Handwritten Digit Recognition

Introduction

- The capacity of computers to read human-written numbers on paper is known as handwritten digit recognition. It is a hard task for the machine because handwritten numerals are not always accurate.

- The handwritten digit recognition is a solution to this issue that makes use of a digit's image to identify the digit that is present in the image

- Applications for digit recognition include form data entry, bank check processing, postal mail sorting, and others.

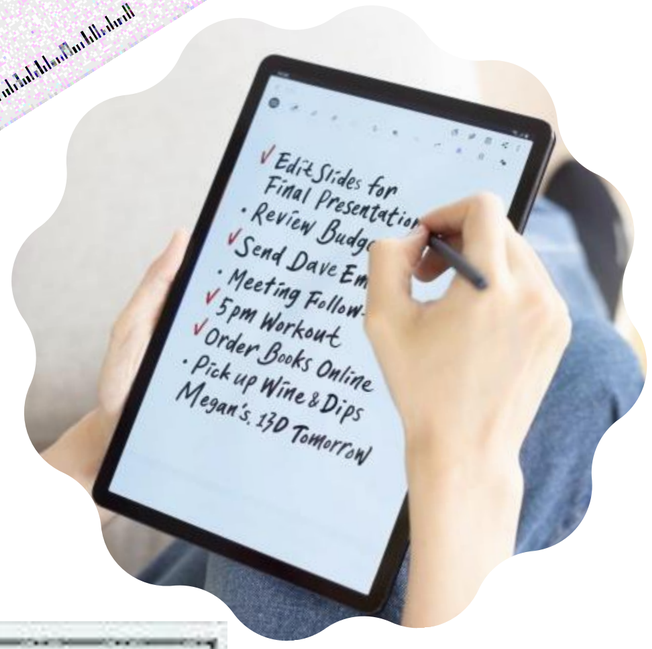
- The method for offline handwritten digit recognition presented in this work is based on various machine learning techniques.

- The major goal of this project is to provide efficient and trustworthy methods for handwritten digit recognition. This project recognizes digits using a variety of machine learning algorithms, Support Vector Classifier, Tensorflow ,Random Forest.

Research papers

- Shamim, S. M. & Miah, Md Badrul & Sarker, Angona & Rana, Masud & Jobair, Abdullah. (2018). Handwritten Digit Recognition Using Machine Learning Algorithms. Indonesian Journal of Science and Technology. 18. 10.17509/ijost.v3i1.10795.

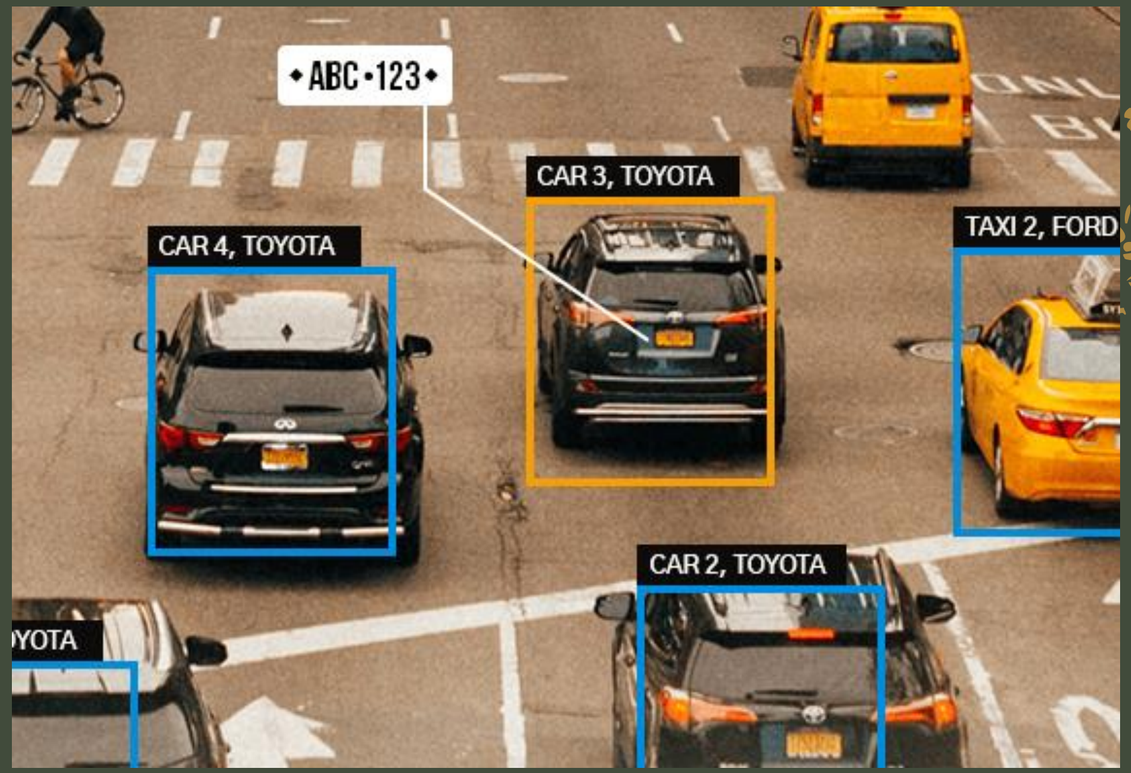
- Wells, Lee & Chen, Shengfeng & Al Mamlook, Rabia & Gu, Yuwen. (2018). Offline Handwritten Digits Recognition Using Machine learning.



