An OpenCV-based Framework for Table Information Extraction

Jiayi Yuan*, Hongye Li*, Meng Wang⊠†‡ Ruyang Liu§, Chuanyou Li†‡ Beilun Wang†‡
*College of Software Engineering, Southeast University, Nanjing, China
†MOE Key Laboratory of Computer Network and Information Integration, China
‡School of Computer Science and Engineering, Southeast University, Nanjing, China
§Chien-Shiung Wu College, Southeast University, Nanjing, China
meng.wang@seu.edu.cn

***Abstract*—Portable Document Format (PDF), as one of the most popular file format, is especially useful for educational documents such as text books, articles, or papers in which we can preserve the original graphic appearance and conveniently share online. Detecting and extracting information from tables in PDF files can provide a plethora of structural data to construct educational knowledge graphs. However, most of the existing methods rely on PDF parsing tools and natural language processing techniques, which generally require training samples and are frail in handling cross-page tables. In light of this, in this paper, we propose a novel OpenCV-based framework to extract the metadata and specific values from PDF tables. Specifically, we first highlight the visual outline of the tables. Then, we locate tables using horizontal and vertical lines and get the coordinates of tabular frames in each PDF page. Once the tables are successfully detected, for each table, we detect the cross-page scenarios and use the Optical Character Recognition (OCR) engine to extract the specific values in each table cell. Differing from other machine learning based methods, the proposed method can achieve table information extraction accurately without labeled data. We conduct extensive experiments on real-world PDF files. The results demonstrate that our approach can effectively deal with cross-page tables and only need 6.12 seconds on average to process a table.**

***Index Terms*—Information Extraction, PDF, OpenCV**

I. Introduction

As a ubiquitous form of information expression, the table is widely used on a variety of different platforms, including scientific publications, web pages and financial reports in PDF format. Tables represent the experimental or statistical data that researchers want to express in a very concise and clear structured form of two dimensions. Table detection and extraction are performed by many researchers to get the information including vital metadata and values, which can sometimes support the construction of knowledge graphs.

However, although it is not difficult for humans to recognize table information from PDF documents, it is difficult to extract these data automatically. The main challenge of extract this task is that table, due to its condensed fashion, usually only occupies a small part of the file so that it will be mixed with other elements, making it hard to find an appropriate approach to detect tables in advance. What's more, tables in PDF documents are untagged in most cases. The two points above render academics great difficulty to propose a suitable table extraction method.

The existing methods (e.g. Yildiz et al. [1] and Ramel et al. [2]) have successfully designed the table extraction technology to a certain extent, but restrictions are still in presence. The restrictions include dependence on the accuracy of PDF to HTML conversion tools, and predefined assumptions on PDF documents. Also, we found that few articles mentioned the extraction of tables with color, ***side-by-side tables*** and ***cross-page tables*** (i.e., tables spanning multiple pages). To overcome these limitations, we proposed our method that not only performs table detection and extraction automatically in electronic documents but also adds processing steps for tables with special table elements and cross-page tables.

Our method consists of generally three consecutive steps for table understanding in PDF documents. Firstly, we implement preprocessing step on each page in PDF files in order to convert special elements in the table into elements we can handle. Secondly, we locate tables using horizontal and vertical lines and get the coordinates of tabular frames in each PDF page. Once the tables are successfully detected, for each table, we obtain all the cells in the table and use the Optical Character Recognition (OCR) engine to identify the characters in each table cell. After the series of operations, we successfully get the metadata (e.g., header, font, data type, coordinates) and data from tables in PDF. Since we extract the tables in the PDF according to the page order, when switching to the next page, we perform our algorithm to judge whether there are cross-page tables.

The contributions of this paper are summarized as below:

- We introduce an OpenCV based framework which is capable of extracting table metadata and data from PDF files automatically.
- We propose a model that can handle tables with colors during table detection and metadata extraction.
- We design an algorithm to identify different types of table including ***side-by-side tables*** and ***cross-page tables***.

This paper is organized as follows. Section II briefly introduces previous work on table processing and a few usage of the OpenCV library. Section III describes our table extraction method in detail. In Section IV we test the accuracy of our method and analyze failure cases and limitations. We conclude our work in Section V.

978-1-7281-8156-1/20/$31.00 ©2020 IEEE
DOI 10.1109/ICBK50248.2020.00093

II. Related Work

In this section, we will discuss previous work on the related research of table processing techniques and systems in the following aspects: the pioneering work on table information extraction, the usage of OpenCV, and the methods of PDF information extraction.

A. The Pioneering Work on Table Information Extraction

A number of surveys have appeared since two decades ago, though many of them have only focused on specific aspects of the problem. Lopresti and Nagy [3] found that the analysis of tables in electronic form could be an essential contributor to a higher level of table understanding. They also looked at the definition of "tabularity" and tabular browsing. Zanibbi et al. [4] researched the table recognition process, including table models, observations, transformations and inferences. They presented the table recognition literature as an interaction of the elements above. Embley et al. [5] analyzed the model proposed by Wang [6], proving that it is effective for useful tasks with less explicit information, and outlined their plans of developing a semi-automated table processing system.

