FASHION CLASS CLASSIFICATION

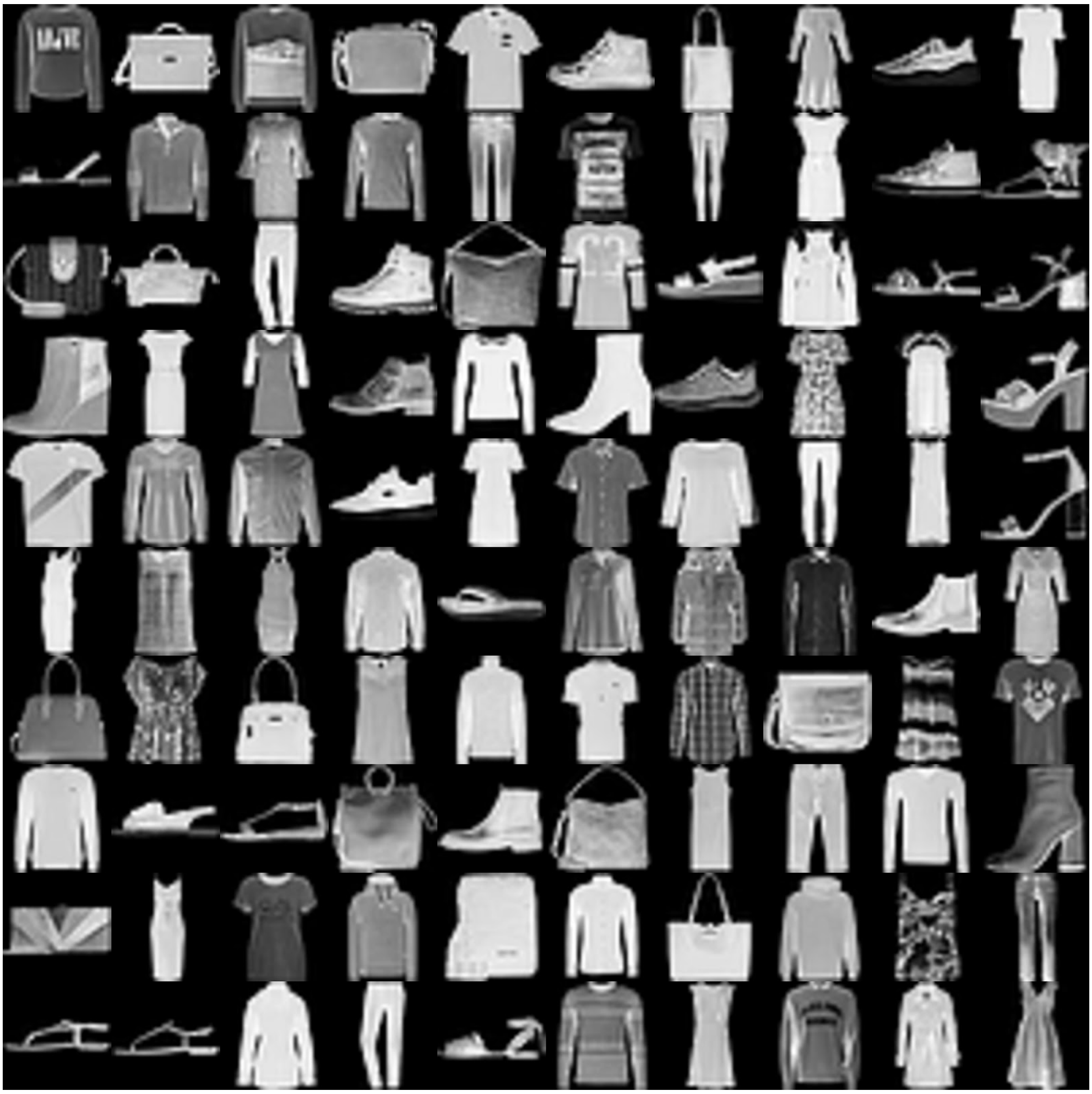
Meet Shah

1. PROBLEM STATEMENT AND BUSINESS CASE UNDERSTANDING

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.