Practical Applications

Handwritten digit recognition in handwriting has existed since the 1980s. The task of handwritten digit recognition using a classifier is very important and has many applications, including online handwriting recognition on computer tablets, sorting postal mail by zip code, processing bank check amounts, and numeric entries in forms filled out by hand (such as tax forms).

Offline handwriting recognition has a wide range of uses, including reading postal addresses, bank check amounts, and forms. Additionally, OCR is crucial for digital libraries because it enables the entry of image textual data into computers using methods for digitization, picture



Importing required libraries

```
import numpy as np #NumPy is a Python library used for working with arrays and matrices
import pandas as pd # read and writing data files
import matplotlib.pyplot as plt #Used for ploting graphs
import os,cv2
import joblib #creating and loading files/pipeling
from sklearn.model_selection import train_test_split    #train
from sklearn import metrics
from sklearn.ensemble import RandomForestClassifier
%matplotlib inline
```

[4] ✓ 1.5s

IMPORTING LIBRARIES

Loading Data

Importing kaggle dataset from downloaded repository

```
train = pd.read_csv('digit-recognizer-kaggledataset/train.csv') #data
test = pd.read_csv('digit-recognizer-kaggledataset/test.csv')
```

[5] ✓ 2.6s

Reading files

```
train
```

✓ 0.4s

	label	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	...	pixel774	pixel775	pixel776	pixel777	pixel778	pix
0	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	
2	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	
3	4	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
41995	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	
41996	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	
41997	7	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	
41998	6	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	
41999	9	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	

42000 rows × 785 columns

Splitting the training and testing data(65:) and storing it.
Displaying head of the stored data and the info

```
x = train.drop('label',axis=1)        #**x_train**: uint8 NumPy array of grayscale image data with shapes, containing the training data. Pixel values range fro
y = train['label'].values             #**y_train**: uint8 NumPy array of digit labels (integers in range 0-9) for the training data.

x_t = test.values                        #**x_test**: uint8 NumPy array of grayscale image data with shapes, containing the test data.

x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.30) # 30 % testing of train.csv | 70 % training

print("x_train => {0}\ny_train => {1}\nx_test => {2}".format(x_train.shape, y_train.shape,x_test.shape))
x_train.head()
```

✓ 0.4s Python

```
x_train => (29400, 784)
y_train => (29400,)
x_test => (12600, 784)
```