In his Ph.D. thesis, Hurst [7] designed an abstract model for table understanding. The model includes 5 components: graphical, physical, functional, structural and semantic components. He [8] divided the table processing task into four procedures: table location, table recognition, functional and structural analysis, and interpretation, making it more convenient to interpret tables. His model is a competitive one, but he mainly concerned about the model itself rather than the detail of table extraction and understanding. Plus, his discussion of the process lacks the ability to locate tables in the document, which is an essential part of our method.

Different table extraction techniques have been used to solve problems in specific document formats, and table information is contained in various document formats, such as HTML, XML. Chen et al. [9] presented their work on extracting tables from large-scale HTML texts. They used string, named entity and number category similarity to judge whether it is a real table. Then they applied those metrics to calculate the cell similarity to judge how to read the table (row-wise or column-wise). Tengli et al. [10] focused on HTML pages in webpages. They used <table>along with other tags (<tr>, <td>, <th>, <caption>) to get the structure of the table and parse its contents. However, PDF files have no backend HTML or XML, their methods cannot be applied directly to PDF files.

Since the problem and application requirements are varied, the works on table extraction also have diverse methods. With regard to the particularity of the documents we deal with, our approach supports table extraction of untagged PDF files with a comprehensive process.

B. The Methods of PDF Table Information Extraction

As for one of the most widespread ways to represent non-editable documents, PDF files have rich table structures and information and several tools and measures for PDF table extraction have emerged. Ramel et al. [2] designed a method based on their representation scheme for tables, using graphic lines for detection and extraction of tables. They also proposed a method dealing tables with little graphic marks by making use of the regularities of the text elements alone.

TableSeer is a table search engine proposed by Liu et al. [11]. It crawls scientific documents (mainly PDF documents) from digital libraries, identifies tables from documents, indexes and ranks tables, and provides a search interface for users. It is built with a box-cutting method that could be used for table detection using table captions, fonts, whitespace, and other elements. Although the system is a competitive one, it relies heavily on precision. It assumes that all tables have captions, as a result, it fails under the condition when the tables are labeled with other keywords, leading to a low recall rate. Similarly, Perez-Arriaga et al. [12] proposed system *TAO* (TAble Organization), which is capable of automatically detecting, extracting and organizing information from tables in PDF documents. However, although *TAO* is superior to *TableSeer* in both table detection and table recognition, this system tends to aggregate global features in a compact representation, making it less suitable for performing object detection.

Graphical models in [13] and [14] have also been used to seek performance improvement. In [14], Pinto et al. used Conditional Random Fields (CRFs), which are undirected graphical models that can be used to calculate the conditional probability of values on designated input nodes, for table extraction. They labeled each line of a document with a tag, marking the boundaries of tables and trained their model to recognize the boundaries, row-column divisions and header cells. The test result was good, but the model knows nothing about columns and is unable to distinguish between data and header cells.

Researchers also tried to convert PDF to XML for table extraction. Yildiz et al. [1] proposed a *pdf2table* system, in which they used the data returned by the *pdftohtml* tool for table recognition (identifying a "construct" as a table). Then they decomposed tables to find out the information in them, including header elements, spanning behavior (i.e., how many columns or rows are spanned), etc. Although it has a graphical user interface to let users make adjustments and overcome limitations, the quality largely depends on the *pdftohtml* tool, under particular circumstances (e.g., the table for extraction is an image) the tool returns no useful information, which cannot be fitted by users.

C. The Related Usage of OpenCV

OpenCV [15] is an open-source computer vision library. It provides many functions, which effectively implements many computer vision algorithms.

Image processing has always been one of the most important functions of OpenCV, and edge detection is a significant segment of it. Xie and Lu [16] implemented a method based on OpenCV to detect and count the exact number of the copper core in the tiny wire. They used morphological opening and closing operations to segment the high-resolution image and count the number through contour tracking. Pásztó and

