Tabla: A framework for Extracting Tables From Images

Ayush Jain, Shubanshu Agarwal, Suyash Mahar

Abstract—Extracting tables from images is a very important task but has limited open source support, currently available frameworks are limited in either performance or usability. The aim of this work is to create a new framework that improves usability while producing high-quality results. We use basic image processing techniques like erosion/dilation, binarization, adaptive thresholding and inversion along with optimizations for broken and misaligned lines to show how even simple methods can result in high-quality results. We find that our tool while being extremely user-friendly in the sense that it doesn't require any tuning from the user, produces high quality results and also supports extracting merged cells in a table. These requirements make Tabla a powerful and versatile tool. Thus we hope that Tabla would help ease the effort required to extract tables from images by being user-friendly as it doesn't require and input other than the image and by using simple image processing to create accurate results.

Index Terms—Table Extraction, OCR, Image Processing

1 Introduction

The need for Tabla arose from the fact that current open source table extractors are severely limited in terms of their performance, the limitations of two of the most popular implementations are listed below:

- **pdftabextract** [1], [2] This framework was developed as a tool to accelerate extraction of information for analysing and archiving, however it suffers from the following major limitations:
  - The framework's performance is limited when a number of fine lines are present in the image, it also classifies text as table border when letter spacing is less than the usual amount.
  - Another limitation is the manual input required by the user for every image to tune the hyper-parameters of the algorithm, this severely limits the usability and hampers productivity.
- **Tabula** [3] This is a full-fledged package for extracting information (i.e. we cannot get intermediate results), however this package suffers from some serious limitation:
  - This framework works only if the text is already embedded in the PDF file and there is no open-source tool available to detect text and embed it directly into the PDF file. This has also been an open GitHub issue on its repository. [4].

Thus, we present 'Tabla', a framework that overcomes all these limitation by the use of simple image processing algorithms and techniques to extract table from an image. Tabla does so without any manual input from the user, easing usability and is highly accurate for images where other frameworks/algorithms fail.

The rest of the paper is: Section 2 talks about how different existing technologies are used, this is followed by detailed explanation of the working of Tabla in Section 3 and finally the results are presented in Section 4.

- *Ayush Jain: 16116013*
- *Shubhanshu Agarwal: 16118078*
- *Suyash Mahar: 16116069*

2 Technology

TABLE 1
Overview of the technologies used

Language	C++ 2011 Standard
Build Tool	CMake (Minimum version 3.10)

2.1 OpenCV [5]

OpenCV being a very powerful tool is extensively used across our project for a wide variety of tasks, some of them are:

- Binary inversion with adaptive Gaussian noise removal filter.
- Contour mapping for identifying the table borders.
- **Morphological operations:** Erosion and Dilation [6] to anything except corresponding to a particular direction, either horizontal or vertical.
- Custom operations over OpenCV matrix of the secluded horizontal lines image and the secluded vertical lines image to extract line coordinates.

2.2 Tesseract-OCR [7]

We also make extensive use of the `tesseract-ocr` library for the following purposes:

- Detecting of blocks of words in the image.
- Detecting bounding boxes of each block of words.
- Language specific (by default, English) LSTM prediction in OCR.

3 WORKING

3.1 Pre-processing

The most important tool used to build this project is OpenCV, an open-source library written in C/C++ with highly optimized algorithms. It has support for almost all the traditional image processing algorithms, as well as, interfaces and modules to use deep learning techniques. For text character recognition, Google's `tesseract-ocr` is being used with customized settings so as to not only detect the word blocks but the pixel coordinates of their bounding boxes too.

3.1.1 Reading an Image

Recommended image type is *Portable Network Graphics (PNG)* for proper recognition of pixel data in case of character recognition. Since a very high-resolution image will increase the execution time, it is expected the image resolution is not greater than 0.4MP.

3.1.2 Grayscaling

To reduce the computational complexity of processing an image, a three-channel RGB image is converted to greyscale using OpenCV, which uses the following relation

$$Y = 0.299R + 0.587G + 0.114B \tag{1}$$

3.1.3 Preparing the image

Real world data is noisy, and a scanned image can have varying intensity in the background itself due to noise in the scanner or cleanliness of the paper that is to be scanned. Since an image of a text-only table does not require intensity variations for distinguishing the data from its background. Thus, the image is binarized with thresholds defined using *adaptive thresholding*, that calculates different thresholds for small chunks of the image, thereby, increasing the accuracy of image binarization for images having varying illumination.

