FASHION CLASS CLASSIFICATION

June 7, 2019

1 FASHION CLASS CLASSIFICATION

2 Md. Manir Uddin

2.1 Introduction:

The truth is, data science and big data analytics play a crucial role today in helping trendsetters pinpoint the ever-evolving shifts and changes present in fashion, and in helping everyone from manufacturers to models tackle the runway and the real world with style and finesse. Here's how they're doing it:

Traditionally, fashion houses and brands kept vital information like sales records and inventory details in-house. But this also meant that they worked in a silo—that much of the colors, style, fit and other decisions for their garments were mostly scattered, unstructured data. They lacked other crucial pieces of the puzzle such as competitive analysis, pricing, trends, insights and other must-have details.

With the fashion industry, every possible facet of a piece of clothing is under scrutiny. From the fabric to the closures to the sizes and the style, everything is collected and analyzed. For those with the right data science degree, this presents an eclectic challenge—how to stay focused and on top of trends before they're forgotten.

Using Fashion MNIST Dataset so we can see small part, how we can use machine learning in the Fashion industry.

3 DESCRIPTION OF THE PROBLEM:

Fashion training set consists of 70,000 images divided into 60,000 training and 10,000 testing samples. Dataset sample consists of 28x28 grayscale image, associated with a label from 10 classes. The 10 classes are as follows: 0 => T-shirt/top 1 => Trouser 2 => Pullover 3 => Dress 4 => Coat 5 => Sandal 6 => Shirt 7 => Sneaker 8 => Bag 9 => Ankle boot

Each image is 28 pixels in height and 28 pixels in width, for a total of 784 pixels in total. Each pixel has a single pixel- value associated with it, indicating the lightness or darkness of that pixel, with higher numbers meaning darker. This pixel- value is an integer between 0 and 255.

We have assigned the following number codes and the objects corresponding to them as follows: - 0 = T-shirt/top - 1 = Trouser - 2 = Pullover - 3 = Dress - 4 = Coat - 5 = Sandal - 6 = Shirt - 7 = Sneaker - 8 = Bag - 9 = Ankle boot

3.0.1 Our model Achieves an accuracy of 91.5%

We can see from the image below that our model recognizes the appropriate images with great accuracy for 15 x 15 plot grid.

4 STEP #1: PROBLEM STATEMENT AND BUSINESS CASE

Fashion training set consists of 70,000 images divided into 60,000 training and 10,000 testing samples. Dataset sample consists of 28x28 grayscale image, associated with a label from 10 classes.

The 10 classes are as follows:
0 => T-shirt/top 1 => Trouser 2 => Pullover 3 => Dress 4 => Coat 5 => Sandal 6 => Shirt 7 => Sneaker 8 => Bag 9 => Ankle boot

Each image is 28 pixels in height and 28 pixels in width, for a total of 784 pixels in total. Each pixel has a single pixel-value associated with it, indicating the lightness or darkness of that pixel, with higher numbers meaning darker. This pixel-value is an integer between 0 and 255.

image.png

Basic review: what is an image? A greyscale image is the system of 256 tones with values ranging from 0–255. '0' represents black and '255' represents white. Numbers in-between represents greys between black and white Binary systems use digits '0' and '1' where '00000000' for black, to '11111111' for white (8-bit image). Note: the binary value of '11111111' is equal to the decimal value of '255.

Our Fashion dataset

Fashion dataset contains 28x28 greyscale image with values ranging from 0–255. '0' represents black and '255' represents white. Each image is represented by a row with 784(i.e:28x28) values.

5 STEP #2: IMPORTING DATA

[1]:
```
# import libraries
import pandas as pd # Import Pandas for data manipulation using dataframes
import numpy as np # Import Numpy for data statistical analysis
import matplotlib.pyplot as plt # Import matplotlib for data visualisation
import seaborn as sns
import random
```

[3]:
```
# dataframes creation for both training and testing datasets
fashion_train_df = pd.read_csv('fashion-mnist_train.csv',sep=',')
fashion_test_df = pd.read_csv('fashion-mnist_test.csv', sep = ',')
```

6 STEP #3: VISUALIZATION OF THE DATASET

[4]:
```
# Let's view the head of the training dataset
# 784 indicates 28x28 pixels and 1 coloumn for the label
# After you check the tail, 60,000 training dataset are present
fashion_train_df.head()
```

[4]:
```
   label  pixel1  pixel2  pixel3  pixel4  pixel5  pixel6  pixel7  pixel8  \
0      2       0       0       0       0       0       0       0       0
1      9       0       0       0       0       0       0       0       0
2      6       0       0       0       0       0       0       0       5
3      0       0       0       0       1       2       0       0       0
4      3       0       0       0       0       0       0       0       0

   pixel9  ...  pixel775  pixel776  pixel777  pixel778  pixel779  pixel780  \
0       0  ...         0         0         0         0         0         0
1       0  ...         0         0         0         0         0         0
2       0  ...         0         0         0        30        43         0
3       0  ...         3         0         0         0         0         1
4       0  ...         0         0         0         0         0         0

   pixel781  pixel782  pixel783  pixel784
0         0         0         0         0
1         0         0         0         0
2         0         0         0         0
3         0         0         0         0
4         0         0         0         0

[5 rows x 785 columns]
```