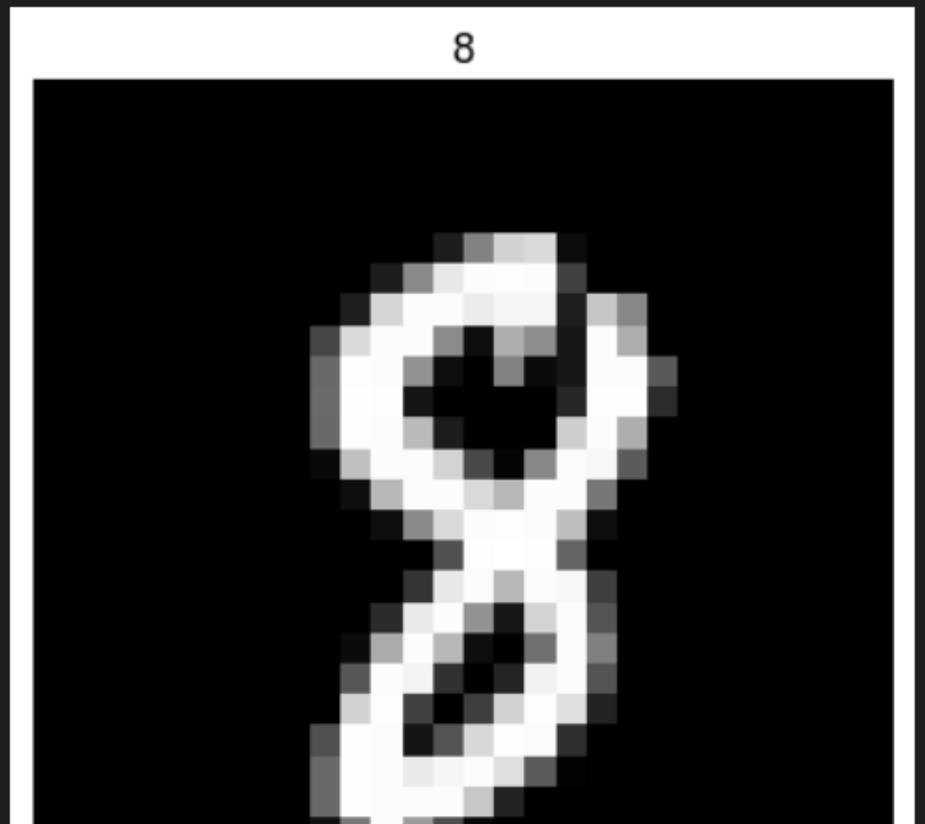
	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	...	pixel774	pixel775	pixel776	pixel777	pixel778	pixel779	pixel780	pixel781	pixe
16100	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	
7563	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	
21061	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	
14829	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	
18123	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	

5 rows × 784 columns

Data Visualization

```python
def display_img(i):
    img = x_train.iloc[i].values.reshape((28,28)) #train.iloc[:,1:].iloc[i].values.reshape(28,28)
    plt.imshow(#x_train[i],
               img,
               cmap = 'gray') #Greys
    plt.title(y_train[i])   #prints number shown in the image
    plt.axis('off')

display_img(5) #diplays the image at index i in train.csv
```

