JOURNAL OF SOFTWARE ENGINEERING & INTELLIGENT SYSTEMS
ISSN 2518-8739
31st August 2017, Volume 2, Issue 2, JSEIS, CADMEI Copyright © 2016-2017
www.jseis.org

Optimized order of character recognition in C# using tesseract

[1]Aleeza Safdar, [2]Usama Mansoor Ali, [3]Shahid Nazir Bhatti
[1,2,3]Dept. of Software Engineering
Bahria University Islamabad, Pakistan
Email: [1]aleezasafdar10@gmail.com, [2]usama_sam92@live.co.uk, [3]snbhattii.buic@bahria.edu.pk

ABSTRACT

There are numerous difficulties in recognizing the textual data from the images and requires great attention. Typical Optical Character Recognition (OCR) systems provide this facility to detect the textual data from image data. The text can also be in handwritten format. In this paper, we proposed the methods or API's that are used in Visual Studio to detect the text from images. Visual Studio only supports MSDN libraries and for this reason the other operations to be performed like, in text recognition OCR simply cannot be implemented without any elaborated API. These API's are basically based on different libraries that are introduced in Visual Studio in order to use the predefined functions of these mentioned libraries.

Keywords: visual studio; API; application programming interface; MSDN; OCR; optical character recognition; OpenCV;

1. INTRODUCTION

Image processing is the analysis and manipulation of a digitized image, especially in order to improve its meaningful quality. C# is the most widely & recently used programming language that creates an environment to solve image processing disciplines [1]. The textual recognition from the image can be achieved by using the 'OpenCV', which is a library of programming functions mainly used at real time computer vision. Further, this library crosses the platform that usually focuses on real time image processing [2, 3]. In C# OpenCV is added along-with its wrapper emgu CV [4]. This wrapper further is precisely written in C# and do not use unsafe code. The steps to extract the text from image are described below:

a) Font based

The text can be detected from an image but with very less accuracy using OpenCV libraries [5]. Most of the letters, words in this cannot be extracted from an image similarly the precise quality of the text extracted is also of very low level. Most of the words either disappear or they are not shown completely in the required language [6, 7].

b) Handwritten text

In this there is another problem that OpenCV libraries do not support handwritten text [8, 9]. Which means that handwritten text cannot be extracted from an image using OpenCV libraries. Even if the image is to be captured using a camera still handwritten text in image cannot be identified [10]. The real challenge is that the adaptive threshold algorithm is needed to be implemented for the handwritten text to address the above-mentioned problem within OpenCV in regard to the handwritten text, but this is an indeed a real challenge to do so [11].

c) Efficient API for C#

Tesseract is an OCR engine for the different operating systems. It is free software and was developed under the Apache License, Version 2.0 and further development has been sponsored by the Google since 2006. Tesseract is considered one of the most accurate open source OCR engines currently available. The Tesseract engine was originally developed as proprietary software at Hewlett Packard labs in Bristol, England and Greeley, Colorado between 1985 and 1994, with some meaningful changes made in 1996 to the ports to Windows, and also migration from C to C++ in 1998 [8, 12, 13]. Thus, a lot of the code was written in C, and then some more was written in C++. Since then all the code has been converted to compile with a C++ compiler, although very little work was done in the following decade. It was then released as open source in 2005 by Hewlett Packard and the University of Nevada, Las Vegas (UNLV) [14, 15]. Tesseract development has been sponsored by Google since 2006 [16].

Tesseract was in top three OCR engines in 1995. It was developed for Linux, Windows and Mac OS but due to limited resources it was only tested by Windows and Ubuntu [2]. Tesseract up to Version 2 could only accept Tagged

Image File Format (TIFF) images including single column of text as input. Then, with the Version 3 output text formatting was also being supported by the Tesseract. There was a library called Leptonica used to add different image formats in Tesseract. The initial version was only able to detect the English language from an image then with the Versions 2 and 3 it was able to detect English, Spanish, German, Italian and many more [1, 12, 14].

If Tesseract is to be used to read the text from right to left such as Arabic or Urdu then it will produce the results from in order that will be from left to right. The Quality of the output being produced from the Tesseract will be quite poor if the image is not pre-processed to suit it. The image should be properly scaled up so that the text (x-height) is up to 20 pixels. If there are dark borders around the image they must be removed and rotation in the image must also be avoided. Due to the dark borders, the Tesseract considers them as any text or special character which can result in the form of inaccurate output while the resizing of the image changes the resolution. Thus, the image must be bright enough to be recognized otherwise no text will be recognized. Tesseract can also be used as a backend, which can help in solving more complicated OCR tasks including layout analysis by using a front end such as OCRopus [17, 18].

