HANDWRITTEN DIGIT RECOGNITION

(REPORT)

- ## INTRODUCTION

  The handwritten digit recognition is the ability of computers to recognize human handwritten digits. It is a hard task for the machine because handwritten digits are not perfect and can be made in many forms. The handwritten digit recognition is the solution to this problem which uses the image of a digit/real-time input and recognizes the digit present in the image.

- ## Prerequisites

  This Python project is built using the Google Colaboratory. Colaboratory (or "Colab" for short) allows anybody to write and execute arbitrary python code through the browser, and is especially well suited to machine learning, data analysis and education. More technically, Colab is a hosted Jupyter notebook service that requires no setup to use, while providing free access to computing resources including GPUs.

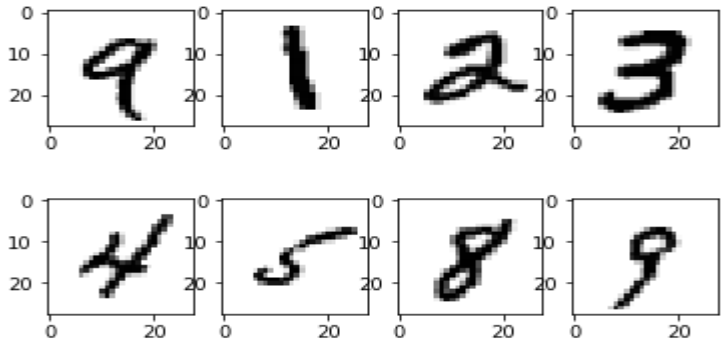
- ## MNIST Dataset

  The MNIST dataset was developed by Yann LeCun, Corinna Cortes and Christopher Burges for evaluating machine learning models on the handwritten digit classification problem.
  The dataset was constructed from a number of scanned document dataset available from the National Institute of Standards and Technology (NIST). This is where the name for the dataset comes from, as the Modified NIST or MNIST dataset.

  Images of digits were taken from a variety of scanned documents, normalized in size and centred. Each image is a 28 by 28 pixel square (784 pixels total). A standard split of the dataset is used to

evaluate and compare models, where 60,000 images are used to train a model and a separate set of 10,000 images are used to test it. It is a digit recognition task. As such there are 10 digits (0 to 9) or 10 classes to predict.

These are some sample images of the handwritten character from MNIST dataset.



- # IMPLEMENTATION

1. Import the libraries and load the dataset

First, we are going to import all the modules that we are going to need for training our model. The Keras library already contains some datasets and MNIST is one of them. So we can easily import the dataset and start working with it. The mnist.load_data() method returns us the training data, its labels and also the testing data and its labels.

```
import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Conv2D, Dropout, Flatten, MaxPooling2D
from keras import backend as K
import matplotlib.pyplot as plt

(x_train, y_train), (x_test, y_test) = mnist.load_data()

print(x_train.shape, y_train.shape)
```

```
(60000, 28, 28) (60000,)
```

2. Pre-process the data

The image data cannot be fed directly into the model so we need to perform some operations and process the data to make it ready for our neural network. The dimension of the training data is (60000, 28, 28). The CNN model will require one more dimension so we reshape the matrix to shape (60000, 28, 28, 1).

```
#Reshaping the array to 4D
x_train = x_train.reshape(x_train.shape[0], 28, 28, 1)
x_test = x_test.reshape(x_test.shape[0], 28, 28, 1)
input_shape = (28, 28, 1)
#Normalizing the RGB codes by dividing it to the max RGB value.
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255
print('x_train shape:', x_train.shape)
print('Number of images in x_train = ', x_train.shape[0])
print('Number of images in x_test = ', x_test.shape[0])
```

```
x_train shape: (60000, 28, 28, 1)
Number of images in x_train =  60000
Number of images in x_test =  10000
```

3. Create the model

Now we will create our CNN model in Python data science project. A CNN model generally consists of convolutional and pooling layers. It works better for data that are represented as grid structures, this is the reason why CNN works well for image classification problems. The dropout layer is used to deactivate some of the neurons and while training, it reduces offer fitting of the model. We will then compile the model with the SGD optimizer.

```
model = Sequential()
model.add(Conv2D(32, kernel_size=(3,3), activation='relu', input_shape=input_shape))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(num_classes, activation='softmax'))
model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.SGD(),metrics=['accuracy'])
```