✓ 0.1s

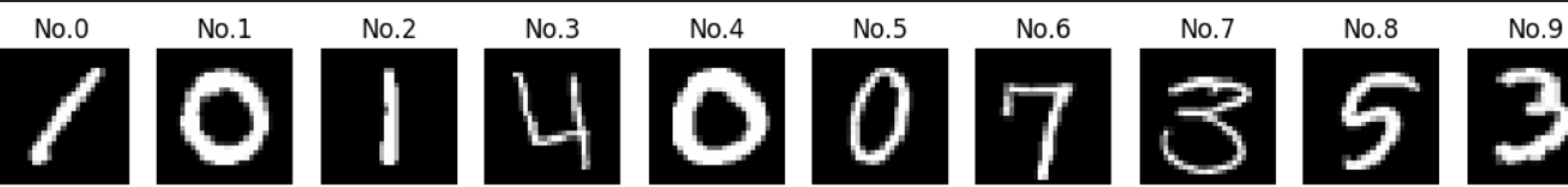


Data Visualization

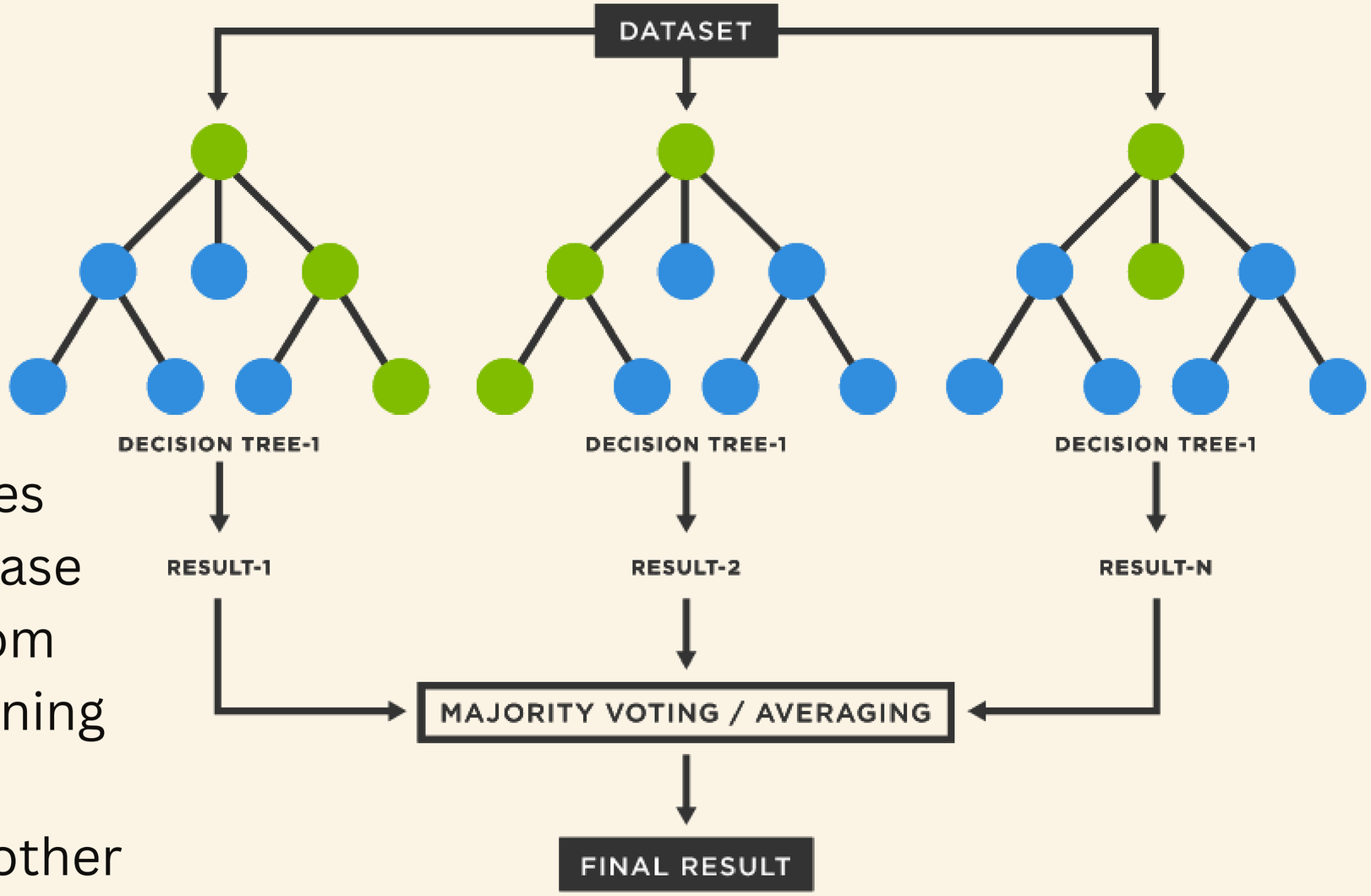
Displaying the first 10 indices

```
plt.figure(figsize=(28,28))
for i in range(10):
    plt.subplot(20, 20, i+1)
    plt.title("No." + str(i))
    plt.axis('off')
    plt.imshow(train.iloc[:,1:].iloc[i].values.reshape(28,28),cmap='gray')
```

✓ 0.4s



Random Forest



A large number of decision trees are built during the training phase of the random forests or random decision forests ensemble learning approach, which is used for classification, regression, and other tasks.

