Step1:

Business Requirenment

We are trying to build a model which detect the each category of images in fashion world

Objective:

If any women like blue dress so we will target the different blue dresses to her with may be different texture ,different material and so on.

The dataset which we are using that is called fashion MNIST dataset and we are having 10 target class which we will get after classifying the image category like Ankle boot, sneakers etc.

Our Fashion Dataset :

```
1. fashion dataset contains 28*28 gray scale images with values ranging from 0-255
2.'0' represents black adn 1 represents the white
3.Each image is representing by a row and 784 (i.e. 28*28 )values
```

Fashion training set consists 70,000 images divided into 60,000 training and 10,000 testing samples. Datset sample consist of 28*28 grayscale images, associated with a labels with 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=>Sneakers 8=>Bag 9=>Ankle boat

Each image is 28 pixel in height and 28 pixel in width. Each pixel has a single pixel-value associated with it,indictaing the lightness or darkness of that pixel with higher number meaning darkness

Step 2: Importing Data

In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
```

In [2]:

```
train_data = pd.read_csv('fashion-mnist_train.csv')
train_data.head()
```

Out[2]:

	label	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	...	pixel775	pixel776	pixel777	pixel778	pixel779	pixel78
0	2	0	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	0	
2	6	0	0	0	0	0	0	0	5	0	...	0	0	0	30	43	
3	0	0	0	0	1	2	0	0	0	0	...	3	0	0	0	0	
4	3	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	

5 rows × 785 columns

Observation:

Each rows represents the images in pixel.

In [3]:

```
test_data = pd.read_csv('fashion-mnist_test.csv')
test_data.head()
```

Out[3]:

	label	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	...	pixel775	pixel776	pixel777	pixel778	pixel779	pixel78
0	0	0	0	0	0	0	0	0	9	8	...	103	87	56	0	0	
1	1	0	0	0	0	0	0	0	0	0	...	34	0	0	0	0	
2	2	0	0	0	0	0	0	14	53	99	...	0	0	0	0	63	5
3	2	0	0	0	0	0	0	0	0	0	...	137	126	140	0	133	22
4	3	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	

5 rows × 785 columns

Step 3 : Visualization of dataset

In [4]:

```
train_data.tail()
```

Out[4]:

	label	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	...	pixel775	pixel776	pixel777	pixel778	pixel779	pi
59995	9	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	
59996	1	0	0	0	0	0	0	0	0	0	...	73	0	0	0	0	
59997	8	0	0	0	0	0	0	0	0	0	...	160	162	163	135	94	
59998	8	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	
59999	7	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	

5 rows × 785 columns

In [5]:

```
test_data.tail()
```

Out[5]:

	label	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	...	pixel775	pixel776	pixel777	pixel778	pixel779	pix
9995	0	0	0	0	0	0	0	0	0	0	...	32	23	14	20	0	
9996	6	0	0	0	0	0	0	0	0	0	...	0	0	0	2	52	
9997	8	0	0	0	0	0	0	0	0	0	...	175	172	172	182	199	
9998	8	0	1	3	0	0	0	0	0	0	...	0	0	0	0	0	
9999	1	0	0	0	0	0	0	0	140	119	...	111	95	75	44	1	

5 rows × 785 columns

In [6]:

```
train_data.shape
```

Out[6]:

```
(60000, 785)
```

In [7]:

```
test_data.shape
```

Out[7]:

Out[7]:

```
(10000, 785)
```

In [8]:

```
training = np.array(train_data,dtype='float32')
```

In [9]:

```
testing = np.array(test_data,dtype='float32')
```

In [10]:

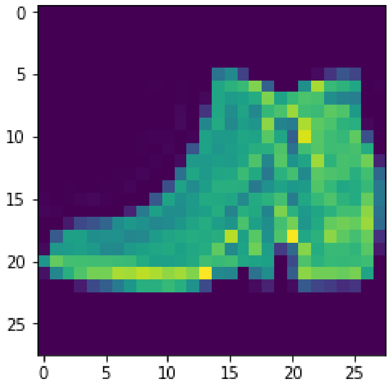
```
import random
i = random.randint(1,60000)
```

In [11]:

