Handwritten Digit Recognition using Python

RA1911003020147 KARTHICK RAJA

RA1911003020148 GOKUL

RA1911003020143 HARISH

Introduction

In this article, we are going to use the MNIST dataset for the implementation of a handwritten digit recognition app. To implement this, we will use a special type of deep neural network called *Convolutional Neural Networks*. In the end, we will also build a Graphical user interface (GUI) where you can directly draw the digit and recognize it straight away. Handwritten digit recognition is the process to provide the ability to machines to recognize human handwritten digits. It is not an easy task for the machine because handwritten digits are not perfect, vary from person-to-person, and can be made with many different flavors.

Prerequisites

Basic knowledge of deep learning with Keras library, the Tkinter library for GUI building, and Python programming are required to run this amazing project.

Commands to Install the necessary libraries for this project:

```
pip install numpy
pip install tensorflow
pip install keras
pip install pillow
```

The MNIST dataset

Among thousands of datasets available in the market, MNIST is the most popular dataset for enthusiasts of machine learning and deep learning. Above

60,000 plus training images of handwritten digits from zero to nine and more than 10,000 images for testing are present in the MNIST dataset. So, 10 different classes are in the MNIST dataset. The images of handwritten digits are shown as a matrix of 28×28 where every cell consists of a grayscale pixel value.

There are 4 files:

```
train-images-idx3-ubyte:  training set images
train-labels-idx1-ubyte:  training set labels
t10k-images-idx3-ubyte:   test set images
t10k-labels-idx1-ubyte:   test set labels
```

The training set contains 60000 examples, and the test set 10000 examples.

The first 5000 examples of the test set are taken from the original NIST training set. The last 5000 are taken from the original NIST test set. The first 5000 are cleaner and easier than the last 5000.

TRAINING SET LABEL FILE (train-labels-idx1-ubyte):

```
[offset] [type]          [value]          [description]
0000     32 bit integer  0x00000801(2049) magic number (MSB first)
0004     32 bit integer  60000            number of items
0008     unsigned byte   ??               label
0009     unsigned byte   ??               label
........
xxxx     unsigned byte   ??               label
```

The labels values are 0 to 9.

TRAINING SET IMAGE FILE (train-images-idx3-ubyte):

```
[offset] [type]          [value]          [description]
0000     32 bit integer  0x00000803(2051) magic number
0004     32 bit integer  60000            number of images
0008     32 bit integer  28               number of rows
0012     32 bit integer  28               number of columns
0016     unsigned byte   ??               pixel
0017     unsigned byte   ??               pixel
........
xxxx     unsigned byte   ??               pixel
```

Pixels are organized row-wise. Pixel values are 0 to 255. 0 means background (white), 255 means foreground (black).

Implementation

Import libraries and dataset

At the project beginning, we import all the needed modules for training our model. We can easily import the dataset and start working on that because the Keras library already contains many datasets and MNIST is one of them. We call mnist.load_data() function to get training data with its labels and also the testing data with its labels.

```
import keras
from keras.datasets import mnist
```

```
from keras.models import Sequential
from keras.layers import Dense Flatten
from keras.layers import Dropout
from keras.layers import Flatten
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras import backend as K
#  to split the data of training and testing sets
(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

The Data Preprocessing

Model cannot take the image data directly so we need to perform some basic operations and process the data to make it ready for our neural network. The dimension of the training data is (60000*28*28). One more dimension is needed for the CNN model so we reshape the matrix to shape (60000*28*28*1).

```
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)
# conversion of class vectors to matrices of  binary class
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255
```

Model Creation

```
batch_size = 128

num_classes = 10

epochs = 10

model = Sequential()

model.add(Conv2D(32, kernel_size=(3,
3),activation='relu',input_shape=input_shape))

model.add(Conv2D(64, (3, 3), activation='relu'))

model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Dropout(0.25))

model.add(Flatten())

model.add(Dense(256, activation='relu'))

model.add(Dropout(0.5))

model.add(Dense(num_classes, activation='softmax'))

model.compile(loss=keras.losses.categorical_crossentropy,optimizer=ker
as.optimizers.Adadelta(),metrics=['accuracy'])
```

Training the model

To start the training of the model we can simply call the model.fit() function of Keras. It takes the training data, validation data, epochs, and batch size as the parameter.

The training of model takes some time. After succesful model training, we can save the weights and model definition in the ‘mnist.h5’ file.