[5]:
```
# Let's view the last elements in the training dataset
fashion_train_df.tail()
```

[5]:
```
       label  pixel1  pixel2  pixel3  pixel4  pixel5  pixel6  pixel7  pixel8  \
59995      9       0       0       0       0       0       0       0       0
59996      1       0       0       0       0       0       0       0       0
59997      8       0       0       0       0       0       0       0       0
59998      8       0       0       0       0       0       0       0       0
59999      7       0       0       0       0       0       0       0       0

       pixel9  ...  pixel775  pixel776  pixel777  pixel778  pixel779  \
59995       0  ...         0         0         0         0         0
59996       0  ...        73         0         0         0         0
59997       0  ...       160       162       163       135        94
59998       0  ...         0         0         0         0         0
59999       0  ...         0         0         0         0         0

       pixel780  pixel781  pixel782  pixel783  pixel784
59995         0         0         0         0         0
```

```
59996         0          0          0          0          0
59997         0          0          0          0          0
59998         0          0          0          0          0
59999         0          0          0          0          0

[5 rows x 785 columns]
```

[6]:
```
# Let's view the head of the testing dataset
fashion_test_df.head()
```

[6]:
```
   label  pixel1  pixel2  pixel3  pixel4  pixel5  pixel6  pixel7  pixel8  \
0      0       0       0       0       0       0       0       0       9
1      1       0       0       0       0       0       0       0       0
2      2       0       0       0       0       0       0      14      53
3      2       0       0       0       0       0       0       0       0
4      3       0       0       0       0       0       0       0       0

   pixel9  ...  pixel775  pixel776  pixel777  pixel778  pixel779  pixel780  \
0       8  ...       103        87        56         0         0         0
1       0  ...        34         0         0         0         0         0
2      99  ...         0         0         0         0        63        53
3       0  ...       137       126       140         0       133       224
4       0  ...         0         0         0         0         0         0

   pixel781  pixel782  pixel783  pixel784
0         0         0         0         0
1         0         0         0         0
2        31         0         0         0
3       222        56         0         0
4         0         0         0         0

[5 rows x 785 columns]
```

[7]:
```
# Let's view the last elements in the testing dataset
fashion_test_df.tail()
```

[7]:
```
      label  pixel1  pixel2  pixel3  pixel4  pixel5  pixel6  pixel7  pixel8  \
9995      0       0       0       0       0       0       0       0       0
9996      6       0       0       0       0       0       0       0       0
9997      8       0       0       0       0       0       0       0       0
9998      8       0       1       3       0       0       0       0       0
9999      1       0       0       0       0       0       0       0     140

      pixel9  ...  pixel775  pixel776  pixel777  pixel778  pixel779  pixel780  \
9995       0  ...        32        23        14        20         0         0
9996       0  ...         0         0         0         2        52        23
9997       0  ...       175       172       172       182       199       222
9998       0  ...         0         0         0         0         0         1
9999     119  ...       111        95        75        44         1         0
```

```
      pixel781  pixel782  pixel783  pixel784
9995         1         0         0         0
9996        28         0         0         0
9997        42         0         1         0
9998         0         0         0         0
9999         0         0         0         0

[5 rows x 785 columns]
```

[8]:
```
fashion_train_df.shape
```

[8]:
```
(60000, 785)
```

[9]:
```
# Create training and testing arrays
training = np.array(fashion_train_df, dtype = 'float32')
testing = np.array(fashion_test_df, dtype='float32')
```

[10]:
```
training.shape
```

[10]:
```
(60000, 785)
```

[11]:
```
training
```

[11]:
```
array([[2., 0., 0., ..., 0., 0., 0.],
       [9., 0., 0., ..., 0., 0., 0.],
       [6., 0., 0., ..., 0., 0., 0.],
       ...,
       [8., 0., 0., ..., 0., 0., 0.],
       [8., 0., 0., ..., 0., 0., 0.],
       [7., 0., 0., ..., 0., 0., 0.]], dtype=float32)
```

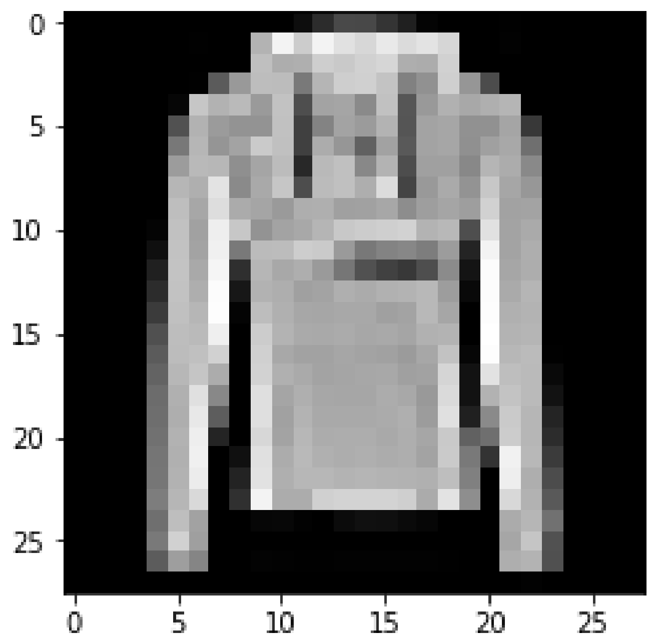
[12]:
```
testing
```

[12]:
```
array([[0., 0., 0., ..., 0., 0., 0.],
       [1., 0., 0., ..., 0., 0., 0.],
       [2., 0., 0., ..., 0., 0., 0.],
       ...,
       [8., 0., 0., ..., 0., 1., 0.],
       [8., 0., 1., ..., 0., 0., 0.],
       [1., 0., 0., ..., 0., 0., 0.]], dtype=float32)
```

