In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [ ]:
```
https://www.youtube.com/watch?v=2C5uGteyhS4----- good youtube
https://www.tensorflow.org/tutorials/images/cnn--- good cnn model of other example
```

In [2]:
```
%matplotlib inline
```

In [3]:
```
fashion_train_df= pd.read_csv('fashion-mnist_train.csv')
```

In [5]:
```
fashion_test_df = pd.read_csv('fashion-mnist_test.csv')
```

In [6]:
```
fashion_train_df.head()
```

Out[6]:

	label	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	...	pixel775	pixel7
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	
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 rows × 785 columns

In [7]:
```
fashion_train_df.tail()
```

Out[7]:

	label	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	...	pixel775	p
59995	9	0	0	0	0	0	0	0	0	0	...	0	
59996	1	0	0	0	0	0	0	0	0	0	...	73	
59997	8	0	0	0	0	0	0	0	0	0	...	160	
59998	8	0	0	0	0	0	0	0	0	0	...	0	
59999	7	0	0	0	0	0	0	0	0	0	...	0	

5 rows × 785 columns

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

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

In [9]:
```
fashion_test_df.shape
```

Out[9]:
```
(10000, 785)
```

In [10]:
```
training = np.array(fashion_train_df,dtype='float32')
testing = np.array(fashion_test_df,dtype='float32')
```

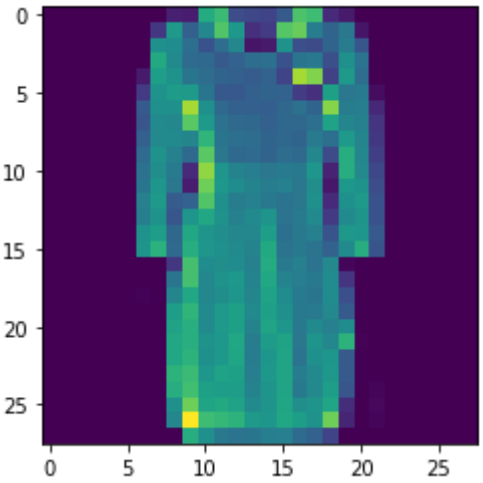
In [11]:
```
training.shape
```

Out[11]:
```
(60000, 785)
```

In [12]:
```
import random
```

In [13]:
```
i = random.randint(0,60001)
plt.imshow(training[i,1:].reshape(28,28))
label = training[i,1]
label
```

Out[13]:
```
0.0
```



i = random.randint(0,60001) plt.imshow(training[i,1:].reshape(28,28)) label = training[i,1] label

In [14]:
```
W_grid = 7
L_grid = 7

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)
```

In [15]:
```
X_train = training[:,1:]/255
y_train = training[:,0]
X_test = testing[:,1:]/255
y_test = testing[:,0]
```

In [16]:
```
from sklearn.model_selection import train_test_split
X_train, X_validate, y_train, y_validate = train_test_split(X_train, y_train,test_
```

In [17]:
```
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 [18]:
```
X_train.shape
```

Out[18]:
```
(48000, 28, 28, 1)
```

In [19]:
```
X_test.shape
```

Out[19]:
```
(10000, 28, 28, 1)
```

In [20]:
```
X_validate.shape
```

Out[20]:
```
(12000, 28, 28, 1)
```

In [21]:
```
import tensorflow as tf
from tensorflow import keras
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_model = Sequential()
cnn_model.add(Conv2D(32,3,3,input_shape = (28,28,1),activation = 'relu'))
cnn_model.add(MaxPooling2D(pool_size= (2,2)))
cnn_model.add(Flatten())
cnn_model.add(Dense(32,activation = 'relu'))
cnn_model.add(Dense(10,activation = 'sigmoid'))
cnn_model.compile(loss ='sparse_categorical_crossentropy',optimizer = Adam(learning
```

In [23]:
```
epochs = 20
```

In [24]:
```
cnn_model.fit(X_train,y_train,batch_size =512,epochs = epochs,verbose = 1,validati
```