2. Importing Essential Libraries & Our Fashion Mnist Data

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
#import plotly.express as px
import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
train_df = pd.read_csv('fashion-mnist_train.csv', sep = ',')
test_df = pd.read_csv('fashion-mnist_test.csv', sep = ',')
```

In [3]:
```
train_df.head(10)
```

Out[3]:

	label	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	...	pixel775	pixel77
0	2	0	0	0	0	0	0	0	0	0	...	0	
1	9	0	0	0	0	0	0	0	0	0	...	0	
2	6	0	0	0	0	0	0	0	5	0	...	0	

	label	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	...	pixel775	pixel77
3	0	0	0	0	1	2	0	0	0	0	...	3	
4	3	0	0	0	0	0	0	0	0	0	...	0	
5	4	0	0	0	5	4	5	5	3	5	...	7	
6	4	0	0	0	0	0	0	0	0	0	...	14	
7	5	0	0	0	0	0	0	0	0	0	...	0	
8	4	0	0	0	0	0	0	3	2	0	...	1	
9	8	0	0	0	0	0	0	0	0	0	...	203	21

10 rows × 785 columns

In [4]:
```
test_df.head(10)
```

Out[4]:

	label	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	...	pixel775	pixel77
0	0	0	0	0	0	0	0	0	9	8	...	103	8
1	1	0	0	0	0	0	0	0	0	0	...	34	
2	2	0	0	0	0	0	0	14	53	99	...	0	
3	2	0	0	0	0	0	0	0	0	0	...	137	12
4	3	0	0	0	0	0	0	0	0	0	...	0	
5	2	0	0	0	0	0	44	105	44	10	...	105	6
6	8	0	0	0	0	0	0	0	0	0	...	0	
7	6	0	0	0	0	0	0	0	1	0	...	174	13