[13]:
```
# Let's view some images!
i = random.randint(1,60000) # select any random index from 1 to 60,000
plt.imshow( training[i,1:].reshape((28,28)) ) # reshape and plot the image

plt.imshow( training[i,1:].reshape((28,28)) , cmap = 'gray') # reshape and plot␣
 ↪the image



# Remember the 10 classes decoding is as follows:
# 0 => T-shirt/top
```

```
# 1 => Trouser
# 2 => Pullover
# 3 => Dress
# 4 => Coat
# 5 => Sandal
# 6 => Shirt
# 7 => Sneaker
# 8 => Bag
# 9 => Ankle boot
```

[13]: <matplotlib.image.AxesImage at 0x4e51b7ccc0>

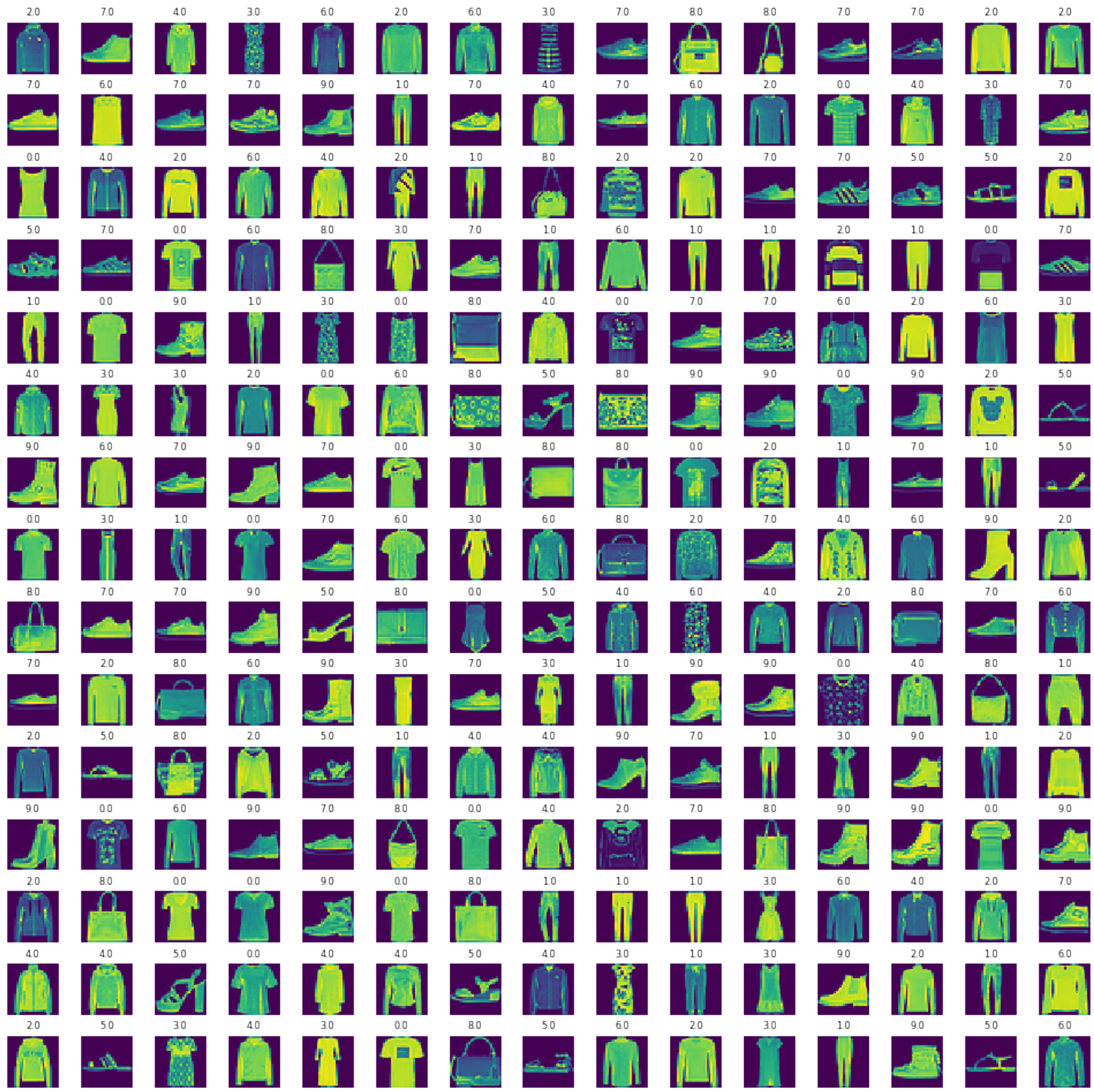


[14]:
```
label = training[i,0]
label
```

[14]: 2.0

[15]:
```
# Let's view more images in a grid format
# Define the dimensions of the plot grid
W_grid = 15
L_grid = 15

# fig, axes = plt.subplots(L_grid, W_grid)
# subplot return the figure object and axes object
# we can use the axes object to plot specific figures at various locations

fig, axes = plt.subplots(L_grid, W_grid, figsize = (17,17))
```

```python
axes = axes.ravel() # flaten the 15 x 15 matrix into 225 array

n_training = len(training) # get the length of the training dataset

# Select a random number from 0 to n_training
for i in np.arange(0, W_grid * L_grid): # create evenly spaces variables

    # Select a random number
    index = np.random.randint(0, n_training)
    # read and display an image with the selected index
    axes[i].imshow( training[index,1:].reshape((28,28)) )
    axes[i].set_title(training[index,0], fontsize = 8)
    axes[i].axis('off')

plt.subplots_adjust(hspace=0.4)

# Remember the 10 classes decoding is as follows:
# 0 => T-shirt/top
# 1 => Trouser
# 2 => Pullover
# 3 => Dress
# 4 => Coat
# 5 => Sandal
# 6 => Shirt
# 7 => Sneaker
# 8 => Bag
# 9 => Ankle boot
```



7 STEP #4: TRAINING THE MODEL

Convolutional neural network- overview

- Convolutional neural network- feature detector
- Convolutional use a kernel matrix to scan a given image and apply a filter to obtain a certain effect.
- An image Kernel is a matrix used to apply effects such as blurring and sharpening
- Kernels are used in machine learning for feature extraction to select most important pixels of an image
- Convolution preserves the spatial relationship between pixels.

[16]:
```
# Prepare the training and testing dataset
X_train = training[:,1:]/255
y_train = training[:,0]

X_test = testing[:,1:]/255
y_test = testing[:,0]
```

[17]:
```
from sklearn.model_selection import train_test_split

X_train, X_validate, y_train, y_validate = train_test_split(X_train, y_train,␣
↪test_size = 0.2, random_state = 12345)
```

[18]:
```
X_train.shape
```

[18]:
```
(48000, 784)
```

[19]:
```
y_train.shape
```

[19]:
```
(48000,)
```

[20]:
```
