Everyone can use CRNN to recognize text images

Mingwei Li*

Abstract

With the increase of data and computation power, deep learning is widely adopted in multiple fields. Text images recognition is a classical and valuable task, and deep neural network is able to provide us with satisfying results. The following 4 pictures demonstrate the performance of our neural network. Our model supports **both** Chinese and English.

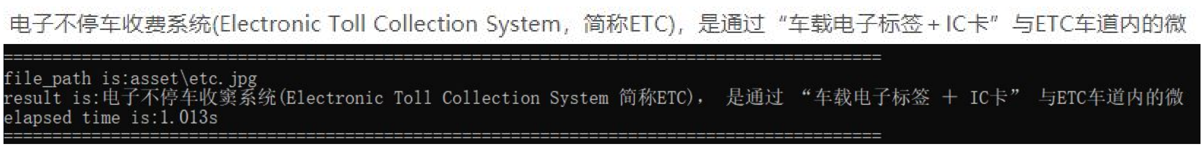

Figure 1: The Performance on Mixed-Language Text Images

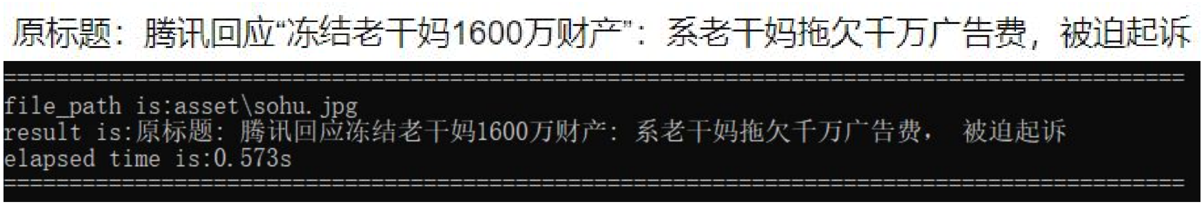

Figure 2: The Performance on Chinese Text Images

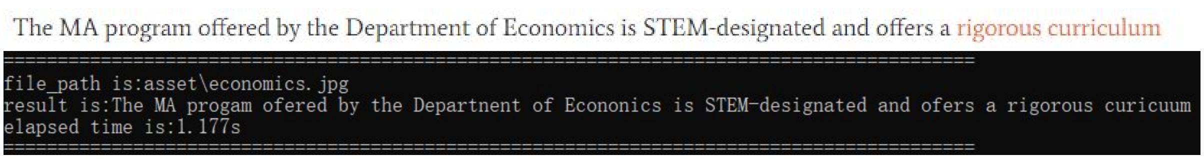

Figure 3: The Performance on English Text Images(1)

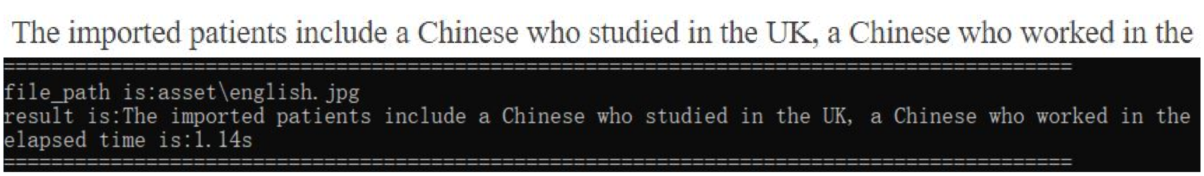

Figure 4: The Performance on English Text Images(2)

Just as the title indicates, this essay aims to tell you how to build an OCR model from nothing. We hope that even new beginner in python can benefit from this work.

The whole passage has 3 chapters. Chapter 1 is the user guidance of the project. It tells you how to use this project to recognize your own images. Chapter 2 is the technical guidance of the project. All technical details to build the model are involved in this chapter. Chapter 3 is an introduction to the principles of CRNN.

