ĐẠI HỌC QUỐC GIA HÀ NỘI
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ

Trần Quang Vinh

NHẬN DIỆN CẤU TRÚC BẢNG TRONG ẢNH TÀI LIỆU

KHÓA LUẬN TỐT NGHIỆP ĐẠI HỌC HỆ CHÍNH QUY
Ngành: Công nghệ thông tin

HÀ NỘI - 2021

ĐẠI HỌC QUỐC GIA HÀ NỘI
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ

TRẦN QUANG VINH

NHẬN DIỆN CẤU TRÚC BẢNG TRONG ẢNH TÀI LIỆU

KHÓA LUẬN TỐT NGHIỆP ĐẠI HỌC HỆ CHÍNH QUY
Ngành: Công nghệ thông tin

Cán bộ hướng dẫn: TS. Nguyễn Thị Ngọc Diệp

Cán bộ đồng hướng dẫn: PGS.TS. Nguyễn Việt Hà

HÀ NỘI - 2021

VIETNAM NATIONAL UNIVERSITY, HA NOI
UNIVERSITY OF ENGINEERING AND TECHNOLOGY

Tran Quang Vinh

TABLE STRUCTURE RECOGNITION IN DOCUMENT IMAGES

BACHELOR'S THESIS
Major: Information Technology

Supervisor: Dr. Nguyen Thi Ngoc Diep

Co-Supervisor: Assoc. Prof. Nguyen Viet Ha

HANOI - 2021

ACKNOWLEDGEMENTS

Firstly, I would like to express my sincere appreciation to my advisor, Dr. Nguyen Thi Ngoc Diep, who has guided and assisted me during the making of this thesis. Her valuable feedback and advice have provided me the tools I needed to complete this research. Without her guidance, this work would not have been possible.

In addition, I would like to thank all of the members of the Faculty of Information and Technology, especially Assoc. Prof. Nguyen Viet Ha, for providing me with invaluable knowledge during my college years. Moreover, my gratitude also goes to my lab mates and classmates, especially those in K62-CACLC3. You are the greatest motivation for me to finish this thesis.

Last but not least, I would like to acknowledge the support and encouragement from my friends and family throughout my study.

TÓM TẮT

Bảng là phương tiện biểu diễn thông tin có cấu trúc phổ biến trong các loại tài liệu. Mặc dù đã được nghiên cứu từ rất lâu nhưng hiện tại, người ta vẫn chưa có cách nào để nhận diện cấu trúc bảng tự động một cách hiệu quả. Việc này khiến cho việc khôi phục, tái sử dụng các bảng từ các tài liệu trở nên tốn rất nhiều thời gian. Gần đây, với sự xuất sắc của kỹ thuật học sâu trong các bài toán nhận diện vật thể, nhiều phương pháp nhận diện cấu trúc bảng bằng mạng nơ ron tích chập đã được giới thiệu và đã đạt được độ chính xác cao hơn các phương pháp truyền thống. Tuy nhiên, việc thiếu các bộ dữ liệu lớn đủ để huấn luyện khiến cho các phương pháp này chưa thể đạt được độ chính xác cao. Khóa luận này sẽ đề xuất sử dụng kỹ thuật học sâu để giải quyết bài toán nhận diện cấu trúc bảng. Đầu tiên, khóa luận sẽ đề xuất phương pháp xử lý bộ dữ liệu nhận diện cấu trúc bảng TableBank được dùng để huấn luyện mô hình học sâu. Bộ dữ liệu này sẽ được xử lý bằng hai kỹ thuật xử lý ảnh và học sâu. Sau đó, bộ dữ liệu này sẽ được dùng với một mô hình mạng nơ ron tích chập để nhận diện từng ô của bảng. Khóa luận này cũng sẽ so sánh hiệu quả của việc dùng một bộ dữ liệu lớn trong việc nhận diện bảng và cấu trúc của bảng so với các phương pháp khác.

Từ khóa: *nhận diện bảng, nhận diện cấu trúc bảng, hiểu biết về bảng*

ABSTRACT

Table is one of the most common ways to represent structured data in documents. Although many researches have been conducted for many years, there is still no effective way to recognize table structure automatically. This turns recovering and reusing tables from documents time into time-consuming tasks. Recently, with the success of deep learning techniques in object detection, many approaches using convolutional neural networks have been proposed and they achieved better accuracy than traditional methods. However, the lack of large datasets for training models in table structure recognition task have hindered their accuracy. This thesis will propose using deep learning techniques to solve this task. First, this thesis will propose a method to process table structure recognition dataset of TableBank, using both digital image processing and deep learning techniques. Then, the dataset will be used to train a convolutional neural network to detect table cells. This thesis will also compare the effects of huge dataset in table detection and structure recognition with other methods.

Keywords: *table detection, table structure recognition, table understanding*

STATUTORY DECLARATION

I hereby declare that the work contained in this thesis is of my own and has not been previously submitted for a degree or diploma at this or any other higher education institution. To the best of my knowledge and belief, the thesis contains no materials previously published or written by another person except where due reference or acknowledgement is made.

Hanoi, April 29, 2021
Student

Tran Quang Vinh

Contents

List of Tables viii

List of Abbreviations ix

List of Figures x

Chapter 1 Introduction 1

1.1 Background of research 1

1.2 Problem statement 2

1.3 Targets and methods 3

1.3.1 Targets 3

1.3.2 Methods 3

1.4 Thesis' structure 3

Chapter 2 Related works 5

2.1 Challenges 5

2.2 Applicability 7

2.3 Applicability after improvements 9

2.4 Approaches 10

2.4.1 Rule-based approaches 11

2.4.2 Data-driven approaches 12

2.5 Limitations of other approaches 14

Chapter 3 Background **17**

3.1 Table 17

3.2 R-CNN and its extensions 18

3.3 Intersection over Union 19

3.4 Cascade Mask R-CNN 20

3.5 Morphological Transformations 21

Chapter 4 Methods **23**

4.1 Method 1 24

4.2 Method 2 28

4.3 Method 3 33

4.4 Method 4 36

4.5 Checking the annotations 39

4.5.1 Method 1: Checking the structure 39

4.5.2 Method 2: Checking the number of cells 40

4.6 Dataset preparation 40

Chapter 5 Experiments and Results **43**

5.1 Experiments 43

5.1.1 Table Detection 43

5.1.2 Table Structure Recognition 43

5.2 Results 44

5.2.1 Table Detection 44

5.2.2 Table Structure Recognition 46

Chapter 6 Conclusion **48**

List of Tables

2.1 Number of tables in popular table structure recognition dataset 15

4.1 Final result on annotation generation. 40

5.1 Results on combined TableBank dataset 44

5.2 Results on LaTeX TableBank dataset 45

5.3 F1-scores on ICDAR19 Track A (Modern) 46

5.4 F1-scores on ICDAR19 Track B2 (Modern) 47

List of Abbreviations

CNN	Convolutional Neural Network
CRAFT	Character-Region Awareness For Text detection
HTML	HyperText Markup Language
IoU	Intersection over Union
mAP	mean Average Precision
PDF	Portable Document Format
R-CNN	Region Based Convolutional Neural Networks

List of Figures

1.1 An example of a table. 2

2.1 Different types of table. 7

3.1 Structure of a table. 18
3.2 Architecture of Faster R-CNN and Cascade R-CNN 21
3.3 Morphological transformation operators. 22

4.1 Overall process of annotation generation. 24
4.2 Process of method 1. 25
4.3 A working example of method 1. 27
4.4 An example that method 1 fails to detect correctly. 28
4.5 The process of method 2. 28
4.6 Conditions check for merging bounding boxes. 29
4.7 A working example of method 2. 31
4.8 Flaws of CRAFT. 32
4.9 Process of method 3. 33
4.10 A working example of method 3. 34
4.11 Method 3 fails to recognize correctly table with spanning cells. 35
4.12 Process of method 4. 36

4.13 Method 4 generates redundant bounding boxes. 37
4.14 An example of method 4. 38
4.15 mAP at different thresholds of table cell detection on sampled data. . . 41

5.1 mAP at different thresholds of table cell detection on new dataset. . . 46

Chapter 1

Introduction

1.1 Background of research

Table is one of the most commonly used methods to represent information by organizing them into rows and columns. Since humans can easily interpret tables, they are widely used in many areas, especially in documents. In recent years, the number of digital documents has increased significantly with businesses, corporations and governments are transitioning to paperless documents. Together with the transition to digital documents is the need to interact with the document as well as automatically processing them. However, since most of these documents are either in image or exchange format like PDF, they do not contain the structure of the documents, making it hard for users to use these files efficiently. For image files, users cannot interact with the document at all because everything is rasterized, rendering operations like select, copy, searching, etc. impossible to do. These drawbacks can be somewhat remedied by performing OCR to add a text layer to the image. However, OCR can only perform text recognition. It cannot recover the layout of documents, especially when it comes to the layout of tables in documents. For the PDF counterpart, while the text is preserved, there are no information to directly locate a table and identify its structure. Since tables usually contain valuable information, researches on how to recognize table structure in a document are necessary.

