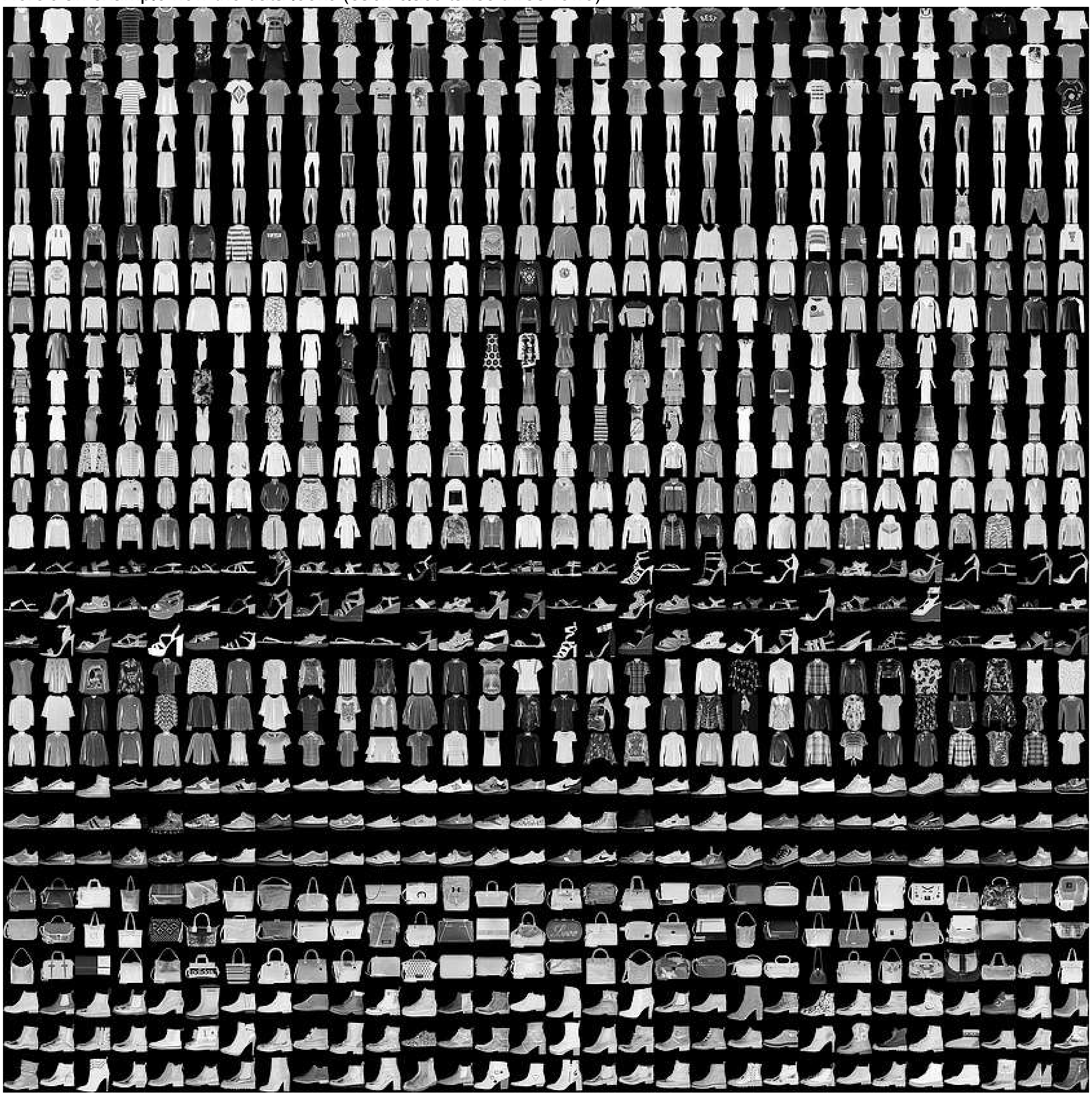
Fashion MNIST Classification

Fashion-MNIST is a dataset of Zalando's article images—consisting of a **training set of 60,000** examples and a **test set of 10,000 examples.** Each example is a **28x28 grayscale** image, associated with a label from **10 classes.** We intend Fashion-MNIST to serve as a direct drop-in **replacement for the original MNIST** dataset for benchmarking machine learning algorithms. It shares the same image size and structure of training and testing splits.