```
Epoch 1/20
94/94 [==============================] - 2s 13ms/step - loss: 1.3759 - accuracy:
0.5645 - val_loss: 0.8201 - val_accuracy: 0.7128
Epoch 2/20
94/94 [==============================] - 1s 13ms/step - loss: 0.7228 - accuracy:
0.7392 - val_loss: 0.6570 - val_accuracy: 0.7579
Epoch 3/20
94/94 [==============================] - 1s 12ms/step - loss: 0.6212 - accuracy:
0.7717 - val_loss: 0.6024 - val_accuracy: 0.7821
Epoch 4/20
94/94 [==============================] - 1s 12ms/step - loss: 0.5735 - accuracy:
0.7907 - val_loss: 0.5632 - val_accuracy: 0.7947
Epoch 5/20
94/94 [==============================] - 1s 12ms/step - loss: 0.5377 - accuracy:
0.8046 - val_loss: 0.5313 - val_accuracy: 0.8084
Epoch 6/20
94/94 [==============================] - 1s 12ms/step - loss: 0.5150 - accuracy:
0.8134 - val_loss: 0.5166 - val_accuracy: 0.8160
Epoch 7/20
94/94 [==============================] - 1s 11ms/step - loss: 0.4982 - accuracy:
0.8174 - val_loss: 0.4940 - val_accuracy: 0.8253
Epoch 8/20
94/94 [==============================] - 1s 11ms/step - loss: 0.4787 - accuracy:
0.8270 - val_loss: 0.4844 - val_accuracy: 0.8259
Epoch 9/20
94/94 [==============================] - 1s 11ms/step - loss: 0.4667 - accuracy:
0.8315 - val_loss: 0.4727 - val_accuracy: 0.8316
Epoch 10/20
94/94 [==============================] - 1s 12ms/step - loss: 0.4553 - accuracy:
0.8350 - val_loss: 0.4593 - val_accuracy: 0.8369
Epoch 11/20
94/94 [==============================] - 1s 11ms/step - loss: 0.4471 - accuracy:
0.8371 - val_loss: 0.4620 - val_accuracy: 0.8298
Epoch 12/20
94/94 [==============================] - 1s 11ms/step - loss: 0.4377 - accuracy:
0.8416 - val_loss: 0.4427 - val_accuracy: 0.8393
Epoch 13/20
94/94 [==============================] - 1s 12ms/step - loss: 0.4310 - accuracy:
0.8436 - val_loss: 0.4357 - val_accuracy: 0.8445
Epoch 14/20
94/94 [==============================] - 1s 12ms/step - loss: 0.4263 - accuracy:
0.8450 - val_loss: 0.4457 - val_accuracy: 0.8392
Epoch 15/20
94/94 [==============================] - 1s 12ms/step - loss: 0.4186 - accuracy:
0.8476 - val_loss: 0.4370 - val_accuracy: 0.8418
Epoch 16/20
94/94 [==============================] - 1s 12ms/step - loss: 0.4153 - accuracy:
0.8487 - val_loss: 0.4252 - val_accuracy: 0.8470
Epoch 17/20
94/94 [==============================] - 1s 11ms/step - loss: 0.4119 - accuracy:
0.8508 - val_loss: 0.4212 - val_accuracy: 0.8495
Epoch 18/20
94/94 [==============================] - 1s 11ms/step - loss: 0.4072 - accuracy:
0.8515 - val_loss: 0.4221 - val_accuracy: 0.8471
Epoch 19/20
94/94 [==============================] - 1s 12ms/step - loss: 0.4032 - accuracy:
0.8540 - val_loss: 0.4180 - val_accuracy: 0.8489
Epoch 20/20
94/94 [==============================] - 1s 12ms/step - loss: 0.4011 - accuracy:
0.8552 - val_loss: 0.4122 - val_accuracy: 0.8508
```

Out[24]:
```
<keras.callbacks.History at 0x27fe7375190>
```

In [25]:
```
evaluation = cnn_model.evaluate(X_test,y_test)
print('Test Accuracy : {:.3f}'.format(evaluation[1]))
```

