LP-5 DEEP LEARNING Practical 3

Convolutional neural network (CNN)

Use MNIST Fashion Dataset and create a classifier to classify fashion clothing into categories.

Name: **ONASVEE BANARSE**

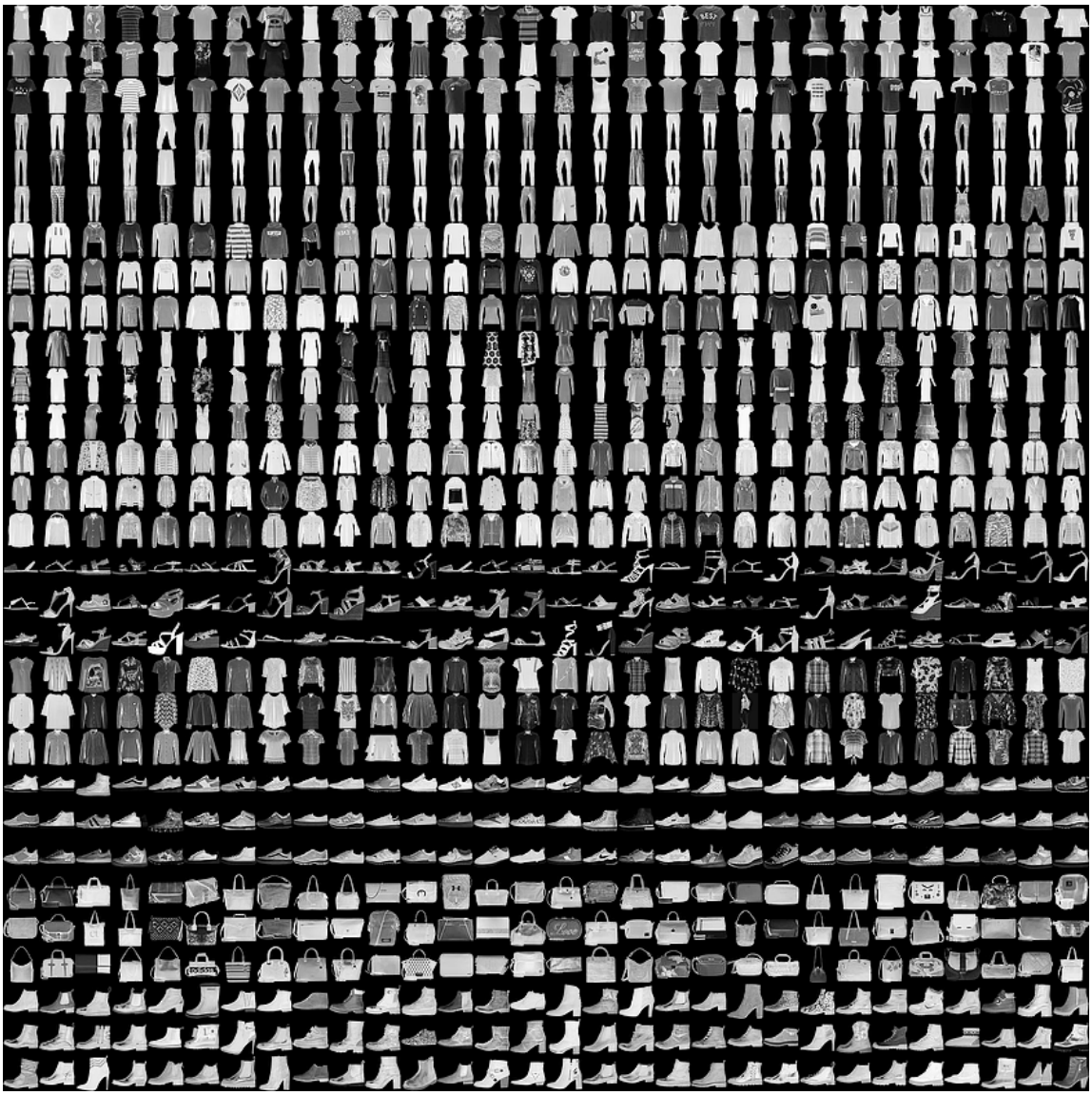
Rollno: **09**

BE COMP 1

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

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

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

In [5]:
```
print(set(fashion_train_df['label']))
```

```
{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
```

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	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

