis=1).min(axis=1).min(),
      fashion_train_df.drop(labels='label', axis=1).max(axis=1).max()])
```

```
[0, 255]
```

So we have 0 to 255 which is the color values for grayscale. 0 being white and 255 being black.

Now lets check some of the rows in tabular format

```
In [ ]: fashion_train_df.head()
```

Out[ ]:

	label	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	...	pixel775	pixel776	pixel777
0	2	0	0	0	0	0	0	0	0	0	...	0	0	0
1	9	0	0	0	0	0	0	0	0	0	...	0	0	0
2	6	0	0	0	0	0	0	0	5	0	...	0	0	0
3	0	0	0	0	1	2	0	0	0	0	...	3	0	0
4	3	0	0	0	0	0	0	0	0	0	...	0	0	0

5 rows × 785 columns

So evry other things of the test dataset are going to be the same as the train dataset except the shape.

```
In [ ]: fashion_test_df.shape
```

```
Out[ ]: (10000, 785)
```

So here we have 10000 images instead of 60000 as in the train dataset.

Lets check first few rows.

```
In [ ]: fashion_test_df.head()
```

Out[ ]:

	label	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	...	pixel775	pixel776	pixel777
0	0	0	0	0	0	0	0	0	9	8	...	103	87	56
1	1	0	0	0	0	0	0	0	0	0	...	34	0	0
2	2	0	0	0	0	0	0	14	53	99	...	0	0	0
3	2	0	0	0	0	0	0	0	0	0	...	137	126	140
4	3	0	0	0	0	0	0	0	0	0	...	0	0	0

5 rows × 785 columns

Step # 3 - Visualization

Now that we have loaded the data and also got somewhat acquainted with it lets visualize the actual images. We are going to use **Matplotlib** library for this.

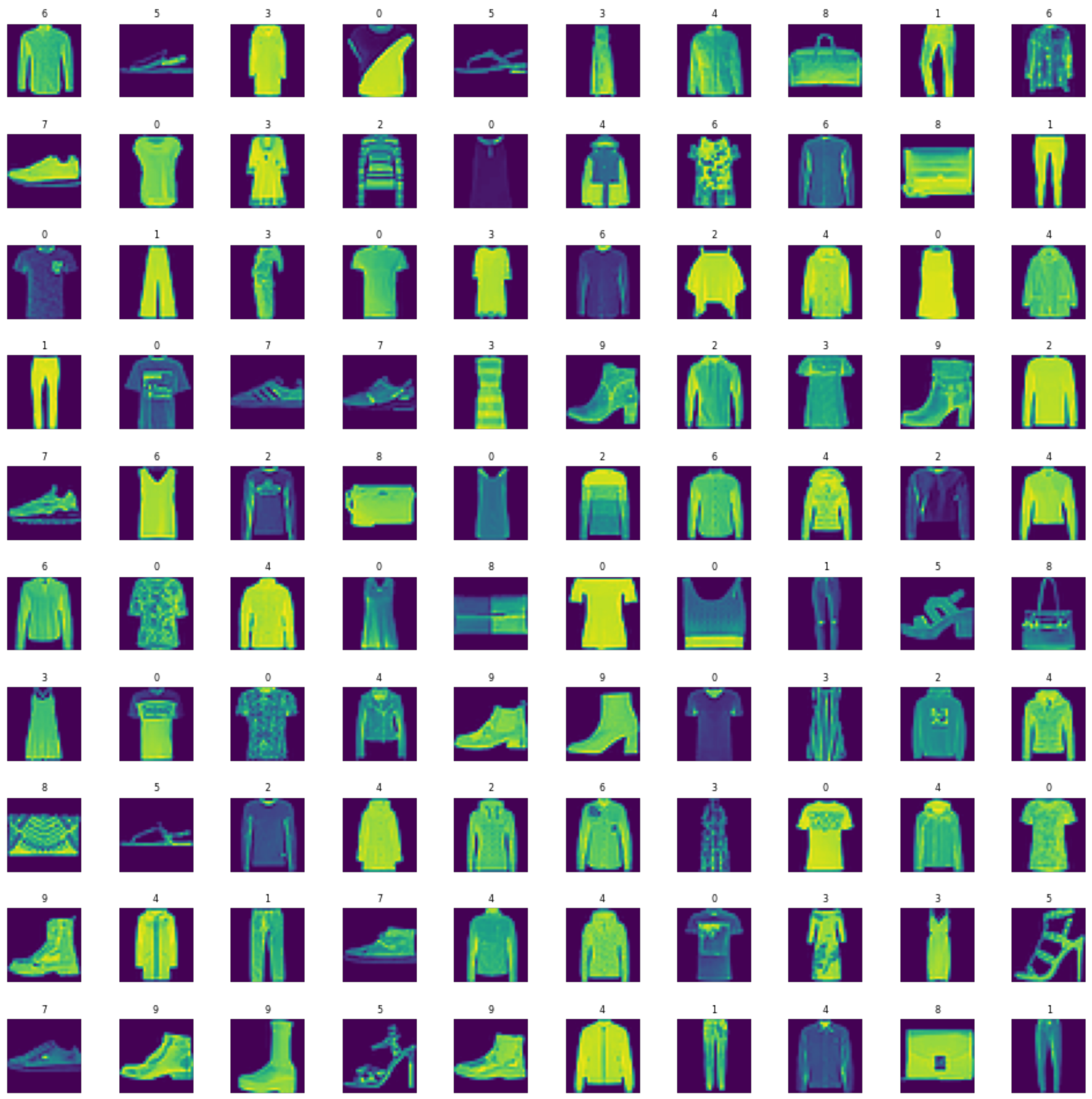
In [ ]:
```python
# Convert the dataframe ti numpy array
training = np.asarray(fashion_train_df, dtype='float32')