Here's an example how the data looks (each class takes three-rows):



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 [1]:
```
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 [2]:
```
fashion_train_df = pd.read_csv('fashion-mnist_train.csv', sep=',')
fashion_test_df = pd.read_csv('fashion-mnist_test.csv', sep=',')
```

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

In [3]:
```
fashion_train_df.shape   # Shape of the dataset
```

Out[3]:
```
(60000, 785)
```

In [4]:
```
fashion_train_df.columns   # Name of the columns of the DataSet.
```

Out[4]:
```
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 [5]:
```
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 [6]:
```
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 [7]:
```
fashion_train_df.head()
```

Out[7]:

	label	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	...	pixel775	pixel776	pixel777	pixel778	pixel77
0	2	0	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	0	
2	6	0	0	0	0	0	0	0	5	0	...	0	0	0	30	4
3	0	0	0	0	1	2	0	0	0	0	...	3	0	0	0	
4	3	0	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 [8]:
```
fashion_test_df.shape
```

Out[8]:
```
(10000, 785)
```

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

Lets check first few rows.

In [9]:
```
fashion_test_df.head()
```

Out[9]:

	label	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	...	pixel775	pixel776	pixel777	pixel778	pixel77
0	0	0	0	0	0	0	0	0	9	8	...	103	87	56	0	
1	1	0	0	0	0	0	0	0	0	0	...	34	0	0	0	
2	2	0	0	0	0	0	0	14	53	99	...	0	0	0	0	6
3	2	0	0	0	0	0	0	0	0	0	...	137	126	140	0	13
4	3	0	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.

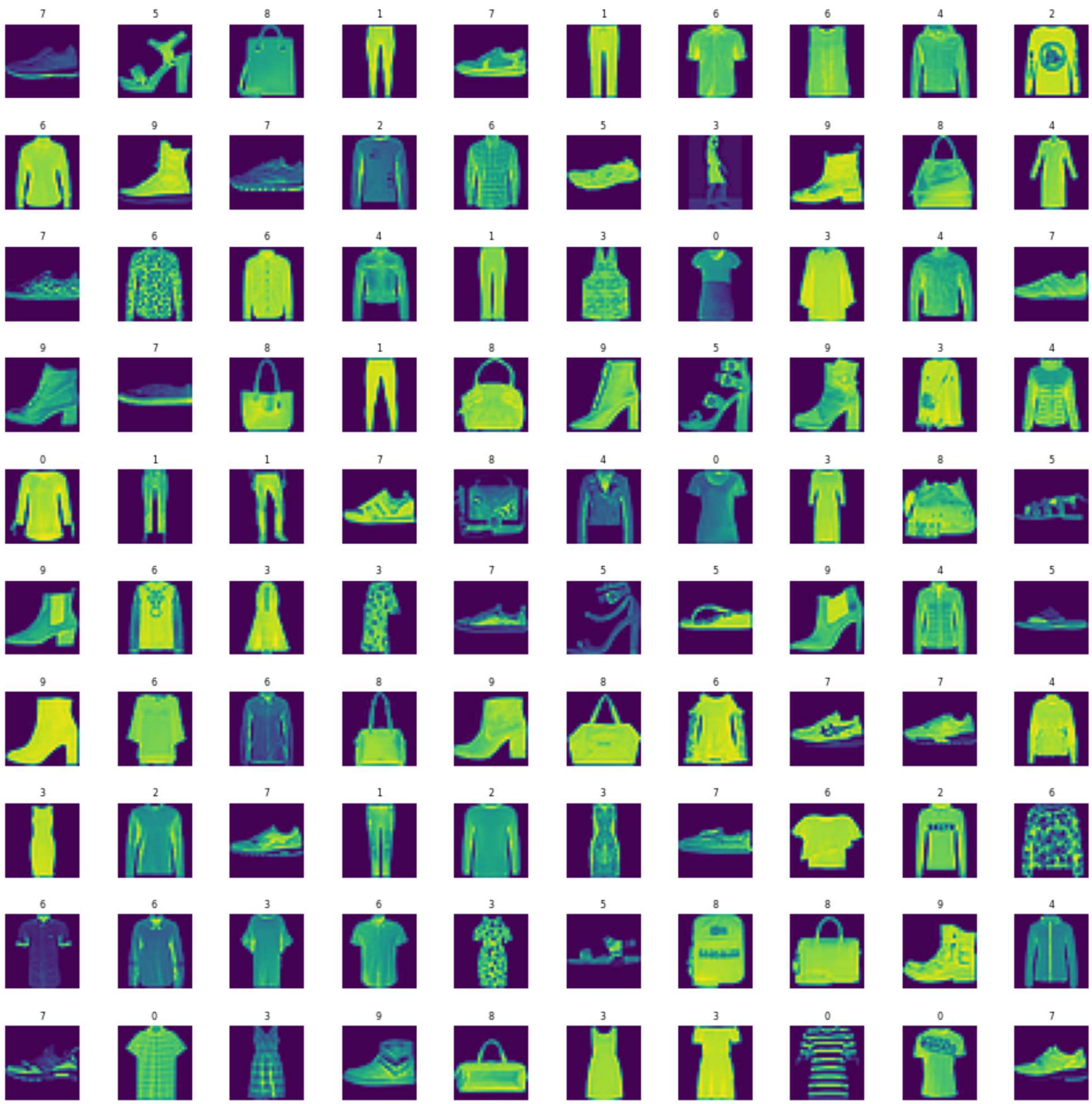
In [10]:
```python
# Convert the dataframe ti numpy array
training = np.asarray(fashion_train_df, dtype='float32')

# 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.

Since we are dealing with image data and our task is to recognize and classify images our model should be a Convolutional Neural Network. For that our images should have atleast 3 dimensions (**height** x **width** x **color_channels**). But our images are flattened in one dimension, **784 pixel (28x28x1)** values per row. So we need to reshape the data into its original format.

In [11]:
```
# 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 [13]:
```
he training set into training and validation sets
X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.2, random_state=5)    # TODO :
```