Tesseract run from the Command Line Interface (CLI) and does not appear with GUI while there are many projects that provide us with GUI for the Tesseract, one common example in this is OCRFeeder is free and open source software which has the property to support all the command line in OCR engines virtually.

d) Tesseract for C#

In order to use Tesseract in our C# project we have downloaded Tesseract-3.02.02-win32-lib-include-dirs.zip. But the major issue is that it is built with the Visual Studio (VS) 2008 and other versions of VS do not support it. Thus, in order to utilize it in VS 2010 or VS 2013 one must look at the '.Net wrapper' for tesseract OCR, that the mentioned project needs to test more VS versions [19]. Tesseract and Leptonica binaries are compiled with VS 2010, so one need to ensure that the VS 2010 runtime is installed. In order to add wrapper in VS 2012 or VS 2010 following procedure must be adopted [20-22].

1) In order to use Tesseract in your C# project you need to make sure that you have downloaded the Tesseract language data files.
2) Then either you can use the internet to download the Tesseract package or you can use the NuGet Package to download it. NuGet Package is a service provided by the Visual Studio Package Manager Console.
3) You have to make sure that Visual Studio 2012 x86 & x64 runtimes are installed in your PC, because this Tesseract package was developed in Visual Studio 2012.
4) Check that the language data files are of Version 3.02. Make sure that all the files in language data folder are set to "Copy to output directory".

2. FUNCTIONAITY OF TESSERACT

Tesseract provides numerous functionalities but the prime functionality is the usage of Tesseract Engine as it takes arguments which include the data folder where all the languages are saved. The language short form for example for the English, "eng" is used and for the Arabic, "ar" is used and so on. It also takes the argument of EngineMode which is of 4 types and each has their own pros and cons [16, 23].

- Default
- Tesseract Only
- CubeOnly
- Tesseract and Cube

Default gives an average result which is neither the fastest nor it is the quickest. Default option instructs the engine to infer the best mode from the other three modes based on the language. TesseractOnly uses the Tesseract part of the engine only which is the fastest in terms of computation. CubeOnly uses the Cube part of the engine only it is slower than Tesseract but has better accuracy [1]. Combined mode runs both Tesseract and Cube modes and combines the results for the best accuracy but it slows down the speed of computation.

It offers a functionality of Object, also the process in which the image is passed as an argument. Make sure to add the effect of black and white to the image for better result. It returns the result to another object from with the text is extracted using 'Object.GetText'. It returns the string which contains the text present in the image [24].

3. IMPLEMENTATION

After downloading tesseract library, we have implemented it using its functions and classes. Firstly, the images are taken which are font based. Tesseract uses a different format for the images used as that of 'Pix'. That is, which is different from that of Bitmap format and they cannot be applied to each other implicitly. For that a "PixToBitmap" and "BitmapToPix" converter is required. The confidence level is being checked that is similarity between the original text that the image possesses and text that is scanned to from image in txt format. In font-based images the confidence level is pretty good. But when it comes to handwritten text again there is an issue regarding the size of image as well as resolution [23]. We have captured the difference taken as sample through camera but when Tesseract functionality is being implemented the image is not detected and nothing is shown as output. Because image resolution that is pixels are very large and unable to be detected by the Tesseract. After resizing the images, again the samples are tested and now the results are much different. Most of the characters are recognized and confidence level is also increased [18]. Means text of handwritten images can easily be recognized using Tesseract library functions. Similarly, there are functions used to crop image. This function is usually used to remove unnecessary borders and to get only the required portion that contains the text so that it can easily be recognized.

a) Training Set

Training Set is an approach of detecting handwritten text from an image. In this approach, a database/folder of similar images of the alphabets is saved (samples of alphabets). All the possible shapes of handwritten alphabets are kept there and from this training set the handwritten alphabets are compared. This is not very efficient way because its accuracy is very low, it operates better in controlled environment which is hard to maintain.

4. RESULTS AND OBSERVATIONS

We took different types of samples which included font based wallpapers, images from Facebook, hand written samples which included simple ones and complex ones. The simple ones were the letters written separately and they were straight (non-italic) and the complex samples included joining handwriting and characters too close to each other. We observed the output of these samples with all the Engine Modes of Tesseract and we observed much deviation, besides we observed other problems and constraints as well.