State/territory[b]	Number			Weight (grams)		
	2004–05	2005–06	% change	2004–05	2005–06	% change
NSW	2,567	3,040	18.4	926,834	677,811	-26.9
Vic	513	795	55.0	1,264,829	493,737	-61.0
Qld	2,399	2,492	3.9	47,276	23,281	-50.8
SA	208	268	28.8	5,391	66,942	1,141.7
WA	2,454	2,891	17.8	19,602	25,165	28.4
Tas	178	109	-38.8	3,084	495	-83.9
NT	92	100	8.7	8,718	7,251	-16.8
ACT	189	292	54.5	406	1,942	378.3
Total	**8,600**	**9,987**	**16.1**	**2,276,140**	**1,296,624**	**-43.0**

Figure 1.1: An example of a table.

1.2 Problem statement

Table structure recognition task, according to many researches [28, 29, 30, 33], involves two sub-tasks:

- **Table Detection:** In this task, one needs to detect the coordinates or region of the tables inside the documents.
- **Table Structure Analysis (also known as Table Structure Recognition):** After the tables are located, one needs to analyze them and recover their structural information. Structural information can be rows', columns' or cells' location, as well as the layout of the table.

In this thesis, the cell recognition task will be the primary objective of the Table Structure Analysis task.

1.3 Targets and methods

1.3.1 Targets

The main objective of this thesis is to propose multiple approaches that can generate annotations for a table structure recognition dataset. As the number of annotations by these approaches is large enough, the new data can be used to train a deep learning model to solve table structure recognition task. Moreover, this research can also assist any future works which aim to solve the problem of table structure recognition but do not have adequate data to do.

1.3.2 Methods

In this thesis, the main focus will be the cell recognition task. The following steps are performed:

1. Convert the annotations of an existing dataset to a friendlier format for cell recognition task.
2. Sample correct annotations and train CascadeTabNet [33] - a cell recognition model using that data.
3. Use the trained model to check the annotations generated earlier.
4. Use all correct annotations to continue training the cell recognition model in a weakly supervised manner.

1.4 Thesis' structure

This thesis is structured as follows:

– **Chapter 2** discusses the challenges, the benefits of dealing with this problem

in real world, as well as the approaches taken by other researchers and their limitations.

– **Chapter 3** explains the background needed to understand this thesis, including tables in documents, the R-CNN model and its extensions, IoU and morphological transformations.

– **Chapter 4** describes how the methods used to prepare the data for training the models of both table detection and table structure recognition.

– **Chapter 5** describes how the experiments are conducted, as well as the results of the method in both Table Detection and Structure Recognition tasks.

– **Chapter 6** concludes the thesis, as well as discusses what can be done to improve the results further in the future.

Chapter 2

Related works

2.1 Challenges

When it comes to digital documents, there are two dominant types of format: PDF and image document. PDF is the most widely used format when it comes to publishing documents, thanks to its capability of preserving layout across different PDF generators. Image documents, on the other hand, are documents which have been converted to digital image formats such as PNG, JPEG, TIFF... They can originate from various sources like document scanners, smartphone cameras or even captured from the PDF files themselves. Compared to PDF files, image ones are rasterized so they do not embed data like text, font, etc.

In order to perform table structure recognition, the table must first be localized in the document. However, in table detection task, both of these formats share the same weakness: they do not contain information about logical structure of tables. Image files store tables as individual pixels. PDF documents, on the other hand, only contain machine-readable text from the tables and draw table lines using vector graphics, which are also used to draw figures, chart, etc. Both formats do not have anything to tell where a table is located in the documents. This makes it difficult to directly locate and recognize logical structure of tables inside both types of document format. For image documents, one can only rely on individual pixels to locate the

tables and for PDF ones, there may be more cues like font size, graphical lines, text arrangements, etc. to depend on.

Another reason that make table detection a challenging task is the two types of variability in the document, namely intra- and inter-class variability. Inter-class variance between tables and other graphical objects such as figures, charts, etc. can affect the detection of tables and make the number of false positives increase. Intra-class variance between various table appearances, such as layout, ruling lines, background color, etc. also makes it hard to correctly detect tables as each heuristic algorithm can only perform well with table appearances that they are optimized for. These are the factors that make table detection a challenging task.

Table structure recognition is also another complicated task to do. First of all, it requires the table detection step to correctly locate the table in the document in order to work. The detected region should only convey the whole table to recognize table structure correctly. Extra elements outside the table can make algorithms fail to detect the structure while not covering enough table can result in missing table rows and columns. Moreover, there have been several researches for the recognition of table structure information but they are limited to only a single format. Methods designed for PDF works only PDF files, making them inapplicable to image documents. On the other hand, methods designed for image documents can work with both image and PDF documents, provided that the documents are converted to images. However, such methods have disadvantages compared to native PDF methods as they cannot leverage embedded information inside the PDF like text, font size, etc.

Even when the table region is correctly localized in the documents, it is still difficult to accurately detect its logical structure. There are many types of tables with different types of structure, as pointed out in figure 2.1. The variation in table structures such as border type, spanning cells, background color, etc. makes it hard for many hand-engineered methods to generalize. For example, algorithm that is optimized for grid-like table like figure 2.1a may detect missing columns in figure 2.1b or fail entirely with figure 2.1d due to the lack of border lines. Spanning cells like in figure 2.1c also complicate the task further. Many heuristics algorithms also fail with tables having color text and background like figure 2.1e because many of them

Areas	# of nodes	# of edges	# of features
Search Engine	269	332	6695
Graph Theory	223	917	6695
Philosophy	303	921	6695

(a) Fully bordered

	x=0	x=0.15	x=0.30
3_BL	-0.16(3)	-0.15(4)	-0.16(4)
9_BL	-0.20(4)	-0.14(4)	-0.22(5)

(b) Missing lines

	SPS beam			HPPS beam		
	Peak I	Peak II	$+F_{3\sigma}$	Peak I	Peak II	$+F_{5\sigma}$
Base	12.5	10.8		26.6	24.5	
GLBOPT	15.3	13.0	+8%	39.2	35.0	+19%
LEOPT	15.1	13.2	+8%	49.3	40.0	+28%
HEOPT	15.3	13.2	+8%	44.0	39.3	+24%

(c) With spanning cells

		+5 -5 cm	-5 -15 cm	-15 -30 cm	-30 -45 cm
Low N	Fa	71.6	16.0	7.0	2.0
	Fl1	83.0	11.3	3.2	1.0
	Fl2	83.9	7.1	4.8	1.7
	Fp	85.7	7.9	3.9	1.0
	Lp	81.6	10.3	5.5	0.9
	avg.	81.2	10.5	4.9	1.3

(d) Borderless

Area (Sq Mtr)	No. of Books	No. of Periodicals	Newspapers subscribed
187	10200	31	7
53	11, 963	8	2

(e) Cells with colored background and text

Figure 2.1: Different types of table.

are designed only for documents with white background and black text. Therefore, to solve table structure recognition task, one has to take these factors into consideration.

2.2 Applicability

Tables usually contain valuable information as they are one of the most popular structured method to organize data in documents and communicate to human readers. While it is easy for human to interpret tabular data from PDF and image documents, machines, on the other hand, are not capable of directly read them. Therefore, being able to automatically recognize table structure in documents has many potential applications in real life and can also pave the way for other researches as well.

Information archiving: The need for storing paper documents digitally is increasing nowadays. To do that, the most common and effortless way is to scan or take a picture of the document. This works by capturing the whole document and rasterize it as an image and store them in a file. However, there are a lot of limitation when

doing this method. Since the documents are saved as image format, it takes a large amount of storage to store individual pixels, even with a good image compression algorithm. In contrast, storing table data in electronic documents format, such as Office Open XML, requires fewer amount of data to be stored, thus saving a lot of storage space, especially when the table contains many rows and columns.

Document reuse: Storing table in image format can also make it hard for users to use table data directly as they cannot perform actions such as searching, selecting and copying. This is a huge obstacle when it comes to indexing image documents. While these operations can be done in PDF documents, due to the lack of table structure information, the result depends heavily on how the PDF generator generate the file. Also since there are no information about how the table are structured in both PDF and image formats, if users want to reuse the table's layout for other documents, they will need to create a new template manually, which can be very time-consuming.

Form automation: Many of the business form are represented as a table. These tableform documents can be automatically processed if their table logical structure is recognized. For example, after a customer has filled a paper tableform document, a business can scan the document, use a handwritten recognition model to automatically gather the information they need without the need to type the answer manually. When the number of documents is huge, this can tremendously reduce amount of time and work needed to input the information from the documents.

Automated machine information retrieval: Knowing about logical structure of tables can also help machine to understand tables and perform automatic information retrieval task. Bao et al. [1] presented an NLP method to generate a natural language sentence to describe a table region when the logical structure of the table is known. Herzig et al. [2] demonstrated a question answering system using tabular data. Similarly, Pasupat and Liang [3] introduced a system to answer complex questions using knowledge from HTML-formatted tables. Jauhar et al. [4] used table as a source of knowledge to train a multiple-choice answering system. Jain et al. [5] proposed a deep learning model to summarize information from multiple tables using fixed schema.

2.3 Applicability after improvements

While this work is primarily aimed at solving table structure recognition task, it will also deal with table detection as well. Since the advent of deep learning, many approaches was invented to deal with object detection task and they have achieved better accuracy compared to traditional methods. In this work, I will use a deep learning model for table detection task that will locate table in a document and try to not miss any table elements, as well as not include anything outside the table. As mentioned earlier, table structure recognition requires good localization of the table so this approach can serve as a base step to produce input for this task, as it can help boost both rule-based and deep learning methods and researchers will only need to focus on table structure recognition task.