4. Train & Save the model

The model.fit() function of Keras will start the training of the model. It takes the training data, validation data, epochs, and batch size.

```
hist = model.fit(x=x_train, y=y_train, batch_size=128, epochs=20,
                 validation_data=(x_test, y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 26s 436us/step - loss: 0.1628 - accuracy: 0.9519
Epoch 2/20
60000/60000 [==============================] - 26s 434us/step - loss: 0.1561 - accuracy: 0.9546
Epoch 3/20
60000/60000 [==============================] - 26s 434us/step - loss: 0.1502 - accuracy: 0.9553
Epoch 4/20
60000/60000 [==============================] - 26s 437us/step - loss: 0.1424 - accuracy: 0.9581
Epoch 5/20
60000/60000 [==============================] - 26s 434us/step - loss: 0.1374 - accuracy: 0.9597
Epoch 6/20
60000/60000 [==============================] - 26s 439us/step - loss: 0.1341 - accuracy: 0.9603
Epoch 7/20
60000/60000 [==============================] - 26s 435us/step - loss: 0.1296 - accuracy: 0.9616
Epoch 8/20
60000/60000 [==============================] - 26s 437us/step - loss: 0.1244 - accuracy: 0.9632
```

It takes some time to train the model. After training, we save the weights and model definition in the 'predict.model' file.

```
#Saving model
model.save('predict.model')
print('MODEL SAVED SUCESSFULLY')
```

```
MODEL SAVED SUCESSFULLY
```

5. Evaluate the model

We have 10,000 images in our dataset which will be used to evaluate how good our model works. The testing data was not involved in the training of the data therefore, it is new data for our model. The MNIST dataset is well balanced so we can get around **97.5%** accuracy.

```
#Checking model
model.evaluate(x_test, y_test)
model_score = model.evaluate(x_test, y_test)
print('Test loss:', model_score[0])
print('Test accuracy:', model_score[1])
```

```
10000/10000 [==============================] - 2s 185us/step
10000/10000 [==============================] - 2s 186us/step
Test loss: 0.0799612998213619
Test accuracy: 0.9749000072479248
```