# * unpack the tuple
X_train = X_train.reshape(X_train.shape[0], *(28, 28, 1))
X_test = X_test.reshape(X_test.shape[0], *(28, 28, 1))
X_validate = X_validate.reshape(X_validate.shape[0], *(28, 28, 1))
```

[21]:
```
X_train.shape
```

[21]:
```
(48000, 28, 28, 1)
```

[22]:
```
X_test.shape
```

[22]:
```
(10000, 28, 28, 1)
```

[23]:
```
X_validate.shape
```

[23]:
```
(12000, 28, 28, 1)
```

[ ]:
```
import keras # open source Neural network library madke our life much easier

# y_train = keras.utils.to_categorical(y_train, 10)
# y_test = keras.utils.to_categorical(y_test, 10)
```

Convolutional neural network- RELU

- RELU layers are used to add non-linearity in the feature map.
- It also enhances the sparsity or how scattered the feature map is.
- The gradient of the RELU does not vanish as we increase x compared to sigmoid function
- Convlutional neural network- maxpooling/flatten
- Pooling or down sampling layers are placed after convolutional layers to reduce feature map dimensionality.
- This improves the computational efficiency while preserving the features.

- Pooling helps the model to generalize by avoiding over-fitting. If one of the pixel is shifted, the pooled feature map will still be the same.
- Max pooling works by retaining the maximum feature response within a given sample size in a feature map.

[24]:
```
# Import train_test_split from scikit library
# Import Keras
import keras
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Dense, Flatten, Dropout
from keras.optimizers import Adam
from keras.callbacks import TensorBoard
```

```
Using TensorFlow backend.
```

[25]:
```
cnn_model = Sequential()

# Try 32 fliters first then 64
cnn_model.add(Conv2D(64,3, 3, input_shape = (28,28,1), activation='relu'))
cnn_model.add(MaxPooling2D(pool_size = (2, 2)))

cnn_model.add(Dropout(0.25))

# cnn_model.add(Conv2D(32,3, 3, activation='relu'))
# cnn_model.add(MaxPooling2D(pool_size = (2, 2)))

cnn_model.add(Flatten())
cnn_model.add(Dense(output_dim = 32, activation = 'relu'))
cnn_model.add(Dense(output_dim = 10, activation = 'sigmoid'))
```

```
C:\Users\Manir Uddin\Anaconda3\envs\tensorflow\lib\site-
packages\ipykernel_launcher.py:4: UserWarning: Update your `Conv2D` call to the
Keras 2 API: `Conv2D(64, (3, 3), input_shape=(28, 28, 1..., activation="relu")`
  after removing the cwd from sys.path.
C:\Users\Manir Uddin\Anaconda3\envs\tensorflow\lib\site-
packages\ipykernel_launcher.py:13: UserWarning: Update your `Dense` call to the
Keras 2 API: `Dense(activation="relu", units=32)`
  del sys.path[0]
C:\Users\Manir Uddin\Anaconda3\envs\tensorflow\lib\site-
packages\ipykernel_launcher.py:14: UserWarning: Update your `Dense` call to the
Keras 2 API: `Dense(activation="sigmoid", units=10)`
```