In this work, I will propose a method to generate annotations automatically for TableBank Table Structure Recognition dataset Li et al. [6], which has unusual and uninformative annotations for the task. Despite having over 140,000 tables, the annotation format of this dataset is a huge obstacle for many researches on this task because it is originally intended for used with an image-to-text model, thus rendering the annotations useless for other methods, especially for deep learning based ones, which requires the file to be annotated as bounding boxes. By generating table annotation using the table image file and verifying the result using the original annotation files, my approach will help future deep learning researches able to utilize this huge dataset without having to manually annotate every single file.

Since this method detects table structure at cells level, its result can be used to deal with other tasks, such as rows and columns detection. This can help reconstruct the whole logical table structure from bottom up. Moreover, the use of deep learning model can also help the OCR process to recognize letters much more accurate thanks to better bounding box results from the approach. When combined with good OCR engine, this will enable works that aim to recover tables as well as their content into an editable format such as Microsoft Word, Excel, etc. so that users can directly manipulate the table. One can also use this work as a base ground to determine table

cell relationship in order to design table understanding systems, enabling various automated information retrieval tasks for the table. Also, since this work is aimed at detecting accurately at cells level, one can combine with OCR engine to reconstruct an original table into another one with various sizes, enabling flexible PDF, which can particularly be useful to mobile users who often have to work with documents optimized for desktop.

2.4 Approaches

There have been several researches on table structure recognition ever since the 90s. Most of these methods share the common workflow: detecting tables and then analyzing their structure. According to many researches, they can be divided into two main categories: rule-based and data-driven approaches.

Rule-based methods are those approaches that rely on hand-crafted features to find tables in documents and recognize their logical structure. These methods often rely on text, graphical lines, arrangements, etc. and they define a set of predefined rules to find the candidate table regions that satisfy their conditions. These features are also used to recognize table structure, such as for separating rows, columns and cells. Since these methods need to define rules, they usually make assumptions about how the documents and the tables look like.

Data-driven approaches are those do not rely on these assumptions at all. Instead of using hand crafted features or metadata, they rely on machine learning, especially deep learning techniques, to locate tables. For table structuring recognition, they either use some heuristic rules with parameters or a deep learning model to locate rows, columns and cells. These methods are trained on large dataset of tables in order to optimize their parameters, hence called data-driven.

2.4.1 Rule-based approaches

Image-based methods: In the early 1990s, Chandran and Kasturi [7] proposed a method to detect tables from binary document images. First it detects horizontal and vertical lines that satisfy predefined conditions. Then, for each missing table line, it finds a white stream separator to substitute them. These demarcations are used to locate blocks, which are used to represent table structure. The way these blocks are labeled will be used in my dataset recognition step. Itonori [8] described a method that detect tables in document images using text-block arrangement and ruled lines. According to them, the way text-blocks are arranged can indicate the presence of a table and the ruled lines can be used to verify that. Since previous works rely heavily on lines, which may cause figures to be detected as tables, Hirayama [9] suggest identifying both areas and then using a set of assumptions to separate them. The authors also proposed using dynamic programming matching method to segment a table into rows and columns. Shafait and Smith [10] proposed a method for detecting tables in different classes of document. It uses Tesseract OCR engine to perform layout analysis, and then uses column partitions and layout with predefined rules to locate table regions. Gatos et al. [11] proposed an approach to detect tables from scanned documents. The method first preprocesses the input to keep only text and graphical areas. Then the table is located by detecting lines and intersection points. Line detection algorithm of this method depends on calculated average character height. Although this is for table detection, in my work, this approach will be slightly modified for use in table structure recognition task. Kieninger and Dengel [12] demonstrated a bottom-up method to recognize logical structure without relying on visual clues like lines. It finds words and then cluster them as blocks. These blocks are then used to form table rows and columns. Van Nguyen et al. [13] proposed a method for extracting table structure from scanned images. The method makes use of CRAFT , a text region detection engine to extract text areas from a document, and then use horizontal and vertical clustering algorithms to find rows and columns, which are later used to reconstruct cells. In my dataset generation step, I will also use CRAFT to deal with images that my initial algorithm cannot correctly annotate.

PDF-based methods: Ramel et al. [14] presented a multi-level representation framework to solve the limitation of existing ones. They located table by detecting all lines and text blocks in documents, then constructing minimum rectangles to determine candidate table regions. Hassan and Baumgartner [15] proposed a method to locate tables and extract their structure from PDF documents without relying on visual features. They segmented each word and clustered them to form columns which are then used together with a set of rules to locate table regions. Then, these candidate columns are used to split table rows, columns and cells. Liu et al. [16] proposed a method to extract table based on PDF metadata. It uses a metadata extractor to segment a document into boxes and also find keywords that may indicate a table to locate table regions. Shigarov et al. [17] exploited the way tables are printed in PDF files to recognize table structure using a set of configurable parameters. This method can also be adapted to various document domains, thanks to the complexity of the parameters, as well as the heavy reliance on font settings. Oro and Ruffolo [18] presented a heuristic approach to extract table structure from PDF files. It works by harvesting basic content elements, and then grouping them based on alignment to form rows and columns, which will be used to rebuild the table.

2.4.2 Data-driven approaches

For data-driven approaches, they can be classified into either traditional machine learning or deep learning method.

Machine Learning: Cesarini et al. [19] used MXY tree as a hierarchical representation to represent elements in documents, as well as reducing prior knowledge for table detection. The MXY tree is then recursively analyzed for table lines to detect the presence of tables using several thresholds. In order to determine the best combination of them, the authors used a supervised learning approach to optimize these thresholds. Kasar et al. [20] also identified table lines and intersecting points to locate tables. However, they generated features from intersecting lines and used a SVM classifier to get coordinates of tables. Wang et al. [21] proposed solving table structure recognition based on statistics. They used a seven-step probability-based

framework to analyze table structure, where the probability are optimized based on training documents.

Deep Learning: Hao et al. [22] was the first to try localizing tables in PDF files using deep learning technique. They still used some predefined rules for each type of tables to locate candidate regions and then used a LeNet-based convolutional neural network to classify whether a region is a table or not. They also feed additional low-level information from the PDF to the neural network to improve the performance. While the previous work still rely on some predefined rules, Gilani et al. [23] proposed the use of Faster R-CNN to locate tables in document images, completely eliminate the need for heuristics rules. The method works by transforming an original document image to a 3-channel image with each channel corresponds to a distance transformation and use it as input for the network. Sun et al. [24] improved Faster R-CNN's performance by using VGG16 as a backbone network and also located table corners using neural network to refine the result. Siddiqui et al. [25] also enhanced Faster R-CNN by implementing deformable convolutional layers in both Faster R-CNN and ResNet, which serves as a base model, as well as in RoI pooling layer to overcome conventional ones' limitations. Huang et al. [26] proposed a method to detect tables using YOLOv3 network with some adjustment to anchor boxes using k-means clustering to make them look like tables. Agarwal et al. [27] presented CDeC-Net, which used Cascade Mask R-CNN with dual ResNeXt network, one assisting and one leading, as backbone and deformable convolutional layers. Since Cascade Mask R-CNN achieved best result across different datasets, my approach will use this model for detecting table region.

Siddiqui et al. [28] proposed using deep learning to detect tables' region as well as their rows and columns. The method uses Faster R-CNN, R-FCN and FPN with deformable variant of ResNet as the base model. Schreiber et al. [29] also used Faster R-CNN and FCN to detect tables and recognize their structure, respectively. However, for the latter task, they considered it as rows and columns segmentation task instead. Since most previous works separately solve the two task one by one, Paliwal et al. [30] proposed a method to solve both tasks in a single inference. The method uses VGG19 to generate features for both tasks, then each segmentation task is solved using a

separated decoder branch in order to generate table and column masks.

Contrast to these top-down approaches, Adiga et al. [31] proposed detecting table cells from PDF documents first. Then, this method will use a Multilayer Feedforward Neural Network to classify the cells relationship. Similarly, Raja et al. [32] proposed using Mask R-CNN to locate table cells in document images. A graph neural network is then used to predict rows and columns relationship. CascadeTabNet by Prasad et al. [33] also uses Cascade Mask R-CNN with HRNet as the backbone network to locate table and generate segmentation mask for table cells in a single run. Since this method achieved state-of-the-art result in table detection, I decided to follow this method in table structure recognition with some additional improvements.

While previous methods work by detect table first then table structure later, Zheng et al. [34] proposed a framework that detect both table and cell regions first to generate refined table region first, then use the result to detect table cells again to yield accurate input for row and column detection.

Qasim et al. [35] argued that graph neural networks are more suited in table structure recognition. Their method only uses CNN to generate vertex features from a table image, and then a graph neural network is used to determine whether a vertices pair share the same row, column or cell. Chi et al. [36] also used graph neural network to predict relationship between table cells in PDF documents. This method also deals with spanning cells, which is an obstacle for existing methods.

2.5 Limitations of other approaches

Although these methods achieve good results, they still have their limitations. These limitations are often shared by the approach they use. A main limitation of these approaches is that they can only work well with the document formats that they are designed for. PDF-based methods like Shigarov et al. [17], Oro and Ruffolo [18] cannot work with image documents or scanned image PDF documents. On the other hand, image-based methods may work with PDF documents but they require conversion and thus, losing many information during the process, making the methods unable

leverage metadata from original PDF files.

