Applying OCR Technology for Receipt Prices Recognition

Alessandro Marzo

Image Analysis, 2016

Contents

1 Abstract

The goal of this project is to develop a system to recognize prices in receipts which then could be implemented in a mobile app to allow easy splitting of expenses between people. To achieve this, we designed different algorithms to find and segment the prices on the image and implemented various preprocessing stages to improve the accuracy of the system. For the recognition stage we relied on the popular OCR engine Tesseract.

2 Defining the problem

Text recognition on receipts is one of the hardest problem for OCR technology to handle. The reasons for this are numerous:

- receipts are usually printed on cheap thin paper.
- they have very large amount of dense text
- existing OCR engines are almost exclusively trained on non-receipt data (books, documents, etc.).
- receipt structure, which is something between tabular and freeform, is hard for any layouting engine to handle.

Working with images from a mobile camera presents a number of challenges for OCR even when the document is clearly readable to a person. Poor image focus causes blurry text that runs together and even minor defects like creases or shadows create noise that can be confused with text. Poor lighting results in low contrast images which can cause speckles in the photo or even broken text.

The proposed algorithm is designed to work under these conditions by taking advantage of the document structure and uses redundant data information to help find the right solution even if poor image quality causes erroneous recognition.

3 Image processing pipeline

Improving the accuracy of the output is divided into several stages:

- Rescaling
- Binarization
- Noise Removal
- Segmentation
- Recognition
- Error Correction

3.1 Rescaling

Tesseract works best on images which have a DPI of at least 300 dpi (dots per inch), so it may be beneficial to resize images if that's not the case. All photos were taken with a smartphone Meizu m2 note, which has a camera with a resolution of 3104x4192 (13 megapixel). This resolution is more than enough for the size of a typical receipt so no resizing is required.

3.2 Binarization

Binarization is the process of converting a pixel image to a binary black and white image. As previously discussed, since the images captured with a mobile camera can have problems such as an illumination gradient or shadows, an adaptive thresholding method is required. Whereas the conventional thresholding operation uses a global threshold for the all image, adaptive thresholding acts locally in each pixels' neighbourhood changing the threshold dynamically over the image. We used the method $ADAPTIVE\_THRESH\_MEAN\_C$ from the function *adaptiveThreshold* of OpenCV [1], given in Eq. 1.

$$dst(x,y) = \begin{cases} 0 & \text{if } src(x,y) > T(x,y) \\ maxValue & \text{otherwise} \end{cases} \tag{1}$$

where the threshold value $T(x,y)$ is a mean of the `blockSize` $\times$ `blockSize` neighbourhood of (x,y) minus a constant value C.

This operation can be sensitive to noise, especially in the uniform regions of the background of the receipt, as shown in Fig. 1.

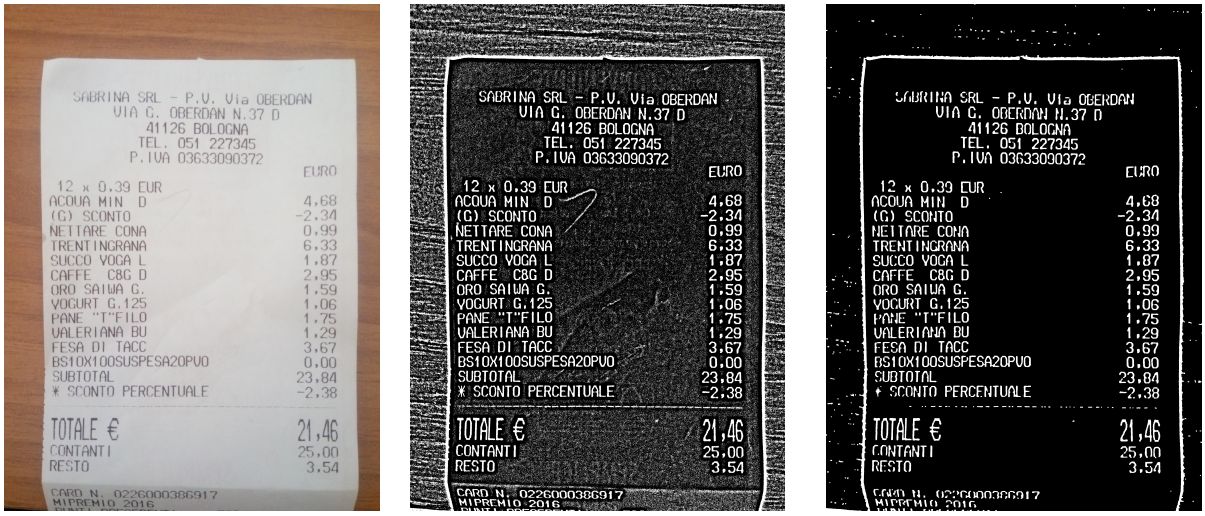

Figure 1: Example of a processed image of a receipt. Right: Starting image. Center: Binary image. Left: Noise removal result.

3.3 Noise Removal

The noise introduced by the thresholding operation is removed using morphological operations: the opening operation followed by the closing operation. Opening removes small

objects from the foreground of an image, placing them in the background, while closing removes small holes in the foreground. The result is shown in Fig 1 on the left. The next step is the segmentation of prices in the image.

3.4 Segmentation

Image segmentation is the process of dividing an image into multiple parts in order to simplify its representation into something that is more meaningful and easier to analyse. To locate the prices in the image we take advantage of the geometry of the document and extra input information from the user, which is required to provide only the initial position of the total price. This information then is used to estimate an approximate area of the image which contains only the total price and the prices above.

Figure 2: Image of an approximate area of the total price indicated by the user.

To achieve the segmentation we implemented two algorithms which have two different purposes. The first is designed to select the window centred around a price in an image of its approximate area.

Figure 3: Automatically selected window containing the total price.

The algorithm divides the image and uses morphological operations to dilate the digits in order to connect the ones the belong to the same price. Then the image is recombined and the $findContours$ and $contourArea$ functions of OpenCV are used to select the object with the largest area, and $boundingRect$ to return its bounding rectangle.

The second algorithm is designed to find the position of each price in an image that contains multiple prices in column.

Figure 4: Approximate area of the image which contains only prices in column.

Projecting all the pixels' intensity values of this image (Fig. 4) on the y-axis we obtain an histogram which indicates the positions of the prices, since the document is structured and all the prices and characters are spaced. This histogram is also thresholded to avoid errors caused by any possible connected components due to noise.

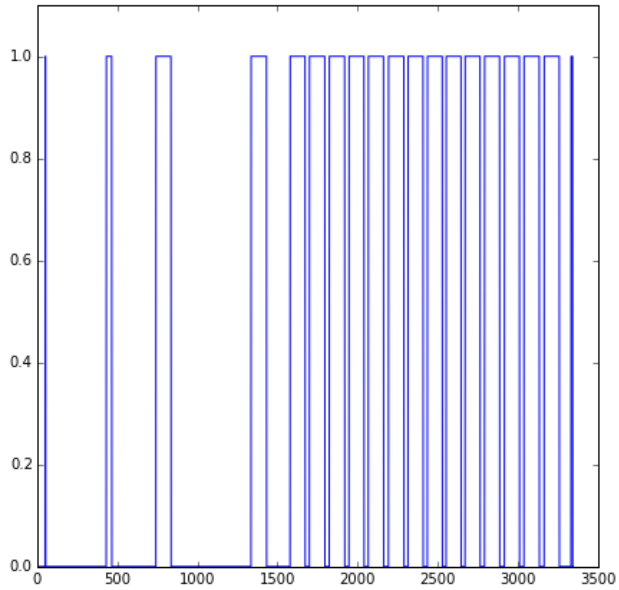

Figure 5: Thresholded histogram of the projected intensity values on the y-axis.

Since this process can select regions of noise and non price objects above the receipt, we filter these objects computing the median of the height of each object in the selected windows and then removing the ones whose height falls outside a confidence value. After this step, the first algorithm is used again to find the windows centred around each price.

3.5 Recognition

The recognition of each segmented price is implemented using Tesseract [2], an open source Optical Character Recognition (OCR) Engine. Each segmented window of prices is fed directly into Tesseract, which then returns a string of text found in the image. To improve its accuracy we selected an appropriate segmentation method instructing Tesseract to treat each image as a single word. By default Tesseract is optimized to recognize sentences of words so we also restricted the characters that Tesseract is able to recognize, since we are trying to recognize prices and we expect the input to contains only digits, points, commas and the minus sign.

3.5.1 Tesseract OCR

Tesseract was originally developed at Hewlett-Packard Laboratories between 1985 and 1994. In 2005 it was open sourced by HP and since then its developing is continued by Google. Tesseract has unicode (UTF-8) support, and can recognize more than 100 languages. It can also be trained to recognize other languages. A lot of the code was written in C, and then some more was written in C++; some unofficial wrappers exist which allow its use Python.

3.6 Error Correction

Despite all the precautions already taken and discussed, the pipeline described is not fool-proof and some adjustments are needed to avoid the more common errors. For example, the segmentation process often can select objects that aren't prices like the word "EURO" or informations about the supermarket above the price list. To remove these objects we first apply Tesseract in its default mode, recognizing both letters and numbers, and then we delete the object windows in which no number was found. Also, if no comma was found in a price we segment each digit of the price and we do a template matching of the comma, replacing the found character. In the end we try to interpret the string of text prices as numbers. If any string of text has not a standard number format, it's discarded. As last step the sum of all prices found is checked with the total price and if their values don't match the user has the option to manually correct any error.

4 Results and Conclusions

We tested the system on various and different receipts from a range of Italian stores. Analysing a total of 9 receipts, the system correctly recognized the price list 6 times after automatic error correction. The other times, some of the errors where caused by erroneous recognition by the OCR engine (the most common error is recognising a 6 as an 8) while the other times the error correction algorithm failed to ignore non price objects.

Some precautions such as good lighting conditions and image focus must be taken while taking the photographs to achieve good results because currently the system recognizes the receipts with very high precision if there are no connected or broken characters. Also, the receipt should be as straight as possible in the image since the system doesn't correct rotation or de-skewing. As possible solution to this problem we could add a stage in which

we correct the rotation/de-skewing of the image maximizing the variance of the brightness sum over the rows of strings. Every stage of recognition in particular situations can face many subproblems, that are fully or partially related to each other. In the end, while the system achieved good performance, there's still a lot of room for improvement.

References

[1] G. Bradski. *Dr. Dobb's Journal of Software Tools.*

[2] Ray Smith. An overview of the tesseract ocr engine. 2007.