It was observed that the images which were to be detected, if they were taken from the camera or they had high resolution, then the results were null. This was due to the difference in the size of fonts. Images having "1920 x 1080" dimension had font size very lard if they were compared to an average image thus, the Tesseract function finds it difficult to detect it. The images were resized to lower resolution which has dimension < 500 pixels and the images were easily detected. Moreover, the blurry images, the images with distorted pixels or text in which alphabets were too much close, gave inaccurate result.

A comparison between the text before resizing and after resizing the image is shown below in the Figures 1 and 2.

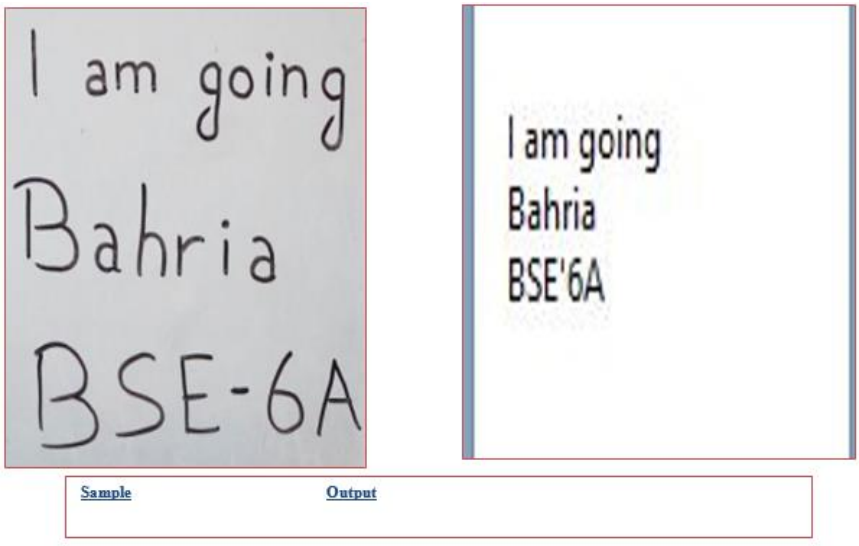

Figure. 1 Test case: a resized image

Figure. 2 Test case: not a resized image

The use of crop, sepia, gray, blur and sharpening in the image made the output accuracy a bit higher [4]. Cropping the image allows us to detect the text from an image where other different backgrounds were there and if we detected the image like that, a lot of errors were expected. Thus, cropping effect allowed us to remove those backgrounds and select only the textual part from the image, from which the detection could be made.

a) Confidence level

Confidence level is the percentage of text recognized by the image out of total text. At first using OpenCV functionality the confidence level was around 50% but with the usage of Tesseract, it improved a lot. The confidence level in case of font based images is from 60 to 90 percent which is fair enough because the text based recognition is basically probabilistic system that is 100% accuracy cannot be achieved in this process. On the other hand, handwritten text has confidence level from 50 to 60% which is approximately good [17, 19].

b) Camera captured images

When the images are captured through a camera and were tested they have shown the results with 0 confidence level. The reason is that the images taken through camera have high resolution power but the images that needed to be tested using Tesseract must have resolution up to 200 pixels. Thus, in order to make them appropriate for testing we have resized the image to get the required results. After resizing again when image is tested it gives the text with the confidence level from 50 to 60% [3, 5].

c) Font based images

All those images those are in the format built-in Calibri or New Times Roman font. They are font based images like the wallpapers we use which have text written on it. These images can easily be detected and give accurate results if being recognized by Tesseract [13].

d) Handwritten images

In this case, we usually take paper and write something using a marker that is easily visible. Then the image is captured using camera and is recognized using Tesseract. The output is usually not exactly similar to the image but there is 50 to 60% similarity which can be increased to 70 to 75% by using better image quality and clear handwritten text. The Figure 3 depicts the accuracy of text in an image.