3.2 Morphological operations

An important step to recognize the table structure in the image is to segregate all the horizontal and vertical structures. Morphological operations like *erosion* and *dilation* when applied with proper kernels can extract straight lines along both the axes.

In case of a curve deviating from the structuring element, an *erosion* operation will remove all the pixel data from that small chunk on which the kernel is applied. Otherwise, for a linear curve along the structuring element that is completely covering the kernel, all the pixel data is set to the maximum value present. The subsequent dilation operation then smooths out the data along the structure by setting the values to maximum if the line is present.

3.3 Edge Recognition and Localization

After segregating the horizontal lines and vertical lines, it is required to localize the endpoints of all the lines, so that, when compared to the detected text's location, the corresponding table cell location can be deduced.

3.3.1 Image traversal

The image is traversed by looping a *pool window*, $k \times k$ kernel, like a look-up lens, on which all the subsequent operations will be executed. This means some linear function over all the grayscale values of the pixels present in the pool will decide whether the centre point represents an endpoint and/or what must be the next step. The traversal of the pool window reduces the complexity as compared to the pixel by pixel traversal.

3.3.2 Detection of end points

A start point, in this case, can be defined as the location of the centre of the pool window for the first time when the white region[1] appears. The appearance of the white region means that it can either be a part of some line or some noise. For acknowledging the endpoint, a threshold is set for the number of white pixels in the white region. Moreover, a buffer of some n number of pool windows is checked if they have crossed the threshold or not. This helps in handling minor glitches or small discontinuities present on the line. Figure 2 explains how introducing a buffer of $n = 3$ pool windows to check for the required region can help in ignoring noise or glitches. Similarly, for locating the end point, the first time when the white region starts disappearing from the pool window can be the required endpoint. Again, buffer handles glitches or noise, if present.

3.3.3 Handling corner cases

Many a time, while traversing the image k pixels at a time, a line might just fall on the edge or one corner of the pool window. In that case, the pool might skip and traverse further, since the threshold for the amount of white region was not crossed. Figure 3 shows that by introducing a calibration, the pool window can be properly positioned to reduce the distance between the actual starting point of the line and the centre of that pool window.

3.3.4 Cleaning the generated output

Sometimes, a table contains a double-line border across its cells or as the main table boundary. Or sometimes, the distance between the detected lines is too small to accommodate any text between it. In such cases, judiciously merging them would solve the problem. This is explained in Figure 4, where the distance is compared to a threshold radius so as to detect the discrepancies.

3.4 Text Recognition and Localisation

`tesseract-ocr` with properly set modes can detect the blocks of text, expected as individual words. The tool also shows the confidence value of the detected text, which can be used to programmatically comment on the accuracy of the detected text.

1. For aesthetic purposes, illustrations present in the region have black lines over white background. In the project, the image is binary inverted.