Traditional rule-based approaches have the most limitations out of the two. First of all, many of these methods do not work well with table that lack some or all table lines. This is because they mainly rely on visual information, which may not exist in some documents. For example, methods like Gatos et al. [11] will fail entirely when working with tables in figure 2.1b and 2.1d because of their reliance on lines and intersection points. Furthermore, these methods requires manually hand crafted features, which are not robust to inter- and intra-class variability mentioned above. This is because when these methods are designed, their authors made assumptions about how the document and table structure is like, which means that these methods are only optimized for the type of documents they are worked on. Therefore, such methods are prone to fail to generalize. For example, many of methods that deal with document images, such as Kasar et al. [20], assume that table background is always white and the text is black. While these methods work well with most documents, they will fail with any table that have other colors like in figure 2.1e.

Table 2.1: Number of tables in popular table structure recognition dataset

Dataset	Number of tables
TableBank	145,463
ICDAR 2013	154
ICDAR 2019	3,587
TabStructDB	1,081

For data-driven approaches, they do not suffer the limitations of rule-based ones as they do not require hand-crafted features to work. These methods can generalize to other document domain, as well as tables with different font styles, color, etc. Furthermore, they can work with documents that lack some or all demarcation lines. However, the quality of these methods is heavily influenced by the quality of the dataset. To train these methods, a dataset with high level of inter- and intra-class variance is needed. Moreover, most approaches often suffer low accuracy in table structure recognition from the lack of data for this task, which can cause overfitting.

For table detection task, experiments by Casado-García et al. [37] showed that by training popular object detection models on TableBank Li et al. [6] dataset with over 270k images and fine tuning on other ones, they can achieve good results when compared to training only on those datasets. Therefore, researchers often combine many datasets to generate a sufficiently large one for training. However, for table structure recognition, only a few datasets for this task is available. This is due to the time needed to manually annotate rows, columns or cells for every single table in documents. Also, the number of tables in many datasets is usually very low (see Table 2.1). For example, ICDAR 2013 by Göbel et al. [38] only contains 238 images with only 154 tables. Moreover, existing datasets have their own annotate format, which make it hard to adapt for these approaches. For example, TableBank Table Structure Recognition dataset ground truths only contain table structure without any coordinates of rows, columns or cells at all. An example is provided in Listing 2.1. This makes the dataset impossible to use without reannotate the dataset. To mitigate this limitation, I will introduce a method that can automatically annotate some images from this dataset and it will use the original ground truth to verify the annotations.

```
<tabular>
   <tbody>
      <tr>
         <tdy/><tdy/><tdy/>
      </tr>
      <tr>
         <tdy/><tdn/><tdy/>
      </tr>
   </tbody>
</tabular>
```

Listing 2.1: An example of TableBank annotation for Table Structure Recognition

Chapter 3

Background

3.1 Table

Table, in terms of documents, is one of the most popular method for visualizing structured data by organizing them into a grid. According to Zanibbi et al. [39], every table have two types of structure: physical and logical structure.

Physical structure: A table's physical structure tells the location of each part constituting the table in a document. For example, coordinates of table, intersection points, separator lines, etc in a document belongs to physical structure.

Logical structure: A table's logical structure describe how a table is formed, for example, the type of each table part, font settings, line styles, etc. to describe how they look like.

Figure 3.1 has shown structure of a table. Logically, the most basic element of a table is cell. A table cell can contains anything, ranging from text, numbers, formulas, figures, etc. or even another table inside. Multiple table cells in the same horizontal line form a row. Similarly, those in the same vertical line form a column. Table rows and columns are often separated by visible separator lines, but in some case, they are not as well. Similarly, a table can be surrounded by a visible boundary or not. In many real world documents, a table have spanning cells. These are the cells that

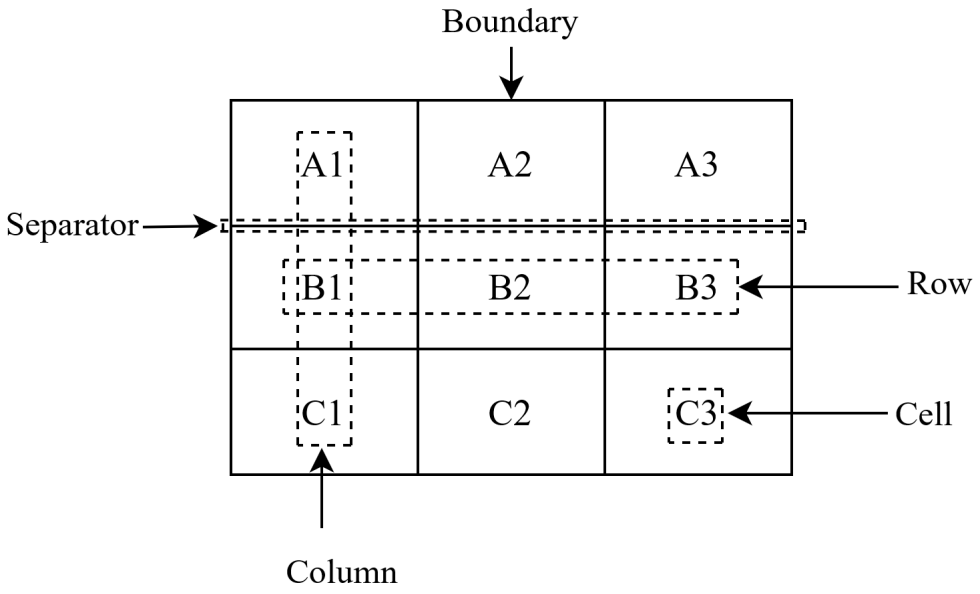


Figure 3.1: Structure of a table.

span across multiple rows and/or columns. They can be considered as a combination or two or more cells. An example of spanning cells is shown in figure 2.1c, where a spanning cell in the header occupies three columns.

3.2 R-CNN and its extensions

In object detection, the output length is not fixed as the number object may appear varies from image to image. However, convolutional neural networks' output size is always fixed. Therefore, it is not possible to directly solve the task with only a CNN. A solution for this is to generate region proposals from the image and then use a CNN to classify whether an object is inside that region or not. Unfortunately, an object can show up in different location in an image and have various aspect ratio, which can create a large number of region proposals to classify.

To deal with this, Girshick et al. [40] proposed R-CNN, which use a selective search algorithm to generate 2,000 candidate region proposals. These regions will be fed to a CNN for feature extraction, which are then classified for object's presence using a SVM. However, R-CNN has some drawbacks. One of them is the time-consuming

training process because each image has 2,000 region proposals and each needs to be calculated in order to train the classifier and regressor.

To mitigate R-CNN's cons, Girshick [41] proposed Fast R-CNN, which runs significantly faster then R-CNN. Instead of calculating features for every region proposal, this method passes the input image to a CNN to calculate a feature map for the whole image once. For each region proposal, a fixed-length feature vector is calculated using a RoI pooling layer, passed to fully connected layers and branched to calculate class probability and bounding box. Not only running faster, Fast R-CNN also achieves better accuracy than the original work.

While both R-CNN and Fast R-CNN rely on time-consuming selective search algorithm to generate region proposals, Ren et al. [42] introduced Faster R-CNN, which uses a Region Proposal Network (RPN) to replace this algorithm. Similar to Fast R-CNN, their method generates a convolutional feature map first from the input image. Then, the RPN is used to generate region proposals from the feature map. This enables the network to learn to generate higher quality proposals, making it slightly better than Fast R-CNN while running significantly faster.

To use Faster R-CNN in semantic segmentation, He et al. [43] proposed Mask R-CNN, an extension of Faster R-CNN with two major adjustment. The first being the replacement of RoI pooling layer with RoIAlign layer to improve the alignments between the input region of interests and the features for better mask generation. The second being the addition of a mask prediction branch after RoIAlign layer to predict segmentation mask.

3.3 Intersection over Union

When an object is detected in an image, it is assigned to a rectangular bounding box $\boldsymbol{b} = (x, y, w, h)$ with x and y are the axis coordinates, w and h are width and height of the bounding box in the image. That object will also be assigned to an object class.

Intersection over Union (IoU) is a popular metric used for evaluating the performance of an object detector. It measures the overlap between two bounding boxes, one

from the ground truth and the other from the prediction of the object detector. Given two bounding boxes GT (ground truth) and Pred (predicted), their IoU is calculated using the following equation:

$$IoU(GT, Pred) = \frac{GT \cap Pred}{GT \cup Pred} \tag{3.1}$$

If $IoU = 1$, it means that the prediction completely match the output. If $IoU = 0$, it means that the predicted bounding boxes does not overlap at all with the ground truth. If the IoU is greater than a specified threshold u, it will be considered as a positive match. The class y of bounding box $Pred$ will be defined as:

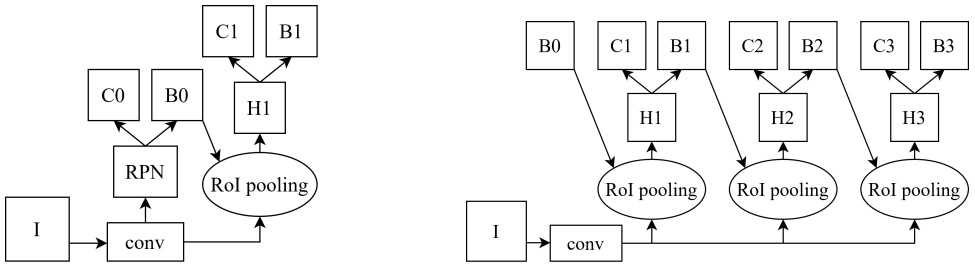
$$y_{\text{Pred}} = \begin{cases} y, & IoU > u \\ 0, & \text{otherwise} \end{cases} \tag{3.2}$$

with y is the class of ground truth bounding box.