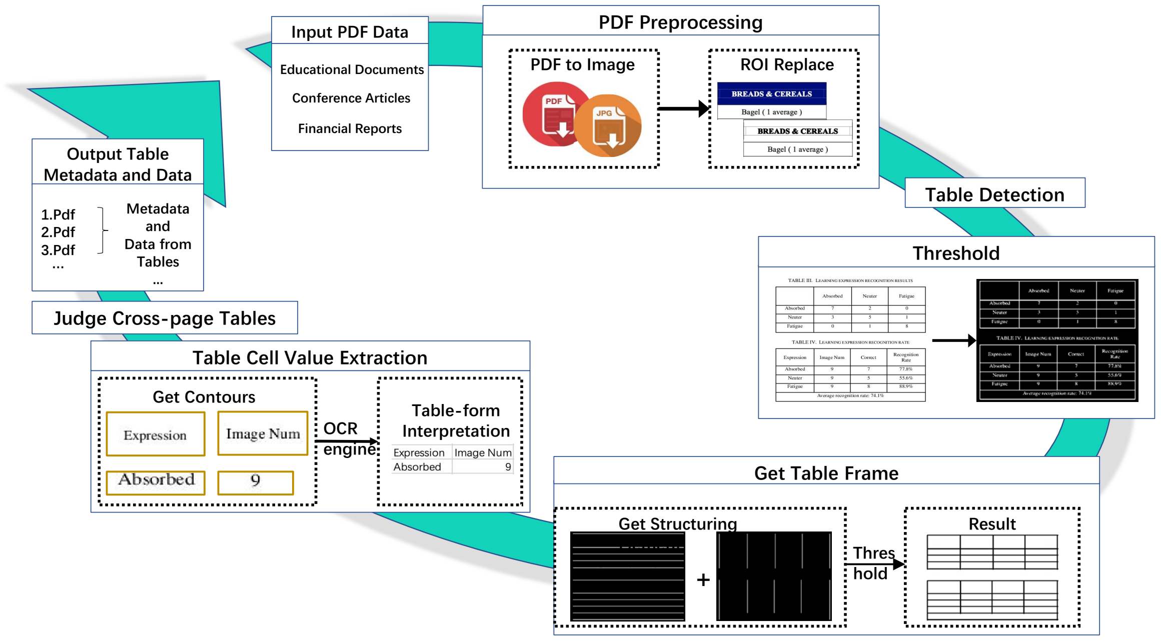


Fig. 1. The Architecture of Our Workflow

Hubinský [17] discussed the possibility of using visual systems for mobile robot navigation. They utilized the Hough transform[1] to detect circles and lines, which is a part of sign detection performing on robot.

Color detection, which is also widely used in OpenCV, is utilized in the field of Traffic Sign Recognition (TSR) in [18], [19], [20] and Skin Detection [21], [22], [23]. In [21], Oliveira and Conci initially convert RGB images to HSV color space to characterize the colors range for skin detection.

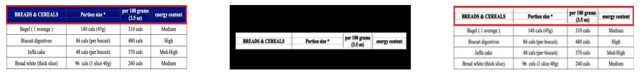
To perform OpenCV, we mainly focus on color detection for PDF preprocessing and horizontal and vertical line detection for table recognition and table cell value extraction.

III. Methodology

We define the workflow of the method as illustrated in Figure 1. The proposed method is composed of PDF preprocessing, table detection and table cell value extraction.

1) **PDF Preprocessing**, to convert PDF documents to images, and conduct pre-processing to clarify the outline of Region Of Interest (ROI).
2) **Table Detection**, to find tables on each page of PDF files using OpenCV based on the images obtained above, and get the information of all the contours in each image.
3) **Table Cell Value Extraction**, to find the cell of each table and extract text from it, at the same time, determine whether there are ***cross-page tables***, if so, merge with the table on the previous page. Through the judgments and table-form interpretation, we get the metadata and data from tables in PDF.

[1]https://en.wikipedia.org/wiki/Hough_transform



(a) Original Image (b) Saturated Image (c) Replaced Image

Fig. 2. Three Types of Image at Different Stages

We utilize the OpenCV library to get access to many prebuilt computer vision functions. Tesseract OCR engine is also used to provide the support for table cell value extraction. Specifically, we will explain how they work as follows.

A. PDF Preprocessing

Firstly, since OpenCV cannot be applied to process the PDF documents directly, we convert PDF files to images page by page.

In special cases, the tables would contain high saturated colors in its rows and columns, making it difficult to apply line detection for table recognition. We use the following steps to solve this problem:

1) **Extract Saturation**. Change colorspace of the "original image"(shown in Figure 2(a)) from BGR (Blue, Green, Red) to HSV (Hue, Saturation, Value) and extract saturation, save the result as "saturated image"(shown in Figure 2(b)).
2) **Obtain Binary Image**. Load the "saturated image", apply grayscale, Gaussian blur[2], then Otsu's threshold to the image.

[2]https://en.wikipedia.org/wiki/Gaussian_blur

3) **Detect Region of Interest (ROI)**. Find contours and then filter using contour approximation with contour perimeter and polygonal curves with the specified precision. With the assumption that the region is a rectangle, if the contour approximation result is **4** then the desired regions are found. Plus, we calculate the area of the contour to filter the noise.
4) **Replace ROI into the Original Image**. Obtain the bounding box coordinates and extract the ROI with numpy slicing so that we can replace ROI into the "original image", save the image as "replaced image"(shown in Figure 2(c)).

Algorithm 1 PDF Preprocessing

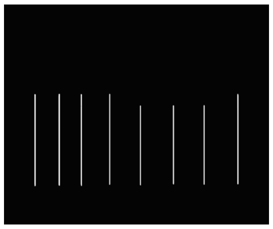
```
Input: A PDF document $D$
Output: A set of images $I$
    $Initialisation : I = \Phi$
    for each page $p$ in $D$ do
        $image \leftarrow pdf\_to\_image(p)$
        if $high\_saturated\_colors()$ then
            $image \leftarrow change\_image\_background(image)$
        end if
        $I = I \cup image$
    end for
    return $I$
```

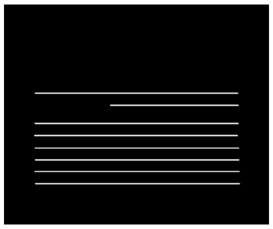
Considering the Algorithm 1, we assuming that *P* is the number of pages in a PDF file, and *C* is the number of contours to be replaced. The complexity of *pdf_to_image* operation is $\mathcal{O}(1)$, and the complexity *change_image_background* operation is $\mathcal{O}(C)$, since it changes every contour in a page that satisfies the requirement. Therefore, the complexity of Algorithm 1 is $\mathcal{O}(PC)$.