In [14]:
```
# 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)
```

```
(38400, 28, 28, 1) (9600, 28, 28, 1) (10000, 28, 28, 1)
(38400,) (9600,) (10000,)
```

Step # 5 - Create and Train the Model

Create the model

In [15]:
```
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 [16]:
```
cnn_model.compile(optimizer=Adam(lr=0.001), loss='sparse_categorical_crossentropy', metrics=['ac
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\admin\anaconda3\lib\site-packages\keras\optimizers\legacy\adam.py:117: UserWarning: The `lr
` argument is deprecated, use `learning_rate` instead.
  super().__init__(name, **kwargs)
```

Train the model

In [17]:
```
cnn_model.fit(x=X_train, y=y_train, batch_size=512, epochs=50, validation_data=(X_val, y_val))
```

```
oss: 0.2723 - val_accuracy: 0.9139
Epoch 45/50
75/75 [==============================] - 20s 264ms/step - loss: 0.1146 - accuracy: 0.9589 - val_l
oss: 0.2765 - val_accuracy: 0.9145
Epoch 46/50
75/75 [==============================] - 20s 272ms/step - loss: 0.1146 - accuracy: 0.9591 - val_l
oss: 0.2777 - val_accuracy: 0.9132
Epoch 47/50
75/75 [==============================] - 20s 273ms/step - loss: 0.1159 - accuracy: 0.9570 - val_l
oss: 0.2761 - val_accuracy: 0.9128
Epoch 48/50
75/75 [==============================] - 21s 278ms/step - loss: 0.1110 - accuracy: 0.9598 - val_l
oss: 0.2960 - val_accuracy: 0.9106
Epoch 49/50
75/75 [==============================] - 22s 290ms/step - loss: 0.1104 - accuracy: 0.9595 - val_l
oss: 0.2875 - val_accuracy: 0.9137
Epoch 50/50
75/75 [==============================] - 20s 272ms/step - loss: 0.1099 - accuracy: 0.9601 - val_l
oss: 0.2833 - val_accuracy: 0.9151
```

Step # 5 - Evaluate the Model

Get the accuracy of the model

In [30]:
```
eval_result = cnn_model.evaluate(X_test, y_test)
print("Accuracy : {:.3f}".format(eval_result[1]))
```