```
hist = model.fit(x_train,
y_train,batch_size=batch_size,epochs=epochs,verbose=1,validation_data=
(x_test, y_test))
```

```
print("The model has successfully trained")
model.save('mnist.h5')
print("Saving the bot as mnist.h5")
```

The model has successfully trained

Saving the bot as mnist.h5

Evaluate the model

To evaluate how accurate our model works, we have around 10,000 images in our dataset. In the training of the data model, we do not include the testing data that's why it is new data for our model. Around 99% accuracy is achieved with this well-balanced MNIST dataset.

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

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/10
60000/60000 [==============================] - 7s 120us/step - loss: 0.2495 - acc:
Epoch 2/10
60000/60000 [==============================] - 7s 117us/step - loss: 0.0763 - acc:
Epoch 3/10
60000/60000 [==============================] - 7s 119us/step - loss: 0.0548 - acc:
Epoch 4/10
60000/60000 [==============================] - 7s 116us/step - loss: 0.0424 - acc:
Epoch 5/10
60000/60000 [==============================] - 7s 116us/step - loss: 0.0350 - acc:
Epoch 6/10
60000/60000 [==============================] - 7s 117us/step - loss: 0.0274 - acc:
Epoch 7/10
60000/60000 [==============================] - 7s 117us/step - loss: 0.0238 - acc:
Epoch 8/10
60000/60000 [==============================] - 7s 117us/step - loss: 0.0209 - acc:
Epoch 9/10
60000/60000 [==============================] - 7s 117us/step - loss: 0.0179 - acc:
Epoch 10/10
60000/60000 [==============================] - 7s 119us/step - loss: 0.0137 - acc:
CNN Error: 0.95%
```

Create GUI to predict digits

The Tkinter library is the part of Python standard library. Our predict_digit() method takes the picture as input and then activates the trained model to predict the digit.

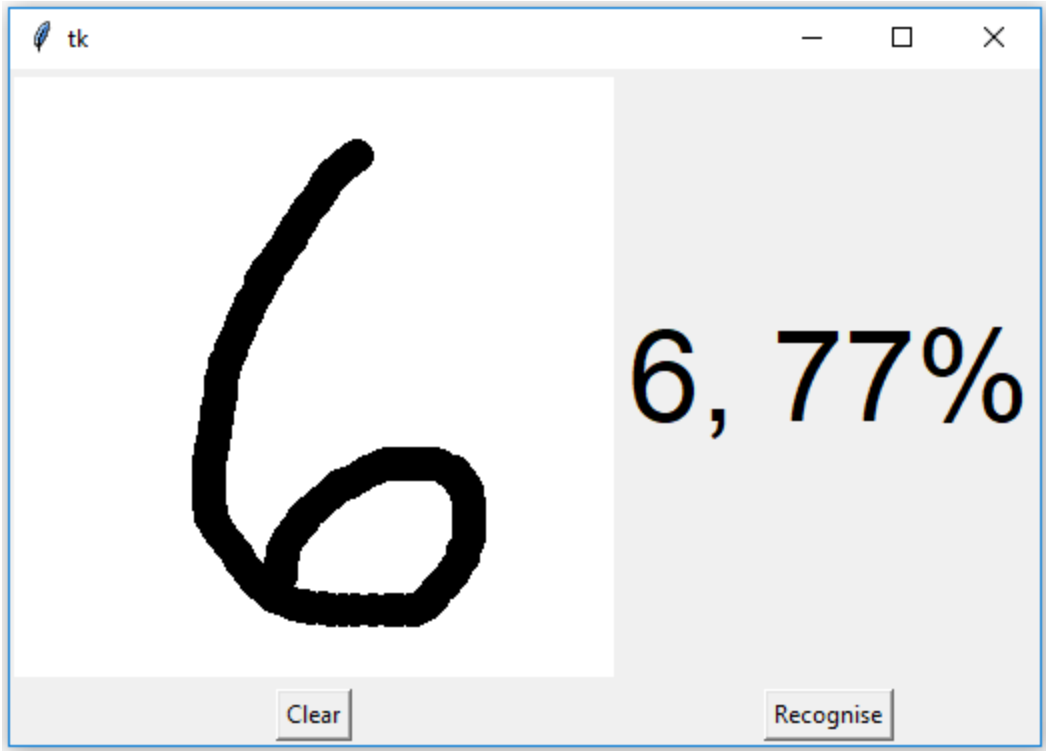
After that to build the GUI for our app we have created the App class. In GUI canvas you can draw a digit by capturing the mouse event and with a button click, we hit the predict_digit() function and show the results.