[26]:
```
cnn_model.compile(loss ='sparse_categorical_crossentropy', optimizer=Adam(lr=0.
↪001),metrics =['accuracy'])
```

[27]:
```
epochs = 50

history = cnn_model.fit(X_train,
                        y_train,
                        batch_size = 512,
                        nb_epoch = epochs,
                        verbose = 1,
                        validation_data = (X_validate, y_validate))
```

```
C:\Users\Manir Uddin\Anaconda3\envs\tensorflow\lib\site-
packages\ipykernel_launcher.py:8: UserWarning: The `nb_epoch` argument in `fit`
has been renamed `epochs`.
```

```
Train on 48000 samples, validate on 12000 samples
Epoch 1/50
48000/48000 [==============================] - 70s 1ms/step - loss: 0.9308 -
acc: 0.6579 - val_loss: 0.5174 - val_acc: 0.8160
Epoch 2/50
48000/48000 [==============================] - 68s 1ms/step - loss: 0.4970 -
acc: 0.8245 - val_loss: 0.4827 - val_acc: 0.8215
Epoch 3/50
48000/48000 [==============================] - 68s 1ms/step - loss: 0.4334 -
acc: 0.8508 - val_loss: 0.4067 - val_acc: 0.8584
Epoch 4/50
48000/48000 [==============================] - 69s 1ms/step - loss: 0.4004 -
acc: 0.8604 - val_loss: 0.3820 - val_acc: 0.8693
Epoch 5/50
48000/48000 [==============================] - 69s 1ms/step - loss: 0.3784 -
acc: 0.8683 - val_loss: 0.3715 - val_acc: 0.8712
Epoch 6/50
48000/48000 [==============================] - 68s 1ms/step - loss: 0.3585 -
acc: 0.8742 - val_loss: 0.3451 - val_acc: 0.8805
Epoch 7/50
48000/48000 [==============================] - 68s 1ms/step - loss: 0.3407 -
acc: 0.8829 - val_loss: 0.3293 - val_acc: 0.8860
Epoch 8/50
48000/48000 [==============================] - 68s 1ms/step - loss: 0.3323 -
acc: 0.8841 - val_loss: 0.3189 - val_acc: 0.8906
Epoch 9/50
48000/48000 [==============================] - 68s 1ms/step - loss: 0.3187 -
acc: 0.8888 - val_loss: 0.3117 - val_acc: 0.8923
Epoch 10/50
48000/48000 [==============================] - 68s 1ms/step - loss: 0.3101 -
acc: 0.8920 - val_loss: 0.3136 - val_acc: 0.8875
Epoch 11/50
48000/48000 [==============================] - 68s 1ms/step - loss: 0.3031 -
acc: 0.8928 - val_loss: 0.2972 - val_acc: 0.8982
```

```
Epoch 12/50
48000/48000 [==============================] - 68s 1ms/step - loss: 0.2925 -
acc: 0.8978 - val_loss: 0.2958 - val_acc: 0.8988
Epoch 13/50
48000/48000 [==============================] - 68s 1ms/step - loss: 0.2849 -
acc: 0.8996 - val_loss: 0.2963 - val_acc: 0.8931
Epoch 14/50
48000/48000 [==============================] - 69s 1ms/step - loss: 0.2816 -
acc: 0.9006 - val_loss: 0.2830 - val_acc: 0.9011
Epoch 15/50
48000/48000 [==============================] - 72s 2ms/step - loss: 0.2732 -
acc: 0.9036 - val_loss: 0.2846 - val_acc: 0.9004
Epoch 16/50
48000/48000 [==============================] - 70s 1ms/step - loss: 0.2674 -
acc: 0.9061 - val_loss: 0.2777 - val_acc: 0.9033
Epoch 17/50
48000/48000 [==============================] - 68s 1ms/step - loss: 0.2595 -
acc: 0.9095 - val_loss: 0.2759 - val_acc: 0.9042
Epoch 18/50
48000/48000 [==============================] - 72s 1ms/step - loss: 0.2584 -
acc: 0.9084 - val_loss: 0.2711 - val_acc: 0.9046
Epoch 19/50
48000/48000 [==============================] - 70s 1ms/step - loss: 0.2549 -
acc: 0.9098 - val_loss: 0.2699 - val_acc: 0.9063
Epoch 20/50
48000/48000 [==============================] - 79s 2ms/step - loss: 0.2483 -
acc: 0.9119 - val_loss: 0.2647 - val_acc: 0.9085
Epoch 21/50
48000/48000 [==============================] - 76s 2ms/step - loss: 0.2486 -
acc: 0.9114 - val_loss: 0.2642 - val_acc: 0.9079
Epoch 22/50
48000/48000 [==============================] - 69s 1ms/step - loss: 0.2403 -
acc: 0.9147 - val_loss: 0.2668 - val_acc: 0.9060
Epoch 23/50
48000/48000 [==============================] - 68s 1ms/step - loss: 0.2403 -