B. Table Detection

Next, we detect tables in the image and obtain their location based on the computer vision technology. When there are multiple tables in the image, we locate the tables and then cut them based on where the rows and columns intersect.

1) **Threshold the Picture**. In OpenCV, finding contours is like finding white objects from the black background. Because we focus more on tables, we use thresholding to convert table lines to white and other backgrounds to black. We perform threshold processing to complete it and we also choose binary images in order to gain higher accuracy,
2) **Identify Horizontal and Vertical Lines**. Then we utilize morphological operations to detect tables. The first thing is to define a rectangle kernel with the length based on the width of the image. We further define two kernels separately detecting the horizontal lines and vertical lines in the image. In this procedure, the key to identification is the morphological operations, which are a series of image processing operations based on shapes, producing output images by applying structural elements to the input images. The idea is using the kernels to do eroding and dilating successively. When trying to identify these lines, the strategy is to choose the most suitable horizontal kernels and vertical kernels. This is great for accurately identifying the lines and pinpointing tables in the image. The result is shown in Figure 3(a) and 3(b), which show the horizontal and vertical lines separately.


(a) Vertical Lines


(b) Horizontal Lines

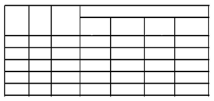

(c) Table Structure from Image

Fig. 3. The Result of Three Table Detection Stages

3) **Get All Contours**. After having obtained two pictures about all the horizontal and vertical lines respectively in the image, we get the sum of the two pictures. In this way, not only the frame of tables in the original image is gained, but also the information in original tables are filtered out. Therefore, we can accurately determine the table and alleviate the problem of noise resulting from wrong extraction. After the series of operations, we successfully get the location and shape of the tables in the image. The consequence is show in Figure 3(c).
Contours can be explained simply as a curve joining all the continuous points (along the boundary), having the same color or intensity. The contours are a useful tool for shape analysis and object detection and recognition [24]. We utilize *findContours* function to get all contours in the original image including the tables themselves and all cells in tables, and we also get the coordinates of cells as well as their width and height. For facilitating subsequent processing, we further sort each contour in an image using the top-to-bottom and left-to-right method.

C. Table Cell Value Extraction

When we have got the information of the table cells in the picture, we will use them effectively and read the text content from them to achieve the purpose of extracting the tables.

Tesseract, which is an optical character recognition (OCR) tool, is able to recognize and "read" the text embedded in different kinds of pictures in various font forms [25]. We utilize Tesseract as a tool to extract text from table cells.

1) **Distinguish Actual Table Cells and the Fake**. Before reading the text from cells, we distinguish the useful information from the noisy lines by checking each of the

obtained contours and judge its width and height obtained from previous steps. If the width or height is less than a certain value, we judge them to be miscellaneous lines rather than a cell in a table. When the filtering process has completed, for each table, we have useful contours, including the table itself and all cells in it. We determine whether it is a table or a cell based on the size of the contour: the larger one is the former and the smaller one is the latter. We will save the table information for later extraction of header and the judgment of cross-page tables.

2) **Read and Judge**. In order to obtain the metadata and data from tables, we present several judgments for table-form interpretation. For each cell, we use the coordinates obtained from the *boundingRect* function, that is, the coordinates of the upper left corner of the cell, as our X and Y coordinates. Since we have sorted the cells, we process each table cell in turn and compare the coordinate of this cell (x_1, y_1) with that of the previous cell (x_2, y_2) in order to solve the problems below.

a) An Independent Table: In most cases, there is only one table at the same height. Under this circumstance, we only need to judge whether two consecutive cells are in the same row or in the same table based on their Y coordinates. The detailed judgments are as follows:

- When $y_1 = y_2$, we consider that the two cells are in the same row. Otherwise, they are regarded in the different rows.
- If the difference between y_1 and y_2 is too large, we determine that they belong to two different tables.

b) Side-by-side Tables: All cells in the same row, whether in the same table or not, are read from left to right, and rows are read from top to bottom. However, when there are side-by-side tables, we cannot only base on Y coordinate to simply judge cells having the same (or very close) Y coordinate are in the same row.

Under this circumstance, when processing cells, we also need to judge their X coordinate. The idea of determining whether there are side-by-side tables is based on the difference of the X coordinate between two consecutive cells (i.e., $x_2 - x_1$). We also take the cell width obtained from the *boundingRect* function into consideration. In this part, we use T_1 and T_2 as representatives for the table on the left and right. We assume that the two tables belong to the two sides of the central axis of the entire page. The detailed rules for judging are as follows:

- If $x_2 - x_1$ is always within a reasonable interval, we hold the belief that there are no side-by-side tables. The hypothetical T_2 does not exist.
- Once $x_2 - x_1$ is much larger than the width of the first cell, it can be assumed that two consecutive cells belong to different tables. When processing each subsequent line, we firstly judge whether the cell being processed is in T_1 or T_2 based on which side of the central axis of the whole page they are on. If it is on the left side, we add it to T_1. If it is on the right side, we assume it is in T_2. After processing each cell, we will explicitly distinguish which table each cell is in.