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	pixel7
0	0	0	0	0	0	0	0	0	9	8	...	103	
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	1
4	3	0	0	0	0	0	0	0	0	0	...	0	

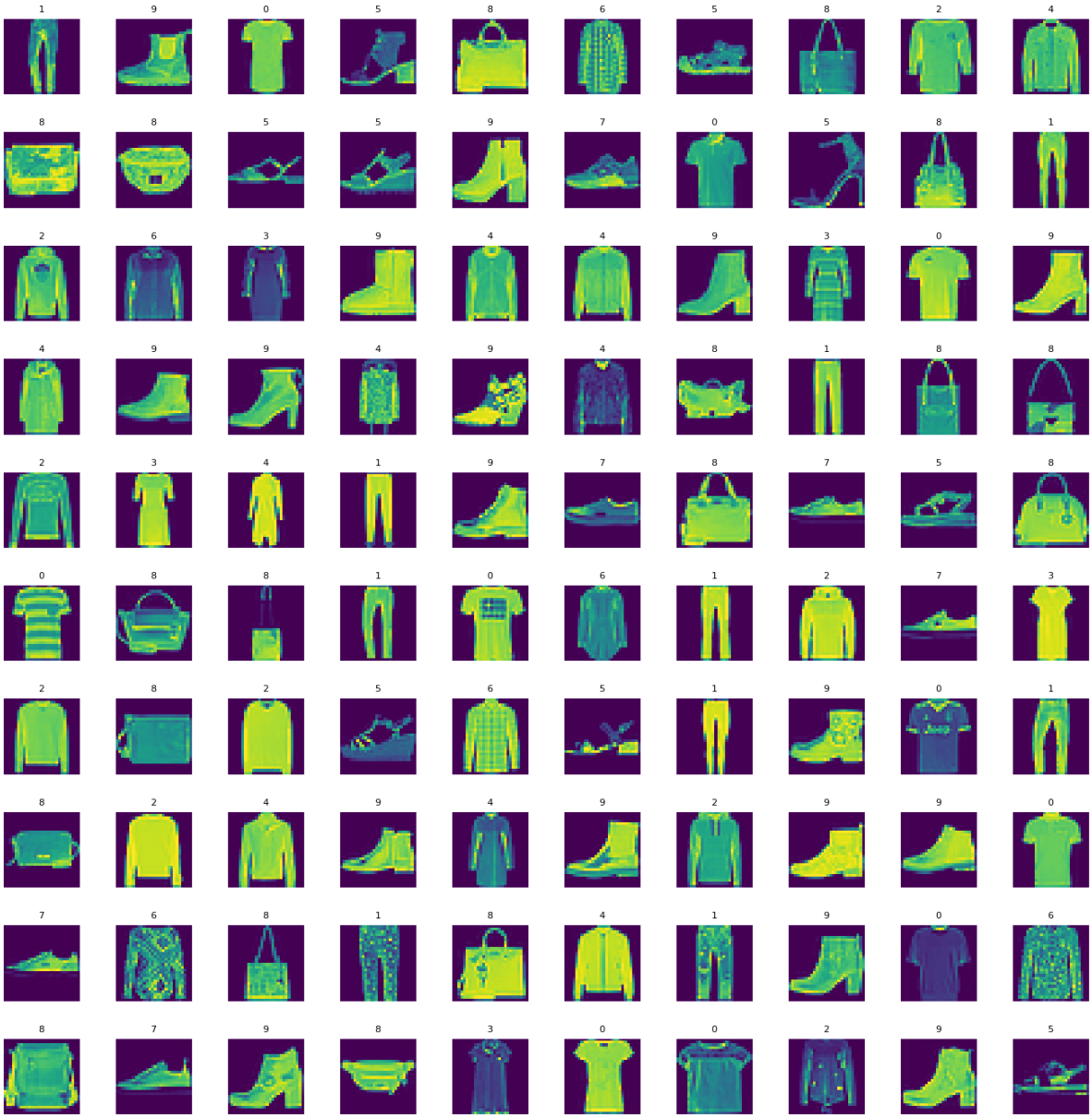
5 rows × 785 columns

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]:

```
# 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

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 [12]:
```
# Split the training set into training and validation sets
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.2,
```

In [13]:
```
# 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)
```