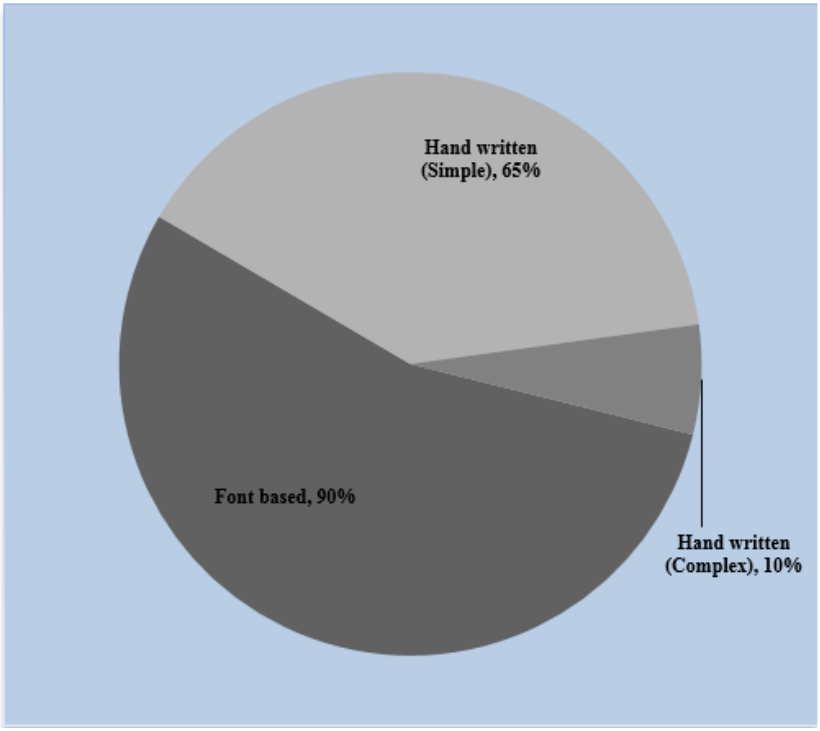

Figure. 3 Depicts the accuracy of text in an image

The different aspects between the outputs from the font based images and the handwritten text are, either complex or simple, shown in Figure 4. One can clearly see that the images from complex handwriting do not make any sense at all. The accuracy is almost 0% here but in some cases, it can increase up to 10 %, although the handwritten simple and font based text give an appreciable result.

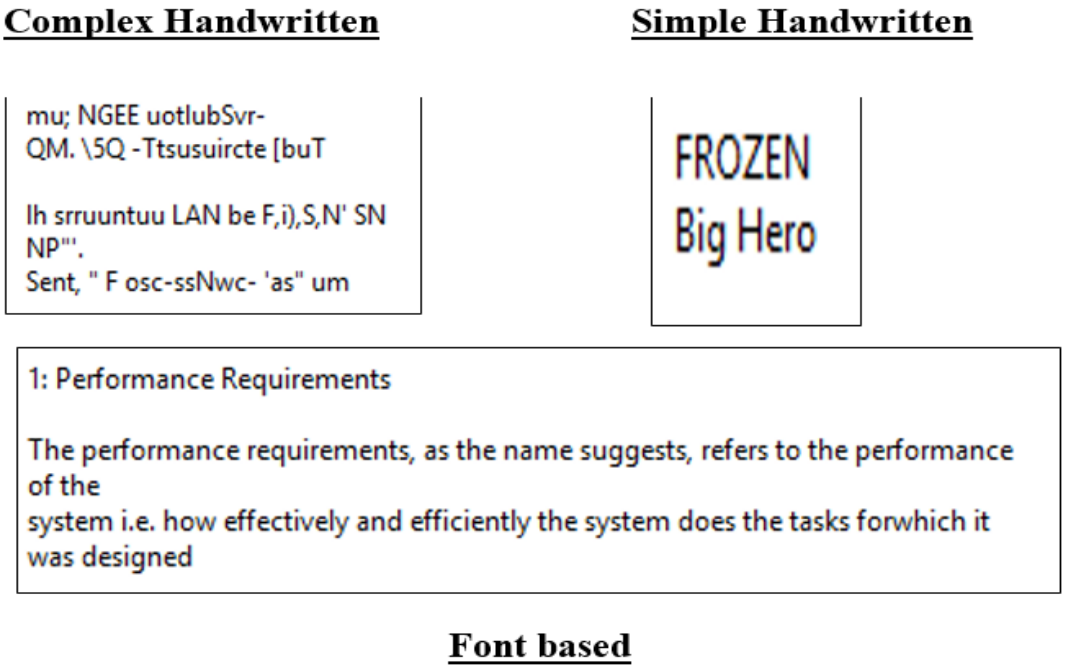

Figure. 4 Output from handwritten and font based images

5. CONCLUSION

Working with the different test scenarios and test cases, we have concluded that the Tesseract (OCR) is the most efficient library available for OCR in C#. Further Tesseract (OCR) has the capacity as well as the capability of improving the efficiency and accuracy with the help of the training sets. It is perfect for the font based scenarios but it is also that of particular interest that the handwritten detection is better as compared to any other present libraries within C#. The accuracy of text detection depends upon the detailed factions and factors discussed in the different sections of this study and it can be improved as well as illustrated with the different samples. The limited case scenarios and samples which we have worked on show the results and assumptions in context to the font based accuracy as well as with the textual based ratios as depicted in Figure 3. Finally, the observed Text detection is also used based on the differently styled number plates of the automobiles detection scenarios and also with the different mobile applications as well.