It is important to note that the chosen threshold u for training the detector will determine how well that detector perform. If u is too high, the quality of prediction hypotheses is often higher, but the number of positive training set may not large enough to train the detector effectively. If u is too low, more positive training set is available but the detector will output more lower quality hypotheses as they can easily satisfy the threshold. For this reason, most object detectors often 0.5 as the IoU threshold.

3.4 Cascade Mask R-CNN

Although the IoU threshold of 0.5 is commonly used in many researches, Cai and Vasconcelos [44] argued that this is rather a loose threshold and can lead to weak hypotheses. Increasing this threshold is also not feasible as this will significantly reduce the number of positive samples, which will worsen the performance of the detector as it needs a lot of data to learn. Also, according to their experiments, increasing the threshold used during training can also hurt the accuracy during inference time as the IoUs of detector and hypotheses available are different. Therefore, Cai and



(a) Faster R-CNN (b) Cascade R-CNN

Figure 3.2: Architecture of Faster R-CNN and Cascade R-CNN. "I": input image. "B": bounding box with B0 is the initial region proposals. "C": classification score. "H": network head. "conv": backbone CNN network for extracting features.

Vasconcelos [44] proposed Cascade R-CNN, an extension of Faster R-CNN with a multi-stage learning architecture to solve these issues. Its architecture is shown in figure 3.2b. In Cascade R-CNN, instead of using a single regressor, multiple different regressors are used to produce hypotheses. Each regressor is trained sequentially with an increasing IoU threshold, using output of the preceding one as the input. The result is that the quality of the output of each network head H improves gradually, meaning that later regressors will have better quality hypotheses as their input. Also, the IoU threshold of each regressor is kept the same during inference time so there is no quality mismatch between the hypotheses and the detectors.

For semantic segmentation task, the segmentation branch can be attached after a network head H, similar to Mask R-CNN. It can be added anywhere and the number of branches to add depends on the user. In this thesis, the segmentation branch is added in parallel to the last detection one.

3.5 Morphological Transformations

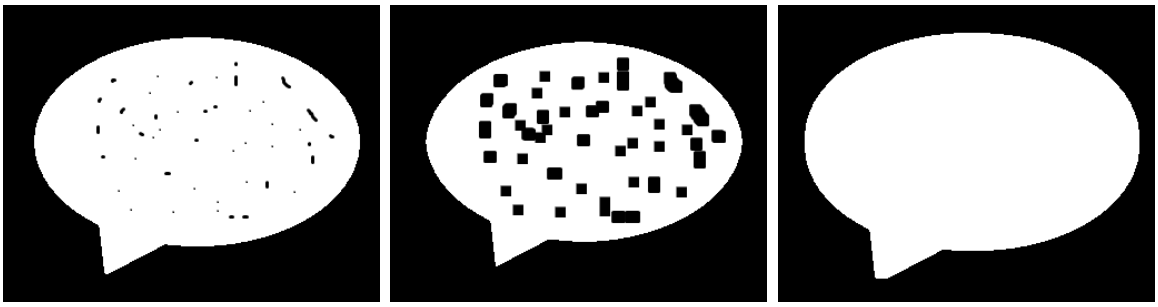
There are four primary operations in morphological transformation: erosion, dilation, opening and closing. Given a binary image (with background in black, foreground in white) and a structuring element (or kernel), each operation works as follows:

– **Erosion:** Erosion operator will erode the foreground by sliding the kernel through the image. If all the pixels of the original in the kernel is 1, the new pixel will be 1, else 0. It means that any pixels surrounding the border of the foreground will be replaced with 0, thus reducing the white regions in the image.

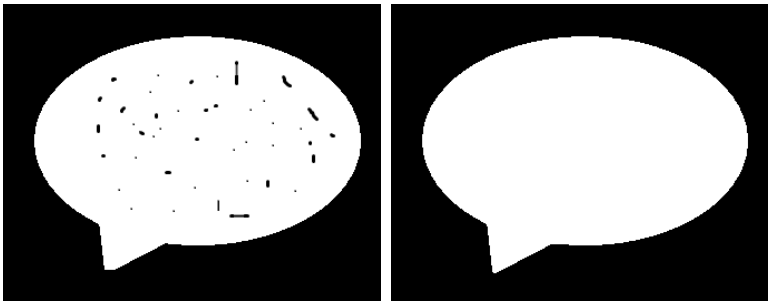
– **Dilation:** Contrary to erosion, dilation operator will dilate the foreground instead. Similarly, if all the pixels of the original in the kernel is 0, the new pixel will be 0, else 1. Opposite to erosion, this will increase the thickness of the foreground in the image.

– **Opening:** Opening operator will perform erosion followed by dilation. It is useful for removing white noises.

– **Closing:** Opposite to opening, closing operator will perform dilation followed by erosion. It is useful for bridging the gap in the foreground object.



(a) Original (b) Erosion (c) Dilation



(d) Opening (e) Closing

Figure 3.3: Morphological transformation operators.

Chapter 4

Methods

As stated in **Chapter 2**, a direct conversion from TableBank's sequence of HTML tags to bounding boxes is not possible due to the lack of coordination. Therefore, to use this dataset for table cell detection, it is needed to find and annotate the location of each cell manually. Since the finding of these cells can generate errors, the original annotation files can be used to check whether the new annotation is correct or not. Moreover, due to the variability of table structure and appearance, using only a single method to find table cells is not enough as it may work with some types of table and not for the others. For that reason, in this thesis, I have devised four different methods which leverage Digital Image Processing and Deep Learning techniques to find the table cells. The overall process to generate annotation files is shown in **Figure 4.1**. Each image will have its bounding boxes generated by all 4 methods. These annotations will then be checked using provided HTML sequences. If one of the method generates correct annotation, that method will be used to generate the XML file for that image. If none of them is correct, the image will simply be ignored.

In the dataset, there are images that have wrong or strange annotations in their XML files and they are not documented in [6]. Since the number of these tables only account for 1% of the dataset, these files are skipped. Also, there are images where some characters are shown as a rectangle because the dataset's authors generated these

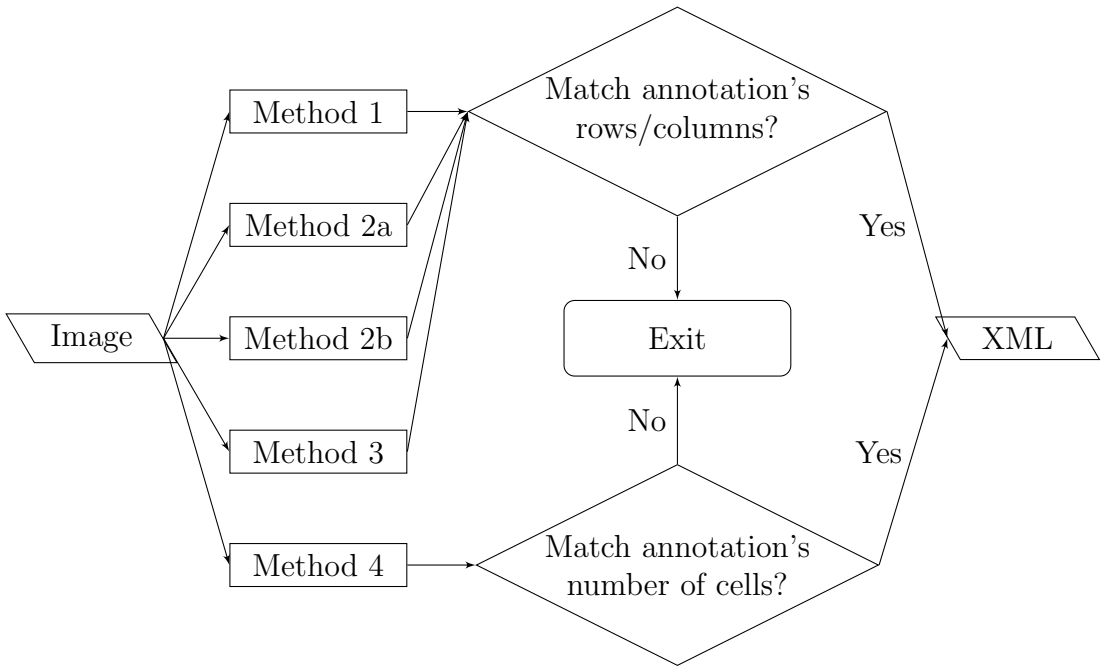


Figure 4.1: Overall process of annotation generation.

files without the necessary fonts. Therefore, these images are also ignored. To do that, I extracted a rectangle from one image and then created another four rectangles that have missing pixel at each corner. These 5 images will be used as a template to detect images with missing fonts using OpenCV's template matching function.

After removing these images from the dataset, the remaining images are processed using 4 different methods. The following sections will describe the details of each method, as well as the methods used to check the new annotations and prepare the data.

4.1 Method 1

The first method works by using Maximally Stable Extremal Regions (MSER) to detect table cells. The steps are shown in **Figure 4.2**.