8	5	0	0	0	0	0	0	0	0	0	...	0	
9	0	0	0	0	0	0	0	0	0	0	...	57	7

10 rows × 785 columns

In [5]:
```
train_df.shape
```

Out[5]:
```
(60000, 785)
```

In [6]:
```
test_df.shape
```

Out[6]:
```
(10000, 785)
```

3. Data Visualization

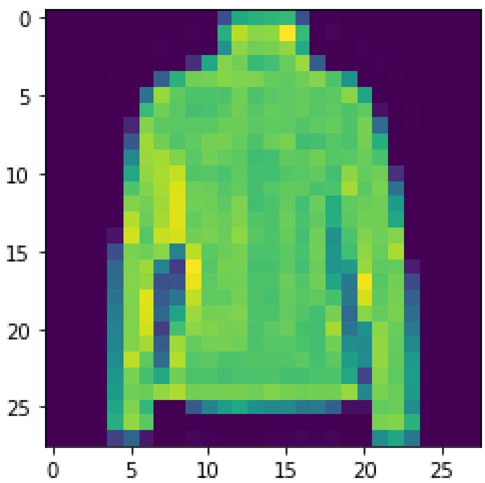
Converting our data from 'dataframe' to an 'array' so that we can visualize our data.

In [7]:
```
training = np.array(train_df, dtype = 'float32')
```

In [8]:
```
test = np.array(test_df, dtype = 'float32')
```

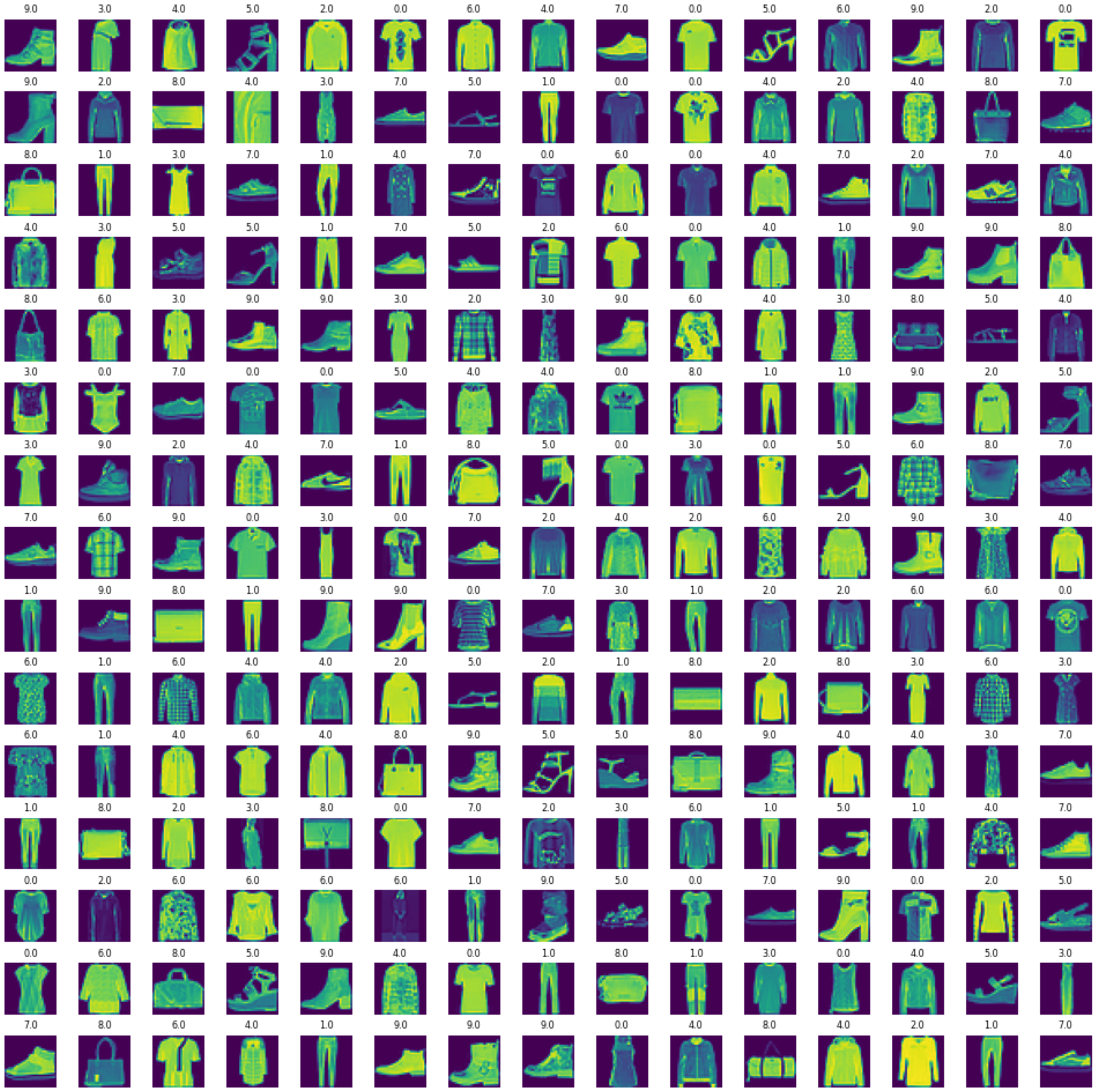
In [9]:
```python
plt.imshow(training[55, 1:].reshape(28,28))
#plt.savefig('Example Image.png')
```

Out[9]: <matplotlib.image.AxesImage at 0x1b2b7395f40>



Moreover, we can use a for loop and create a grid of random images.

In [10]:
```python
W_grid = 15
L_grid = 15
fig, axes = plt.subplots(L_grid, W_grid, figsize = (17,17))
axes = axes.ravel()
n_training = len(training)
for i in np.arange(0, W_grid * L_grid):
    index = np.random.randint(0, n_training)
    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)
#fig.savefig('Grid Image.png')
```



Model Building & Training

In [11]:
```
# Splitting the Training Set
X_train = training[:, 1:]/255
y_train = training[:, 0]