Algorithm 2 Table Detection and Table Cell Value Extraction

```
Data: The set of images $I$
Result: Metadata and Data from tables in $I$
  for each $img \in \mathcal{I}$ do
    $img \leftarrow img$ after $cv2.\_threshold()$
    define $vertical\_kernel, horizontal\_kernel$
    $horizontal\_lines\_img, vertical\_lines\_img$ : result of
    morphological operations based on kernels
    $final\_img \leftarrow add\ the\ two\ images\ above$
    $final\_img \leftarrow final\_img$ after $cv2.\_threshold()$
    $contours \leftarrow cv2.findContours(final\_img)$
    sort $contours$ by method
    for each contour $c$ in $contours$ do
      if $c$ is the contour of a table then
        get $table\_head\_text$ from the area above $c$
        determine if there is a cross-page table
        determine if there are many tables in one page
      else if $c$ is the contour of a cell then
        extract $text$ from the cell
        judge if it is the same row with the previous cell
        determine if there are side-by-side tables
      end if
      get the metadata and data from tables in the $img$
    end for
    all metadata and data from tables in $I$
  end for
```

c) Table Header Extraction: As mentioned above, the extraction of the table's header is always considered necessary. Then we also set rules to make it possible before extract tables' contents. Because we already know the coordinates of each table cell, we only need to extract the text content of the specified area above the table as the header of the table.

d) Text Recognition: We use Tesseract to identify the text contents in the table cells to complete the last step of table-form interpretation.

3) **Judge Cross-page Tables**. When we have finished the table extraction of the first picture and want to continue processing the second one, it is necessary to make a key judgment about whether there are cross-page tables. If a cross-page table exists, that is, the last table in the first picture and the first table in the second picture are actually from the same table, then connect them. Otherwise, two tables are totally separate. In the method, the following rules are formulated to determine whether there are cross-page tables.

- Only tables on two consecutive pages may be cross-page tables.
- The second table must be the same width with the first table, then they can become cross-page tables.
- The bottom of the first table should be below its page.
- The top of the second table should be in the upper area

of its page.

Considering the Algorithm 2, we assume that P is the number of pages in a PDF file, and C is the number of contours to be processed. The complexity of *sort contours* operation is $\mathcal{O}(C \log C)$. Since the number of images is equal to the number of pages, the complexity of Algorithm 2 is $\mathcal{O}(PC \log C)$.

IV. Experiments and Analysis

A. Experimental Setup

In this section, we will introduce how we set up our experiment, including the introduction of datasets, the standards we used and the technologies we employed. Our codes and datasets are available at https://github.com/Sevenbule/An-OpenCV-based-Framework-for-Table-Information-Extraction.

Dataset. In our experiments, we use three datasets to test our algorithm. Tables in these datasets usually provide useful information for the construction of educational knowledge graphs. 1) 52 educational documents with 119 tables collect from Web. 2) 57 articles from different academic conferences of computer science with 191 tables. 3) The dataset of the fifth task in the evaluation task of the China Conference on Knowledge Graph and Semantic Computing 2019 (CCKS 2019), including financial reports from many companies in PDF format. Table I summarizes the detailed statistics of the dataset. What's more, many cross-page tables are involved in this dataset, enabling us to test our algorithm on judging cross-page tables.

Employed Technologies. The employed technologies in the experimental environment and their versions are listed below.

- Python (version 3.7.5).
- OpenCV Library of Python (version 3.4.2).
- Tesseract OCR engine (version 4.1.1).

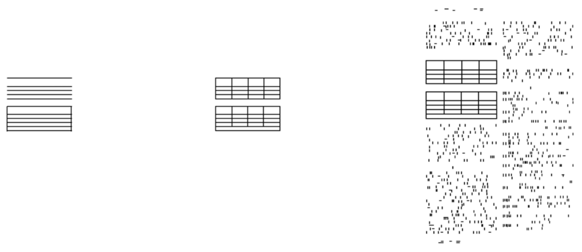
Standards. Because there is no ground truth provided, the correctness of comparisons are judged by human understanding. We use three standards: recall, precision and F1 score, to evaluate the performance of the last two steps of our table extraction method. They are calculated as below:

$$Recall = \frac{number\ of\ correctly\ identified\ tables}{number\ of\ tables\ in\ the\ PDF} \quad (1)$$

$$Precision = \frac{number\ of\ correctly\ identified\ tables}{number\ of\ identified\ tables} \quad (2)$$

$$F1 = \frac{2 * (Precision * Recall)}{Precision + Recall} \quad (3)$$

Since our main tasks are detecting tables and extracting metadata and data from tables in PDFs in order to find a better way for table-form interpretation (i.e., whether they are from the same table and in correct horizontal and vertical order), when we calculate the standards, the accuracy of *extracted text* in table cells, which relies heavily on the internal algorithms of Tesseract OCR engine, is not considered here.



