Volume 6, Issue 2
April 2017

ITEE Journal
Information Technology & Electrical Engineering

ISSN: - 2306-708X

Object Character Recognition in C# using Tesseract

Aleeza Safdar
Research Assistant,
Dept. of Software Engineering
Bahria University Islamabad (BUI), Pakistan
aleezasafdar10@gmail.com

Dr. Shahid N. Bhatti
Senior Assistant Professor,
Dept. of Software Engineering
Bahria University Islamabad (BUI), Pakistan
snbhattii.buic@bahria.edu.pk

U. Mansoor Ali
Research Assistant,
Dept. of Software Engineering
Bahria University Islamabad (BUI), Pakistan
usama_sam92@live.co.uk

Dr. S. Asim Ali Shah
Senior Assistant Professor,
Dept. of Electrical Engineering
Bahria University Islamabad, Pakistan
asimshah.buic@bahria.edu.pk

ABSTRACT

There are numerous difficulties in recognizing the textual data from the images and requires great attention. Typical OCR (optical character recognition) systems provide this facility to detect the textual data from that of image data. The text can also be in handwritten format. In this paper we propose the methods or API's that are used in Visual Studio to detect the text from that of images. As Visual Studio only supports MSDN libraries and for this reason the other operations to be performed like, in text recognition Optical Character recognition (OCR) simply cannot be implemented without any elaborated API. These API's are basically based on different libraries that are to be 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. 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. In C# OpenCV is added along with its wrapper emgu CV. This wrapper further is precisely written in C# and do not use unsafe code

2. EXTRACTING TEXT FROM IMAGE

A. Font based

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

B. Hand Written Text

In this another problem with OpenCV libraries is that they do not support handwritten text. That is 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. 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.

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. It is the most accurate OCR engine available now days. 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 [2] [10] [8]. 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 at least 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

Volume 6, Issue 2
April 2017

ITEE Journal
Information Technology & Electrical Engineering

ISSN: - 2306-708X

Nevada, Las Vegas (UNLV) [3] [21]. Tesseract development has been sponsored by Google since 2006.

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 [21]. Tesseract up to version 2 could only accept TIFF (tagged image file format) 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 Lepontica used to add different image format in Tesseract. The initial version was only able to detect the English language from an image then with the version 2 and 3 it was able to detect English, Spanish, German, Italian and many more [3] [11] [16].

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 preprocessed to suit it. The image should be properly scaled up to 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. As 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 (the further details in this are elaborated in section 5 of this paper). 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 frontend such as OCRopus [20] [13].

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 to download tesseract-3.02.02-win32-lib-include-dirs.zip. But the major issue is that it is built with the Visual Studio 2008 and other VS do not support it. Thus in order to utilize it in VS 2010 or VS 2013 we will have to look at the '.Net wrapper' for tesseractOCR that the mentioned project has to be testing more VS versions. As Tesseract and leptonica binaries have been 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 [21] [9] [3].

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"

FUNCTIONALITY 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 etc. It also takes the argument of EngineMode which is of 4 types and each has their own pros and cons [21].

- Default
- Tesseract Only
- CubeOnly
- Tesseract And Cube

Default give 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. 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.

IMPLEMENTATION

After downloading tesseract library we have implemented it using its functions and classes. Firstly the images are taken which are fonts 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 i.e. 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. 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 i.e. pixels were very large and is unable to be detected by Tesseract. After resizing the images again those samples are tested now the results are much different. Most of the characters are recognized and confidence level is also being increased. Means text of handwritten images can easily be recognized using

Volume 6, Issue 2
April 2017

ITEE Journal
Information Technology & Electrical Engineering

ISSN: - 2306-708X

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 are being 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.

RESULT AND OBSERVATION

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 one which had 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 result 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 so the Tesseract function finds it difficult to detect it. The images were resized to lower resolution which has dimension < 500 pixels were easily detected. More over 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 figure 1.

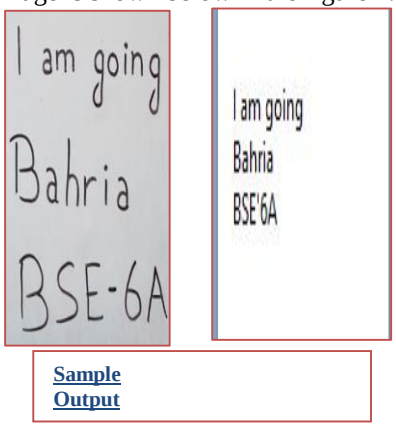

Fig.1. Test case: a resized image

B. Images

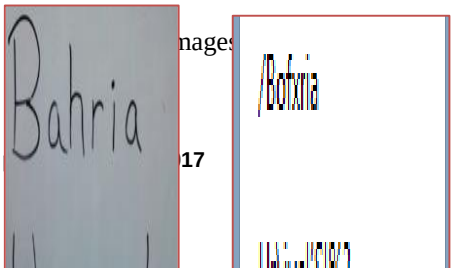

Fig.2. Test case: not a resized image

The use of crop, sepia, Gray, blur and sharpening in the image made the output accuracy a bit higher. 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 i.e. 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 [11].