# Lets show multiple images in a 15x15 grid
height = 10
width = 10

fig, axes = plt.subplots(nrows=width, ncols=height, figsize=(17,17))
axes = axes.ravel()  # this flattens the 15x15 matrix into 225
n_train = len(training)

for i in range(0, height*width):
    index = np.random.randint(0, n_train)
    axes[i].imshow(training[index, 1:].reshape(28,28))
    axes[i].set_title(int(training[index, 0]), fontsize=8)
    axes[i].axis('off')

plt.subplots_adjust(hspace=0.5)
```



Step # 4 - Preprocess Data

Great! We have visualized the images. So now we can start preparing for creating our model. But before that we need to preprocess our data so that we can fit our model easily. Lets do that first.

In [ ]:
```
# convert to numpy arrays and reshape
training = np.asarray(fashion_train_df, dtype='float32')
X_train = training[:, 1:].reshape([-1,28,28,1])
X_train = X_train/255   # Normalizing the data
y_train = training[:, 0]

testing = np.asarray(fashion_test_df, dtype='float32')
X_test = testing[:, 1:].reshape([-1,28,28,1])
X_test = X_test/255    # Normalizing the data
y_test = testing[:, 0]
```

Also we need to have three different sets of data for **training, validatin** and **testing**. We already have different sets for training and testing. So we are going to split the training dataset further into two sets and will use one set of training and the other for validation.

In [ ]:
```
# Split the training set into training and validation sets
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.2, random_sta
```

In [ ]:
```
# Lets check the shape of all three datasets
print(X_train.shape, X_val.shape, X_test.shape)
print(y_train.shape, y_val.shape, y_test.shape)
```