(a) kernel_size=62 (b) kernel_size=15 (c) kernel_size=7

Fig. 4. Outcomes of Different Values of kernel_size

TABLE I
Description of Two Datasets

Dataset	Amount of Files	Amount of Tables	Amount of Cross-page Tables
Educational Documents	52	119	–
Conference Articles	57	191	–
CCKS Financial Reports	10	337	96

B. Baseline

We use the Camelot[3] engine to evaluate its performance to compare with our method. Camelot is a popular tool for extracting tables. At the same time, we also use Tabula[4] to test its performance. Tabula is able to detect tables in many forms, and when dealing with tables with ruling lines, they detect them with Hough Transform technique. The two tools above both have two types of work mechanism, *stream* and *lattice*. Because lattice mode is mainly implemented based on OpenCV, we use this mode to compare our algorithms with theirs.

C. Results Analysis

In this section, we firstly do the analysis of the parameter sensitivity and then analyze our experimental results.

Parameter Sensitivity. When doing table detection, the size of the kernel plays a crucial role in detecting the horizontal and vertical lines, thus having a huge impact on the precision of recognizing the cells in tables.

The example Figure 4 is shown to clarify the influence of the size of the kernel. Through our experiments, we can draw a conclusion that the size of the kernel has a greater influence on vertical line extraction than horizontal line extraction. We separate the influence into two categories:

- Small kernel size leads to more noisy vertical lines appearing on the image. This is because characters like '1','I' or 'T' are contained in texts, leading to misrecognition of OpenCV.
- Large kernel size leads to the loss of table lines, especially vertical lines.

[3]https://camelot-py.readthedocs.io/en/master/
[4]https://tabula.technology/

TABLE II
RESULTS OF TABLE EXTRACTION

Dateset	Method	Recall	Precision	F1 score
Educational Documents	Ours	79.8%	85.6%	82.6%
CCKS Financial Reports	Ours	82.8%	90.3%	86.4%
Conference Articles	Ours	82.2%	94.0%	87.7%
	Tabula (lattice)	51.5%	61.3%	56.0%
	Camelot (lattice)	50.6%	42.3%	46.1%

TABLE III
RESULTS OF JUDGMENT ON CROSS-PAGE TABLES

Amount of Cross-page tables	Correctly Recognized	Recall
96	84	87.5%

Therefore, we set a proper value based on the width of the image and get relatively satisfactory results.

Results. The results of table extraction with different methods on the datasets are shown in Table II. And the result of the judgment on cross-page tables as shown in Table III.

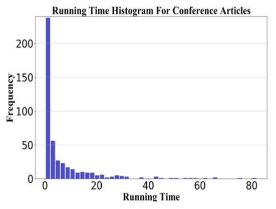
Our method obtained more than 85% F1 score at around 82% precision and more than 90% recall. For Camelot and Tabula, they only work on text-based PDFs and sometimes fails to extract tables that contain rows or columns spanning several cells. As a result, we only test them on conference articles. Despite being tested only on one dataset, Tabula and Camelot performed poorly.

In Tabula, the *autoselect* feature (i.e., the table position in PDF will be automatically framed) is useful for table extraction, but it will fail in particular cases, which causes the last row of the table to be undetected when we identify the table. Using Camelot specific text areas are mistaken for tables. The reasons above lead to bad performances.

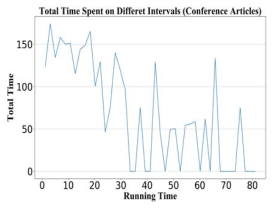
Time Efficiency Analysis[5]. In the previous sections, we analyzed the time complexity of our algorithm. In this part we give the time efficiency a more in-depth analysis.

Since the number of tables contained in different pages varies, when we applied our method to PDF files, the time efficiency of extracting tables from different pages may vary greatly. The spread of the recognition time is illustrated by a histogram (shown in Figure 5(a) and 5(c)) of the number of pages with the recognition time that fell into one of the intervals. We also draw line charts (shown in Figure 5(b) and 5(d)) to demonstrate the total time spent on the recognition of all pages from the corresponding interval. Because there are relatively more tables on each page of financial reports, the processing time of each page is relatively long: recognition of all 508 pages takes 10350.13 seconds, and the average recognition time of one page is 20.37 seconds. The time efficiency of processing conference papers is relatively higher: the total time for processing 456 pages is 2795.25 seconds, with an average processing time of 6.12 seconds per page.