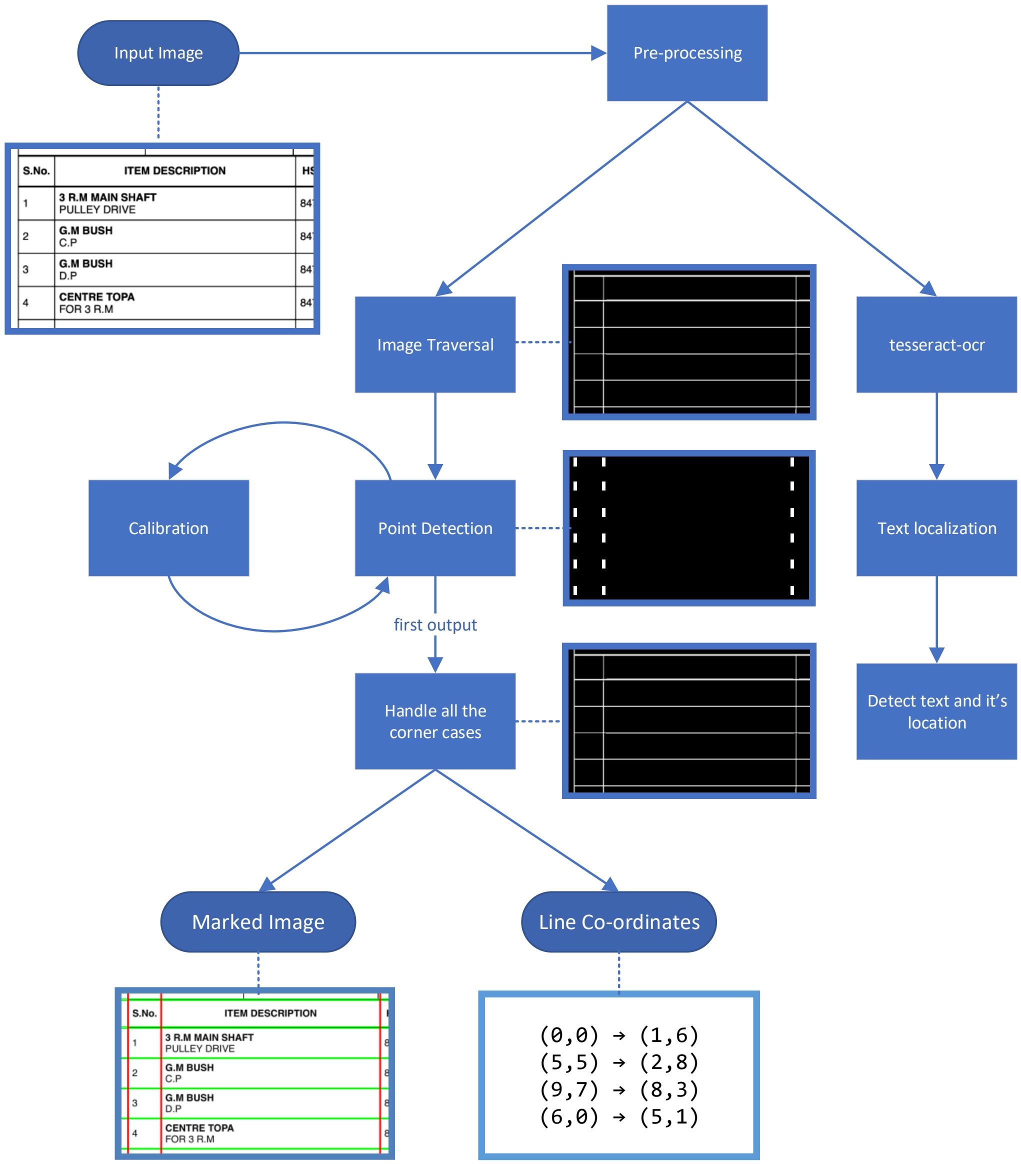

Fig. 1. Overview of the tabla algorithm with an example demonstrating various operations performed on the image.

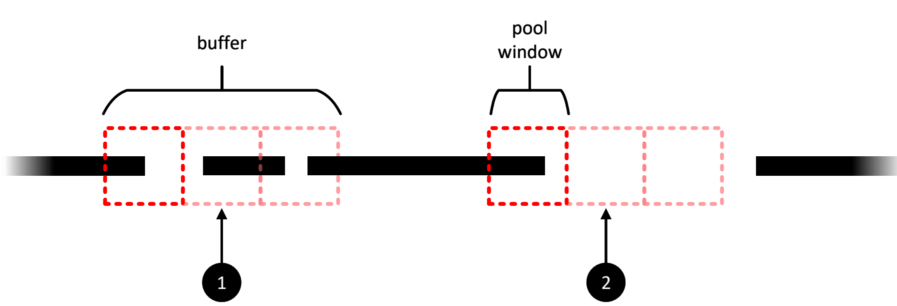

Fig. 2. Application of pool buffer. Checking the subsequent pool windows indicates whether it's an endpoint or a minor glitch. In 1: the buffer detects non-zero values in next pool windows, thus, indicating a glitch. And in 2: the buffer detects no values in the next pool windows, indicating that the current one is an endpoint.

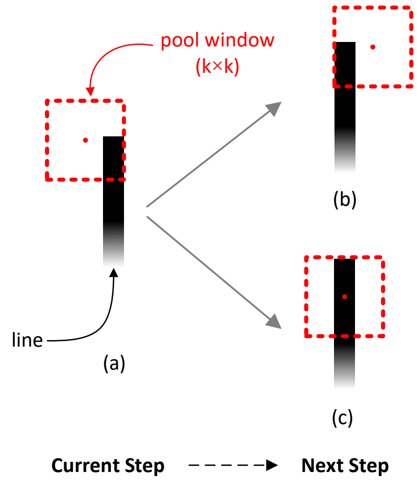

Fig. 3. Tabla calibrates the pool window to prevent windows with lower than threshold line content to be skipped. (a) Current state of the pool window. Notice how the line's end lies at the extreme corner. (b) **Without calibration:** Pool window might skip the detected region. (c) **With calibration:** Pool window calibrates to match its centre.