```
plt.imshow(training[i,1:].reshape(28,28))
```

Out[11]:

```
&lt;matplotlib.image.AxesImage at 0x1146b4506d8&gt;
```



In [12]:

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

Out[12]:

```
9.0
```

In [13]:

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

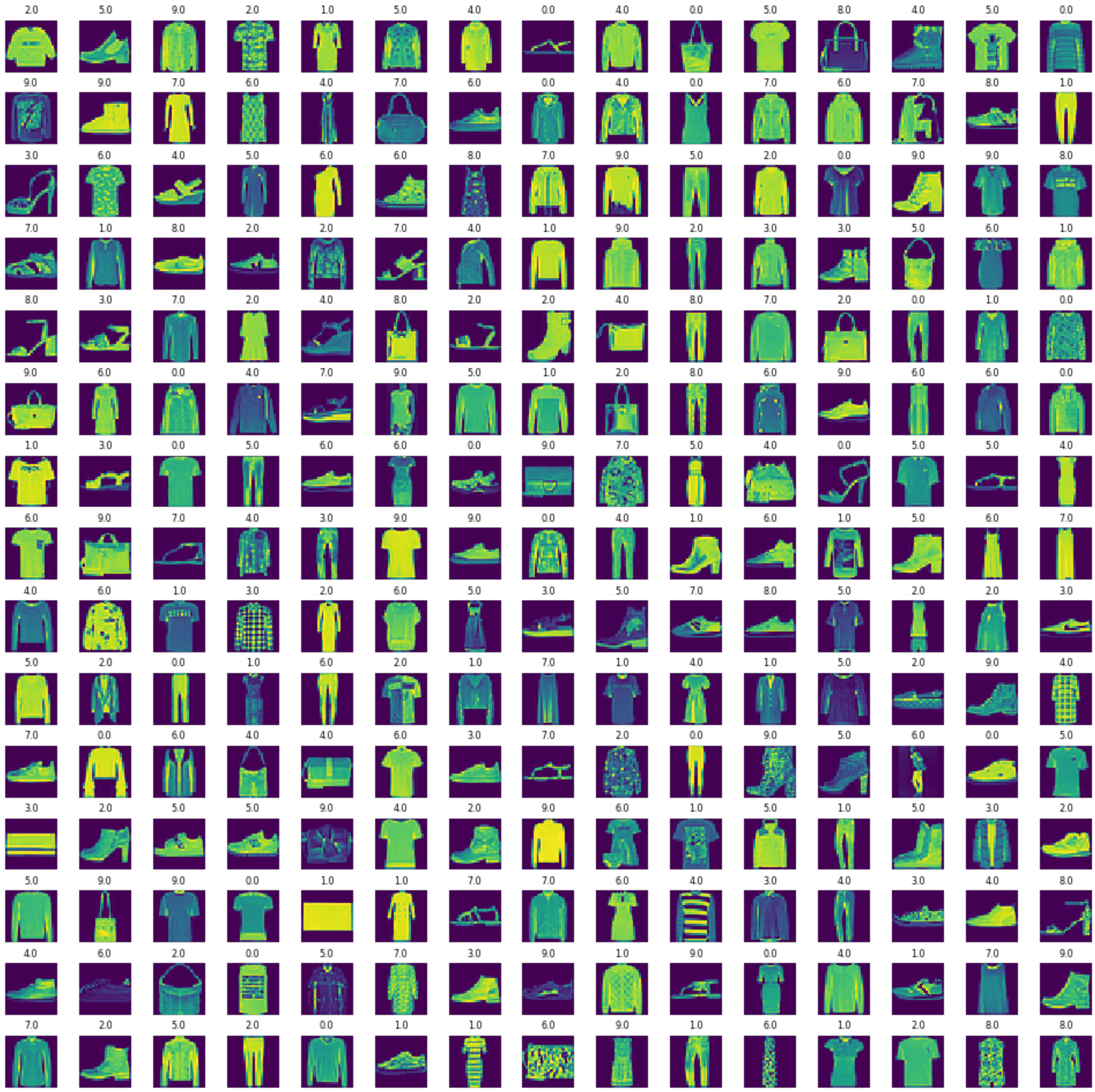
In [14]:

```
# let's view more images in a grid format
#define the dimensions of plot grid
W_grid = 15
L_grid = 15
```

```
fig,axes = plt.subplots(L_grid,W_grid,figsize =(17,17))
axes = axes.ravel() #flatten the 17 * 17 matrix into 255 array
n_training = len(training)#get the length of training dataset
#select  the random number from 0 to n_training

for i in np.arange(0,W_grid*L_grid):
    #create evenly space variable
    #slect a random variable
    index = np.random.randint(0,n_training)
    #read and display an image with selected index
    axes[i].imshow(training[i,1:].reshape(28,28))
    axes[i].set_title(training[index,0],fontsize=8)
    axes[i].axis('off')

plt.subplots_adjust(hspace=0.4)
```



CNN :

When we deal with images we need to preserve or called as special dependent on pixels .So if we take a bag all the pixels are dependent other pixel around it that' why we need to perform kind of another option before actually we feed the pixel directly our network we want to perform called convolution that's where the CNN came into play

Step 4 :Training The model

In [16]:

```
X_train = training[:, 1:]/255 # we are taking all row except one c0lumn that is target and we norm
alize it
Y_train = training[:,0] # actually need column number 0
```

In [17]:

```
X_test = testing[:, 1:]/255 # we are taking all row except one clumn that is target and we
normalize it
Y_test = testing[:,0] # actually need column number 0
```

In [18]:

```
from sklearn.model_selection import train_test_split
```

In [19]:

```
X_train,X_validate,Y_train,Y_validate = train_test_split(X_train,Y_train,test_size = 0.2,
random_state = 5 )
```

In [20]:

```
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 [21]:

```
X_train.shape
```

Out[21]:

```
(48000, 28, 28, 1)
```

We are having 48000 sample and each of them 28 by 28 by 1.Baiscally it is gray scale image 28 by 28

In [22]:

```
X_test.shape
```

Out[22]:

```
(10000, 28, 28, 1)
```

In [23]:

```
X_validate.shape
```

Out[23]:

```
(12000, 28, 28, 1)
```

In [24]:

```
import warnings
warnings.filterwarnings("ignore")
import keras
```

```
Using TensorFlow backend.
```

In [25]:

```
from keras.models import Sequential
```

```
from keras.layers import Conv2D,MaxPooling2D,Dense,Flatten,Dropout
from keras.optimizers import Adam # we import the Adam optimizer
from keras.callbacks import TensorBoard
```

We used the sequential to build the network followed by maxpooling,dropout,flatten and dense(fully connected network) and we used tensorboard for callbacks

In [26]:

```
import warnings
warnings.filterwarnings("ignore")
# To build our model kind of Sequential form
cnn_model = Sequential() # we call the Sequential and then we start building on top of that
cnn_model.add(Conv2D(32,3,3,input_shape = (28,28,1),activation = 'relu')) # going to add Convolutio
n Layer first
#we specified the 32 kernal with size 3 by 3.input shape (28,28,1) it is size about image then spe
cify the relu activation
#function
```

```
WARNING:tensorflow:From C:\Anaconda3\lib\site-packages\keras\backend\tensorflow_backend.py:74: The
name tf.get_default_graph is deprecated. Please use tf.compat.v1.get_default_graph instead.

WARNING:tensorflow:From C:\Anaconda3\lib\site-packages\keras\backend\tensorflow_backend.py:517: Th
e name tf.placeholder is deprecated. Please use tf.compat.v1.placeholder instead.

WARNING:tensorflow:From C:\Anaconda3\lib\site-packages\keras\backend\tensorflow_backend.py:4138: T
he name tf.random_uniform is deprecated. Please use tf.random.uniform instead.
```

In [27]:

```
import warnings
warnings.filterwarnings("ignore")
#Going to specify the maxpooling layer
cnn_model.add(MaxPooling2D(pool_size = (2,2)))
```

```
WARNING:tensorflow:From C:\Anaconda3\lib\site-packages\keras\backend\tensorflow_backend.py:3976: T
he name tf.nn.max_pool is deprecated. Please use tf.nn.max_pool2d instead.
```

In [28]:

```
cnn_model.add(Flatten())
```

In [29]:

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

In [30]:

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

In [31]:

```
cnn_model.compile(loss = 'sparse_categorical_crossentropy',optimizer=Adam(lr=0.001),metrics = ['ac
curacy'])
```

```
WARNING:tensorflow:From C:\Anaconda3\lib\site-packages\keras\optimizers.py:790: The name
tf.train.Optimizer is deprecated. Please use tf.compat.v1.train.Optimizer instead.

WARNING:tensorflow:From C:\Anaconda3\lib\site-packages\keras\backend\tensorflow_backend.py:3341: T
he name tf.log is deprecated. Please use tf.math.log instead.
```

In [32]:

```
epochs = 50
```

In [33]:

```
cnn_model.fit(X_train,Y_train,batch_size=512,nb_epoch = epochs,verbose=1,validation_data=(X_validat
e,Y_validate))
```

```
WARNING:tensorflow:From C:\Anaconda3\lib\site-packages\tensorflow\python\ops\math_grad.py:1250: ad
d_dispatch_support.<locals>.wrapper (from tensorflow.python.ops.array_ops) is deprecated and will
be removed in a future version.
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
WARNING:tensorflow:From C:\Anaconda3\lib\site-packages\keras\backend\tensorflow_backend.py:986: Th
e name tf.assign_add is deprecated. Please use tf.compat.v1.assign_add instead.

Train on 48000 samples, validate on 12000 samples
Epoch 1/50
48000/48000 [==============================] - 18s 376us/step - loss: 1.0315 - acc: 0.5988 - val_l
oss: 0.5359 - val_acc: 0.8105
Epoch 2/50
48000/48000 [==============================] - 17s 363us/step - loss: 0.4704 - acc: 0.8349 - val_l
oss: 0.4427 - val_acc: 0.8472
Epoch 3/50
48000/48000 [==============================] - 17s 362us/step - loss: 0.4123 - acc: 0.8568 - val_l
oss: 0.4135 - val_acc: 0.8522
Epoch 4/50
48000/48000 [==============================] - 17s 362us/step - loss: 0.3813 - acc: 0.8678 - val_l
oss: 0.3894 - val_acc: 0.8653
Epoch 5/50
48000/48000 [==============================] - 17s 360us/step - loss: 0.3607 - acc: 0.8753 - val_l
oss: 0.3656 - val_acc: 0.8729
Epoch 6/50
48000/48000 [==============================] - 17s 361us/step - loss: 0.3503 - acc: 0.8783 - val_l
oss: 0.3623 - val_acc: 0.8724
Epoch 7/50
48000/48000 [==============================] - 17s 363us/step - loss: 0.3336 - acc: 0.8847 - val_l
oss: 0.3545 - val_acc: 0.8763
Epoch 8/50
48000/48000 [==============================] - 17s 362us/step - loss: 0.3214 - acc: 0.8891 - val_l
oss: 0.3408 - val_acc: 0.8821
Epoch 9/50
48000/48000 [==============================] - 17s 362us/step - loss: 0.3114 - acc: 0.8920 - val_l
oss: 0.3356 - val_acc: 0.8810
Epoch 10/50
48000/48000 [==============================] - 17s 362us/step - loss: 0.3052 - acc: 0.8932 - val_l
oss: 0.3321 - val_acc: 0.8837
Epoch 11/50
48000/48000 [==============================] - 17s 364us/step - loss: 0.2953 - acc: 0.8967 - val_l
oss: 0.3277 - val_acc: 0.8843
Epoch 12/50
48000/48000 [==============================] - 17s 363us/step - loss: 0.2900 - acc: 0.8989 - val_l
oss: 0.3201 - val_acc: 0.8883
Epoch 13/50
48000/48000 [==============================] - 17s 362us/step - loss: 0.2810 - acc: 0.9018 - val_l
oss: 0.3161 - val_acc: 0.8875
Epoch 14/50
48000/48000 [==============================] - 17s 362us/step - loss: 0.2755 - acc: 0.9035 - val_l
oss: 0.3083 - val_acc: 0.8925
Epoch 15/50
48000/48000 [==============================] - 17s 362us/step - loss: 0.2699 - acc: 0.9051 - val_l
oss: 0.3106 - val_acc: 0.8883
Epoch 16/50
48000/48000 [==============================] - 17s 361us/step - loss: 0.2636 - acc: 0.9075 - val_l
oss: 0.3087 - val_acc: 0.8925
Epoch 17/50
48000/48000 [==============================] - 17s 362us/step - loss: 0.2590 - acc: 0.9088 - val_l
oss: 0.3052 - val_acc: 0.8898
Epoch 18/50
48000/48000 [==============================] - 17s 362us/step - loss: 0.2538 - acc: 0.9105 - val_l
oss: 0.3078 - val_acc: 0.8915
Epoch 19/50
48000/48000 [==============================] - 17s 362us/step - loss: 0.2498 - acc: 0.9114 - val_l
oss: 0.2986 - val_acc: 0.8952
Epoch 20/50
48000/48000 [==============================] - 17s 360us/step - loss: 0.2474 - acc: 0.9127 - val_l
oss: 0.2954 - val_acc: 0.8963
Epoch 21/50
```

```
Epoch 21/50
48000/48000 [==============================] - 17s 363us/step - loss: 0.2400 - acc: 0.9152 - val_l