The source code of this essay is available at here. Please star my project if you feel it's helpful, thanks!!

*Mingwei Li is a master student of economics, The University of Texas at Austin

Contents

1 User Guidance

The project is adequate to recognize your own images as the model weight is preserved in our github repository. You need to download train data if you want train your own CRNN model.

1.1 File Paths

At first, you need to save all your images under **asset/** directory. **Please note!** This model could only recognize single line of text. Because of the parameter sharing mechanism of RNN and CNN, our model is able to recognize single text lines of flexible length. Specifically, you can use this model to recognize a sentence consisted of more than 1000 words. However, the input image should have only **one** text line!

Figure 5: Asset Images

Secondly, you should execute order line **python demo_infer.py**, and the recognition results of all images under **asset/** directory would be presented as followed.

Figure 6: Recognition Results

The model weight of this project is saved as **weights/CRNN.h5**.

1.2 Infer Details

(1) Image Processing: At first, all images are converted to gray-scale images. Then, they are resized to make sure the height of input images is exactly 32 pixels. The corresponding function is at here.

(2) Basic Division: The greatest weakness of CRNN is that such neural network can't recognize the space between words naturally. Moreover, the white space between different words may harm the performance of CRNN. Therefore, it's necessary to conduct some basic division. The corresponding function is at here. The input image is binarized twice. Cut lines would be added to spaces which have more than 5 consecutive pure black pixels. Figure

7 demonstrates the performance of division. **Please note!** CRNN doesn't need to divide a sentence into single characters. However, huge space between different words should be deleted.

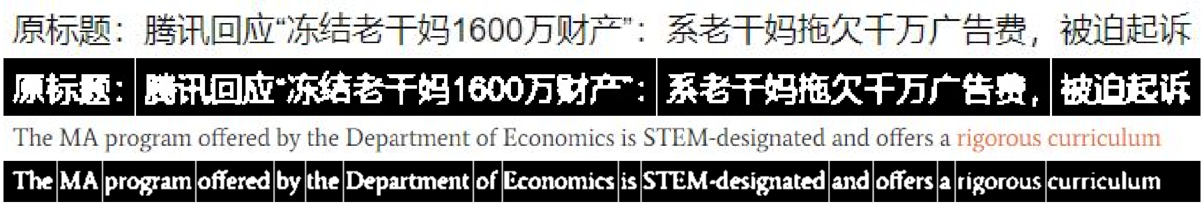

Figure 7: Division Performance

(3) Recognize small pieces: The whole image is divided into small pieces after the previous 2 steps. Then, those small pieces are sent to neural network. Background color is added to the right side of each single piece, and the corresponding function is at here. The left sub-figure of figure 8 is the original piece, and the right sub-figure is the piece after color adding. According to our experiment, color adding is able to increase recognition performance slightly. The reason of this phenomenon may be the fact that some training images have blank space on their right sides.

Figure 8: The Performance of color adding

Figure 9 is a part of training data. In fact, most of the training data is Chinese rather than English. Therefore, our model performs better on Chinese recognition than English. More English data is needed to obtain a more robust model.

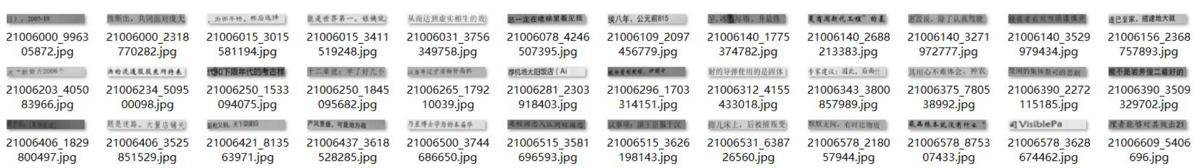

Figure 9: Part of the Training Data

2 Technical Guidance

This chapter aims to direct you to build a CRNN model from data management to image inference. The contents of this chapter is carefully designed to make sure that even new beginners in python can follow

2.1 Data Management

More than 3.6 million images are involved in the process of model training. Therefore, some special methods are required to manage data efficiently. The whole data set is uploaded at here, and the download key is **thxb**. The data on baiducloud can be used to substitute the original **data/** directory of our project.