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. So 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% [11] [20].

All those images those are in the format built-in Calibri or New Times Roman font etc. They are font based images like the wallpapers we use which have text written on it. These images

Volume 6, Issue 2
April 2017

ITEE Journal
Information Technology & Electrical Engineering

ISSN: - 2306-708X

can easily be detected and give accurate results if being recognized by Tesseract.

C. Handwritten images

In this case we usually take paper, writes 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 Fig 3 (graph) depicts the accuracy of text in an image. depicts the accuracy of text in an image.

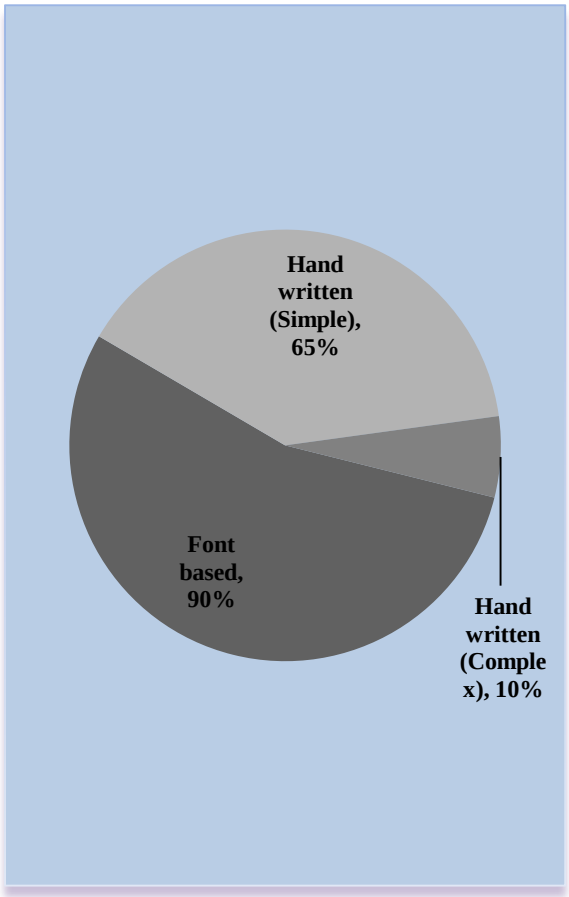

Fig.3.Depicts the accuracy of text in an image

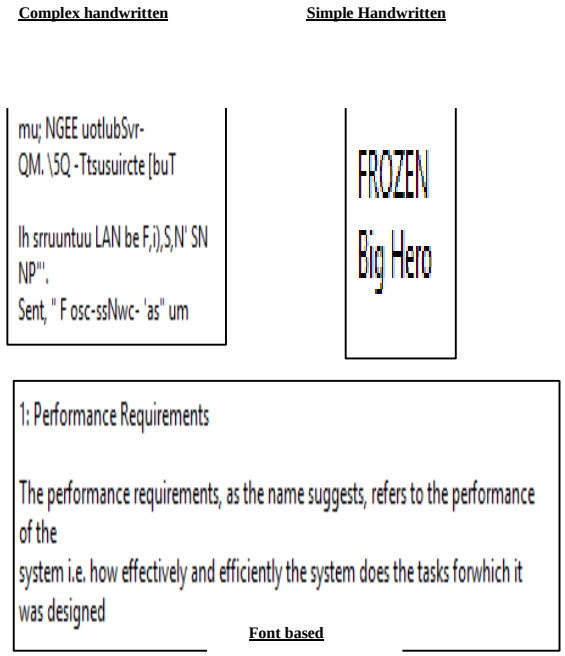

Fig.4. Output from Handwritten & Font based images

The different aspects between the outputs from the font based images and the handwritten text are (either complex or simple) are shown in the following 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 gives an appreciable result.

CONCLUSION

In the following while 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 (work) and it can be improved as well as illustrated with the different samples. The following limited case scenario (experiment) and samples which we have worked on show the results & assumptions in context to the font based accuracy as well as with the textual based ratios as depicted in the particular graph in section 5. Finally the observed Text detection is also used in the different number automobiles plate detections scenarios and also with the different mobile applications as well.

REFERENCES

[1] Dino Esposito, Andrea Saltarello, 2014. Microsoft .NET - Architecting Applications for the Enterprise, 2nd Edition. Microsoft Press.

[2] John M. Blain, 2014. The Complete Guide to Blender Graphics, Second Edition. Computer Modeling and Animation 2nd Edition. A K Peters/CRC Press.

[3] Alex Okita, 2014. Learning C# Programming with Unity 3D, 1st Edition, A K Peters/CRC Press.