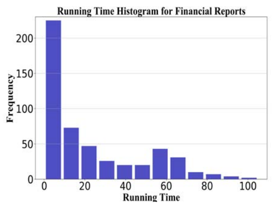
[5]Time performance evaluation was conducted on a single thread on Macintosh OS X 10.15.4 with two 3.1GHz Intel core i5



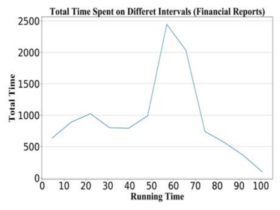
(a) Conference Articles Histogram



(b) Conference Articles Line Chart



(c) CCKS Histogram



(d) CCKS Line Chart

Fig. 5. Statistics of Experiment Time

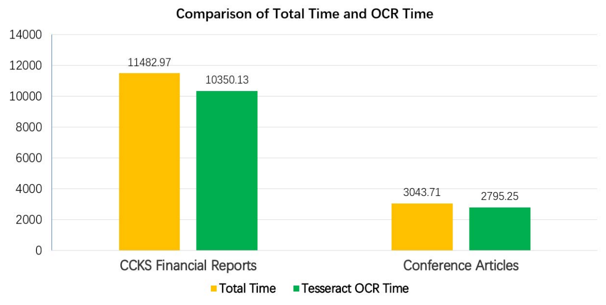


Fig. 6. Comparison of Total Time and OCR Time

Although time complexity is an important aspect of the time efficiency of an algorithm, we cannot completely rely on it because the time efficiency of certain steps in the algorithm is not high. Therefore, we give Tesseract OCR step, which is an indispensable part of our method, careful consideration. It turns out that OCR takes up most of the time (shown in Figure 6).

Failure Case. When testing our method on the datasets, we found that we couldn't successfully extract any table from specific PDF files (shown in Figure 7), which also affected our accuracy of extracting forms to a certain degree. As we continue to examine these files, we find that the tables contained in these files are all of one style - the height of the cells contained in them are very small, making it hard to reach the lower limit of cell height standard, which is used to filter useless contours. Also, these special style tables contain many discontinuous lines, making the size of the kernel too large for them to be recognized. Therefore, due to the preset standard, no content can be extracted from these tables. To address this problem, we plan to set up a kernel size that adapts automatically to the table styles.

Limitation Analysis. Although we established rules to extract tables and recognize the contents including the judgment

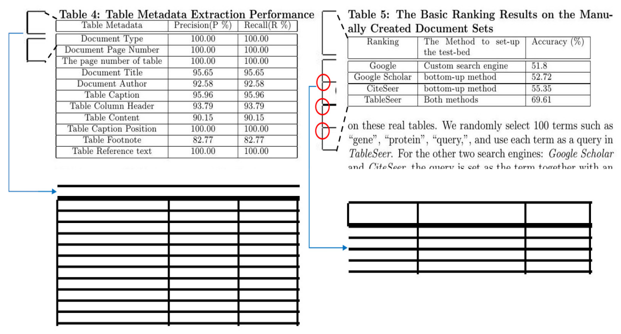


Table 4: Table Metadata Extraction Performance

Table Metadata	Precision(P %)	Recall(R %)
Document Type	100.00	100.00
Document Page Number	100.00	100.00
The page number of table	100.00	100.00
Document Title	95.65	95.65
Document Author	92.58	92.58
Table Caption	95.96	95.96
Table Column Header	93.79	93.79
Table Content	90.15	90.15
Table Caption Position	100.00	100.00
Table Footnote	82.77	82.77
Table Reference text	100.00	100.00

Table 5: The Basic Ranking Results on the Manually Created Document Sets

Ranking	The Method to set-up the test-bed	Accuracy (%)
Google	Custom search engine	51.8
Google Scholar	bottom-up method	52.72
CiteSeer	bottom-up method	55.35
TableSeer	Both methods	69.61

on these real tables. We randomly select 100 terms such as "gene", "protein", "query,", and use each term as a query in *TableSeer*. For the other two search engines: *Google Scholar* and *CiteSeer* the query is set as the term together with an

Fig. 7. The example of specific PDF tables. The line of the left table is not correctly identified due to the cell height of the first line, and the red circles demonstrate the discontinuity of the table line on the right table, leading to misrecognition of table line.

of cross-page tables and side-by-side tables, errors are still inevitable in the following case. 1) When the width of the table cell is less than the range we set, the cell will be regarded as disordered lines and filtered out. 2) When there are side-by-side tables, but the distance between them is very small, they will mix up and can not be recognized correctly. 3) Specific tables may be mistaken for cross-page tables when they satisfy the rules we established. However, they may just have the same format but they have totally different content. 4) This method is not suitable for the situation where the table is almost frameless.

V. Conclusion and Future Work

In this paper, we propose an effective method for detecting and extracting tables in PDF files via OpenCV for horizontal and vertical line detection and Tesseract OCR engine for text recognition. We also design our own algorithm of preprocessing PDF files, eliminating noisy lines, and judging cross-page tables and side-by-side tables. Our method has achieved satisfactory accuracy. We notice that when it comes to tables without gridlines and three-line tables, the outcome of this method is difficult to guarantee, and when we have got the table cells, the accuracy of table content recognition is mainly based on our choice of Tesseract. In future work, we will try to introduce machine learning methods to build several kinds of target detection model that can be used to perform area detection and content recognition on complex table types. Also, we will try to achieve better recognition of text content in pictures with higher accuracy.

Acknowledgment

This work was supported by the National Science Foundation of China with Grant Nos. 61906037 and U1736204; National Key Research and Development Program of China with Grant Nos. 2018YFC0830201 and 2017YFB1002801.