# Splitting the Test Set
X_test = test[:, 1:]/255
y_test = test[:, 0]
```

In [12]:
```
from sklearn.model_selection import train_test_split
X_train, X_validate, y_train, y_validate = train_test_split(X_train, y_train, test_s
```

In [13]:
```
X_train.shape
```

Out[13]: (48000, 784)

In [14]:
```
y_train.shape
```

Out[14]: (48000,)

In [15]:
```
# Remember that * unpacks 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))
```

In [16]:
```
X_train.shape
```

Out[16]: (48000, 28, 28, 1)

In [17]:
```
X_test.shape
```

Out[17]: (10000, 28, 28, 1)

In [18]:
```
X_validate.shape
```

Out[18]: (12000, 28, 28, 1)

In [19]:
```
import tensorflow
```

In [20]:
```
tensorflow.__version__
```

Out[20]: '2.3.1'

In [21]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dense, Flatten, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import TensorBoard
```

In [22]:
```
cnn = Sequential()
```

In [23]:
```
# Was first run with filter = 32 and then later changed it to 64 for better accuracy
cnn.add(Conv2D(filters = 64, kernel_size = 3, input_shape = [28, 28, 1], activation
```

In [24]:
```
cnn.add(MaxPooling2D(pool_size = (2, 2)))
```

In [25]:
```
cnn.add(Flatten())
```

In [26]:
```
cnn.add(Dense(units = 32, activation = 'relu'))
```

In [27]:
```
cnn.add(Dense(units = 10, activation = 'sigmoid'))
```

In [ ]:
```
#cnn.add(Dropout(0.25))
```

In [28]:
```
cnn.compile(loss = 'sparse_categorical_crossentropy', optimizer = Adam(lr=0.001), me
```