```
313/313 [==============================] - 0s 1ms/step - loss: 0.3963 - accuracy:
0.8528
Test Accuracy : 0.853
```

In [26]:
```
predicted_classes = np.argmax(cnn_model.predict(X_test),axis=-1)
```

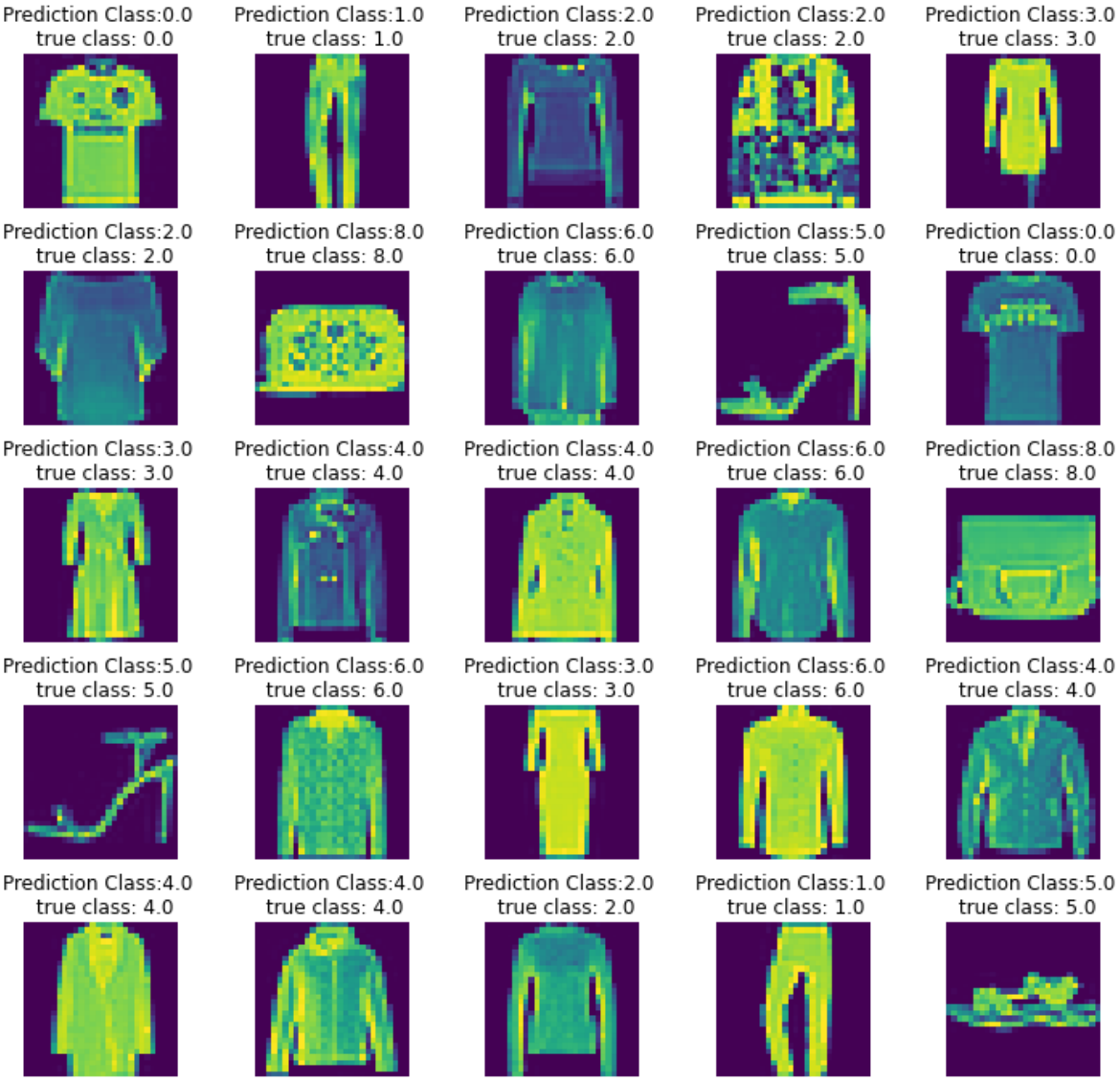
```
313/313 [==============================] - 0s 1ms/step
```

In [27]:
```
predicted_classes
```

Out[27]:
```
array([0, 1, 2, ..., 8, 8, 1], dtype=int64)
```

In [28]:
```
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:{1} \n true class: {1}'.format(predicted_cl
    axes[i].axis('off')
plt.subplots_adjust(wspace = 0.5)
```



In [29]:
```
from sklearn.metrics import classification_report

classes = 10
targets = ["Class {}".format(i) for i in range(classes)]
print(classification_report(y_test, predicted_classes, target_names = targets))
```

```
              precision    recall  f1-score   support

     Class 0       0.82      0.79      0.80      1000
     Class 1       0.97      0.97      0.97      1000
     Class 2       0.79      0.74      0.77      1000
     Class 3       0.89      0.85      0.87      1000
     Class 4       0.73      0.81      0.77      1000
     Class 5       0.95      0.93      0.94      1000
     Class 6       0.59      0.59      0.59      1000
     Class 7       0.90      0.93      0.91      1000
     Class 8       0.95      0.97      0.96      1000
     Class 9       0.95      0.94      0.94      1000

    accuracy                           0.85     10000
   macro avg       0.85      0.85      0.85     10000
weighted avg       0.85      0.85      0.85     10000
```