ACKNOWLEDGEMENT

The authors gave special thanks to the Department of Software Engineering at Bahria University Islamabad, Pakistan in providing resources to complete this research.

REFERENCES

1. Okita, A., *Learning C# programming with Unity 3D*. 2014: CRC Press.
2. Chancellor, J., *Rapid C# Windows Development*. 1st ed. 2006: Lulu Pubs.
3. Esposito, D. and A. Saltarello, *Microsoft. net-architecting applications for the enterprise*. 2014: Microsoft Press.
4. Smet, B.J.F.D., *C# 5.0 Unleashed*. 1st ed. 2013: Sams Publishing.
5. Richter, J., *CLR via C# (Developer Reference)*. 4th ed. 2012: Pearson Education.
6. Hall, G.M., *Adaptive Code via C#: Agile coding with design patterns and SOLID principles*. 2014: Pearson Education.
7. Skeet, J., *C# in Depth*. 3rd ed. 2013: Manning Publications.
8. Blain, J.M., *The Complete Guide to Blender Graphics: Computer Modeling & Animation*. 2016: CRC Press.
9. Sharp, J., *Microsoft Visual C# 2013 Step by Step*. 2013: Pearson Education.
10. Bevis, T., *C# Design Pattern Essentials*. 2012, UK: Ability First Limited.
11. Littleboy, D.M., *Numerical techniques for eigenstructure assignment by output feedback in aircraft applications*. 1994, University of reading.
12. Bhatti, S.N., *Deducing the complexity to quality of a system using UML.* ACM SIGSOFT Software Engineering Notes, 2009. **34**(3): p. 1-7.
13. Bhatti, S.N. and A.M. Malik, *An XML-based framework for bidirectional transformation in model-driven architecture (MDA).* ACM SIGSOFT Software Engineering Notes, 2009. **34**(3): p. 1-5.
14. Asghar, A.R., et al., *The Impact of Analytical Assessment of Requirements Prioritization Models: An Empirical Study.* INTERNATIONAL JOURNAL OF ADVANCED COMPUTER SCIENCE AND APPLICATIONS, 2017. **8**(2): p. 303-313.
15. Sultan, Z., et al., *Analytical Review on Test Cases Prioritization Techniques: An Empirical Study.* INTERNATIONAL JOURNAL OF ADVANCED COMPUTER SCIENCE AND APPLICATIONS, 2017. **8**(2): p. 293-302.
16. Williamson, J., *Character development in Blender 2.5*. 2011: Cengage Learning.
17. Wagner, B., *More Effective C#: 50 Specific Ways to Improve Your C*. 2010: Pearson Education.
18. Albahari, J. and B. Albahari, *C# 5.0 in a Nutshell: The Definitive Reference*. 2012: " O'Reilly Media, Inc.".
19. Dowland, P. and S. Furnell, *Advances in Communications, Computing, Networks and Security 5*. Vol. 5. 2008: Lulu. com.
20. Butt, F.L., et al., *Optimized Order of Software Testing Techniques in Agile Process-A Systematic Approach.* International Journal of Computer Science and Information Security, 2016. **14**(12): p. 509.
21. Nystrom, R., *Game programming patterns*. 2014: Genever Benning.
22. Holley, R., *How good can it get? Analysing and improving OCR accuracy in large scale historic newspaper digitisation programs.* D-Lib Magazine, 2009. **15**(3/4).
23. Whitaker, R., *The C# Player's Guide*. 2015: Starbound Software.
24. Asghar, A.R., et al., *Role of Requirements Elicitation & Prioritization to Optimize Quality in Scrum Agile Development.* INTERNATIONAL JOURNAL OF ADVANCED COMPUTER SCIENCE AND APPLICATIONS, 2016. **7**(12): p. 300-306.

AUTHORS PROFILE

Dr. Shahid Nazir Bhatti is currently working as Senior Assistant Professor in the Software Engineering Department at Bahria University Islamabad, Pakistan. He has obtained his PhD from Johannes Kepler University Linz, Austria in year 2007 in the area of Software Engineering. Dr. Bhatti has more than 14 years of teaching, research & industrial experience. Dr. Bhatti's current research activities are in the field of System Engineering, Requirement Engineering, Software Metrics & Applications, Software Quality Engineering, Information Systems & Software Reengineering.