In [29]:
```
cnn.fit(X_train, y_train, epochs = 50, batch_size = 512, verbose = 1, validation_dat
```

```
Epoch 1/50
94/94 [==============================] - 41s 431ms/step - loss: 0.8849 - accuracy:
0.6748 - val_loss: 0.4992 - val_accuracy: 0.8219
Epoch 2/50
94/94 [==============================] - 34s 364ms/step - loss: 0.4484 - accuracy:
0.8426 - val_loss: 0.4152 - val_accuracy: 0.8547
Epoch 3/50
94/94 [==============================] - 34s 361ms/step - loss: 0.3906 - accuracy:
0.8638 - val_loss: 0.3742 - val_accuracy: 0.8714
Epoch 4/50
94/94 [==============================] - 35s 376ms/step - loss: 0.3556 - accuracy:
0.8764 - val_loss: 0.3692 - val_accuracy: 0.8717
Epoch 5/50
```

```
94/94 [==============================] - 34s 361ms/step - loss: 0.3331 - accuracy:
0.8849 - val_loss: 0.3280 - val_accuracy: 0.8832
Epoch 6/50
94/94 [==============================] - 34s 363ms/step - loss: 0.3139 - accuracy:
0.8901 - val_loss: 0.3213 - val_accuracy: 0.8885
Epoch 7/50
94/94 [==============================] - 35s 371ms/step - loss: 0.3022 - accuracy:
0.8943 - val_loss: 0.3119 - val_accuracy: 0.8907
Epoch 8/50
94/94 [==============================] - 34s 358ms/step - loss: 0.2880 - accuracy:
0.8996 - val_loss: 0.3050 - val_accuracy: 0.8939
Epoch 9/50
94/94 [==============================] - 34s 366ms/step - loss: 0.2782 - accuracy:
0.9022 - val_loss: 0.2902 - val_accuracy: 0.8979
Epoch 10/50
94/94 [==============================] - 34s 363ms/step - loss: 0.2670 - accuracy:
0.9062 - val_loss: 0.2968 - val_accuracy: 0.8951
Epoch 11/50
94/94 [==============================] - 34s 364ms/step - loss: 0.2608 - accuracy:
0.9079 - val_loss: 0.2808 - val_accuracy: 0.9022
Epoch 12/50
94/94 [==============================] - 34s 364ms/step - loss: 0.2502 - accuracy:
0.9119 - val_loss: 0.2795 - val_accuracy: 0.9021
Epoch 13/50
94/94 [==============================] - 33s 348ms/step - loss: 0.2441 - accuracy:
0.9138 - val_loss: 0.2921 - val_accuracy: 0.8962
Epoch 14/50
94/94 [==============================] - 33s 356ms/step - loss: 0.2398 - accuracy:
0.9141 - val_loss: 0.2721 - val_accuracy: 0.9038
Epoch 15/50
94/94 [==============================] - 34s 356ms/step - loss: 0.2289 - accuracy:
0.9197 - val_loss: 0.2735 - val_accuracy: 0.9023
Epoch 16/50
94/94 [==============================] - 36s 385ms/step - loss: 0.2252 - accuracy:
0.9210 - val_loss: 0.2593 - val_accuracy: 0.9084
Epoch 17/50
94/94 [==============================] - 34s 362ms/step - loss: 0.2164 - accuracy:
0.9243 - val_loss: 0.2598 - val_accuracy: 0.9078
Epoch 18/50
94/94 [==============================] - 33s 356ms/step - loss: 0.2134 - accuracy:
0.9249 - val_loss: 0.2567 - val_accuracy: 0.9087
Epoch 19/50
94/94 [==============================] - 34s 357ms/step - loss: 0.2111 - accuracy:
0.9255 - val_loss: 0.2551 - val_accuracy: 0.9090
Epoch 20/50
94/94 [==============================] - 33s 356ms/step - loss: 0.2013 - accuracy:
0.9291 - val_loss: 0.2563 - val_accuracy: 0.9078
Epoch 21/50
94/94 [==============================] - 33s 355ms/step - loss: 0.2027 - accuracy:
0.9281 - val_loss: 0.2604 - val_accuracy: 0.9081
Epoch 22/50
94/94 [==============================] - 33s 355ms/step - loss: 0.1920 - accuracy:
0.9321 - val_loss: 0.2521 - val_accuracy: 0.9093
Epoch 23/50
94/94 [==============================] - 34s 365ms/step - loss: 0.1875 - accuracy:
0.9339 - val_loss: 0.2531 - val_accuracy: 0.9084
Epoch 24/50
94/94 [==============================] - 33s 356ms/step - loss: 0.1832 - accuracy:
0.9364 - val_loss: 0.2563 - val_accuracy: 0.9094
Epoch 25/50
94/94 [==============================] - 33s 356ms/step - loss: 0.1789 - accuracy:
0.9383 - val_loss: 0.2498 - val_accuracy: 0.9115
Epoch 26/50
94/94 [==============================] - 33s 355ms/step - loss: 0.1757 - accuracy:
0.9381 - val_loss: 0.2470 - val_accuracy: 0.9120
Epoch 27/50
94/94 [==============================] - 33s 355ms/step - loss: 0.1675 - accuracy:
0.9421 - val_loss: 0.2486 - val_accuracy: 0.9128
Epoch 28/50
```

```
94/94 [==============================] - 33s 356ms/step - loss: 0.1674 - accuracy: 0.9408 - val_loss: 0.2552 - val_accuracy: 0.9125
Epoch 29/50
94/94 [==============================] - 33s 355ms/step - loss: 0.1638 - accuracy: 0.9422 - val_loss: 0.2497 - val_accuracy: 0.9114
Epoch 30/50
94/94 [==============================] - 33s 356ms/step - loss: 0.1580 - accuracy: 0.9440 - val_loss: 0.2635 - val_accuracy: 0.9093
Epoch 31/50
94/94 [==============================] - 33s 355ms/step - loss: 0.1543 - accuracy: 0.9458 - val_loss: 0.2558 - val_accuracy: 0.9081
Epoch 32/50
94/94 [==============================] - 34s 365ms/step - loss: 0.1506 - accuracy: 0.9477 - val_loss: 0.2515 - val_accuracy: 0.9103
Epoch 33/50
94/94 [==============================] - 33s 349ms/step - loss: 0.1479 - accuracy: 0.9483 - val_loss: 0.2482 - val_accuracy: 0.9139
Epoch 34/50
94/94 [==============================] - 33s 354ms/step - loss: 0.1466 - accuracy: 0.9490 - val_loss: 0.2708 - val_accuracy: 0.9063
Epoch 35/50
94/94 [==============================] - 33s 355ms/step - loss: 0.1404 - accuracy: 0.9508 - val_loss: 0.2637 - val_accuracy: 0.9089
Epoch 36/50
94/94 [==============================] - 33s 352ms/step - loss: 0.1365 - accuracy: 0.9532 - val_loss: 0.2634 - val_accuracy: 0.9093
Epoch 37/50
94/94 [==============================] - 33s 347ms/step - loss: 0.1343 - accuracy: 0.9532 - val_loss: 0.2572 - val_accuracy: 0.9103
Epoch 38/50
94/94 [==============================] - 34s 364ms/step - loss: 0.1294 - accuracy: 0.9551 - val_loss: 0.2561 - val_accuracy: 0.9128
Epoch 39/50
94/94 [==============================] - 33s 355ms/step - loss: 0.1292 - accuracy: 0.9543 - val_loss: 0.2584 - val_accuracy: 0.9133
Epoch 40/50
94/94 [==============================] - 34s 358ms/step - loss: 0.1240 - accuracy: 0.9568 - val_loss: 0.2574 - val_accuracy: 0.9138
Epoch 41/50
94/94 [==============================] - 34s 359ms/step - loss: 0.1203 - accuracy: 0.9590 - val_loss: 0.2619 - val_accuracy: 0.9113
Epoch 42/50
94/94 [==============================] - 34s 362ms/step - loss: 0.1166 - accuracy: 0.9602 - val_loss: 0.2631 - val_accuracy: 0.9118
Epoch 43/50
94/94 [==============================] - 34s 360ms/step - loss: 0.1146 - accuracy: 0.9604 - val_loss: 0.2787 - val_accuracy: 0.9084
Epoch 44/50
94/94 [==============================] - 33s 352ms/step - loss: 0.1123 - accuracy: 0.9617 - val_loss: 0.2659 - val_accuracy: 0.9139
Epoch 45/50
94/94 [==============================] - 36s 381ms/step - loss: 0.1059 - accuracy: 0.9648 - val_loss: 0.2669 - val_accuracy: 0.9127
Epoch 46/50
94/94 [==============================] - 34s 362ms/step - loss: 0.1077 - accuracy: 0.9640 - val_loss: 0.2693 - val_accuracy: 0.9121
Epoch 47/50
94/94 [==============================] - 33s 351ms/step - loss: 0.1034 - accuracy: 0.9648 - val_loss: 0.2665 - val_accuracy: 0.9135
Epoch 48/50
94/94 [==============================] - 33s 356ms/step - loss: 0.1005 - accuracy: 0.9659 - val_loss: 0.3021 - val_accuracy: 0.9014
Epoch 49/50
94/94 [==============================] - 33s 352ms/step - loss: 0.1001 - accuracy: 0.9656 - val_loss: 0.2731 - val_accuracy: 0.9146
Epoch 50/50
94/94 [==============================] - 33s 352ms/step - loss: 0.0950 - accuracy: 0.9679 - val_loss: 0.2716 - val_accuracy: 0.9162
```