oss: 0.2966 - val_acc: 0.8958
Epoch 22/50
48000/48000 [==============================] - 17s 364us/step - loss: 0.2371 - acc: 0.9164 - val_l
oss: 0.2915 - val_acc: 0.8982
Epoch 23/50
48000/48000 [==============================] - 17s 360us/step - loss: 0.2336 - acc: 0.9169 - val_l
oss: 0.2966 - val_acc: 0.8963
Epoch 24/50
48000/48000 [==============================] - 17s 358us/step - loss: 0.2289 - acc: 0.9191 - val_l
oss: 0.3025 - val_acc: 0.8929
Epoch 25/50
48000/48000 [==============================] - 17s 360us/step - loss: 0.2253 - acc: 0.9201 - val_l
oss: 0.2858 - val_acc: 0.9002
Epoch 26/50
48000/48000 [==============================] - 17s 359us/step - loss: 0.2199 - acc: 0.9230 - val_l
oss: 0.2892 - val_acc: 0.8983
Epoch 27/50
48000/48000 [==============================] - 17s 361us/step - loss: 0.2204 - acc: 0.9219 - val_l
oss: 0.2929 - val_acc: 0.8967
Epoch 28/50
48000/48000 [==============================] - 17s 360us/step - loss: 0.2129 - acc: 0.9253 - val_l
oss: 0.2816 - val_acc: 0.9003
Epoch 29/50
48000/48000 [==============================] - 17s 359us/step - loss: 0.2083 - acc: 0.9265 - val_l
oss: 0.2839 - val_acc: 0.9012
Epoch 30/50
48000/48000 [==============================] - 17s 359us/step - loss: 0.2068 - acc: 0.9269 - val_l
oss: 0.2814 - val_acc: 0.9035
Epoch 31/50
48000/48000 [==============================] - 17s 359us/step - loss: 0.2049 - acc: 0.9273 - val_l
oss: 0.2908 - val_acc: 0.8958
Epoch 32/50
48000/48000 [==============================] - 17s 360us/step - loss: 0.2017 - acc: 0.9291 - val_l
oss: 0.2801 - val_acc: 0.9008
Epoch 33/50
48000/48000 [==============================] - 17s 353us/step - loss: 0.1961 - acc: 0.9320 - val_l
oss: 0.2790 - val_acc: 0.9016
Epoch 34/50
48000/48000 [==============================] - 17s 353us/step - loss: 0.1936 - acc: 0.9315 - val_l
oss: 0.2814 - val_acc: 0.9028
Epoch 35/50
48000/48000 [==============================] - 17s 349us/step - loss: 0.1897 - acc: 0.9335 - val_l
oss: 0.2783 - val_acc: 0.9054
Epoch 36/50
48000/48000 [==============================] - 17s 352us/step - loss: 0.1899 - acc: 0.9335 - val_l
oss: 0.2789 - val_acc: 0.9030
Epoch 37/50
48000/48000 [==============================] - 17s 358us/step - loss: 0.1849 - acc: 0.9347 - val_l
oss: 0.2804 - val_acc: 0.9045
Epoch 38/50
48000/48000 [==============================] - 17s 352us/step - loss: 0.1811 - acc: 0.9366 - val_l
oss: 0.2783 - val_acc: 0.9047
Epoch 39/50
48000/48000 [==============================] - 18s 371us/step - loss: 0.1787 - acc: 0.9375 - val_l
oss: 0.2803 - val_acc: 0.9052
Epoch 40/50
48000/48000 [==============================] - 18s 369us/step - loss: 0.1768 - acc: 0.9373 - val_l
oss: 0.2828 - val_acc: 0.9057
Epoch 41/50
48000/48000 [==============================] - 17s 352us/step - loss: 0.1723 - acc: 0.9399 - val_l
oss: 0.2747 - val_acc: 0.9079
Epoch 42/50
48000/48000 [==============================] - 17s 361us/step - loss: 0.1714 - acc: 0.9409 - val_l
oss: 0.2790 - val_acc: 0.9053
Epoch 43/50
48000/48000 [==============================] - 18s 368us/step - loss: 0.1673 - acc: 0.9425 - val_l
oss: 0.2834 - val_acc: 0.9042
Epoch 44/50
48000/48000 [==============================] - 18s 371us/step - loss: 0.1643 - acc: 0.9422 - val_l
oss: 0.2790 - val_acc: 0.9049
Epoch 45/50
48000/48000 [==============================] - 17s 361us/step - loss: 0.1664 - acc: 0.9414 - val_l
oss: 0.2807 - val_acc: 0.9027
Epoch 46/50
48000/48000 [==============================] - 18s 367us/step - loss: 0.1621 - acc: 0.9429 - val_l
oss: 0.2756 - val_acc: 0.9072
```

```
oss: 0.2756 - val_acc: 0.9072
Epoch 47/50
48000/48000 [==============================] - 18s 367us/step - loss: 0.1566 - acc: 0.9463 - val_l
oss: 0.2779 - val_acc: 0.9068
Epoch 48/50
48000/48000 [==============================] - 17s 357us/step - loss: 0.1553 - acc: 0.9461 - val_l
oss: 0.2846 - val_acc: 0.9027
Epoch 49/50
48000/48000 [==============================] - 17s 355us/step - loss: 0.1522 - acc: 0.9473 - val_l
oss: 0.2781 - val_acc: 0.9049
Epoch 50/50
48000/48000 [==============================] - 17s 359us/step - loss: 0.1489 - acc: 0.9486 - val_l
oss: 0.2807 - val_acc: 0.9072
```