CREATING THE MODEL & LOADING THE MODEL

Creating Model

```
classifier = RandomForestClassifier()
classifier.fit(x_train, y_train)
prediction = classifier.predict(x_test)
joblib.dump(classifier, "models/rfc_model")
```

[7] ✓ 15.8s

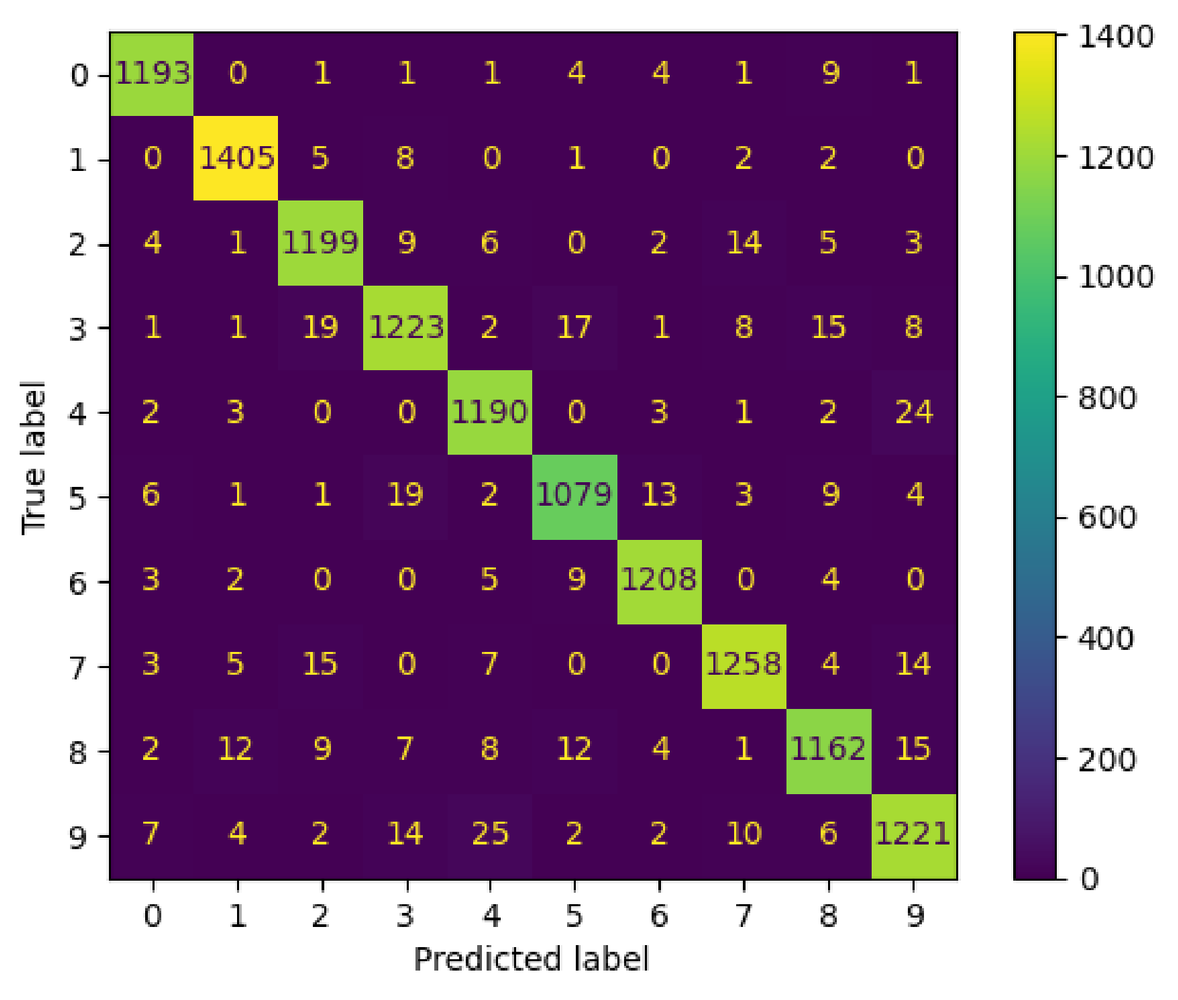
```
['models/rfc_model']
```

Loading model

```
model_load = joblib.load("models/rfc_model")
prediction=model_load.predict(x_test)
```