Out[29]:
```
<tensorflow.python.keras.callbacks.History at 0x1b2f6313ee0>
```

In [30]:
```
evaluation = cnn.evaluate(X_test, y_test)
print('Test Accuracy : {:.3f}'.format(evaluation[1]))
```

```
313/313 [==============================] - 3s 8ms/step - loss: 0.2745 - accuracy: 0.
9172
Test Accuracy : 0.917
```

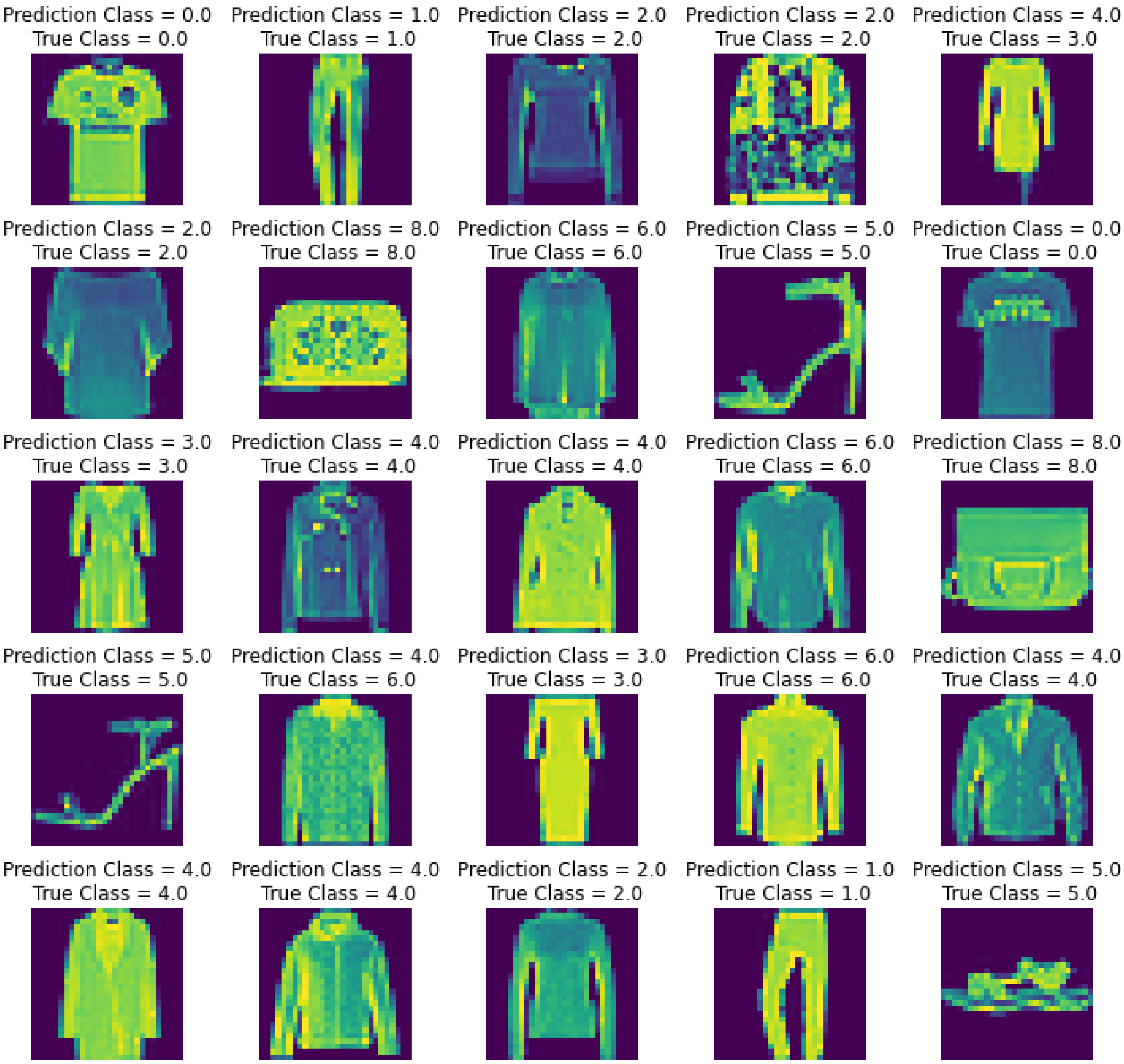
In [31]:
```
predicted_classes = cnn.predict_classes(X_test)
```

```
WARNING:tensorflow:From <ipython-input-31-ef5bdc73df9d>:1: Sequential.predict_classe
s (from tensorflow.python.keras.engine.sequential) is deprecated and will be removed
after 2021-01-01.
Instructions for updating:
Please use instead:* `np.argmax(model.predict(x), axis=-1)`,   if your model does mu
lti-class classification   (e.g. if it uses a `softmax` last-layer activation).* `(m
odel.predict(x) > 0.5).astype("int32")`,   if your model does binary classification
(e.g. if it uses a `sigmoid` last-layer activation).
```