In the first method, the image is binary thresholded and inverted. Binary thresholding helps remove blurry pixels surrounding edges as it turns any pixel with value

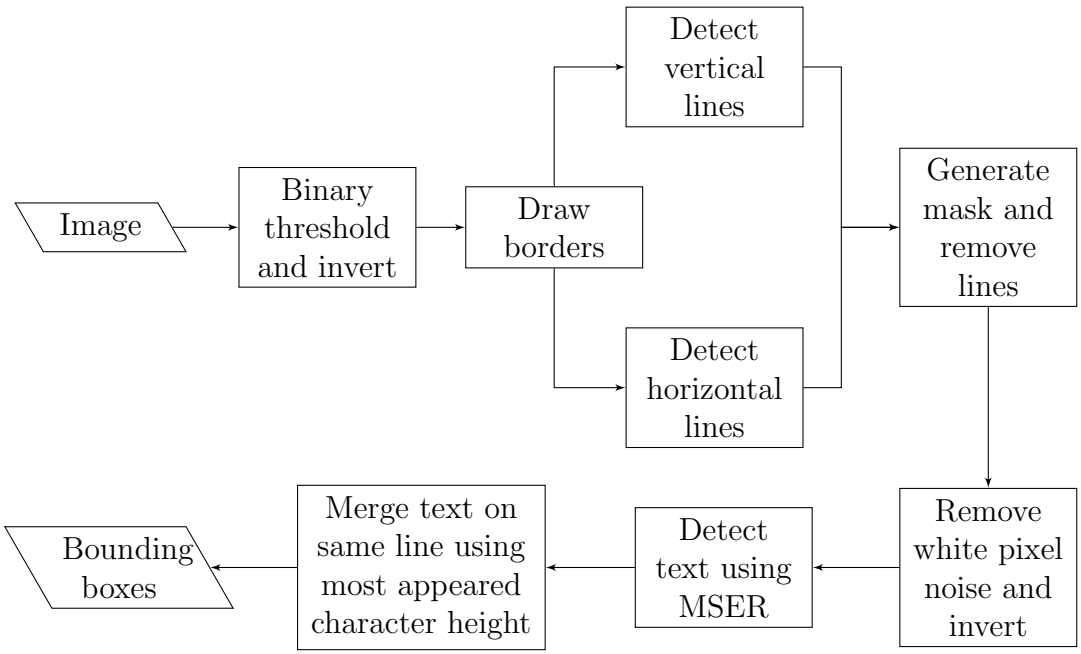


Figure 4.2: Process of method 1.

below the threshold value to black (0), while others are set to white (255). In my experiments, the value 225 is chosen as threshold because this method can generate the most files using this value. I also experimented with Otsu's thresholding but the number of annotations generated using this method was significantly lower, only about 2,000 images when compared to the chosen value. This is because the threshold calculated from Otsu's method can be incorrect due to the domination of background pixels (table background) compared to object pixels (text, lines). This results in many gray pixels become white, which may cause the loss of gray lines and characters' pixels, making the structure generated later on incorrect.

After the image is inverted, a border surrounding the image will be drawn. This step is needed to ensure the consistency between examples in the dataset as some images have fully drawn lines and some don't, which can cause issues when removing table lines. To generate masks for these lines, first, both vertical and horizontal lines are detected using erosion and then dilation, each runs for three iterations to make sure that the text is removed completely while retaining the lines. For horizontal lines, the horizontal kernel has the shape of $(horizontal\_kernel\_length, 1)$. For vertical

lines, the kernel's shape is $(1, vertical\_kernel\_length)$. $horizontal\_kernel\_length$ and $vertical\_kernel\_length$ are calculated as follows:

$$horizontal\_kernel\_length = \lfloor \lfloor \frac{image\ width}{max.\ number\ of\ columns} \rfloor * \frac{1}{4} \rfloor \tag{4.1}$$

$$vertical\_kernel\_length = \lfloor \lfloor \frac{image\ height}{number\ of\ rows} \rfloor * \frac{1}{4} \rfloor * 3 \tag{4.2}$$

with *max. number of columns* is the maximum number cells of table rows and *number of rows* is number of table rows taken from original annotation files. After generating horizontal and vertical lines masks, they are combined using bitwise OR operation to generate a table line mask. Subtracting the inverted image earlier with this mask will remove the table lines from the image while keeping only text. Because some images may have some noisy small pixels near the table lines, a structuring element is used to remove these noise using opening operation, which erode and then dilate the image using the provided kernel. Because the images in the dataset have small resolution, $(1, 1)$ is chosen as the size of the structuring element to ensure that the noise will be removed without affecting too much to the text area.

Then, the image is inverted and MSER is used to detect text areas in the image. Any areas smaller than 4 pixels is set to be pruned to ignore any noise that are not detected in the above step, while keeping small hyphens in many tables of the dataset. For each region detected by MSER, it will be drawn in a white image and the box is filled black for the merging process. Inspired by [11], I will also use bounding boxes' height to merge them. However, instead of using the average height like the paper, my method will use the most appeared one instead. The reason is that when a table have some big bounding boxes, if it has large amount of little bounding boxes, the merging can still correct as the average value is small. But, when the number of little bounding boxes is small, the merging can be incorrect because the average value in this case will be high, which may incorrectly merge bounding boxes of different cells. Therefore, using most appeared bounding box height can tackle this issue. The structuring element used for merging has the shape of $(most\ appeared\ height * 1.3, 1)$. This will be used to merge any bounding boxes on the same line using closing operation, which is

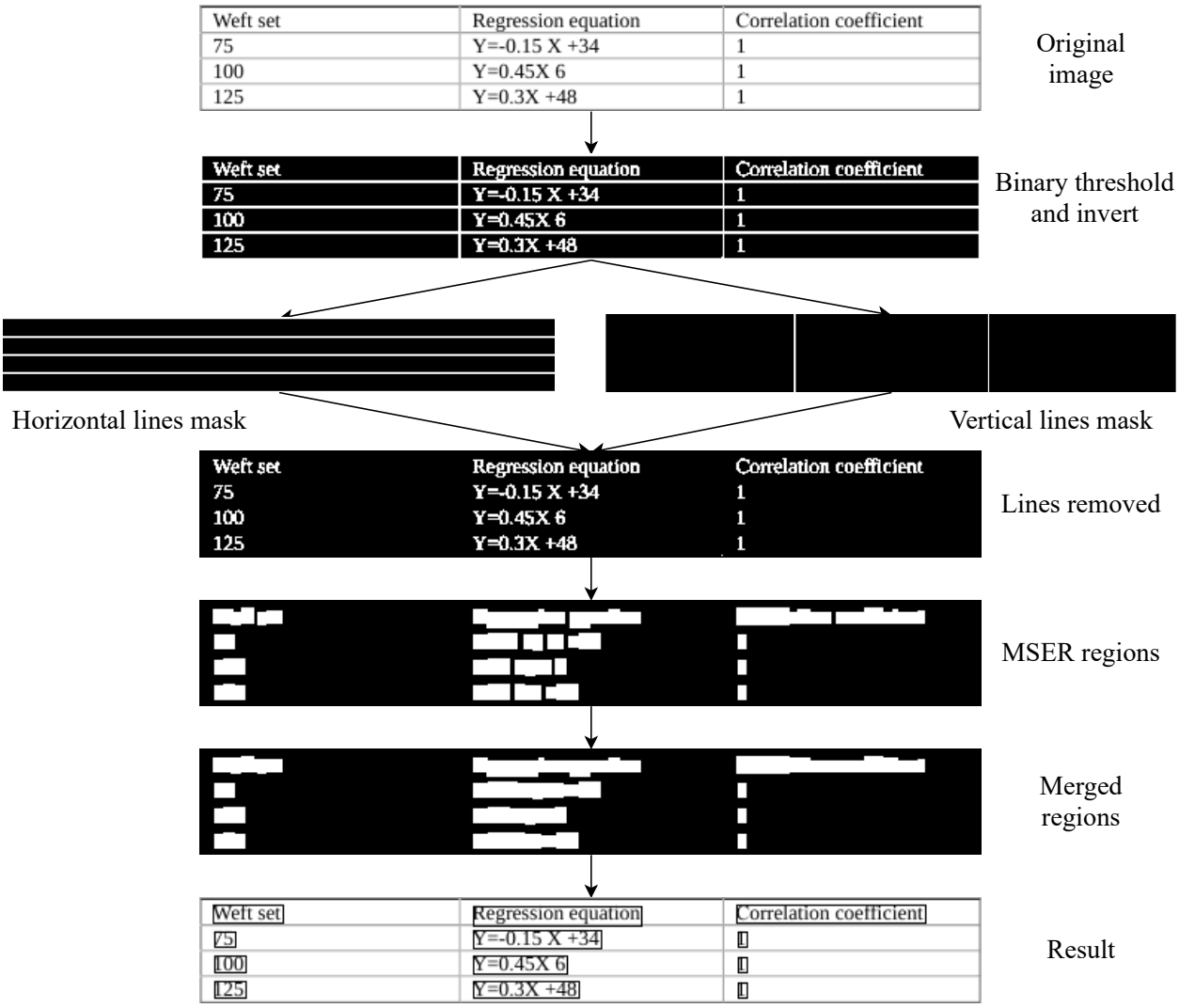


Figure 4.3: A working example of method 1.

the reversed of opening. Finally, connected components in the image is extracted as rectangles and used as the final bounding boxes result.

