HANDWRITTEN CHARACTER RECOGNITION USING PYTHON

Submitted by

AFNAN NAQSHBANDI

1. INTRODUCTION

1.1 Project Overview

Handwriting recognition has been one of the most fascinating and challenging research areas in field of image processing and pattern recognition in the recent years. It contributes immensely to the advancement of automation process and improves the interface between man and machine in numerous applications. The development of handwriting recognition systems began in the 1950s when there were human operators whose job was to convert data from various documents into electronic format, making the process quite long and often affected by errors. Automatic text recognition aims at limiting these errors by using image pre-processing techniques that bring increased speed and precision to the entire recognition process.

1.2 Problem Statement

Given a handwritten character, the system needs to predict the type of the character. In other words if we can write the character “A” the system predict the character that it is truly “A” or the input character is nearer to “A” or something else. The purpose of this project is to take handwritten English characters as input, process the character, train the neural network algorithm, to recognize the pattern and modify the character to a beautified version of the input. This project is aimed at developing software which will be helpful in recognizing characters of English language. This project is restricted to English characters only. It can be further developed to recognize the characters of different languages. It engulfs the concept of neural network. One of the primary means by which computers are endowed with humanlike abilities is through the use of a neural network. Neural networks are particularly useful for solving problems that cannot be expressed as a series of steps, such as recognizing patterns, classifying them into groups, series prediction and data mining. Pattern recognition is perhaps the most common use of neural networks. The neural network is presented with a target vector and also a vector which contains the pattern information, this could be an image and hand written data. The neural network then attempts to determine if the input data matches a pattern that the neural network has memorized. A neural network trained for classification is designed to take input samples and classify them into groups. These groups may be fuzzy, without clearly defined boundaries. This project concerns detecting free handwritten characters.

2. Dataset used

https://www.kaggle.com/sachinpatel21/az-handwritten-alphabets-in-csv-format which contains 26 folders (A-Z) containing handwritten images in size 28*28* pixels, each alphabet in the image is centre fitted to 20*20* pixel box.

3. Libraries used

Numpy, openCV, Keras, Tensorflow (Keras uses TensorFlow in backend and for some image preprocessing) , Matplotlib and Pandas were used to achieve the goal of handwritten character recognition.

4. Source Code

```
[ ]  from keras.datasets import mnist
     import matplotlib.pyplot as plt
     import cv2
     import numpy as np
     from keras.models import Sequential
     from keras.layers import Dense, Flatten, Conv2D, MaxPool2D, Dropout
     from keras.optimizers import SGD, Adam
     from keras.callbacks import ReduceLROnPlateau, EarlyStopping
     from keras.utils import to_categorical
     import pandas as pd
     import numpy as np
     from sklearn.model_selection import train_test_split
     from keras.utils import np_utils
     import matplotlib.pyplot as plt
     from tqdm import tqdm_notebook
     from sklearn.utils import shuffle
```

```
train_x, test_x, train_y, test_y = train_test_split(X, y, test_size = 0.2)
train_x = np.reshape(train_x.values, (train_x.shape[0], 28,28))
test_x = np.reshape(test_x.values, (test_x.shape[0], 28,28))
print("Train data shape: ", train_x.shape)
print("Test data shape: ", test_x.shape)
```

```
Train data shape:  (75637, 28, 28)
Test data shape:  (18910, 28, 28)
```

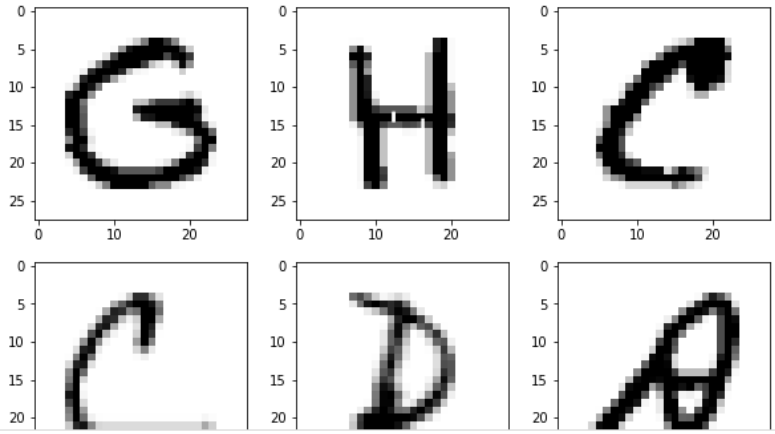
```
[ ]
word_dict = {0:'A',1:'B',2:'C',3:'D',4:'E',5:'F',6:'G',7:'H',8:'I',9:'J',10:'K',11:'L',12:'M',13:'N',14:'O',15:'P',16:'Q',
```

```
[ ]  #Shuffling the data ...
     shuff = shuffle(train_x[:100])

     fig, ax = plt.subplots(3,3, figsize = (10,10))
     axes = ax.flatten()

     for i in range(9):

         axes[i].imshow(np.reshape(shuff[i], (28,28)), cmap="Greys")
     plt.show()
```



```
model = Sequential()

model.add(Conv2D(filters=32, kernel_size=(3, 3), activation='relu', input_shape=(28,28,1)))
model.add(MaxPool2D(pool_size=(2, 2), strides=2))

model.add(Conv2D(filters=64, kernel_size=(3, 3), activation='relu', padding = 'same'))
model.add(MaxPool2D(pool_size=(2, 2), strides=2))

model.add(Conv2D(filters=128, kernel_size=(3, 3), activation='relu', padding = 'valid'))
model.add(MaxPool2D(pool_size=(2, 2), strides=2))

model.add(Flatten())

model.add(Dense(64,activation ="relu"))
model.add(Dense(128,activation ="relu"))

model.add(Dense(26,activation ="softmax"))
```

```
[ ]  #Making model predictions...

     pred = model.predict(test_X[:9])
     print(test_X.shape)
```

```
[ ]  fig, axes = plt.subplots(8,8, figsize=(15,18))
     axes = axes.flatten()

     for i,ax in enumerate(axes):
         img = np.reshape(test_X[i], (28,28))
         ax.imshow(img, cmap="Greys")
         pred = word_dict[np.argmax(test_yOHE[i])]
         ax.set_title("Prediction: "+pred)
         ax.grid()
```

5. PREDICTIONS