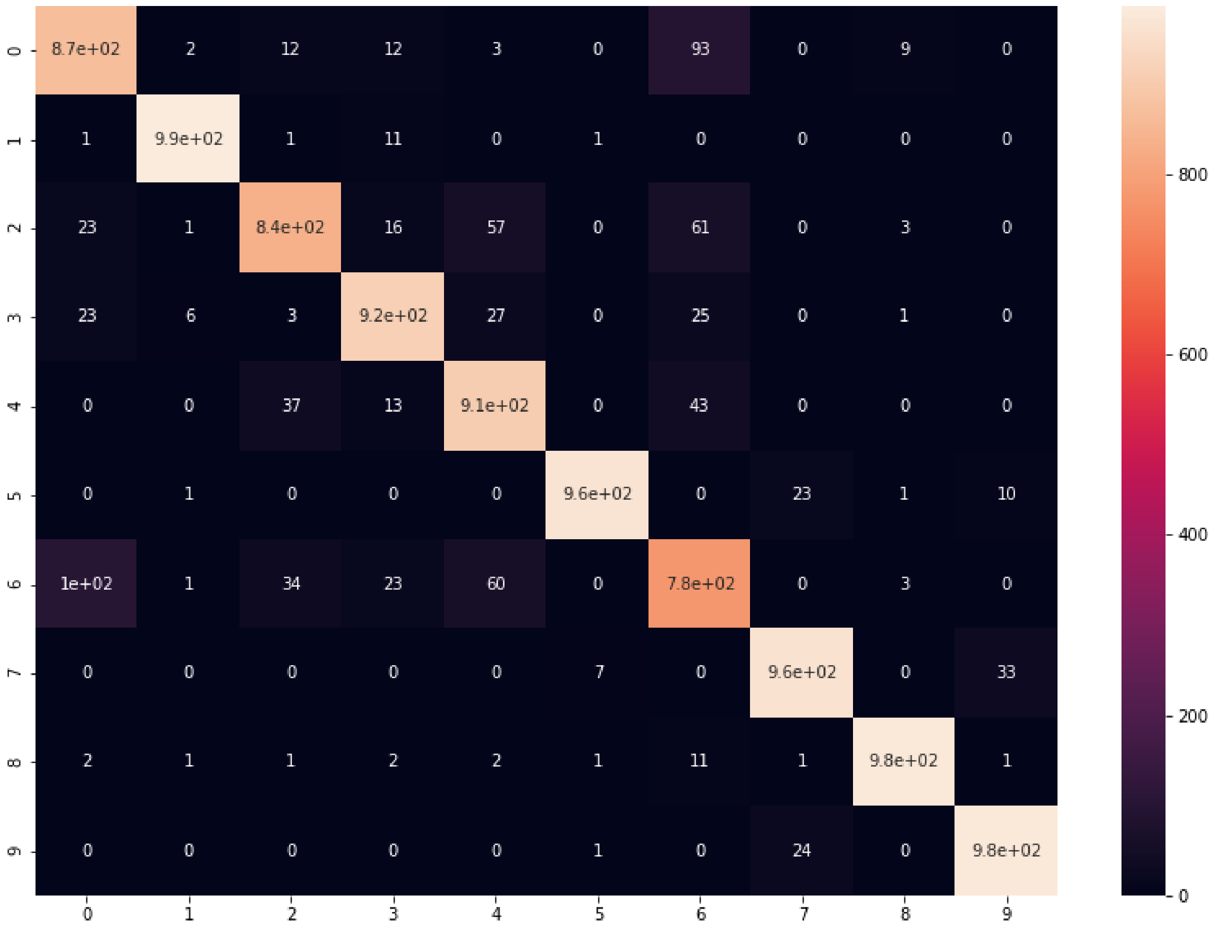
In [32]:
```
predicted_classes
```

Out[32]:
```
array([0, 1, 2, ..., 8, 8, 1], dtype=int64)
```

In [33]:
```
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(pre
    axes[i].axis('off')
plt.subplots_adjust(wspace = 0.5)
#fig.savefig('Grid Image Showing Predicted Class vs True Class.png')
```



In [34]:
```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, predicted_classes)
plt.figure(figsize = (14, 10))
sns.heatmap(cm, annot = True)
#plt.savefig('Heatmap Showing Predicted class vs True Class.png')
```



In [35]:
```python
from sklearn.metrics import classification_report


target_names = ["Class {}".format(i) for i in range(1, 11)]
print(classification_report(y_test, predicted_classes, target_names = target_names))
```

```
              precision    recall  f1-score   support

     Class 1       0.85      0.87      0.86      1000
     Class 2       0.99      0.99      0.99      1000
     Class 3       0.91      0.84      0.87      1000
     Class 4       0.92      0.92      0.92      1000
     Class 5       0.86      0.91      0.88      1000
     Class 6       0.99      0.96      0.98      1000
     Class 7       0.77      0.78      0.77      1000
     Class 8       0.95      0.96      0.96      1000
     Class 9       0.98      0.98      0.98      1000
    Class 10       0.96      0.97      0.97      1000

    accuracy                           0.92     10000
   macro avg       0.92      0.92      0.92     10000
weighted avg       0.92      0.92      0.92     10000
```

Conclusion

As seen in output no. 29 & 30, our accuracies are as follows:

1. Training Data Accuracy - 96.79%
2. Validation Data Accuracy - 91.62%
3. Testing Data Accuracy - 91.7%

So when we print out our classification report we finally get an average accuracy of 92%.