[4] Aneesa Rida Asghar, Shahid Nazir Bhatti, S. Asim Ali Shah, "The Impact of Analytical Assessment of Requirements Prioritization Models: An Empirical Study" International Journal of Advanced Computer Science and Applications(IJACSA), 8(2), 2017.

[5] Rose Holly, 2009. How good can it Get, D-Lib Magazine.

[6] Jonathan Williamson, Character Development in Blender 2.5, 1st Edition. Cengage Learning PTR.

[7] Farrukh L. Butt, Shahid Nazir Bhatti, Sohail Sarwar, Amr Mohsen Jadi and Abdul Saboor, "Optimized Order of Software Testing Techniques in Agile Process – A Systematic Approach" International Journal of Advanced Computer Science and Applications(ijacsa), 8(1), 2017.

[8] Jon Skeet, 2013. C# in Depth, Third Edition. Manning Publications.

[9] Joseph Chancellor, 2006. "Rapid C# Windows Development", First Edition, Lulu Pubs.

[10] Shahid Nazir Bhatti, 2009. Deducing the complexity to quality of a system using UML. ACM SIGSOFT Software Engineering Notes 34(3): 1-7 (2009). DOI= http://dl.acm.org/citation.cfm?doid=1527202.1527207

[11] Paul Dowland, Steven Furnell, 2009, Advances in Communications, Computing, Networks and Security, Volume 6, Proceedings of the MSc/MRes programmes from the School of Computing, Communications and Electronics, UK.

[12] John Sharp, 2013. Microsoft Visual C# 2013 Step by Step, 1st Edition, Microsoft Press.

[13] Joseph Albahari, Ben Albahari, 2012. C# 5.0 in a Nutshell: The Definitive Reference. 5th Edition, O'Reilly Media.

[14] RB Whitaker, 2012. The C# Player's Guide, Starbound Software.

[15] Robert Nystrom, 2014. Game Programming Patterns. 1st Edition, Genever Benning.

[16] Shahid Nazir Bhatti, Asif Muhammad Malik, 2009. An XML-based framework for bidirectional transformation in model-driven architecture (MDA). ACM SIGSOFT Software Engineering Notes 34(3): PP 1-5.

[17] Aneesa Rida Asghar, Shahid Nazir Bhatti, Atika Tabassum, "Role of Requirements Elicitation & Prioritization to Optimize Quality in Scrum Agile Development" International Journal of Advanced Computer Science and Applications(IJACSA), 7(12), 2016.

[18] Bart J.F. De Smet, 2013. C# 5.0 Unleashed, First Edition, Sams Publishing.

[19] Darren M Littleboy. 2012, "Numerical Techniques for Eigenstructure Assignment by Output Feedback in Aircraft Applications"

[20] Bill Wagner, 2010. Effective C#: 50 Specific Ways to Improve Your C#, 2nd Edition, Addison-Wesley Professional.

[21] Jeffrey Richter, 2012. CLR via C#, (Developer Reference), 4th Edition, Microsoft Press.

[22] Tony Bevis, 2012. C# Design Pattern Essentials. Ability First Limited

[23] Gary McLean Hall, 2014. Adaptive Code via C#: Agile coding with design patterns and SOLID principles, (Developer Reference), 1st Edition. Microsoft Press.

[24] Shahid N. Bhatti, Maria Usman, Amr A. Jadi, 2015, Validation to the Requirement Elicitation Framework via Metrics. ACM SIGSOFT Software Engineering Notes 40(5): 17, USA.

[25] Zainab Sultan, Rabiya Abbas, Shahid Nazir Bhatti and S. Asim Ali Shah, "Analytical Review on Test Cases Prioritization Techniques: An Empirical Study" International Journal of Advanced Computer Science and Applications (IJACSA), 8(2), 2017.

AUTHOR 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 System 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, CBSD & Software Reengineering.

Aleeza Safdar received the degree in Software Engineering from Bahria University, Islamabad in 2016. Her research interest areas are Software Engineering, Quality Engineering, Simulations and Modelling.

Usama Mansoor Ali received the degree in Software Engineering from Bahria University, Islamabad in 2016. He is a research student of Masters in Software Engineering. Currently he is a Teaching/Research Assistant at Bahria University, Islamabad

Dr. Syed Asim Ali Shah is currently working as Senior Assistant Professor in the Electrical Engineering Department at Bahria University Islamabad, Pakistan. He received his degree of Doctor of Philosophy in Simulation and Modelling from Sheffield Hallam University, United Kingdom in 2016. He received his MS in Electronics and Information Technology majoring in Artificial Intelligence in 2004 from UK. He is a member of Pakistan Engineering Council (PEC). He has over 13 years of teaching, research & industrial experience. His current research activities are in the field of Simulation and Modelling, Artificial Intelligence, Robotics and Control Systems.