References

[1] B. Yildiz, K. Kaiser, and S. Miksch, "pdf2table: A method to extract table information from pdf files," in *IICAI*, 2005, pp. 1773–1785.

[2] J.-Y. Ramel, M. Crucianu, N. Vincent, and C. Faure, "Detection, extraction and representation of tables," in *Seventh International Conference on Document Analysis and Recognition, 2003. Proceedings.* IEEE, 2003, pp. 374–378.

[3] D. Lopresti and G. Nagy, "A tabular survey of automated table processing," in *International Workshop on Graphics Recognition*. Springer, 1999, pp. 93–120.

[4] R. Zanibbi, D. Blostein, and J. Cordy, "A survey of table recognition: Models, observations, transformations, and inferences, 2003," *Online: http://www. cs. queensu. ca/˜ cordy/Papers/IJDAR_ Tables. pdf, Last Checked*, pp. 12–01, 2007.

[5] D. W. Embley, D. Lopresti, and G. Nagy, "Notes on contemporary table recognition," in *International Workshop on Document Analysis Systems*. Springer, 2006, pp. 164–175.

[6] X. Wang, "Tabular abstraction, editing, and formatting," 2016.

[7] M. F. Hurst, "The interpretation of tables in texts," 2000.

[8] M. Hurst, "Layout and language: Challenges for table understanding on the web," in *Proceedings of the International Workshop on Web Document Analysis*, 2001, pp. 27–30.

[9] H.-H. Chen, S.-C. Tsai, and J.-H. Tsai, "Mining tables from large scale html texts," in *Proceedings of the 18th conference on Computational linguistics-Volume 1*. Association for Computational Linguistics, 2000, pp. 166–172.

[10] A. Tengli, Y. Yang, and N. L. Ma, "Learning table extraction from examples," in *Proceedings of the 20th international conference on Computational Linguistics*. Association for Computational Linguistics, 2004, p. 987.

[11] Y. Liu, K. Bai, P. Mitra, and C. L. Giles, "Tableseer: automatic table metadata extraction and searching in digital libraries," in *Proceedings of the 7th ACM/IEEE-CS joint conference on Digital libraries*, 2007, pp. 91–100.

[12] M. O. Perez-Arriaga, T. Estrada, and S. Abad-Mota, "Tao: system for table detection and extraction from pdf documents," in *The Twenty-Ninth International Flairs Conference*, 2016.

[13] S. Shetty, H. Srinivasan, M. Beal, and S. Srihari, "Segmentation and labeling of documents using conditional random fields," in *Document Recognition and Retrieval XIV*, vol. 6500. International Society for Optics and Photonics, 2007, p. 65000U.

[14] D. Pinto, A. McCallum, X. Wei, and W. B. Croft, "Table extraction using conditional random fields," in *Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*, 2003, pp. 235–242.

[15] G. Bradski and A. Kaehler, *Learning OpenCV: Computer vision with the OpenCV library*. " O'Reilly Media, Inc.", 2008.

[16] G. Xie and W. Lu, "Image edge detection based on opencv," *International Journal of Electronics and Electrical Engineering*, vol. 1, no. 2, pp. 104–6, 2013.

[17] P. Pásztó and P. Hubinskỳ, "Application of a visual system for mobile robot navigation (opencv)," *AT&P Journal Plus*, vol. 1, pp. 62–64, 2010.

[18] P. Shopa, N. Sumitha, and P. Patra, "Traffic sign detection and recognition using opencv," in *International conference on information communication and embedded systems (ICICES2014)*. IEEE, 2014, pp. 1–6.

[19] M. Russell and S. Fischaber, "Opencv based road sign recognition on zynq," in *2013 11th IEEE International Conference on Industrial Informatics (INDIN)*. IEEE, 2013, pp. 596–601.

[20] A. Lorsakul and J. Suthakorn, "Traffic sign recognition for intelligent vehicle/driver assistance system using neural network on opencv," in *The 4th International Conference on Ubiquitous Robots and Ambient Intelligence*, 2007.

[21] V. Oliveira and A. Conci, "Skin detection using hsv color space," in *H. Pedrini, & J. Marques de Carvalho, Workshops of Sibgrapi*. Citeseer, 2009, pp. 1–2.

[22] K. Nikolskaia, N. Ezhova, A. Sinkov, and M. Medvedev, "Skin detection technique based on hsv color model and slic segmentation method⋆," in *Proceedings of the 4th Ural Workshop on Parallel, Distributed, and Cloud Computing for Young Scientists, Ural-PDC*, 2018, pp. 123–135.

[23] M. S. Kalas, "Real time face detection and tracking using opencv," *international journal of soft computing and Artificial Intelligence*, vol. 2, no. 1, pp. 41–44, 2014.

[24] A. Mordvintsev and K. Abid, "Opencv-python tutorials documentation," *Obtenido de https://media. readthedocs. org/pdf/opencv-python-tutroals/latest/opencv-python-tutroals. pdf*, 2014.

[25] https://pypi.org/project/pytesseract/.