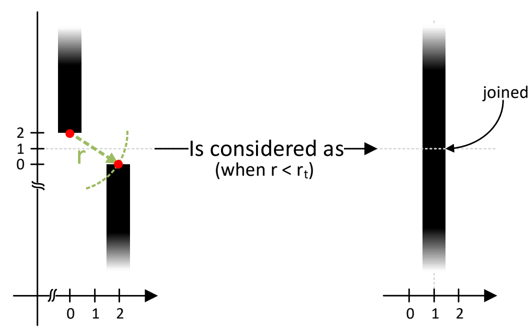

Fig. 4. Tabla merges lines that have distance between their endpoints and centre points on one of the axes, less than a certain thresholds.

3.5 Errors and Limitations

Despite the improved performance and simple optimizations Tabla's performance can be limited in a few cases, some of them are listed below:

- A glitch of size greater than the distance threshold cannot be handled.
- Complexity of the algorithm is not highly optimised, therefore, execution time is very high for high resolution images.
- Detected text using `tesseract-ocr` did not produce reliable results, some characters like `[`, `]`, `|`, etc., are additionally detected for the table edges.
- Works really well on high contrast images with white background, not so reliable otherwise.

4 Results

We have evaluated Tabla on a number of different images to evaulate it's effectiveness and performance, the results

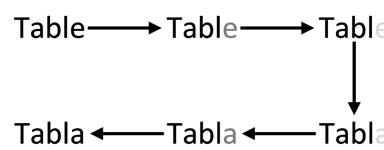

Fig. 5. Naming of our framework.

might vary significantly from image to image, keeping this and the length limitation we have presented 4 representative results in Table 2. The results show how Tabla is effective in extracting tables from images compared to the current state of the art (`pdftabextract`).

Conclusion and Future Work

Few of the future extensions of this project are listed below:

- Auto image formatting (like output provided by CamScanner [8], FastScanner [9], etc)
- De-skewing of slightly rotated image.
- Cell boundary confirmation using joints.
- Bolding edges through contrast manipulation.

References

[1] M. Konrad, "pdftabextract - A set of tools for data mining (OCR-processed) PDFs," https://github.com/WZBSocialScienceCenter/pdftabextract, 2017.
[2] ——, "Data Mining OCR PDFs - Using pdftabextract to liberate tabular data from scanned documents," Feb. 2017. [Online]. Available: https://datascience.blog.wzb.eu/2017/02/16/data-mining-ocr-pdfs-using-pdftabextract-to-liberate-tabular-data-from-scanned-documents/
[3] "Tabula is a tool for liberating data tables locked inside PDF files." Jun. 2018. [Online]. Available: {https://tabula.technology/}
[4] Tabulapdf, "Feature request : OCR mode: Integrate Tabula and Tesseract OCR to facilitate scanned tabular document OCR Issue #409 tabulapdf/tabula." [Online]. Available: https://github.com/tabulapdf/tabula/issues/409
[5] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.
[6] R. C. Gonzalez, "Richard e. woods," *Digital image processing*, vol. 2, pp. 550–570, 2002.
[7] Google LLC, "tesseract-ocr/tesseract," Apr. 2019. [Online]. Available: https://github.com/tesseract-ocr/tesseract
[8] "CamScanner." [Online]. Available: https://www.camscanner.com/
[9] "Fastscanner." [Online]. Available: https://www.coolmobilesolution.com/

5 Appendix

5.1 The Name 'Tabla'

After much deliberation and spending more time on naming than actually working on the project we decided to name our project 'Tabla', it's derived from the word `table` but has been modified to represent our framework. The letter `e` has been dropped (which signified **e**ffort that the user had to put in) and has been replaced by `a` to create `tabla`. The letter `a` in the new name signifies the framework's accessibility. This is demonstrated much clearly in the Figure 5.

TABLE 2
Comparison of the current state of the art method (`pdftabextract`) and Tabla over four representative images.

Original Image	Current state of the art method (`pdftabextract`)	Tabla