OPTICAL CHARACTER RECOGNITION FOR EXPIRATION DATES VALIDATION

Mitja Avgunštinčič[1], *Adam Peichl*[2]

[1]*University of Ljubljana, mitja.avgustincic@gmail.com*
[2]*Czech Technical University in Prague, adam.peichl@fs.cvut.cz*

Abstract: This paper focuses on custom OCR algorithm created for expiration dates validation using mainly open source libraries. Two main approaches are proposed in this work: recognising every character separately and unified characters method. To boost accuracy of OCR algorithm, several preprocessing methods and algorithms are also introduced. The final module is written in python and reaches satisfying accuracy considering all the difficulties, which have come across during the implementation process.
Keywords: OCR, character recognition, image preprocessing, OpenCV

1 Introduction

In recent years a lot of development and action was focused on the *optical character recognition* (OCR). Starting in the 1950, with the purpose of helping vision impaired persons to read, the OCR has developed and greatly improved throughout the decades, leaving its marks mainly in industry.

In nowadays industry, the OCR is generally used as an aid for automatization of the processes, with the processing speed being its stronghold. Primarily OCR is being used for document scanning, recording and tracking. One of the most used applications of the OCR system is in the banking, where the OCR typed fonts were developed for easier and more accurate character detection [1].

Among the other applications we can also find license plate recognition as surveillance technique and industrial packages validation, where product numbers (barcodes) and expiration dates are validated.

In our project we focused on the optical character recognition of the expiration dates on food packages with the purpose of validating the expiration date correctness in the production industry. Another possible application of our project would be to integrate our system into smart refrigerators, where every package is scanned for expiration date, before being put in the fridge, to ease the package and data management of the stored products.

The main focus was put on creating software for automatizating the process of reading an expiration date from image and its implementation in python. Our system takes an image as an input, process it using different algorithms and methods, and with the use of open source libraries recognises the characters in the expiration date. Two approaches were tested with the intention to compare them between themselves and find the method with the higher accuracy.

2 Optical character recognition (OCR)

Optical character recognition (OCR) is a process of classification of optical patterns contained in a digital image corresponding to alphanumerical or other characters. The character recognition is achieved through important steps of segmentation, feature extraction and classification [2].

OCR systems, with its capability of identifying graphic symbols that make up an alphabet of any language, simulate the human ability to read. These systems start from the acquisition of an image through optical equipment connected to a device to capture the graphic symbols (usually by a camera or a scanner). Digital image processing techniques, computer vision and pattern recognition are applied to the image in order to obtain a code, such as ASCII, which will represent each character [2].

OCR systems have become one of the most successful applications of technology in the field of pattern recognition and artificial intelligence and have also been widely applied in various areas, such as industrial applications

(Markov models, license plate recognition vertical, traffic sign recognition, etc). It is needed when the information should be readable both, to humans and to machine, and alternative inputs cannot be predefined. There are numerous commercial systems, but none so far have achieved recognition rates close to humans, especially when the system works in an uncontrolled environment, with the inevitable presence of noise. However, these systems have the advantage of total insensibility to eyestrain and in most cases outperform humans at speed of recognition [2], [3].

The algorithm described in this article can be divided into 3 stages, first we used various techniques to preprocess the image, second step was using algorithms for localization and segmentation of the image and the last step was character recognition using open source OCR software.

3 Experimental setup and used software

For our project, several random food packages with expiration dates were used. Images were obtained using the mobile phone *Huawei P9* with 12 megapixel *Leica* camera with lens aperture f/2.2. Data set of expiration date images was created for further processing and validation.

Software was created using *python 3.7 programming language*. There is a whole range of OCR software available today in the markets like: Desktop OCR, Server OCR, Web OCR, Mobile OCR etc. When considering possible OCR applications for our project, we tried different web OCR (websites), add-ons (Adobe Reader, Word) and even applications. Accuracy of extracting text of any of these OCR tool varies from 71 % to 95 %, with proper image preprocessing. Many OCR tools are available as paid and work really well but only few of them are open source and free.

Among many libraries and OCR software provided on the internet, the most popular and widely used library is *tesseract* [4]. The tesseract library was chosen based on some key properties such as being open source, high accuracy, constant development and improvement (currently under Google development) and different programming language integration (such as python, C++ and Java).