Out[33]:

```
<keras.callbacks.History at 0x1143cf5e710>
```

when epoch number is increasing accuracy is also increasing

Step 5 : Evaluting the model

In [35]:

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

```
10000/10000 [==============================] - 2s 169us/step
Test Accuracy : 0.912300
```

In [36]:

```python
predicted_class = cnn_model.predict_classes(X_test)
```

In [37]:

```python
predicted_class
```

Out[37]:

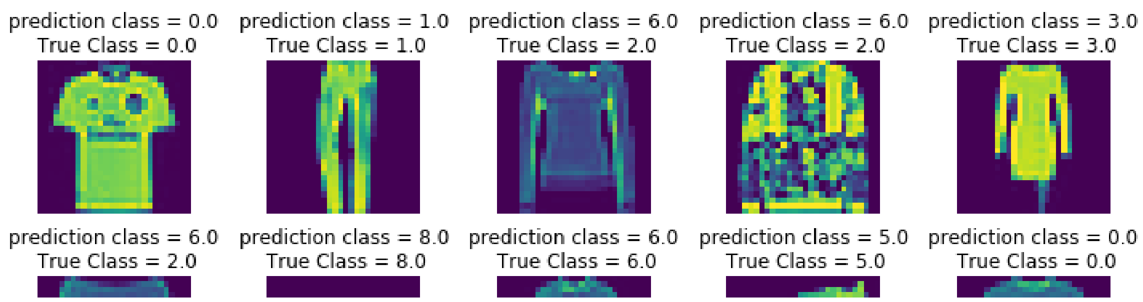
```
array([0, 1, 6, ..., 8, 8, 1], dtype=int64)
```