[8] ✓ 0.6s

Confusion Matrix



The confusion matrix show the accuracy and performance of the model

RACY AND SION

Accuracy measuring

```
print(metrics.classification_report(y_true=y_test, y_pred=prediction))
print(f"Accuracy = {metrics.accuracy_score(prediction, y_test)*100}")
```

[9] ✓ 0.6s

	precision	recall	f1-score	support
0	0.98	0.98	0.98	1472
1	0.98	0.98	0.98	1639
2	0.95	0.95	0.95	1473
3	0.95	0.93	0.94	1502
4	0.95	0.97	0.96	1467
5	0.96	0.96	0.96	1319
6	0.97	0.98	0.97	1397
7	0.97	0.96	0.96	1534
8	0.95	0.94	0.95	1445
9	0.94	0.93	0.94	1452
accuracy			0.96	14700
macro avg	0.96	0.96	0.96	14700
weighted avg	0.96	0.96	0.96	14700

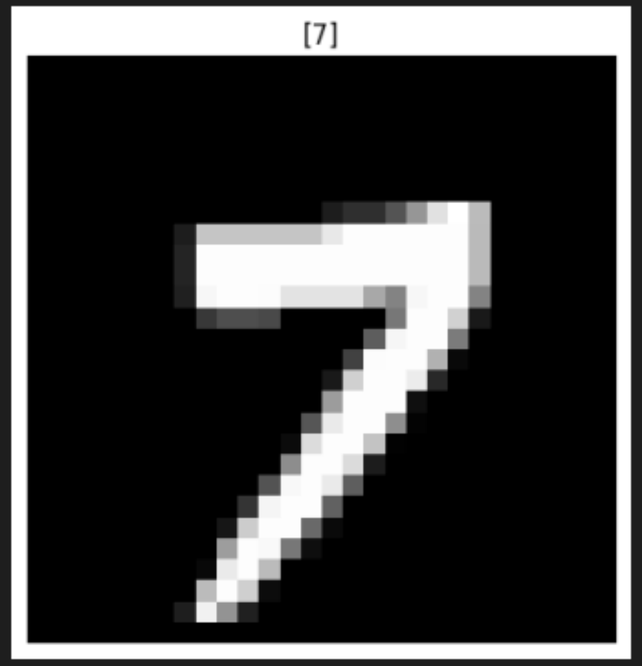
```
Accuracy = 95.97959183673468
```

Random image from the test data set is drawn and passed through the model

```
from random import randrange
img = randrange(28001)

prediction_sample = model_load.predict(x_t[[img]])

print("The answer is",int(prediction_sample))

plt.imshow(x_t[[img]].reshape(28,28),cmap='gray')
plt.title(prediction_sample)
plt.axis('off')
```

10] ✓ 0.1s

```
The answer is 7

c:\Users\Vanish\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\base.py:450: UserWarning: X does not have valid feature names, but RandomForestClassifier was fitted wi
  warnings.warn(
c:\Users\Vanish\AppData\Local\Programs\Python\Python310\lib\site-packages\matplotlib\text.py:1241: FutureWarning: elementwise comparison failed; returning scalar instead, but in the fut
  if s != self._text:

(-0.5, 27.5, 27.5, -0.5)
```



Random image from the test data set

CONCLUSION

Handwritten digit recognition plays an important part in processing bank cheque,zip codes on mails,goverment forms,postal letters ,vehicle plate detection etc.As such i have made a basic model using random forest classifier achieving an accuracy of 96%.