3.1 OpenCV

Open Source Computer Vision Library (OpenCV) [5] is one of the most widely used libraries in image processing, which is developed and released by *Intel Corporation*. It is distributed under a BSD style license which allows for royalty free commercial or research use with no requirement that the user's code be free or open. OpenCV is written in C and C++ and runs under Windows, Linux or MAC OS, but the code is well behaved and has ported to many other operating systems [6].

OpenCV contains an optimized collection of C libraries spanning a wide range of computer vision algorithms, including motion segmentation and pose recognition, multi-projector display system, object and face recognition, and 3D reconstruction, etc. The broad functional areas supported by OpenCV include:

- basic structures and array manipulations
- image processing and analysis
- object structural analysis
- motion analysis and object tracking
- object and face recognition
- camera calibration and 3D reconstruction
- stereo, 3D tracking and statistically boosted classifiers
- user interface and video acquisition support

There is active development on interfaces for Python, Ruby, Matlab, and other languages [6].

3.2 Tesseract

Tesseract is an optical character recognition engine for various operating systems. It is free software, released under the *Apache License*, Version 2.0. Since 2006, it's development has been sponsored by Google.

We are using python library PyTesser [7], which is an optical character recognition module for Python. It takes an image as input and outputs a string. PyTesser uses the Tesseract OCR engine, converting images to an accepted format and calling the Tesseract executable as an external script.

Tesseract's output will have very poor quality if the input images are not preprocessed properly. Images (especially screenshots) must be scaled up so that the text x-height is at least 20 pixels, any rotation or skew must be corrected or no text will be recognized, dark borders must be manually removed, or they will be misinterpreted as characters, etc.

4 Image Preprocessing

In order to preserve high accuracy, image preprocessing must be made. We used different methods and algorithms in order to prepare the image for letter recognition with the highest possible accuracy.

Usually, in a computer vision system, the preprocessing step aims to improve the quality of the digital image. Therefore, algorithms are applied to eliminate undesirable regions (noise), which arise due the conditions or the method of acquisition. *Low-pass filters* are among the most commonly used techniques to eliminate and reduce noise in an image. This type of filter smooths the image and reduces the number of gray levels and consequently minimizes the noise. High frequencies, which correspond to fast transitions are attenuated. *Median* and *Gaussian* are two types of low-pass filters widely used [3].

4.1 Gaussian Blur

A Gaussian filter softens the image based on the mathematical theory of the Gauss curve. This filter is applied on the convolution matrix defined as mask, and depending mainly on the standard deviation σ. Equation 1 represents the Gaussian filter in two dimensions, which means, that it is the product of two such Gaussian functions, one in each dimension:

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{1}$$

where x is the distance from the origin in the horizontal axis, y is the distance from the origin in the vertical axis, and σ is the standard deviation of the Gaussian distribution. When using two dimensional blurring, values from Gaussian distribution are used to build a convolution matrix which is applied to the original image. Each pixels new value is set to a weighted average of that pixels neighborhood. The original pixels value receives the heaviest weight (having the highest Gaussian value) and neighboring pixels receive smaller weights as their distance to the original pixel increases. This results in a blur that preserves boundaries and edges better than other, more uniform blurring filters.

4.2 Color to Grayscale Conversion

The next step in image preprocessing was conversion from color to grayscale.

Grayscale image is one in which the value of each pixel is a single sample representing only an amount of light. The image itself only carries intensity information.

Input images have *RGB* color channels. Classically, the grayscale image is obtained by a linearly weighted transformation:

$$\mathbf{J}(x,y) = \alpha \cdot \mathbf{R}(x,y) + \beta \cdot \mathbf{G}(x,y) + \gamma \cdot \mathbf{B}(x,y) \tag{2}$$

The most popular method selects the weighted values of α, β and γ by eliminating the hue and saturation information while retaining the luminance.

To this end, a color pixel is first transformed to the so-called NTSC color space from the RGB space by the standard NTSC conversion formula:

$$\begin{bmatrix} \mathbf{Y}(x,y) \\ \mathbf{I}(x,y) \\ \mathbf{Q}(x,y) \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ 0.596 & -0.274 & -0.322 \\ 0.211 & -0.523 & 0.312 \end{bmatrix} \begin{bmatrix} \mathbf{R}(x,y) \\ \mathbf{G}(x,y) \\ \mathbf{B}(x,y) \end{bmatrix} \tag{3}$$

where $\mathbf{Y}$,$\mathbf{I}$ and $\mathbf{Q}$ represent the NTSC luminance, hue, and saturation components, respectively. Then the luminance is used as the grayscale signal $\mathbf{J}(x,y) = \mathbf{Y}(x,y)$. Thus we have:

$$\alpha = 0.299, \quad \beta = 0.587, \quad \gamma = 0.114$$

With the grayscaling we reduced the noise of saturation and hue in the image, leaving only luminance on the image [8].

