Mobile Code Executer

Alok Kumar Rai, Deepmani Bhardwaj, Arpan Srivastava and Vinita*
Department of of Computer Science and Engineering
IMS Engineering College, Ghaziabad
Dr APJ Abdul Kalam Technical University, India
*vinita.chauhan@imsec.ac.in

Abstract

This paper introduces a system of processing handwritten text into compliable source code. In order to generate syntactically correct source code, near perfect accuracy is required. Especially for handwritten text, these accuracies are beyond the capabilities of current OCR engines. We overcome the limited accuracy of the Tesseract OCR engine by using an unambiguous font, preprocessing the input image before it is sent to Tesseract to remove dust and correct skew, modifying Tesseract's dictionary and increase dictionary strength, post-processing Tesseract's output, use feedback from the compiler to automatically make further corrections. The performance is tested on two different datasets, one consisting of samples collected from the known users (those who prepared the training data samples) and the other consisting of handwritten data samples of unknown users.

Keywords -Image Processing, Segmentation, Character Recognition, Linguistic Analyser, Matlab

Introduction

Mobile code executor is an android application that provides a solution for situation where an individual is asked to write a code on a piece of paper especially in interviews which enable the interviewees to see the general logic of the code but cannot actually execute it and proof of its correctness. It allows the user to take a photo of the code by android phone. The application will then process the photo and display execution output of the code.

The language chosen for this process is C language because of its simple syntax. C's grammar is relatively strict and is quite small compared to other modern programming languages. Missing declaration, missing semicolon or wrong variable type, etc. will all lead to compilation error (Yi and Tian, 2011).

An OCR system eases the barrier of the keyboard interface between man & machine to a great extent and help in office automation with huge saving of time and human effort. Such a system allows desired manipulation of the scanned text as the output is coded with some character code from the paper based input text. For a specific language based on some alphabet, OCR techniques are either aimed at printed text or handwritten text (Pingping *et al.*, 2012). The key component of such application software is an OCR engine, enabled with the key functional modules like line extraction, line to line segmentation, word to word character segmentation , character recognition using standard dictionaries.

OVERVIEW OF THE TESSERACT OCR ENGINE AND ITS WORKING

Tesseract (Rakshit *et al.*, 2008) is an open source (under apache license) offline optical character recognition engine, originally developed at Hewlett-Packard from 1984 to 1994. Like any standard OCR engine, tesseract is developed on top of the key functional modules like line and word finder, word recognizer, static character classifier, linguistic analyzer and an adaptive classifier (Gonzalez *et al.*, 2013)

Collection of the data set

For the collection of the dataset, we have concentrated on handwritten script. Some printed and handwritten document pages were collected from the different users. Foe each user, pages are collected for preparing the training set for the testing purposes, we have considered two types of users i.e known users and unknown users. Known users are the users who participated in preparation of the training samples. Unknown users are the users which do not participate in the preparation of the training set but are providing the input for enhancing the efficiency of the tesseract engine (Jhajj *et al.*,2010).

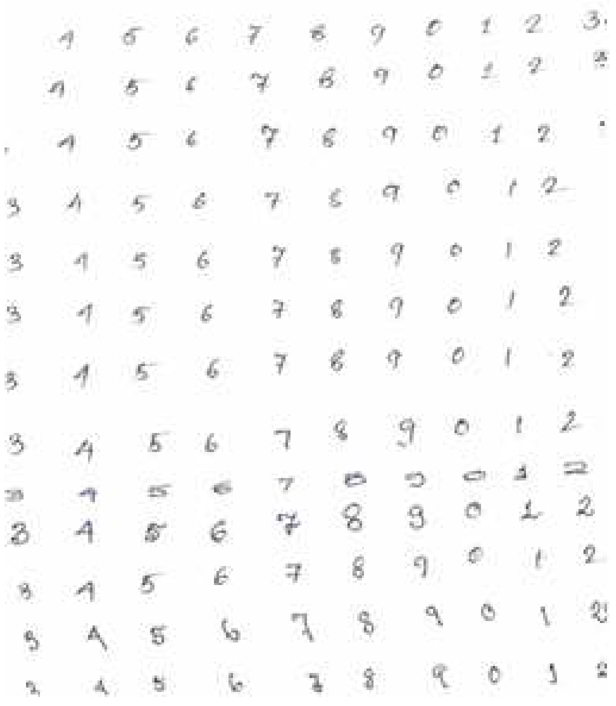


Figure 1: Sample document page containing free flow text

Process Overview

Since tesseract OCR would require high computation complexity, running the entire project on mobile phone is infeasible. Therefore, the project is implemented based on client-server communication model. The client side, android device, is in charge of capturing image and displaying result to user, while the server side is mainly responsible for most processing jobs.

The major stages in this process includes:

1. Capture and upload image: Using android device to capture the image and upload it to the server.

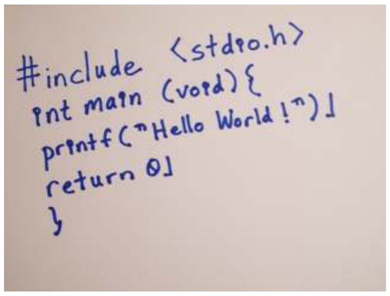
2. Matlab Preprocessing: In this step, several image processing methods are applied to the uploaded image to make it ready for recognition. These methods include: Binarization, small region removal and morphological opening. Preprocessing is an essential step to obtain accurate text recognition results with the Tesseract OCR tool, because a code image captured by a handheld camera is far from ideal input for an OCR engine.