```
(48000, 28, 28, 1) (12000, 28, 28, 1) (10000, 28, 28, 1)
(48000,) (12000,) (10000,)
```

Step # 5 - Create and Train the Model

Create the model

In [14]:
```
cnn_model = Sequential()
cnn_model.add(Conv2D(filters=64, kernel_size=(3,3), input_shape=(28,28,1), activati
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 [15]:
```
cnn_model.compile(optimizer=Adam(learning_rate=0.001), loss='sparse_categorical_cr
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
_________________________________________________________________
```

Train the model

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

Step # 5 - Evaluate the Model

Get the accuracy of the model

In [17]:
```
eval_result = cnn_model.evaluate(X_test, y_test)
print("Accuracy : {:.3f}".format(eval_result[1]))
```

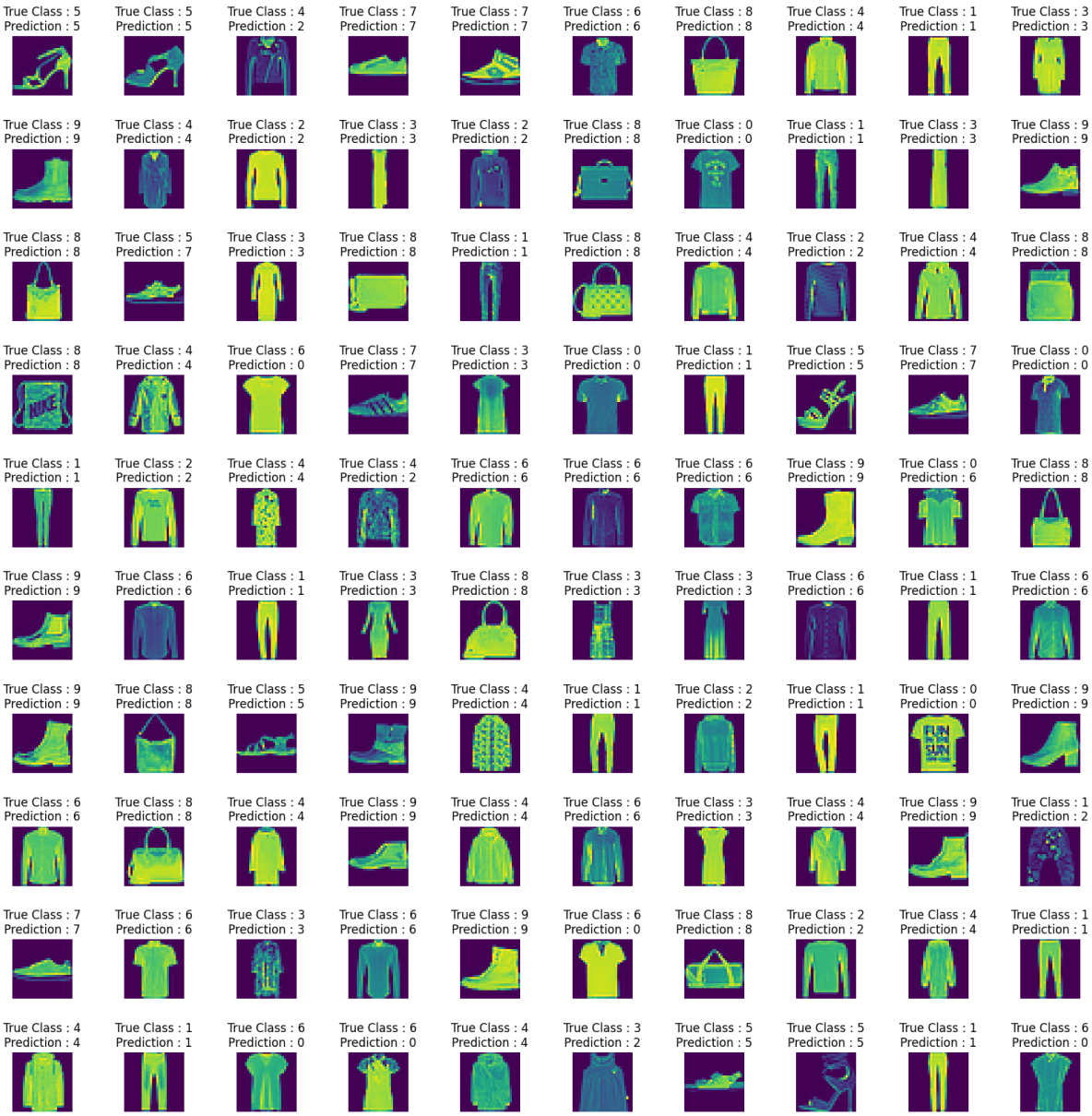
```
313/313 [==============================] - 1s 4ms/step - loss: 0.2491 - accuracy:
0.9191
Accuracy : 0.919
```

Visualize the model's predictions

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

```
313/313 [==============================] - 1s 4ms/step
```

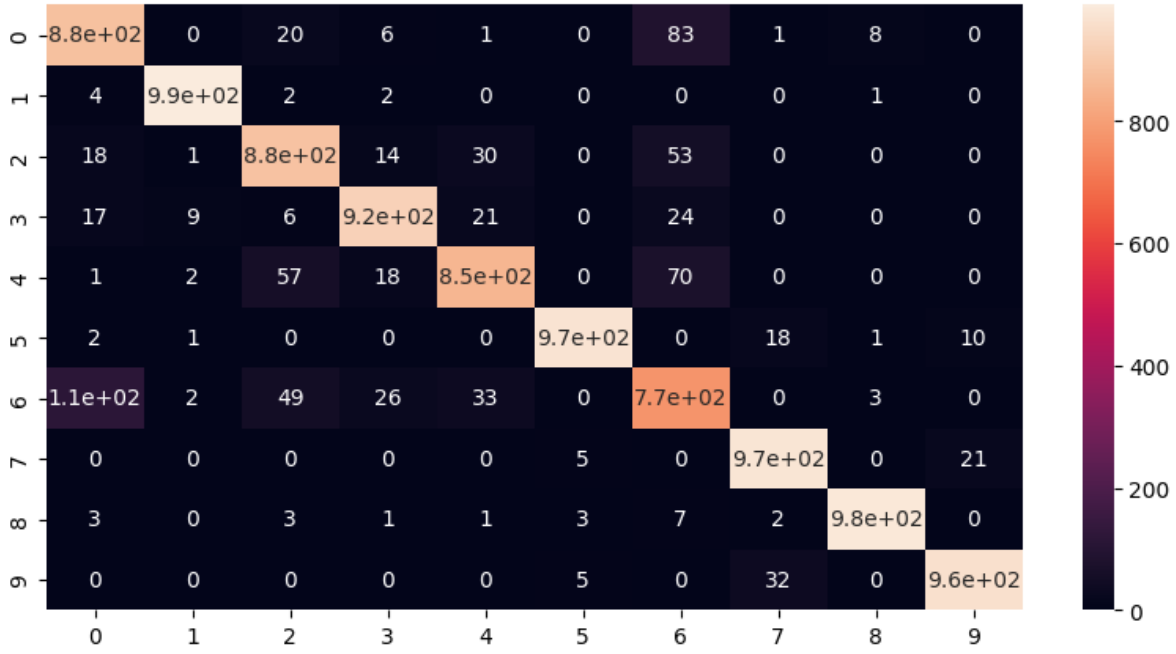
In [28]:
```
height = 10
width = 10

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(classes_x))
    axes[i].imshow(X_test[index].reshape((28,28)))
    axes[i].set_title("True Class : {:0.0f}\nPrediction : {:d}".format(y_test[inde
    axes[i].axis('off')
plt.subplots_adjust(hspace=0.9, wspace=0.5)
```



Plot Confusin Matrix

In [29]:
```
cm = confusion_matrix(y_test, classes_x)
plt.figure(figsize=(10,5))
sbn.heatmap(cm, annot=True)
```

Out[29]: <AxesSubplot: >



Classification Report

In [30]:
```
num_classes = 10
class_names = ["class {}".format(i) for i in range(num_classes)]
cr = classification_report(y_test, classes_x, target_names=class_names)
print(cr)
```

```
              precision    recall  f1-score   support

     class 0       0.85      0.88      0.86      1000
     class 1       0.99      0.99      0.99      1000
     class 2       0.87      0.88      0.87      1000
     class 3       0.93      0.92      0.93      1000
     class 4       0.91      0.85      0.88      1000
     class 5       0.99      0.97      0.98      1000
     class 6       0.77      0.77      0.77      1000
     class 7       0.95      0.97      0.96      1000
     class 8       0.99      0.98      0.98      1000
     class 9       0.97      0.96      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
```