acc: 0.9145 - val_loss: 0.2905 - val_acc: 0.8964
Epoch 24/50
48000/48000 [==============================] - 68s 1ms/step - loss: 0.2375 -
acc: 0.9153 - val_loss: 0.2702 - val_acc: 0.9037
Epoch 25/50
48000/48000 [==============================] - 76s 2ms/step - loss: 0.2320 -
acc: 0.9168 - val_loss: 0.2640 - val_acc: 0.9078
Epoch 26/50
48000/48000 [==============================] - 75s 2ms/step - loss: 0.2285 -
acc: 0.9174 - val_loss: 0.2559 - val_acc: 0.9107
Epoch 27/50
48000/48000 [==============================] - 75s 2ms/step - loss: 0.2245 -
acc: 0.9205 - val_loss: 0.2624 - val_acc: 0.9070
```

```
Epoch 28/50
48000/48000 [==============================] - 69s 1ms/step - loss: 0.2230 -
acc: 0.9200 - val_loss: 0.2575 - val_acc: 0.9106
Epoch 29/50
48000/48000 [==============================] - 68s 1ms/step - loss: 0.2193 -
acc: 0.9216 - val_loss: 0.2560 - val_acc: 0.9113
Epoch 30/50
48000/48000 [==============================] - 68s 1ms/step - loss: 0.2161 -
acc: 0.9222 - val_loss: 0.2535 - val_acc: 0.9107
Epoch 31/50
48000/48000 [==============================] - 69s 1ms/step - loss: 0.2104 -
acc: 0.9243 - val_loss: 0.2532 - val_acc: 0.9117
Epoch 32/50
48000/48000 [==============================] - 68s 1ms/step - loss: 0.2103 -
acc: 0.9237 - val_loss: 0.2555 - val_acc: 0.9117
Epoch 33/50
48000/48000 [==============================] - 68s 1ms/step - loss: 0.2101 -
acc: 0.9245 - val_loss: 0.2533 - val_acc: 0.9116
Epoch 34/50
48000/48000 [==============================] - 68s 1ms/step - loss: 0.2059 -
acc: 0.9250 - val_loss: 0.2598 - val_acc: 0.9092
Epoch 35/50
48000/48000 [==============================] - 69s 1ms/step - loss: 0.2004 -
acc: 0.9271 - val_loss: 0.2580 - val_acc: 0.9097
Epoch 36/50
48000/48000 [==============================] - 68s 1ms/step - loss: 0.1998 -
acc: 0.9282 - val_loss: 0.2591 - val_acc: 0.9089
Epoch 37/50
48000/48000 [==============================] - 70s 1ms/step - loss: 0.2010 -
acc: 0.9268 - val_loss: 0.2532 - val_acc: 0.9122
Epoch 38/50
48000/48000 [==============================] - 69s 1ms/step - loss: 0.1976 -
acc: 0.9293 - val_loss: 0.2570 - val_acc: 0.9115
Epoch 39/50
48000/48000 [==============================] - 68s 1ms/step - loss: 0.1986 -
acc: 0.9276 - val_loss: 0.2533 - val_acc: 0.9127
Epoch 40/50
48000/48000 [==============================] - 69s 1ms/step - loss: 0.1907 -
acc: 0.9309 - val_loss: 0.2512 - val_acc: 0.9132
Epoch 41/50
48000/48000 [==============================] - 68s 1ms/step - loss: 0.1891 -
acc: 0.9313 - val_loss: 0.2614 - val_acc: 0.9077
Epoch 42/50
48000/48000 [==============================] - 68s 1ms/step - loss: 0.1879 -
acc: 0.9322 - val_loss: 0.2606 - val_acc: 0.9103
Epoch 43/50
48000/48000 [==============================] - 68s 1ms/step - loss: 0.1839 -
acc: 0.9334 - val_loss: 0.2491 - val_acc: 0.9141
```

```
Epoch 44/50
48000/48000 [==============================] - 69s 1ms/step - loss: 0.1832 -
acc: 0.9346 - val_loss: 0.2587 - val_acc: 0.9102
Epoch 45/50
48000/48000 [==============================] - 68s 1ms/step - loss: 0.1816 -
acc: 0.9336 - val_loss: 0.2495 - val_acc: 0.9142
Epoch 46/50
48000/48000 [==============================] - 69s 1ms/step - loss: 0.1805 -
acc: 0.9345 - val_loss: 0.2572 - val_acc: 0.9112
Epoch 47/50
48000/48000 [==============================] - 68s 1ms/step - loss: 0.1785 -
acc: 0.9342 - val_loss: 0.2694 - val_acc: 0.9071
Epoch 48/50
48000/48000 [==============================] - 68s 1ms/step - loss: 0.1763 -
acc: 0.9366 - val_loss: 0.2526 - val_acc: 0.9126
Epoch 49/50
48000/48000 [==============================] - 69s 1ms/step - loss: 0.1713 -
acc: 0.9372 - val_loss: 0.2559 - val_acc: 0.9117
Epoch 50/50
48000/48000 [==============================] - 68s 1ms/step - loss: 0.1735 -
acc: 0.9363 - val_loss: 0.2533 - val_acc: 0.9127
```