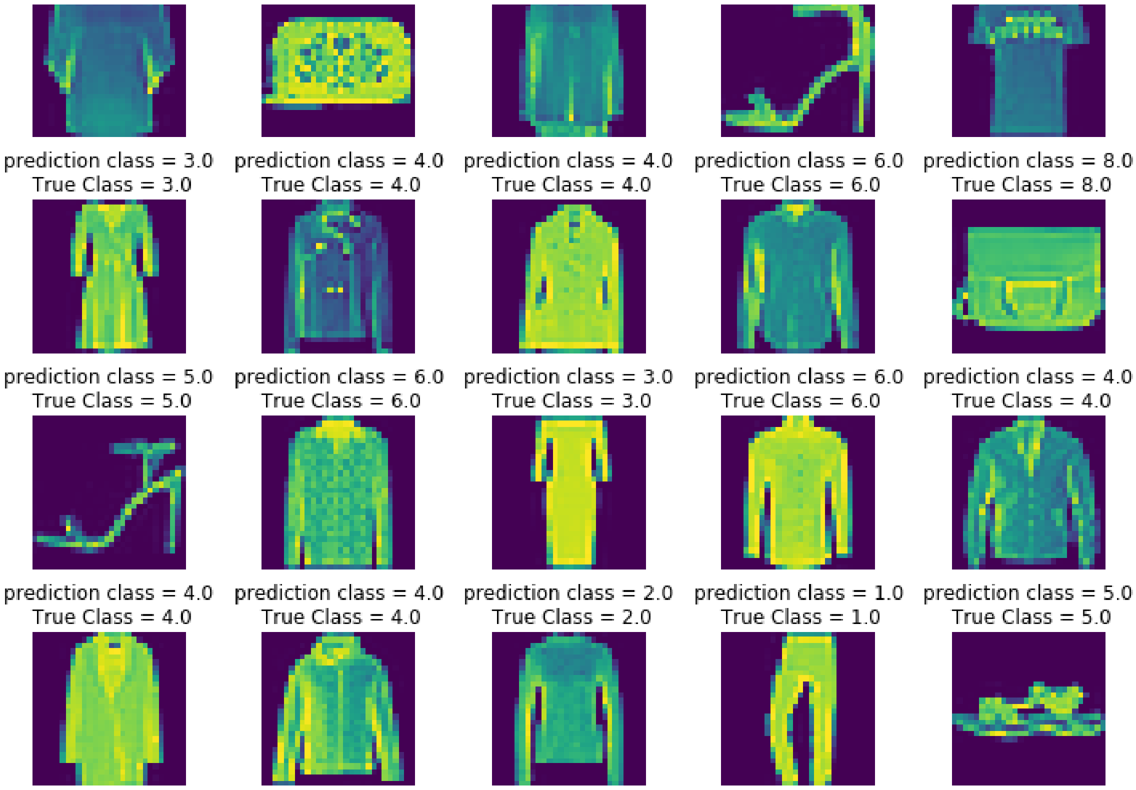
I am having 0 to 8 classes

In [41]:

```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_class[i]
,Y_test[i]))
    axes[i].axis("off")

plt.subplots_adjust(wspace=0.5)
```