```
(48000, 28, 28, 1) (12000, 28, 28, 1) (10000, 28, 28, 1)
(48000,) (12000,) (10000,)
```

Step # 5 - Create and Train the Model

Create the model

In [ ]:
```
cnn_model = Sequential()
cnn_model.add(Conv2D(filters=64, kernel_size=(3,3), input_shape=(28,28,1), activation='relu')
cnn_model.add(MaxPooling2D(pool_size = (2,2)))
cnn_model.add(Dropout(rate=0.3))
cnn_model.add(Flatten())
cnn_model.add(Dense(units=32, activation='relu'))
cnn_model.add(Dense(units=10, activation='sigmoid'))
```

compile the model

In [ ]:
```
cnn_model.compile(optimizer=Adam(lr=0.001), loss='sparse_categorical_crossentropy', metrics=[
cnn_model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d (Conv2D)             (None, 26, 26, 64)        640       
                                                                 
 max_pooling2d (MaxPooling2D  (None, 13, 13, 64)       0         
 )                                                               
                                                                 
 dropout (Dropout)           (None, 13, 13, 64)        0         
                                                                 
 flatten (Flatten)           (None, 10816)             0         
                                                                 
 dense (Dense)               (None, 32)                346144    
                                                                 
 dense_1 (Dense)             (None, 10)                330       
                                                                 
=================================================================
Total params: 347,114
Trainable params: 347,114
Non-trainable params: 0
_________________________________________________________________
C:\Users\gdalv\AppData\Roaming\Python\Python310\site-packages\keras\optimizers\legacy\adam.p
y:117: UserWarning: The `lr` argument is deprecated, use `learning_rate` instead.
  super().__init__(name, **kwargs)
```

Train the model

```
In [ ]: cnn_model.fit(x=X_train, y=y_train, batch_size=512, epochs=50, validation_data=(X_val, y_val)
Epoch 1/50
94/94 [==============================] - 17s 174ms/step - loss: 0.7018 - accuracy: 0.7616 - v
al_loss: 0.4662 - val_accuracy: 0.8353
Epoch 2/50
94/94 [==============================] - 15s 164ms/step - loss: 0.4260 - accuracy: 0.8514 - v
al_loss: 0.3990 - val_accuracy: 0.8609
Epoch 3/50
94/94 [==============================] - 15s 163ms/step - loss: 0.3726 - accuracy: 0.8700 - v
al_loss: 0.3512 - val_accuracy: 0.8805
Epoch 4/50
94/94 [==============================] - 15s 158ms/step - loss: 0.3419 - accuracy: 0.8804 - v
al_loss: 0.3298 - val_accuracy: 0.8857
Epoch 5/50
94/94 [==============================] - 16s 173ms/step - loss: 0.3238 - accuracy: 0.8864 - v
al_loss: 0.3099 - val_accuracy: 0.8937
Epoch 6/50
94/94 [==============================] - 16s 172ms/step - loss: 0.3051 - accuracy: 0.8930 - v
al_loss: 0.3034 - val_accuracy: 0.8957
Epoch 7/50
94/94 [==============================] - 16s 167ms/step - loss: 0.2909 - accuracy: 0.8980 - v
al_loss: 0.2871 - val_accuracy: 0.8987
Epoch 8/50
94/94 [==============================] - 16s 174ms/step - loss: 0.2796 - accuracy: 0.9011 - v
al_loss: 0.2848 - val_accuracy: 0.9015
Epoch 9/50
94/94 [==============================] - 16s 168ms/step - loss: 0.2731 - accuracy: 0.9034 - v
al_loss: 0.2767 - val_accuracy: 0.9045
Epoch 10/50
94/94 [==============================] - 16s 170ms/step - loss: 0.2616 - accuracy: 0.9070 - v
al_loss: 0.2680 - val_accuracy: 0.9067
Epoch 11/50
94/94 [==============================] - 16s 165ms/step - loss: 0.2545 - accuracy: 0.9089 - v
al_loss: 0.2749 - val_accuracy: 0.9032
Epoch 12/50
94/94 [==============================] - 16s 170ms/step - loss: 0.2532 - accuracy: 0.9091 - v
al_loss: 0.2583 - val_accuracy: 0.9103
Epoch 13/50
94/94 [==============================] - 15s 165ms/step - loss: 0.2407 - accuracy: 0.9140 - v
al_loss: 0.2623 - val_accuracy: 0.9082
Epoch 14/50
94/94 [==============================] - 16s 172ms/step - loss: 0.2387 - accuracy: 0.9144 - v
al_loss: 0.2633 - val_accuracy: 0.9071
Epoch 15/50
94/94 [==============================] - 16s 169ms/step - loss: 0.2316 - accuracy: 0.9168 - v
al_loss: 0.2545 - val_accuracy: 0.9087
Epoch 16/50
94/94 [==============================] - 16s 169ms/step - loss: 0.2265 - accuracy: 0.9182 - v
al_loss: 0.2452 - val_accuracy: 0.9130
Epoch 17/50
94/94 [==============================] - 16s 169ms/step - loss: 0.2220 - accuracy: 0.9206 - v
al_loss: 0.2482 - val_accuracy: 0.9130
Epoch 18/50
94/94 [==============================] - 16s 168ms/step - loss: 0.2173 - accuracy: 0.9212 - v
al_loss: 0.2413 - val_accuracy: 0.9163
Epoch 19/50
94/94 [==============================] - 17s 178ms/step - loss: 0.2124 - accuracy: 0.9235 - v
al_loss: 0.2423 - val_accuracy: 0.9152
Epoch 20/50
94/94 [==============================] - 18s 188ms/step - loss: 0.2065 - accuracy: 0.9264 - v
al_loss: 0.2441 - val_accuracy: 0.9149
Epoch 21/50
94/94 [==============================] - 16s 174ms/step - loss: 0.2057 - accuracy: 0.9252 - v
al_loss: 0.2375 - val_accuracy: 0.9188
Epoch 22/50
94/94 [==============================] - 16s 175ms/step - loss: 0.2006 - accuracy: 0.9274 - v
al_loss: 0.2367 - val_accuracy: 0.9169
Epoch 23/50
94/94 [==============================] - 16s 173ms/step - loss: 0.1995 - accuracy: 0.9275 - v
al_loss: 0.2392 - val_accuracy: 0.9172
```

```
Epoch 24/50
94/94 [==============================] - 16s 173ms/step - loss: 0.1908 - accuracy: 0.9315 - v
al_loss: 0.2424 - val_accuracy: 0.9161
Epoch 25/50
94/94 [==============================] - 16s 171ms/step - loss: 0.1879 - accuracy: 0.9323 - v
al_loss: 0.2387 - val_accuracy: 0.9169
Epoch 26/50
94/94 [==============================] - 16s 176ms/step - loss: 0.1869 - accuracy: 0.9316 - v
al_loss: 0.2329 - val_accuracy: 0.9196
Epoch 27/50
94/94 [==============================] - 16s 167ms/step - loss: 0.1837 - accuracy: 0.9333 - v
al_loss: 0.2342 - val_accuracy: 0.9183
Epoch 28/50
94/94 [==============================] - 15s 164ms/step - loss: 0.1819 - accuracy: 0.9341 - v
al_loss: 0.2364 - val_accuracy: 0.9164
Epoch 29/50
94/94 [==============================] - 16s 174ms/step - loss: 0.1781 - accuracy: 0.9356 - v
al_loss: 0.2393 - val_accuracy: 0.9148
Epoch 30/50
94/94 [==============================] - 16s 168ms/step - loss: 0.1733 - accuracy: 0.9368 - v
al_loss: 0.2498 - val_accuracy: 0.9128
Epoch 31/50
94/94 [==============================] - 15s 162ms/step - loss: 0.1728 - accuracy: 0.9367 - v
al_loss: 0.2373 - val_accuracy: 0.9177
Epoch 32/50
94/94 [==============================] - 15s 163ms/step - loss: 0.1699 - accuracy: 0.9376 - v
al_loss: 0.2579 - val_accuracy: 0.9073
Epoch 33/50
94/94 [==============================] - 15s 159ms/step - loss: 0.1663 - accuracy: 0.9399 - v
al_loss: 0.2331 - val_accuracy: 0.9207
Epoch 34/50
94/94 [==============================] - 15s 159ms/step - loss: 0.1645 - accuracy: 0.9393 - v
al_loss: 0.2306 - val_accuracy: 0.9204
Epoch 35/50
94/94 [==============================] - 15s 161ms/step - loss: 0.1620 - accuracy: 0.9404 - v
al_loss: 0.2388 - val_accuracy: 0.9183
Epoch 36/50
94/94 [==============================] - 15s 161ms/step - loss: 0.1632 - accuracy: 0.9398 - v
al_loss: 0.2353 - val_accuracy: 0.9187
Epoch 37/50
94/94 [==============================] - 16s 172ms/step - loss: 0.1551 - accuracy: 0.9437 - v
al_loss: 0.2310 - val_accuracy: 0.9220
Epoch 38/50
94/94 [==============================] - 17s 181ms/step - loss: 0.1559 - accuracy: 0.9426 - v
al_loss: 0.2321 - val_accuracy: 0.9220
Epoch 39/50
94/94 [==============================] - 17s 176ms/step - loss: 0.1502 - accuracy: 0.9442 - v
al_loss: 0.2296 - val_accuracy: 0.9208
Epoch 40/50
94/94 [==============================] - 16s 173ms/step - loss: 0.1528 - accuracy: 0.9442 - v
al_loss: 0.2375 - val_accuracy: 0.9211
Epoch 41/50
94/94 [==============================] - 16s 172ms/step - loss: 0.1468 - accuracy: 0.9458 - v
al_loss: 0.2363 - val_accuracy: 0.9211
Epoch 42/50
94/94 [==============================] - 16s 170ms/step - loss: 0.1429 - accuracy: 0.9475 - v
al_loss: 0.2366 - val_accuracy: 0.9203
Epoch 43/50
94/94 [==============================] - 16s 171ms/step - loss: 0.1428 - accuracy: 0.9485 - v
al_loss: 0.2366 - val_accuracy: 0.9208
Epoch 44/50
94/94 [==============================] - 16s 172ms/step - loss: 0.1393 - accuracy: 0.9481 - v
al_loss: 0.2446 - val_accuracy: 0.9203
Epoch 45/50
94/94 [==============================] - 16s 169ms/step - loss: 0.1392 - accuracy: 0.9485 - v
al_loss: 0.2397 - val_accuracy: 0.9208
Epoch 46/50
94/94 [==============================] - 16s 172ms/step - loss: 0.1372 - accuracy: 0.9503 - v
al_loss: 0.2405 - val_accuracy: 0.9197
```

```
Epoch 47/50
94/94 [==============================] - 16s 171ms/step - loss: 0.1338 - accuracy: 0.9516 - v
al_loss: 0.2386 - val_accuracy: 0.9211
Epoch 48/50
94/94 [==============================] - 16s 166ms/step - loss: 0.1324 - accuracy: 0.9511 - v
al_loss: 0.2394 - val_accuracy: 0.9212
Epoch 49/50
94/94 [==============================] - 16s 171ms/step - loss: 0.1293 - accuracy: 0.9530 - v
al_loss: 0.2435 - val_accuracy: 0.9215
Epoch 50/50
94/94 [==============================] - 16s 167ms/step - loss: 0.1272 - accuracy: 0.9520 - v
al_loss: 0.2500 - val_accuracy: 0.9186
```

Out[ ]:

```
<keras.callbacks.History at 0x25621b32fb0>
```

Step # 5 - Evaluate the Model

Get the accuracy of the model

In [ ]:

```
eval_result = cnn_model.evaluate(X_test, y_test)
print("Accuracy : {:.3f}".format(eval_result[1]))
```

```
313/313 [==============================] - 2s 5ms/step - loss: 0.2454 - accuracy: 0.9186
Accuracy : 0.919
```