(1) Unpacking Data: All images are saved in 7z format as followed:

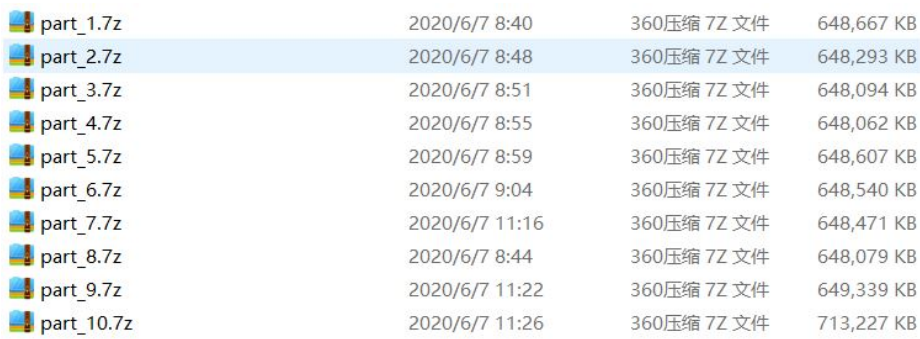

Figure 10: Training Data

Then, you should execute the order line **python unpacking_data.py**, and your program would execute 7z commands automatically to extract data. It may takes hours to finish this process. The whole training data is divided into 101 batches after finishing unpacking.

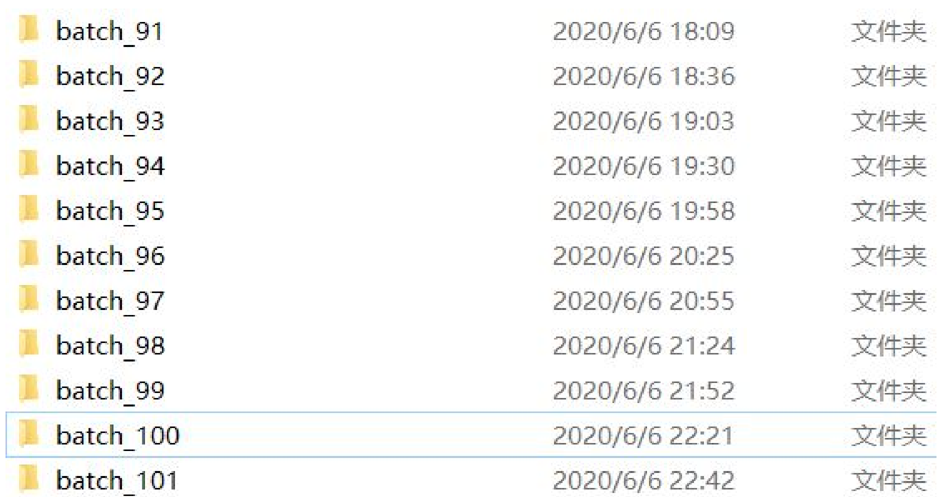

Figure 11: Training Data

If we gather the 3.6 million images into one directory, the data loading speed of computer memory would be extremely low. Therefore, we have to divide the whole data set into 101 batches.

(2) Loading Data: The core data loading module is at here. This class inherits from **tf.keras.utils.Sequence**. The **__len__** method returns the total number of steps. The **__getitem__** method returns a batch of data. Users only need to specify these 2 methods as **tf.keras.utils.Sequence** has incorporated other fundamental methods.