3. Text recognition: In this step, we run the tesseract OCR engine (Mithe *et al.*,2013) to extract text from the preprocessed image. Tesseract cannot recognize handwritten text originally and some training process is required to make it capable of doing so. Tesseract requires a box file associates with each training image. Basically, each box file contains all coordinates and corresponding characters in the image, one per each row. Box file is not generated automatically but need produced by the manual labelling.

4. Post processing: In this step, some algorithms are applied to the retrieved text. The general idea is to make the text looks more like a piece of C code. The text returned by Tesseract contains mistakes in most case. Before pass it to human adjustment, we apply some heuristic text-level processing on it to avoid as much manual modification as possible.

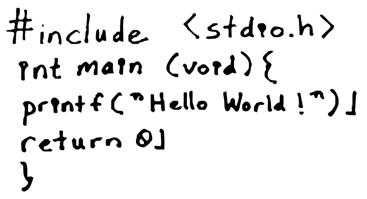
5. Manual adjustment: Even with all the previous steps, the system still cannot guarantee 100% correctness of recognition and even a single error would lead to a compilation failure. (Chucai, 2014) Therefore, server will send the final text back to the client and let the client do the final check. This step should only involve slight modification.

6. Compile, execute & display: After manually modified by user, ideally everything should be correct now. The client then sends text back to the server. The server compiles and executes the code and passes the result to the client for display.



```
#include <stdio.h>
int main (void){
printf("Hello World!");
return 0;
}
```

Figure 2: Image before pre-processing



```
#include <stdio.h>
int main (void){
printf("Hello World!");
return 0;
}
```

Figure 3: Image after preprocessing

Implementation

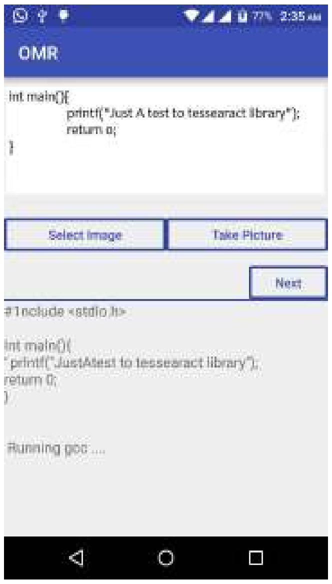
Codeable is implemented entirely in Python and Matlab. Codeable uses the python-requests module to communicate over HTTP with the webcam, mlabwrap module to communicate with Matlab, and python-tesseract module to communicate with the Tesseract API. Codeable is modular and consists of the user interface loop codeable.py and several components: webcam.py, normalize_image.m, ocr.py, postprocess.py, compiler.py, and feedback_postprocess.py.

Conclusion

This application may be quite useful in future and will come up with better result by training the engine more effectively. User should also maintain its handwriting as if the handwriting will be neat then it will be easy for the OCR engine to recognize the text more easily and for the compiler as well to recognize the word. Usage of ambiguous font should be minimized so as to enhance the efficiency of the application by providing the accurate output. Even a few misclassified characters, or missing characters that slips through the cracks of the post processing engine causes compilation to fail. Every aspect of the codeable image processing steps can be improved upon.

Result

Tesseract is originally designed for printed text recognition, but after proper training, it can also be used for handwritten text recognition. Some datasets have already been collected for training purpose. Since there should be some common key words for a certain programming language e.g. 'def' in Python, it needs to build a dictionary that contains such key words and run the spell corrector on each recognized term.



```
int main(){
        printf("Just A test to tessearact library");
        return o;
}
```

```
#1nclude <stdio.h>

int main(){
"printf("JustAtest to tessearact library");
return 0;
}


Running gcc ....
```

Figure 4: Final output

References

Gonzalez, Brian M. 2012. Iris: A Solution for Executing Handwritten Code." University of Adger, 01 June 2012. Web. 6 June 2013.

Jhajj, P., Sharma, D. 2010. Recognition of Isolated Handwritten Characters in Gurmukhi Script. *International Journal of Computer Applications,* 4(8), 9-17.

Mithe, R., Indalkar, S., Divekar, N. 2013. Optical Character Recognition. *International Journal of Recent Technology and Engineering,* 2(1).

Pingping Xiu, Henry S. B. 2012. Whole-Book Recognition. *IEEE Transactions on pattern analysis and machine intelligence,* 34(12).

Rakshit, Sandip, and Basu, S. 2008. Recognition of Handwritten Roman Script Using Tesseract Open Source OCR Engine, *Proc. National Conference on NAQC.*

Yi, C., Tian, Y. 2011.Text string detection from natural scenes by structure based partition and grouping. *IEEE Trans. Image Process.* 20(9), 2594–2605.

Yi, C., Tian, Y., Arditi, A. 2014. Portable Camera-Based Assistive Text and Product Label Reading From Hand-Held Objects for Blind Persons, *IEEE/ASME Transactions on Mechatronics,* 19(3).

https://code.googke.com/p/tesseract-ocr/wiki/TrainingTesseract3 (accessed on 5th January 2017)