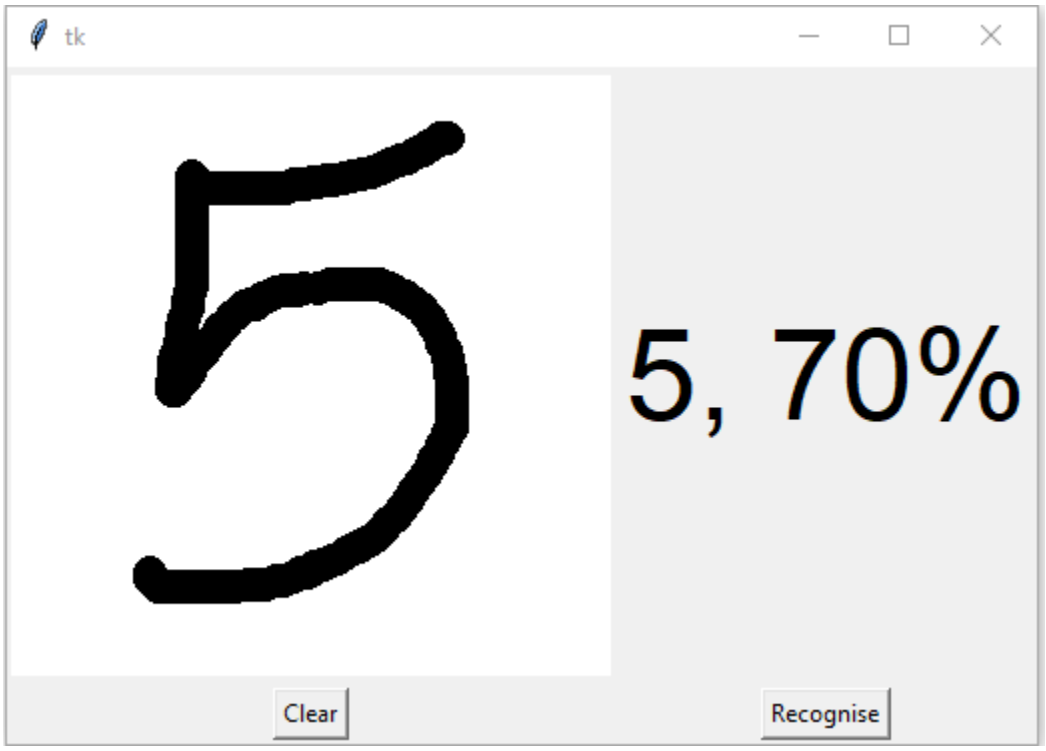
```
from keras.models import load_model
from Tkinter import *
import Tkinter successful  as tk
import win32gui
from PIL import ImageGrab, Image
import numpy as np
model = load_model('mnist.h5')
def predict_digit(img):
    #resize image to 28x28 pixels
    img = img.resize((28,28))
    #convert rgb to grayscale
    img = img.convert('L')
    img = np.array(img)
    #reshaping for model normalization
    img = img.reshape(1,28,28,1)
    img = img/255.0
    #predicting the class
    res = model.predict([img])[0]
    return np.argmax(res), max(res)
class App(tk.Tk):
```

```
    def __init__(self):
        tk.Tk.__init__(self)
        self.x = self.y = 0
        # Creating elements
        self.canvas = tk.Canvas(self, width=200, height=200, bg =
"black", cursor="cross")
        self.label = tk.Label(self, text="Analyzing..",
font=("Helvetica", 48))
        self.classify_btn = tk.Button(self, text = "Searched", command
=         self.classify_handwriting)
        self.button_clear = tk.Button(self, text = "Dlt", command =
self.clear_all)
        # Grid structure
        self.canvas.grid(row=0, column=0, pady=2, sticky=W, )
        self.label.grid(row=0, column=1,pady=2, padx=2)
        self.classify_btn.grid(row=1, column=1, pady=2, padx=2)
        self.button_clear.grid(row=1, column=0, pady=2)
        #self.canvas.bind("", self.start_pos)
        self.canvas.bind("", self.draw_lines)
    def clear_all(self):
        self.canvas.delete("all")
    def classify_handwriting(self):
        Hd = self.canvas.winfo_id() # to fetch the handle of the
canvas
        rect = win32gui.GetWindowRect(Hd) # to fetch the edges of the
canvas
        im = ImageGrab.grab(rect)
        digit, acc = predict_digit(im)
        self.label.configure(text= str(digit)+', '+
str(int(acc*100))+'%')
    def draw_lines(slf, event):
        slf.x = event.x
```

```
        slf.y = event.y
        r=8
        slf.canvas.create_oval(slf.x-r, slf.y-r, slf.x + r, slf.y + r,
fill='black')
app = App()
mainloop()
```

Output





Conclusion

We have created and deployed a successful deep learning project of digit recognition. We build the GUI for easy learning where we draw a digit on the canvas then we classify the digit and show the results.