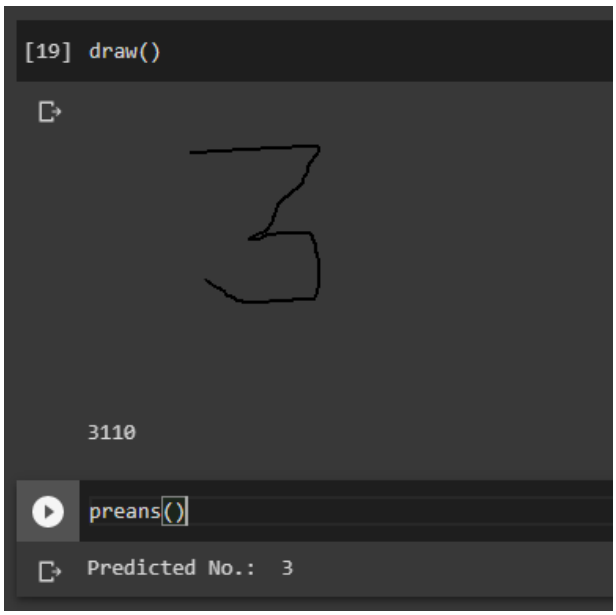
6. Create Script to predict the digits

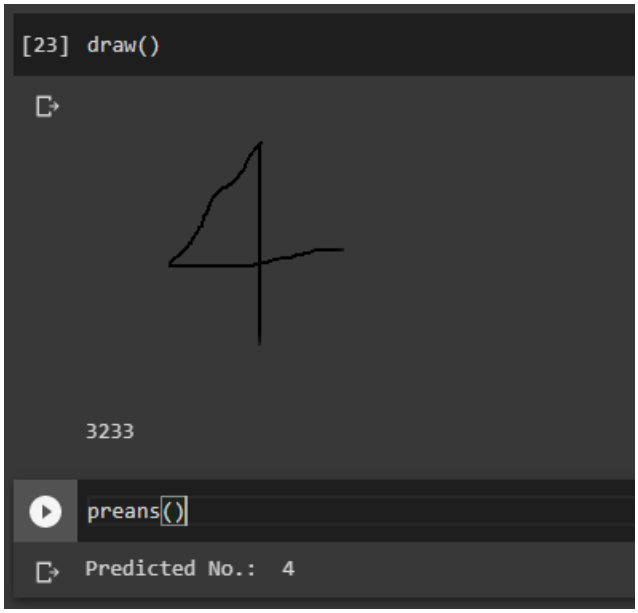
To provide freehand drawing support in the Google Colab, draw() function from draw.py by Korakot Chaovavanich is used.
(Ref. https://gist.github.com/korakot/8409b3feec20f159d8a50b0a811d3bca)

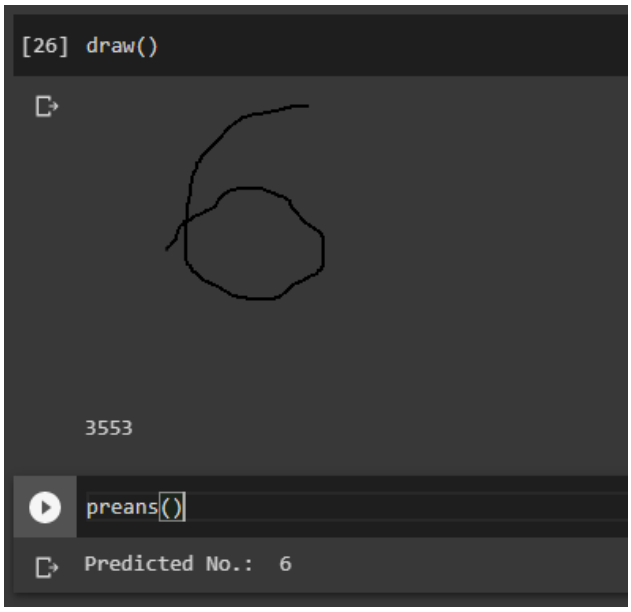
Then we have created a function preans() that takes the image as input and then uses the trained model to predict the digit.

```
import numpy as np
import cv2
def preans():
    image = cv2.imread('num.png', cv2.IMREAD_UNCHANGED)
    trans_mask = image[:,:,3] == 0
    image[trans_mask] = [255, 255, 255, 255]
    new_img = cv2.cvtColor(image, cv2.COLOR_BGRA2BGR)
    grey = cv2.cvtColor(image.copy(), cv2.COLOR_BGR2GRAY)
    ret, thresh = cv2.threshold(grey.copy(), 75, 255, cv2.THRESH_BINARY_INV)
    contours, _ = cv2.findContours(thresh, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
    cv2.drawContours(image, contours, -1, (0, 255, 0), 3)
    for c in contours:
        x,y,w,h = cv2.boundingRect(c)
        # Creating a rectangle around the digit in the original image
        # (for displaying the digits fetched via contours)
        cv2.rectangle(image, (x,y), (x+w, y+h), color=(0, 255, 0), thickness=2)
        # Cropping out the digit from the image corresponding to the current
        # contours in the for loop
        digit = thresh[y:y+h, x:x+w]
        # Resizing that digit to (18, 18)
        resized_digit = cv2.resize(digit, (18,18))
        # Padding the digit with 5 pixels of black color (zeros)
        # in each side to finally produce the image of (28, 28)
        padded_digit = np.pad(resized_digit, ((5,5),(5,5)), "constant", constant_values=0)
        # Adding the preprocessed digit to the list of preprocessed digits
        ppd = (padded_digit)
    img = ppd.reshape(1, 28, 28, 1)
    img = img
    # Predicting the digit
    result = model.predict([img])[0]
    print('Predicted No.: ',np.argmax(result))
```

- **SCREENSHOTS**







****