The following code is critical to our training process. It changes the background color of given text. 255 stands for white while 0 represents black. The ideal model should recognize both white text and black text. If the following code isn't involved, the train loss would decrease to less than 0.10 quickly. After the following code is incorporated in our program, the train loss stabilizes at 0.25. As a result, change the background color of images randomly may increase the robustness of our model.

```
for name in name_list:
    img = cv2.imread(name)
    if np.random.randint(0, 100) > 50:
        img = 255-img
```

Another important point is the use of **data generator**. The corresponding code is at here. In comparing with fitting after loading the data into memory, fit directly on data generator would double the training speed. **tensorflow** is able to synchronize reading data and training model effectively.

2.2 Net Structure

The whole structure of CRNN is at here. The input shape and output shape of every single module are specified in the annotations. The train model is as followed. There are four inputs and one output.

```
self.train_core = Model([vgg_input, labels, input_length, label_length], loss_out)
```

vgg_input: Gray-scale images whose shape is (batch_size,32,280,1).
labels: The ground-truth labels whose shape is (batch_size,10).
input_length: The input of CTC-loss function. Its shape is (batch_size,1).
label_length: The input of CTC-loss function. Its shape is (batch_size,1).
loss_out: Return of CTC-loss function.

The details of CTC-loss would be demonstrated in chapter 3. There are 2 critical aspects concerning network structure: 1.The height of input image ought to be restricted to 32 pixels. 2.The width of input image is flexible in both training and inferring process. However, in order to train in batches, we set all training images and labels to the same shape.

The most innovative point of CRNN is at here. The pooling window of (2,1) is able to capture text features effectively.

```
l2 = layers.MaxPooling2D(pool_size=(2,1), name='max3')(l2)
```

CRNN.predict_core is used for further inference on new images, and **CRNN.train_core** is used for fitting on training data. **_load_weights** method demonstrates the relationship between these 2 **cores**.

```
def _load_weights(self,path):
    self.train_core.load_weights(path)
    output = self.train_core.get_layer("blstm2_out").output
    img_input = self.train_core.input[0]
    self.predict_core = Model(img_input,output)
```

3 The Principle of CRNN

CRNN is a neural network consisted of CNN and RNN. The parameter sharing mechanism of CNN and RNN enables CRNN to recognize images of flexible width. The most important aspect of CRNN is the use of **CTC-loss**. Classical OCR has to divide a word into single characters before text recognition. With the help of **CTC-loss**, CRNN don't need to conduct characters dividing task, which is much more difficult than single character recognition.

CTC-loss is a special loss different from classical loss like mean squared error. However, just like other loss function, **CTC-loss** is also a measurement of the difference between **y_true** and **y_pred**. As for our OCR task, the definitions of **y_true** and **y_pred** are demonstrated below.

y_true: An array of 10 integers. Each integer represents a certain Chinese or English letter. For instance, 20455828_2605100732.jpg corresponds to [263,82,29,56,35,435,890,293,126,129]. Figure 12 is the Chinese text represented by the integer array.

Figure 12: 20455828_2605100732.jpg

y_pred: A tensor whose shape is (69,5991). Dimension one(69) stands for different positions in horizontal direction(width). Dimension two(5991) is the output of softmax layer, stands for the probability of 5991 characters on 69 different positions.

CTC-loss: For any given **y_pred**, we can decode **y_pred** to get the final prediction. For instance, *-s-t-aatte* and *-s-tta-tte* would both lead to *state*. We assume *state* as the ground truth, and multiple **y_pred** can be decoded as *state*. Since **y_pred** is the probability of different letters, we are able to calculate the following probability by greedy algorithm.

$$P_{ctc} = \sum_{Decode(y\_pred)==y\_true} P(y\_pred)$$

CTC-loss is defined as followed:

$$CTCloss = -log(P_{ctc})$$

During the decode process, the blank letter - is abandoned. Therefore, CRNN is not able to recognize the space between different words naturally.

4 Prospect

The training data of our project is mainly consisted of Chinese rather than English. Therefore, the CRNN model performs better on Chinese. More English images should be incorporated into our training data to make the model more robust.

In addition, as the structure of English letter is very different from Chinese letter, it's reasonable to train multiple models to deal with various languages.