By using this method, I was able to generate annotations for about 26.66% of the whole dataset (38,783/145,463 images). An example of how method 1 works is shown in **Figure 4.3**. This method works with both fully/semi-bordered and borderless tables as it turns all bordered tables into borderless and the latter steps are designed to work without borders. However, this method assumes that cells' background pixel values are above threshold while the texts' are below so it does not work well when a cell's background color is darker than the chosen threshold value as this will result in

thresholding step incorrectly set the pixels of the background to black and may cause the loss of text inside that table cell. Similarly, this will not work when the text color is brighter than the threshold value because they will be treated as background. **Figure 4.4** illustrates an example where method 1 fails to detect the structure correctly due to the background and text color.

Space Use Division	2007	% Employment 2007	2012	% Employment 2012	Change 2007-2012	% Change 2007-2012
Office	15,650	78.8%	21,554	74.3%	5,904	37.7%
Shop/Showroom	417	2.1%	856	3.0%	439	105.3%
Storage	257	1.3%	235	0.8%	-22	-8.6%
Industrial	599	3.0%	783	2.7%	184	30.7%
Entertainment/Leisure	2,237	11.3%	3,139	10.8%	902	40.3%
Restaurant/Eating	608	3.1%	2,040	7.0%	1,432	235.5%
Community	72	0.4%	260	0.9%	188	261.1%
Utilities	10	0.1%	21	0.1%	11	110.0%
Transport	0	0.0%	29	0.1%	29	NA
Other	19	0.1%	93	0.3%	74	389.5%
Total	19,869	100.0%	29,010	100.0%	9,141	46.0%

Figure 4.4: An example that method 1 fails to detect correctly.

4.2 Method 2

In order to deal with the aforementioned weaknesses of previous method 1, method 2 is designed to use a deep learning model to detect the text regions inside tables, regardless of their structure and appearances. Similar to [13], this method also uses CRAFT (Character-Region Awareness For Text detection) to detect text areas and then locate the cell location of the table. CRAFT is particular helpful in cases where table cells and text are colored as it is capable of detecting them, regardless of their colors. Overall process of this method is shown in **Figure 4.5**. An example is shown in **Figure 4.7**.

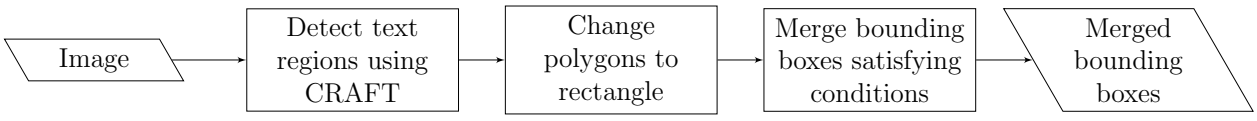


Figure 4.5: The process of method 2.

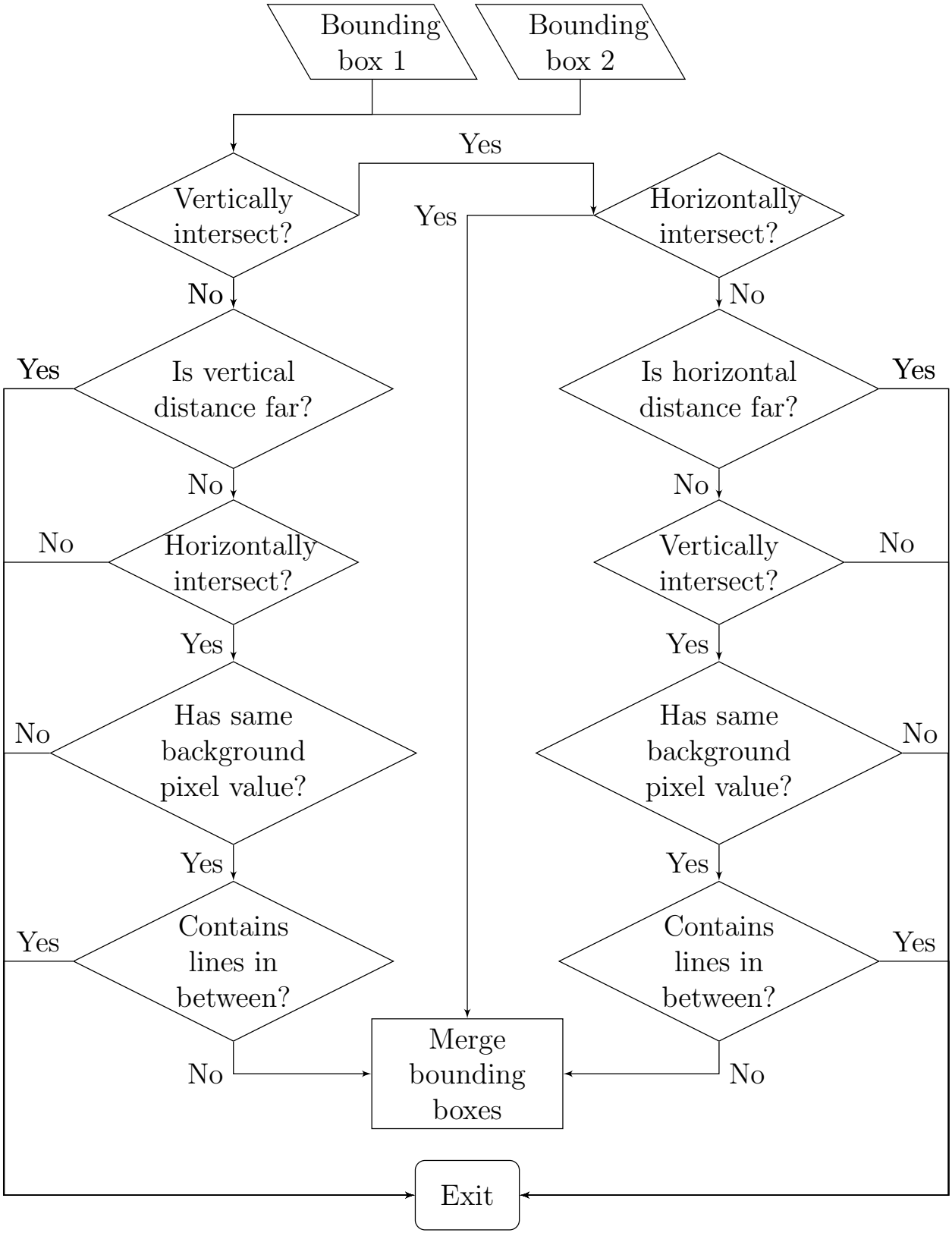


Figure 4.6: Conditions check for merging bounding boxes.

First, the image is passed to CRAFT model to generate text region proposals. Since the results of CRAFT are polygons that has angle, they are converted to rectangle bounding boxes instead because all images in the dataset are computer generated, meaning that the text are perfectly aligned. The conversion is done by taking the most top and left, bottom and right as the top left and bottom right coordinates of the bounding box, respectively.

Then, these bounding boxes are merged by pair if they satisfy predefined conditions. The condition checking process is shown in **Figure 4.6**. For every pair of bounding boxes, if they are both vertically and horizontally intersected, they will be merged as this indicates that they share the same table cell. If they are both not vertically and horizontally intersected, they will not be merged as most pairs of bounding boxes that not satisfy this condition does not belong to the same cell. If a pair is not vertically but horizontally intersected or vice versa, they will be checked to make sure that their distance is not too far. The distance threshold is set to $\frac{\text{minimum height of 2 bounding boxes}}{4}$ for both horizontal and vertical distance in method 2a, while method 2b has vertical distance of $\frac{\text{minimum height of 2 bounding boxes}}{3}$ and horizontal distance of $\frac{\text{minimum height of 2 bounding boxes}}{2}$. The purpose of two sets of value is to generate correct annotations as many as it can with this method.

After checking the distance, the bounding boxes are checked if they belong to the same cell. First, their background pixels value is checked to see if they match. I sampled some regions and found out that for most cases, the background pixel value is always the one with the most pixels in its histogram. By comparing these values from the histograms of the regions, it is possible to determine if they are in the same cell or not because a table cell can only have the same color value. If the values are different between two regions, it means that they are located in different cells. An example for this is the first cell of the first and second rows of the table in **Figure 4.4**. However, only having the same background does not guarantee that two regions belong to the same cell as two different cells may have the same background. To counter that, it is necessary to check whether or not there are any lines in the area between the bounding boxes. When the area only has one unique pixel value, if the value is equal to the background pixel value of both boxes, this means that no lines exist in the