4.3 Thresholding

Another very common process in the preprocessing step is thresholding. This technique has set of values as the threshold, usually in gray levels, which are maintained in an interval of image tones [3].

If a pixel value is greater than a threshold value, it is assigned one value (may be white), else another value is assigned (may be black).

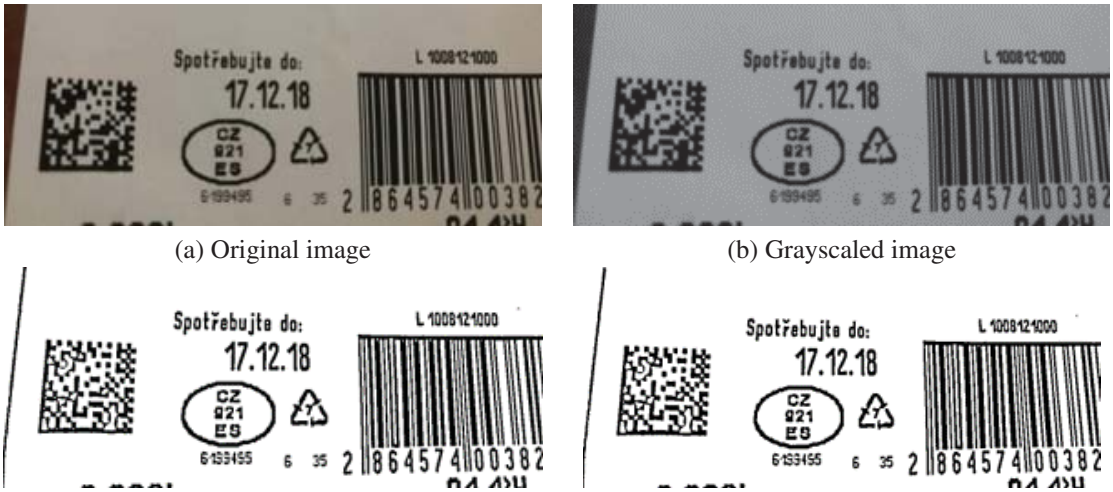
In our case the adaptive Gaussian threshold was used with the binary thresholding function. Binary threshold function returns only values that are either black (0) or white (255).

Normal thresholding uses a global value as threshold value. It turns out that this may result in poor results when dealing with images with different lightning conditions in different areas. Adaptive threshold on the other hand calculates the threshold for small regions of the image resulting in different thresholds for different regions of the same image giving us better results for images with varying illumination [9].

4.4 Image Closing

The last image preprocessing algorithm, we need to use, is image closing. Image closing is morphological operation. Morphological operations can be used to construct filters similar in concept to the *spatial filters*. The objective of the morphological operation is to eliminate the noise and its effects on the print while distorting it as little as possible.

The image closing is usually used in image processing that tends to smooth sections of contours. However it generally fuses narrow breaks and long thin gulfs, eliminating small holes and filling the gaps in the contour. For closing operations 3*x*3 kernel was used [10].



(a) Original image (b) Grayscaled image

(c) Thresholded image (d) Closed image

Table 1: Sequence of preprocessing of an image necessary before recognition

4.5 Segmentation

In the next step all of the contours in the image were found using OpenCV function *findContours* [11]. These contours can be divided into two groups:

1. contours of the characters we want to recognise
2. contours we recognise as a noise

There are many different approaches to split the contours into mentioned groups. The one used here takes the boundaries of width and height of contours as parameters, which are set in accordance to the width and height of the target characters on the preprocessed image, resulting in extracting of these character contours from every possible contour in the image.

OpenCV also includes function called *boundingRect* [12]. This function was used to determinate the position and the area of the contours in terms of coordinates. With information about the characters area and location, the letter extraction follows [12].

In order to extract letter, white image was created that would serve as a canvas, as an matrix with all values of 255 (white) and the same resolution of an input image. Later on, the position and area of each bounding box

around a character contour was taken and used to overwrite the values of the bounding box on the canvas white image in the same position as in original image. With this technique, we end up with image that only contains characters we want to recognise. The noise has been generally removed.