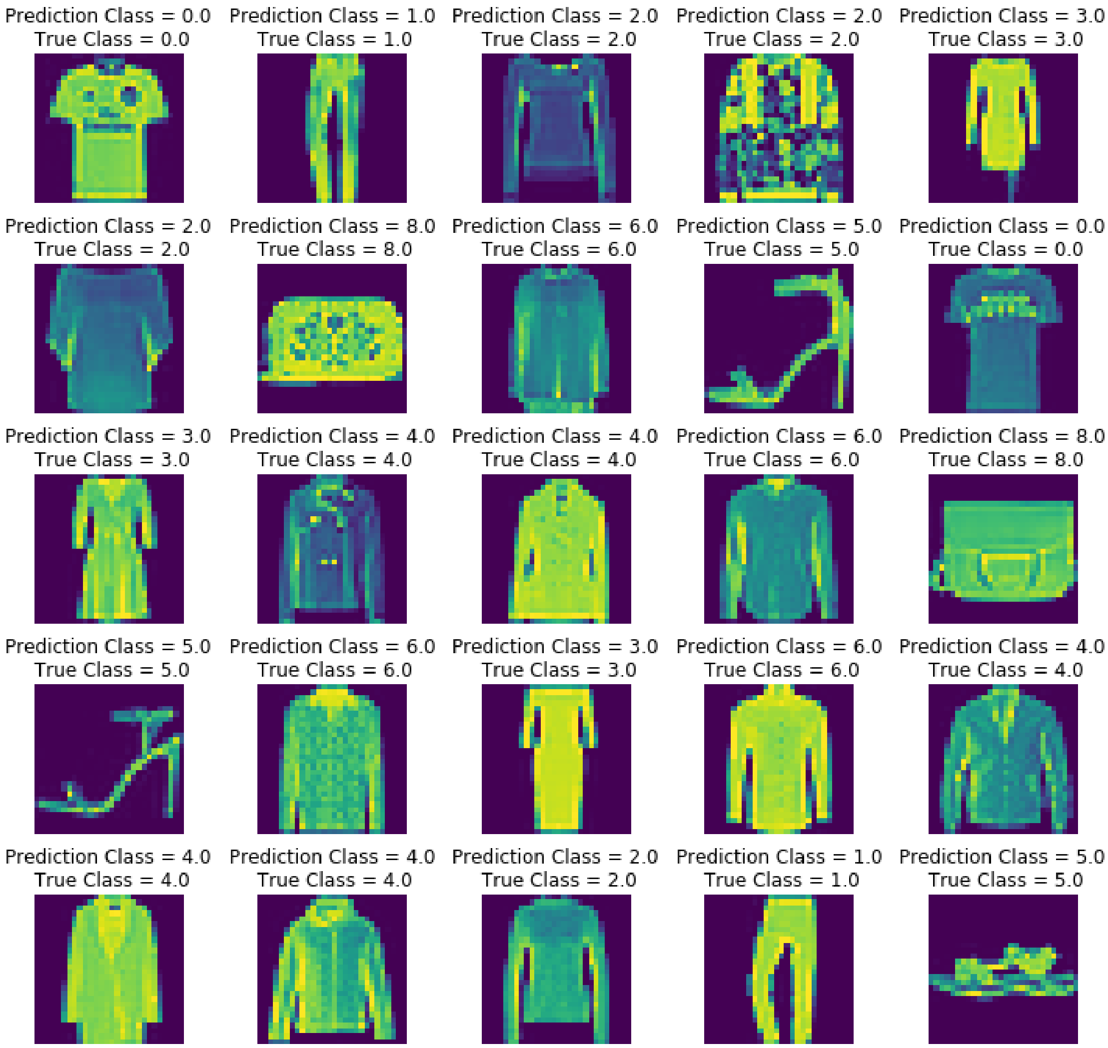
8 STEP #5: EVALUATING THE MODEL

[28]:
```python
evaluation = cnn_model.evaluate(X_test, y_test)
print('Test Accuracy : {:.3f}'.format(evaluation[1]))
```

```
10000/10000 [==============================] - 4s 360us/step
Test Accuracy : 0.913
```