```
313/313 [==============================] - 2s 6ms/step - loss: 0.2629 - accuracy: 0.9193
Accuracy : 0.919
```

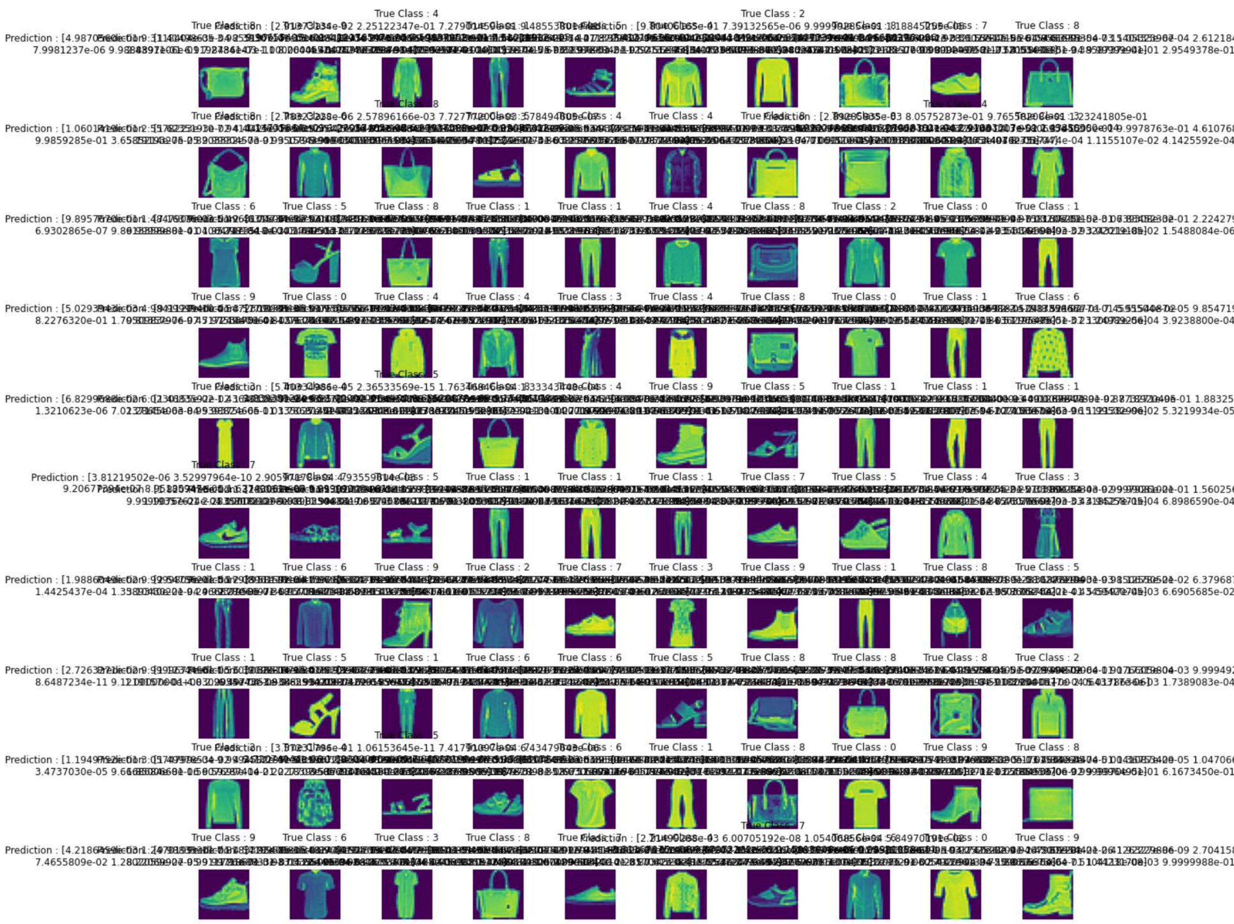
Visualize the model's predictions

In [44]:
```
y_pred = cnn_model.predict(x=X_test)
```

```
313/313 [==============================] - 2s 5ms/step
```

In [20]:
```
height = 10
width = 10

fig, axes = plt.subplots(nrows=width, ncols=height, figsize=(20,20))
axes = axes.ravel()
for i in range(0, height*width):
    index = np.random.randint(len(y_pred))
    axes[i].imshow(X_test[index].reshape((28,28)))
    axes[i].set_title("True Class : {:0.0f}\nPrediction : {:}".format(y_test[index],y_pred[index
    axes[i].axis('off')
plt.subplots_adjust(hspace=0.8, wspace=0.5)
```



Plot Confusin Matrix