(a) Original image

(b) Image of possible contours

(c) Image of possible characters

(d) Image of characters

Table 2: Segmentantion

5 Letter extraction

Final step in our software we had to take was the use of the library tesseract to recognise the characters from preprocessed image and return it as a string.

Although we reduced most of the noise from the image, it is still possible that there is a noise in the image. If we still have noise in the image or if the characters are not clearly shown, tesseract either recognises nothing or it recognises non ASCII characters. The noise removal in this case was solved using returned string and remove all the blank spaces and non ASCII characters as well as ASCII letters. The ending result is expiration date number in the following format DDMMYYYY [13].

6 Unified characters method

In the beginning of this research an assumption was made, that recognising every character separately can have its cons. These cons were confirmed with testing the software and are generally thought to be:

1. recognising more noise if the expiration dates characters are the same font size as noise text
2. lower accuracy because of punctuation[1]
3. lower accuracy because of different date formats

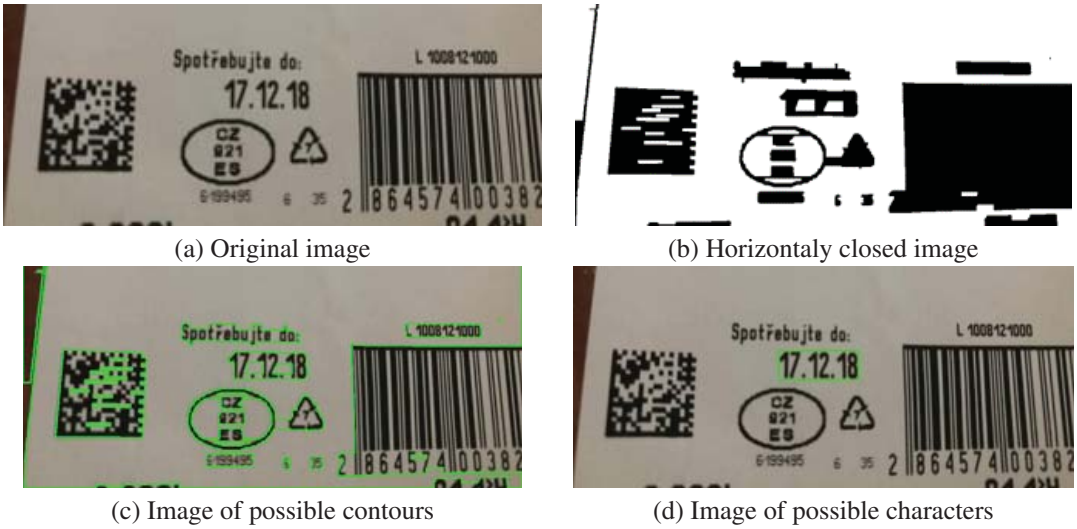
In order to improve the software and to avoid the cons used in *separate character recognition* method, an alternative approach was made.

Instead of using 3x3 kernel in image closing, horizontal size 30x5 kernel was chosen. With selecting kernel crucially bigger horizontally than vertically, we achieve image closing of the horizontal printed text. With this method, horizontally aligned text is merged. Contour functions are applied on the horizontally merged text and when we have the contours, we implement them on the original preprocessed image with 3x3 image closing kernel.

In this case, instead of filtering the contours using the height and width of the characters, we can filter the width of the whole expiration date. The input parameters are in this case easier to determine, since the expiration date is usually stand alone print on the package and it usually has different text width than noise in the image.

The rest of the algorithm, including image processing, segmentation and tesseract character recognition remain the same.

[1]For example character ≫.≪ can be recognised as ≫0≪



(a) Original image (b) Horizontaly closed image

(c) Image of possible contours (d) Image of possible characters

Table 3: States of Unified characters method

7 Conclusions

One of the first realisation made was that the accuracy of the both methods is the same, 66.7 %. In both cases low accuracy was achieved due to different lightning and other surrounding effects on the images. It should be also taken into consideration the cameras automatic mode for photograph optimisation, that changes parameters differently for every single image. One of the reason for not recognised expiration dates is also the bad quality of the printing expiration dates on the packages, because some of them were partially washed or brushed off.

When analysing the single OCR method conclusion was made, that it is easier and much more accurate to recognise the expiration dates on the packages, where expiration date font is sufficiently bigger than the other text. There are also eventual problems, with noise in the close region of the character, because simple long thin noise contour can be easily recognised as »I« or »1«. This kind of missinterpretation are not in the unified OCR method, because the tesseract library, when being fed with merged letters recognises and distinguish between characters and punctuation and sometimes even noise. However in some cases although the single OCR recognises the photo, no matter of the input parameters, the unified OCR recognises nothing.

Some general conclusions regarding the expiration dates were also made. The most clearly date expiration dates are the one who are printed with black color on white background. The OCR accuracy also depends on the material, the expiration date is printed on, the most success we get with carton, while plastics are harder to read due to reflective light our camera captures. Dot matrix character expiration dates are one of the most hard to read since for proper OCR it is needed to have closed contours. Stamped expiration dates also cause problems reading the expiration dates, since many of the character details are washed or brushed off, leaving incomplete letters, our program fails to recognize.

In order to have better OCR expiration date recognition in industry for validation purposes, the products provider should take an extra step in designing more OCR friendly packages. Some of the possible solutions are: different font sizes for expiration dates, OCR typed fonts, material where expiration date is printed (paper sticker instead of plastics) and most important of all, the controlled space where capturing and validation will be made.

Some of the future work considerations are using grayscale camera of lower resolution than 12 MP. With this action we reduce the computing time, making it more suitable for real-time industry and simplification of the software could be made with removing some of its features. Secondly, bigger amount of data set could be used in order to obtain better accuracy and one type of food packaging, captured in constant not-changing space should be used.

Acknowledgements

This project was supported by grant *SGS18/177/OHK2/3T/12*. The project was programmed in *Python* and source codes are available on request.

References

[1] Amarjot Singh, Ketan Bacchuwar, and Akshay Bhasin. A survey of ocr applications. *International Journal of Machine Learning and Computing*, 2(3):314, 2012.

[2] Pratixa Badelia Soumya K. Ghosh Arindam Chaudhuri, Krupa Mandaviya. *Optical Character Recognition Systems for Different Languages with Soft Computing*. Springer International Publishing AG 2017, 2017.

[3] Gabriel B Holanda, João Wellington M Souza, Daniel A Lima, Leandro B Marinho, Anaxágoras M Girão, João Batista Bezerra Frota, and Pedro P Rebouças Filho. Development of ocr system on android platforms to aid reading with a refreshable braille display in real time. *Measurement*, 120:150–168, 2018.

[4] pytesseract. https://pypi.org/project/pytesseract/. Accessed: 2019-02-19.

[5] Opencv. https://opencv.org/. Accessed: 2019-02-20.

[6] Opencv - about. https://opencv.org/about/. Accessed: 2019-02-20.

[7] Pytesser. https://pypi.org/project/PyTesser/. Accessed: 2019-02-22.

[8] Color conversions. https://docs.opencv.org/3.1.0/de/d25/imgproc_color_conversions.html. Accessed: 2019-02-21.

[9] Image thresholding. https://docs.opencv.org/3.4.0/d7/d4d/tutorial_py_thresholding.html. Accessed: 2019-02-21.

[10] Rafael C. Gonzalez and Richard E. Woods. *Digital image processing*. Prentice Hall, Upper Saddle River, N.J., 2008.

[11] Opencv - findcontours. https://docs.opencv.org/2.4/doc/tutorials/imgproc/shapedescriptors/find_contours/find_contours.html. Accessed: 2019-02-25.

[12] Opencv - boundingrect. https://docs.opencv.org/3.1.0/dd/d49/tutorial_py_contour_features.html. Accessed: 2019-02-25.

[13] python.org - string. https://docs.python.org/2/library/string.html. Accessed: 2019-02-25.

Selected article from
Tento dokument byl publikován ve sborníku

Nové metody a postupy v oblasti přístrojové techniky, automatického řízení a informatiky 2019

New Methods and Practices in the Instrumentation, Automatic Control and Informatics 2019

27. 5. – 29. 5. 2019, Zvíkovské Podhradí

ISBN 978-80-01-06617-1

Web page of the original document:
http://iat.fs.cvut.cz/nmp/2019.pdf

Obsah čísla/individual articles:
http://iat.fs.cvut.cz/nmp/2019/

Ústav přístrojové a řídicí techniky, FS ČVUT v Praze, Technická 4, Praha 6