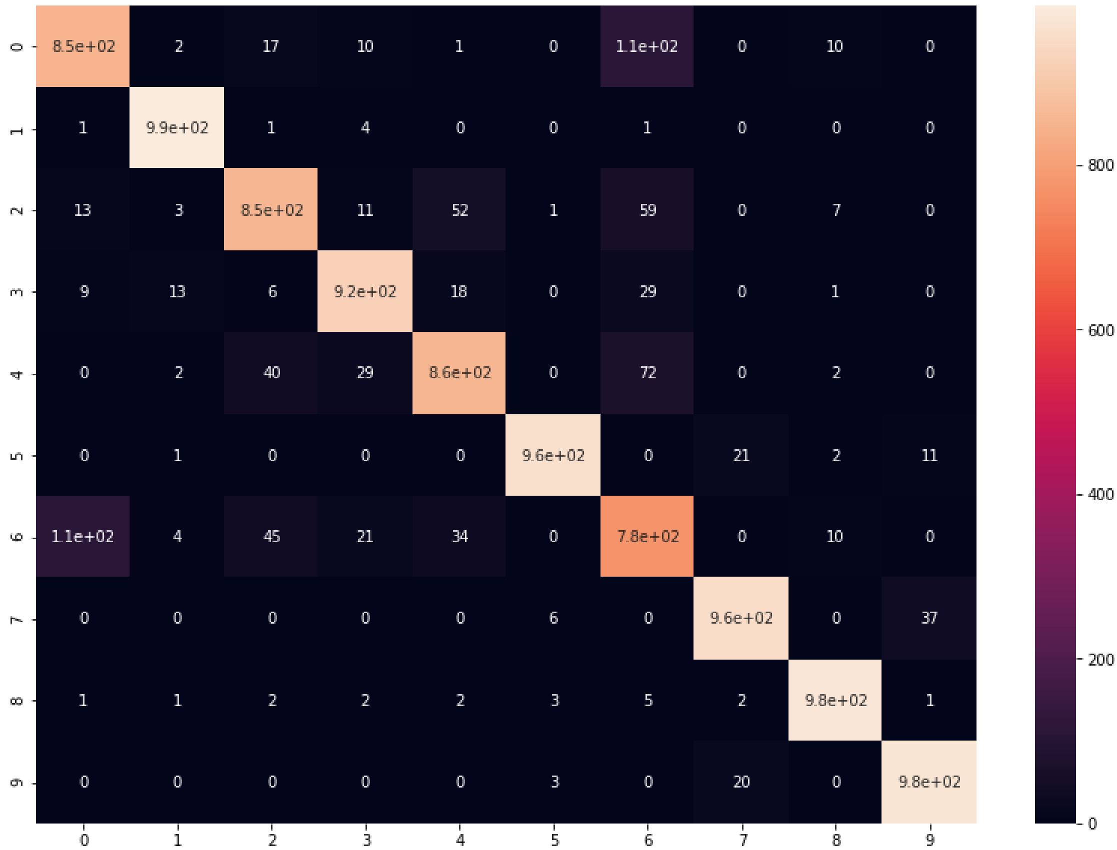
[29]:
```python
# get the predictions for the test data
predicted_classes = cnn_model.predict_classes(X_test)
```

[30]:
```python
L = 5
W = 5
fig, axes = plt.subplots(L, W, figsize = (12,12))
axes = axes.ravel() #

for i in np.arange(0, L * W):
    axes[i].imshow(X_test[i].reshape(28,28))
    axes[i].set_title("Prediction Class = {:0.1f}\n True Class = {:0.1f}".
 ↪format(predicted_classes[i], y_test[i]))
    axes[i].axis('off')

plt.subplots_adjust(wspace=0.5)
```



[31]:
```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, predicted_classes)
plt.figure(figsize = (14,10))
sns.heatmap(cm, annot=True)
# Sum the diagonal element to get the total true correct values
```

[31]: <matplotlib.axes._subplots.AxesSubplot at 0x4e4ddeb128>



[32]:
```
from sklearn.metrics import classification_report

num_classes = 10
target_names = ["Class {}".format(i) for i in range(num_classes)]

print(classification_report(y_test, predicted_classes, target_names =
 ↪target_names))
```

```
              precision    recall  f1-score   support

     Class 0       0.86      0.85      0.86      1000
     Class 1       0.97      0.99      0.98      1000
     Class 2       0.88      0.85      0.87      1000
     Class 3       0.92      0.92      0.92      1000
     Class 4       0.89      0.85      0.87      1000
     Class 5       0.99      0.96      0.98      1000
     Class 6       0.74      0.78      0.76      1000
     Class 7       0.96      0.96      0.96      1000
     Class 8       0.97      0.98      0.97      1000
     Class 9       0.95      0.98      0.96      1000
```

```
   micro avg       0.91      0.91      0.91     10000
   macro avg       0.91      0.91      0.91     10000
weighted avg       0.91      0.91      0.91     10000
```

9 STEP #6: IMPROVING THE MODEL

9.1 Convolutional Neural Networks - Increase Filters/Dropout

- Improve accuracy by adding more feature detectors/fillters or adding a dropout.
- Dropout refers to dropping out units in a neural network.
- Neurons develop co-dependency amongst each other during training.
- Dropout is a regularization technique for reducing over-fitting in neural networks.
- It enables training to occur on several architectures of the neural network.

10 Dropout :

- Dropout is a regularization technique which prevents over-fitting of the network. As the name suggests, during training a certain number of neurons in the hidden layer is randomly dropped. This means that the training happens on several architectures of the neural network on different combinations of the neurons. You can think of drop out as an ensemble technique, where the output of multiple networks is then used to produce the final output.

https://www.analyticsvidhya.com/blog/2017/05/25-must-know-terms-concepts-for-beginners-in-deep-learning/dropout-2/

11 Conclusion

- Advanced technique using more rich dataset can be used to analyses the color, texture and style besides the categorical classification.