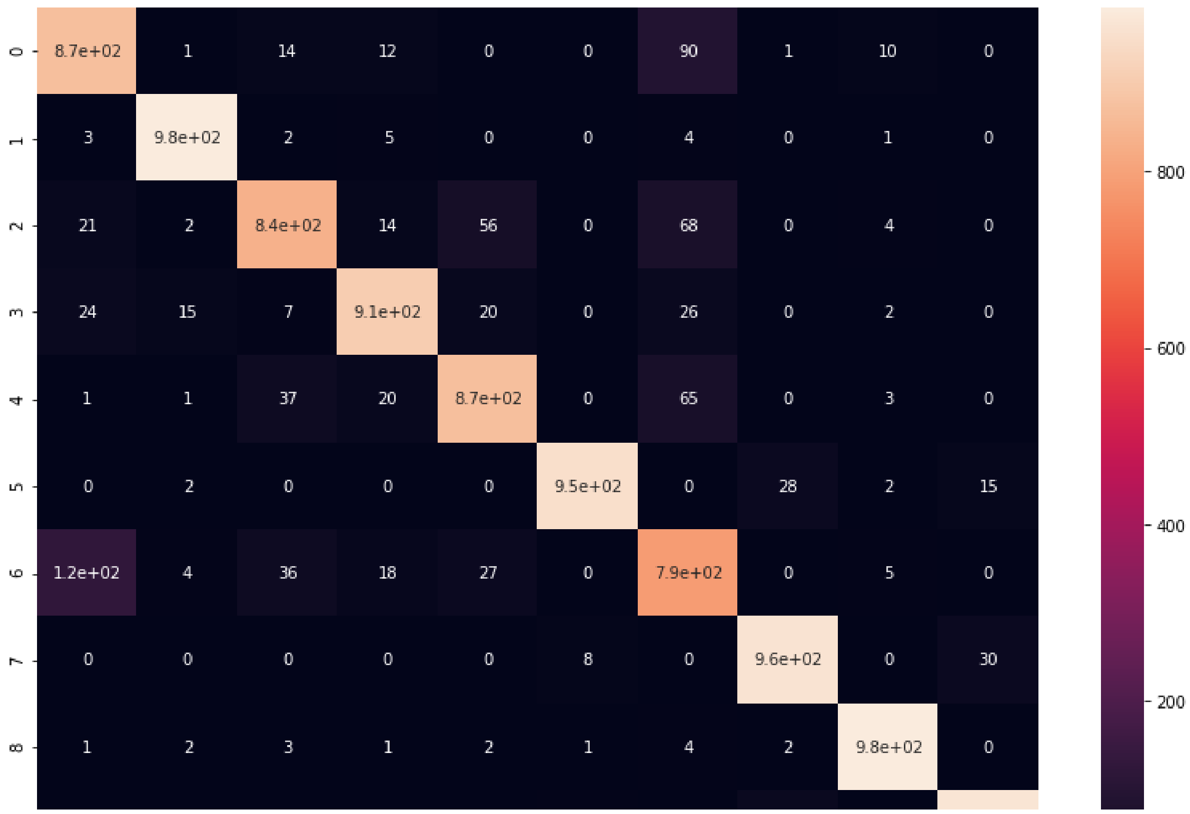


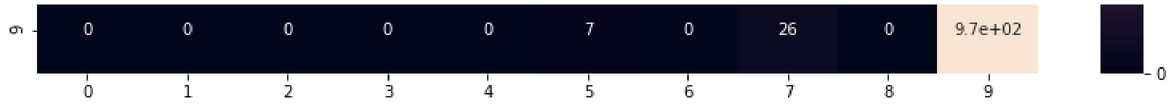
In [43]:

```
from sklearn.metrics import confusion_matrix
cn = confusion_matrix(Y_test,predicted_class)
plt.figure(figsize=(14,10))
sns.heatmap(cn,annot=True)
#we will use seaborn heatmap to show the number of sample which correctly classified and number of
sample which is not correctly
#classified
```

Out[43]:

<matplotlib.axes._subplots.AxesSubplot at 0x11467ee80b8>





In [45]:

```
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_class,target_names=target_names))
```

```
              precision    recall  f1-score   support

     Class 0       0.83      0.87      0.85      1000
     Class 1       0.97      0.98      0.98      1000
     Class 2       0.89      0.83      0.86      1000
     Class 3       0.93      0.91      0.92      1000
     Class 4       0.89      0.87      0.88      1000
     Class 5       0.98      0.95      0.97      1000
     Class 6       0.75      0.79      0.77      1000
     Class 7       0.94      0.96      0.95      1000
     Class 8       0.97      0.98      0.98      1000
     Class 9       0.96      0.97      0.96      1000

    accuracy                           0.91     10000
   macro avg       0.91      0.91      0.91     10000
weighted avg       0.91      0.91      0.91     10000
```

In [ ]:

```
#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
```

We found that sandal is classfied correctly. We can see sandal properly. Sneaker ,Bag we can see properly

Improving the Model

We can improve the model by adding kernal or adding a dropout(Dropout is regularization technique for reducing the overfitting in neural networks)

In [ ]: