Competation title-- Digit Recognizer

https://www.kaggle.com/c/digit-recognizer

Importing

Importing module for this learning process

We will implement **NEURAL NETWORKS** for this purpose

```
import tensorflow as tf
from keras.utils.np_utils import to_categorical
from sklearn.model_selection import train_test_split
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.optimizers import Adam
```

Loading Dataset Path

Calling both the present dataset train and test from the kaggle directory

```
TRAIN_PATH = "/content/train.csv"
TEST_PATH = "/content/test.csv"
```

1. Reading Data

Reading both the present csv by using the pandas module and checking the content of the given data

```
train = pd.read_csv(TRAIN_PATH)
test = pd.read_csv(TEST_PATH)
print('Training Data \n')
print(train.head(5))
print('Testing Data \n')
print(test.head(5))
print(test.shape)
```

```
Training Data 

   label  pixel0  pixel1  pixel2  ...  pixel780  pixel781  pixel782  pixel783
0      1       0       0       0  ...         0         0         0         0
1      0       0       0       0  ...         0         0         0         0
2      1       0       0       0  ...         0         0         0         0
3      4       0       0       0  ...         0         0         0         0
4      0       0       0       0  ...         0         0         0         0
```

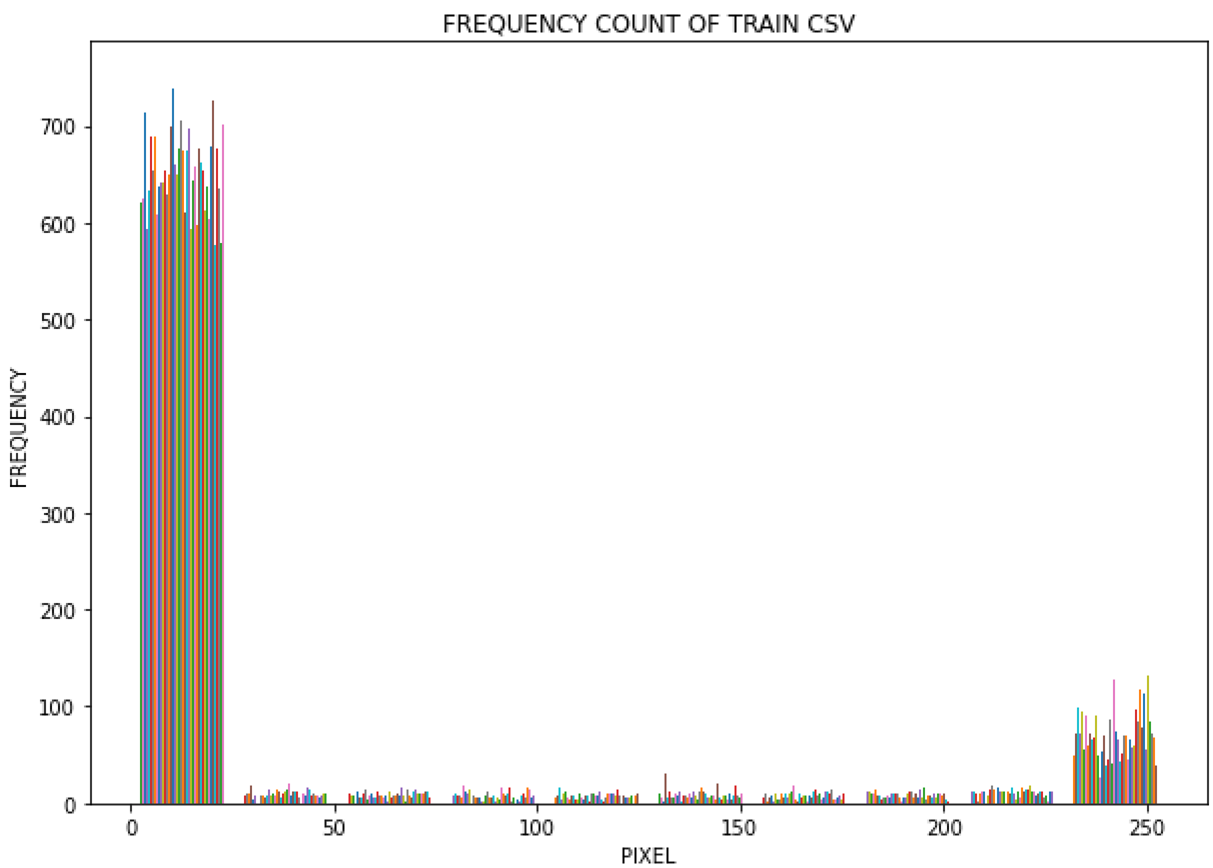
```
[5 rows x 785 columns]
Testing Data

   pixel0  pixel1  pixel2  pixel3  ...  pixel780  pixel781  pixel782  pixel783
0       0       0       0       0  ...         0         0         0         0
1       0       0       0       0  ...         0         0         0         0
2       0       0       0       0  ...         0         0         0         0
3       0       0       0       0  ...         0         0         0         0
4       0       0       0       0  ...         0         0         0         0

[5 rows x 784 columns]
(28000, 784)
```

2. Visualing data

The Training Dataset consist of **42000 rows & 785 columns**out of 785 columns 1 column is label and the rest 784 is pixel value ranging from (0,255)

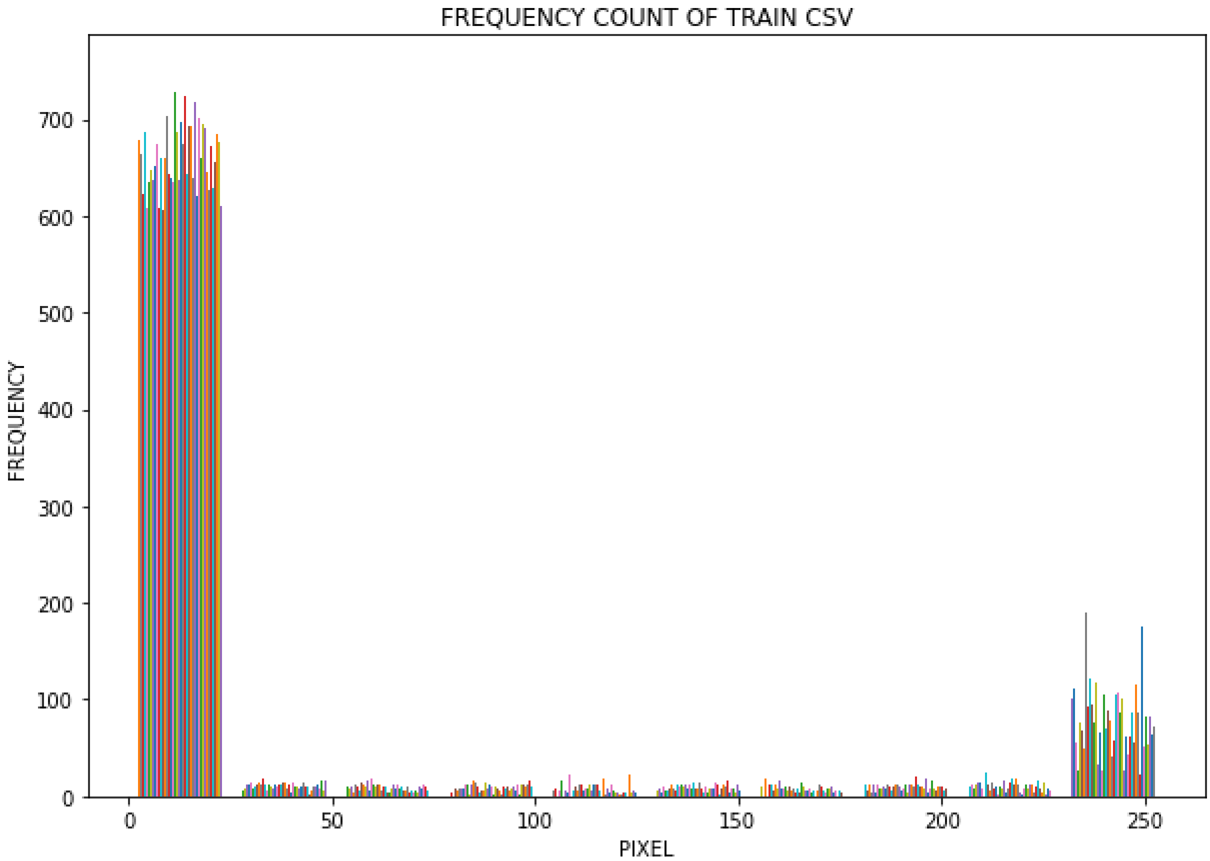
The Testing Dataset consist of **28000 rows & 784 columns**

```
plt.figure(figsize = (10,7))
plt.hist(train.drop(['label'],axis=1))
plt.title('FREQUENCY COUNT OF TRAIN CSV')
plt.xlabel('PIXEL')
plt.ylabel('FREQUENCY')
plt.show()
```



After Visulazing the train dataset we can see that most of pixel is in between the range (1,10) and we minor hike at pixel count of (230,250)

```
plt.figure(figsize = (10,7))
plt.hist(test)
plt.title('FREQUENCY COUNT OF TRAIN CSV')
plt.xlabel('PIXEL')
plt.ylabel('FREQUENCY')
plt.show()
```



After Visulazing the test dataset we can see similar result as train i.e most of pixel is in between the range (1,10) and we minor hike at pixel count of (230,250)

3. Preprocessing

Part converting the pixel in the range of 0 and 1 and then converting the labels to cateogorical values, As the shape of the pixel is not appropiate then convert it into the required shape i.e. of (28,28)

```
y=train['label']
X=train.drop(['label'],axis=1)
X=X/255
test=test/255
```

```
X = X.values.reshape(-1,28,28,1)
#test = test.values.reshape(-1,28,28,1)
```

4. Module Making

This process consist of various steps such as

1. Spliting the data
2. Visualizing the split data
3. Defining the Module
4. Training the Module
5. Visualizing accuracy & loss of Module

Spliting the data

Using the **train_test_split** from sklearn to split the data into **training and validation set i.e (80% of training & 20% of validation)**

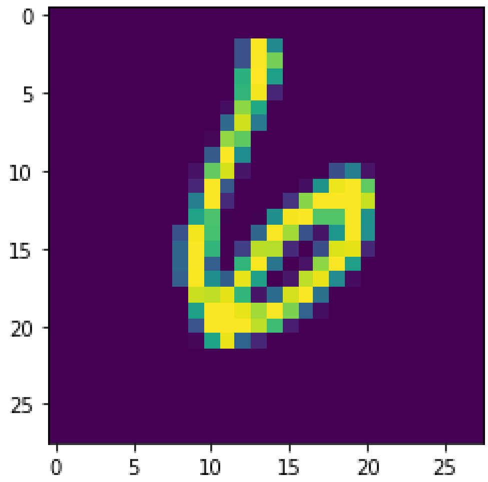
```
x_train,x_val,y_train,y_val=train_test_split(X,y,test_size=0.2, random_state=42)
```

Visualizing the split data

We visualize the label and the pixel value of the training data of the first image

```
plt.imshow((tf.squeeze(x_train[0])))
```

```
<matplotlib.image.AxesImage at 0x7fda81677f10>
```



Defining Module

For this purpose we will use the tensorflow and declare our layers and convolutional

```
model = tf.keras.Sequential([
    tf.keras.layers.Conv2D(32,(3,3),activation='relu',input_shape=(28,28,1)),
    tf.keras.layers.MaxPool2D(2,2),
    tf.keras.layers.Conv2D(32,(3,3),activation='relu'),
    tf.keras.layers.MaxPool2D(2,2),
    tf.keras.layers.Conv2D(16,(3,3),activation='relu'),
    tf.keras.layers.MaxPool2D(2,2),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(10, activation='sigmoid')
])

model.compile(
    optimizer='adam',
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
)
model.summary()
'''from tensorflow.keras.utils import plot_model
plot_model(model)'''
```

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_12 (Conv2D)          (None, 26, 26, 32)        320

 max_pooling2d_12 (MaxPoolin  (None, 13, 13, 32)       0
 g2D)

 conv2d_13 (Conv2D)          (None, 11, 11, 32)        9248

 max_pooling2d_13 (MaxPoolin  (None, 5, 5, 32)         0
 g2D)

 conv2d_14 (Conv2D)          (None, 3, 3, 16)          4624

 max_pooling2d_14 (MaxPoolin  (None, 1, 1, 16)         0
 g2D)

 flatten_4 (Flatten)         (None, 16)                0

 dense_8 (Dense)             (None, 128)               2176

 dense_9 (Dense)             (None, 10)                1290

=================================================================
Total params: 17,658
Trainable params: 17,658
Non-trainable params: 0
_________________________________________________________________
'from tensorflow.keras.utils import plot_model\nplot_model(model)'
```

Training the module

Using ImageDataGenerator from tensorflow to train the training and the validation data

```
train_datagen = ImageDataGenerator(featurewise_center=False,
                             samplewise_center=False,
                             featurewise_std_normalization=False,
                             samplewise_std_normalization=False,
                             zca_whitening=False,
                             rotation_range=10,
                             zoom_range=0.1,
                             width_shift_range=0.1,
                             height_shift_range=0.1,
                             horizontal_flip=False,
                             vertical_flip=False
                            )
train_generator = train_datagen.flow(x_train, y_train,
                                     batch_size=32,
                                     shuffle=True)

val_datagen = ImageDataGenerator()
val_generator = val_datagen.flow(x_val, y_val,
                                 batch_size=32,
                                 shuffle=True)
history=model.fit(
                train_generator,
                epochs=30,
                validation_data=val_generator,
                verbose=1)
```

```
Epoch 2/30
1050/1050 [==============================] - 21s 20ms/step - loss: 0.1787 - ac
Epoch 3/30
1050/1050 [==============================] - 21s 20ms/step - loss: 0.1616 - ac
Epoch 4/30
1050/1050 [==============================] - 21s 20ms/step - loss: 0.1511 - ac
Epoch 5/30
1050/1050 [==============================] - 21s 20ms/step - loss: 0.1429 - ac
Epoch 6/30
1050/1050 [==============================] - 21s 20ms/step - loss: 0.1275 - ac
Epoch 7/30
1050/1050 [==============================] - 21s 20ms/step - loss: 0.1195 - ac
Epoch 8/30
1050/1050 [==============================] - 21s 20ms/step - loss: 0.1224 - ac
Epoch 9/30
1050/1050 [==============================] - 22s 20ms/step - loss: 0.1175 - ac
Epoch 10/30
1050/1050 [==============================] - 21s 20ms/step - loss: 0.1114 - ac
Epoch 11/30
1050/1050 [==============================] - 22s 21ms/step - loss: 0.1069 - ac
Epoch 12/30
1050/1050 [==============================] - 22s 20ms/step - loss: 0.0979 - ac
Epoch 13/30
1050/1050 [==============================] - 21s 20ms/step - loss: 0.1013 - ac
Epoch 14/30
1050/1050 [==============================] - 21s 20ms/step - loss: 0.0947 - ac
Epoch 15/30
1050/1050 [==============================] - 21s 20ms/step - loss: 0.0914 - ac

Epoch 16/30
1050/1050 [==============================] - 21s 20ms/step - loss: 0.0879 - ac
```

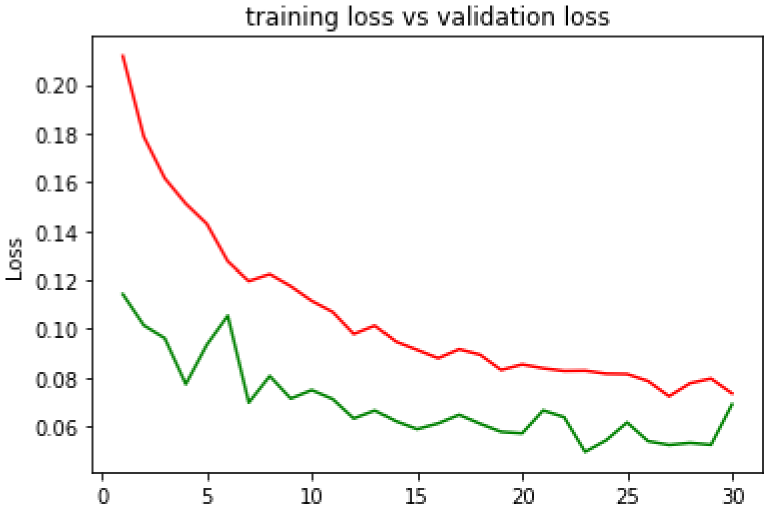
```
Epoch 17/30
1050/1050 [==============================] - 21s 20ms/step - loss: 0.0916 - ac
Epoch 18/30
1050/1050 [==============================] - 21s 20ms/step - loss: 0.0894 - ac
Epoch 19/30
1050/1050 [==============================] - 21s 20ms/step - loss: 0.0831 - ac
Epoch 20/30
1050/1050 [==============================] - 21s 20ms/step - loss: 0.0854 - ac
Epoch 21/30
1050/1050 [==============================] - 21s 20ms/step - loss: 0.0838 - ac
Epoch 22/30
1050/1050 [==============================] - 21s 20ms/step - loss: 0.0827 - ac
Epoch 23/30
1050/1050 [==============================] - 21s 20ms/step - loss: 0.0828 - ac
Epoch 24/30
1050/1050 [==============================] - 21s 20ms/step - loss: 0.0816 - ac
Epoch 25/30
1050/1050 [==============================] - 21s 20ms/step - loss: 0.0814 - ac
Epoch 26/30
1050/1050 [==============================] - 21s 20ms/step - loss: 0.0786 - ac
Epoch 27/30
1050/1050 [==============================] - 21s 20ms/step - loss: 0.0723 - ac
Epoch 28/30
1050/1050 [==============================] - 21s 20ms/step - loss: 0.0777 - ac
Epoch 29/30
1050/1050 [==============================] - 21s 20ms/step - loss: 0.0796 - ac
Epoch 30/30
1050/1050 [==============================] - 21s 20ms/step - loss: 0.0735 - ac
```

Visualizing accuracy, loss of Module& Confusion Matrix

In this portion we willl visulaize the various parameters of the the we obtained after trainig the model and we find the relation between them to visualization is the best way to check weather the model is overfitting , underfitting or justperfect. In Deep Learning, the loss function is used by the model to learn. The goal of the model is to minimize the value of the loss. This is done by using techniques such as gradient descent, which changes the model parameters using the information of the result of the loss.

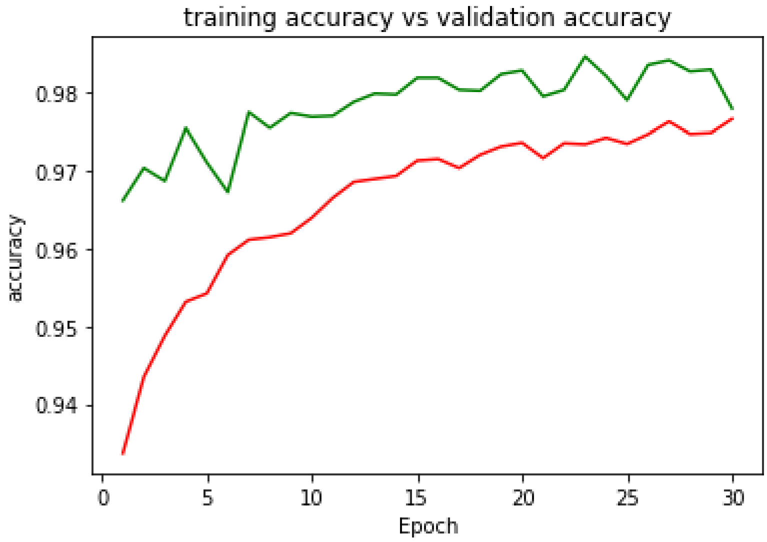
From the Chart of losses of training and validation we can see that both the losses gradually decrease which indicate the model is doing well we can see the drastic changes indicating the model is tring to learn.

```
training_loss = history.history['loss']
val_loss=history.history['val_loss']
epoch_count = range(1, len(training_loss) + 1)
#plt.figure(figsize = (10,7))
plt.plot(epoch_count, training_loss, 'r')
plt.plot(epoch_count, val_loss, 'g')
plt.title('training loss vs validation loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.show();
```



From the Chart of training accuracy and validation accuracy we can se the gradual increase in the acurracy of both the training aswell as validation accuracy thus showing there is no underfitting as well as overfitting

```
training_loss = history.history['accuracy']
val_loss=history.history['val_accuracy']
epoch_count = range(1, len(training_loss) + 1)
#plt.figure(figsize = (10,7))
plt.plot(epoch_count, training_loss, 'r')
plt.plot(epoch_count, val_loss, 'g')
plt.title('training accuracy vs validation accuracy')
plt.xlabel('Epoch')
plt.ylabel('accuracy')
plt.show();
```



Predicting the values and ploting confusing matrix

As we can see that all the diagonal elements are correct predictions, for example, we correctly predicted the number 0, 801 times.

Here the labels are the actual values and the predictions are the values that model predicted

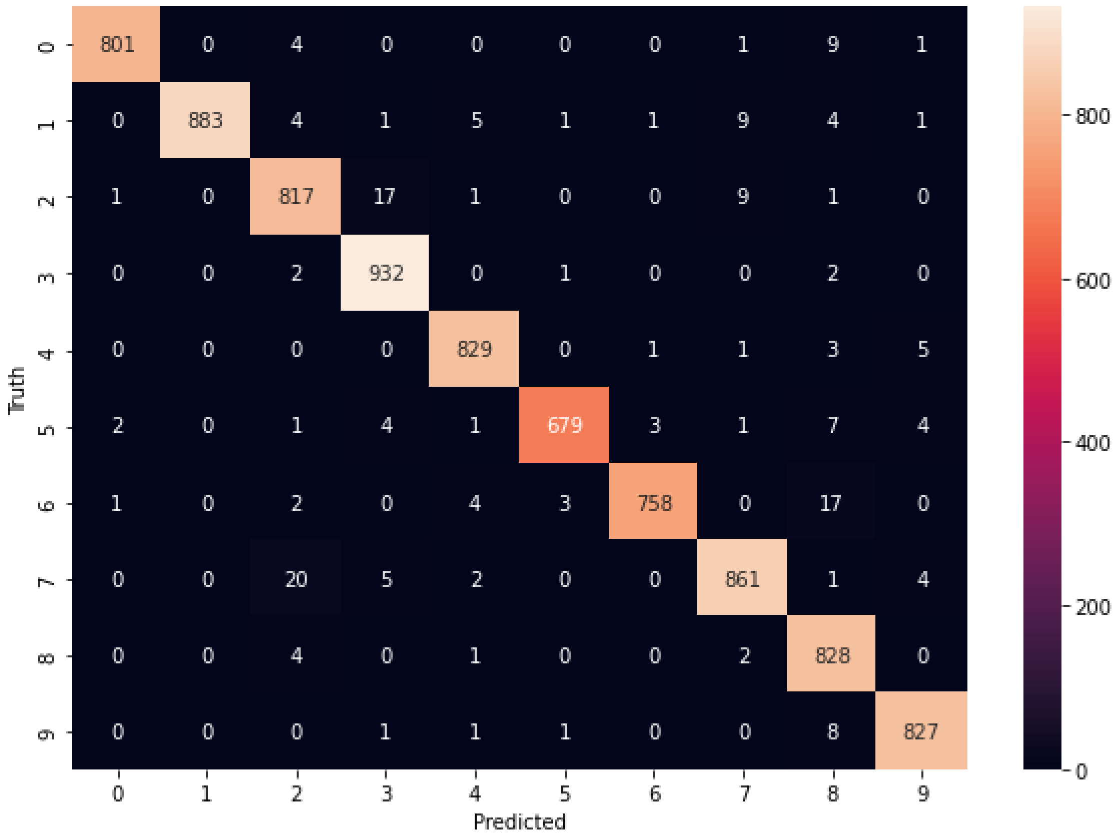
```
y_predicted = model.predict(x_val)
y_predicted_labels = [np.argmax(i) for i in y_predicted]
cm = tf.math.confusion_matrix(labels=y_val, predictions=y_predicted_labels)
print(cm)
```

```
tf.Tensor(
[[801   0   4   0   0   0   0   1   9   1]
 [  0 883   4   1   5   1   1   9   4   1]
 [  1   0 817  17   1   0   0   9   1   0]
 [  0   0   2 932   0   1   0   0   2   0]
 [  0   0   0   0 829   0   1   1   3   5]
 [  2   0   1   4   1 679   3   1   7   4]
 [  1   0   2   0   4   3 758   0  17   0]
 [  0   0  20   5   2   0   0 861   1   4]
 [  0   0   4   0   1   0   0   2 828   0]
 [  0   0   0   1   1   1   0   0   8 827]], shape=(10, 10), dtype=int32)
```

ploating the confusion value using the seaborn library taking input as the confusion matrix from the above cell o/p

```
import seaborn as sn
plt.figure(figsize = (10,7))
sn.heatmap(cm, annot=True, fmt='d')
plt.xlabel('Predicted')
plt.ylabel('Truth')
```

```
Text(69.0, 0.5, 'Truth')
```



✓ 0s completed at 7:37 PM