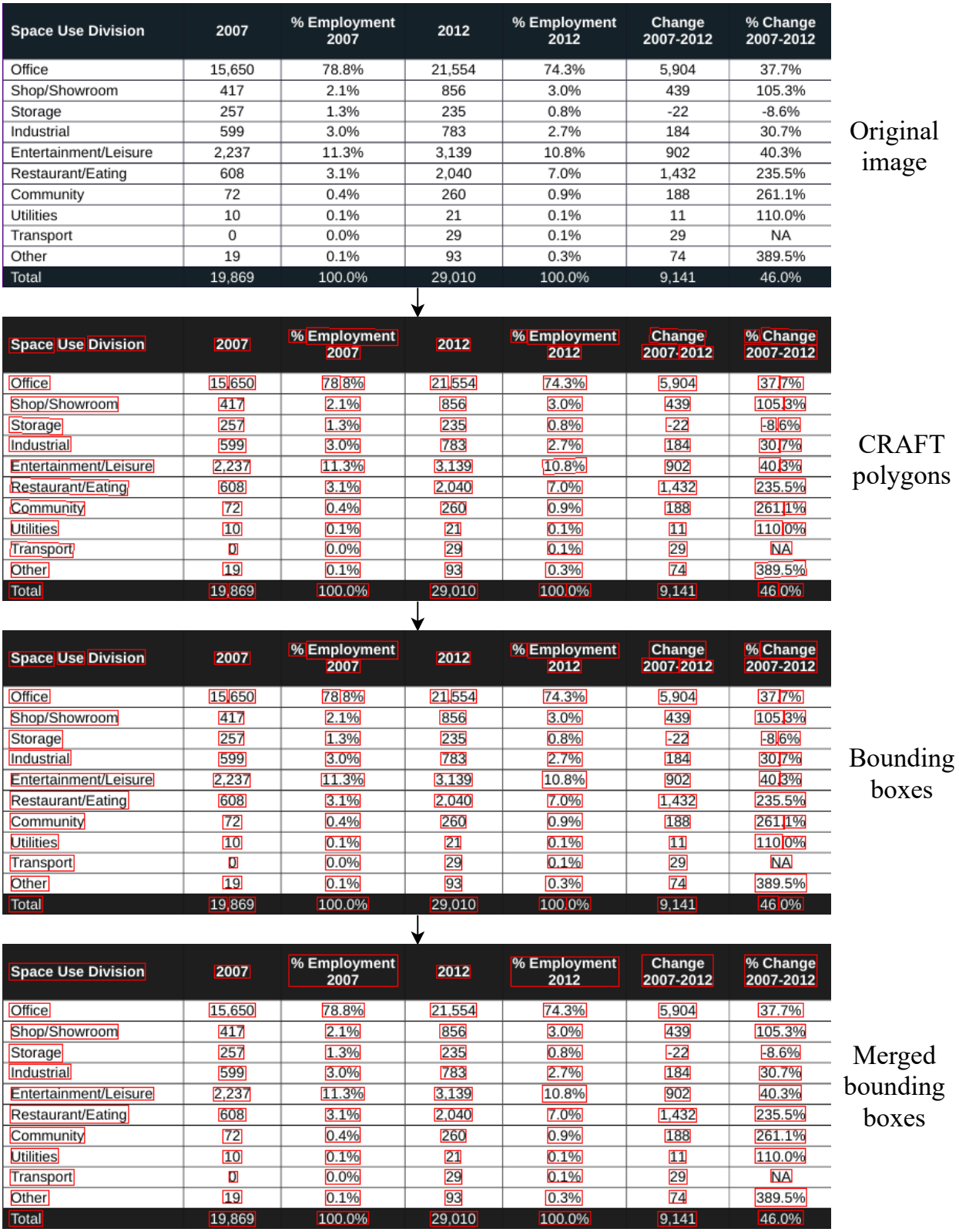
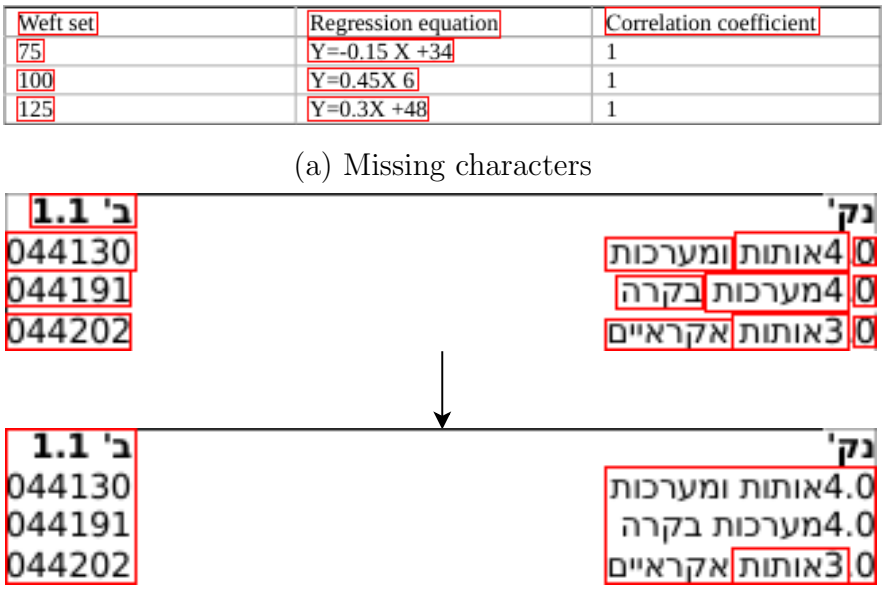


Figure 4.7: A working example of method 2.

area and vice versa. There are cases where the region in between has multiple unique pixel values because the space between boxes is large or CRAFT misses small symbols like dot, comma, etc. For these cases, I perform a loop check of every horizontal (for bounding boxes which are to be merged vertically) or vertical (for those which are to be merged horizontally) line and see if the total number of background pixel value divided by 4 is smaller than the total number of other pixels. If all lines pass this condition, it means that no lines are present in between and they can be merged.

By using this method, I was only able to generate annotations for 3,301 (2.26%) images using 2a's configuration and 3,374 (2.31%) images using 2b's. Method 2 can only produce annotations for 4,424 (3.04%) images, which is significantly lower than other methods. The main culprit for its weak performance is the CRAFT model. Since CRAFT are trained for scene text detection, it does not always detect math symbols in the image, as well as some text areas (**Figure 4.8a**). Also, most images in the dataset have rather small resolution so the bounding boxes detected using CRAFT tend to have short distance to the others, which causes them to be merged in the process (**Figure 4.8b**).



(a) Missing characters

(b) Bounding boxes' distance too short

Figure 4.8: Flaws of CRAFT.

4.3 Method 3

Bordered table can help detecting its cells because its lines provide clues on how to locate the cells. However, method 1 and 2 do not utilize them yet. Therefore, in this method, I will use such table lines to find the table cells. The overall process is shown in **Figure 4.9**.

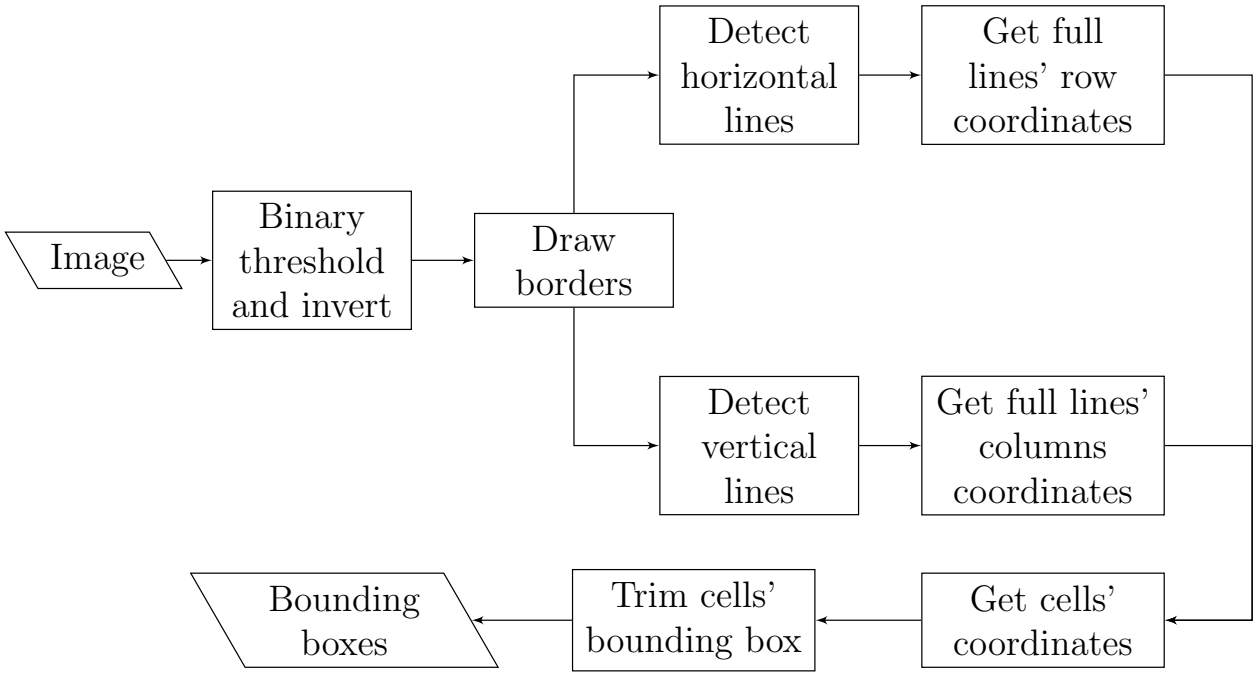


Figure 4.9: Process of method 3.

Similar to method 1, the first step involves binary thresholding and inverting the image to reduce the complexity when working with the image. Instead of 225 like the previous method, the chosen thresholding value for this approach is 192 because it can generate the most correct annotations. Then, the border surrounding the image is also drawn to help the process of locating cells later on. Like method 1, I used a vertical and a horizontal kernel with the same size calculated in **Equation 4.1** and **4.2** for horizontal and vertical one, respectively. Then erosion and dilation are performed to the image, each for three iterations to detect the lines in the image.

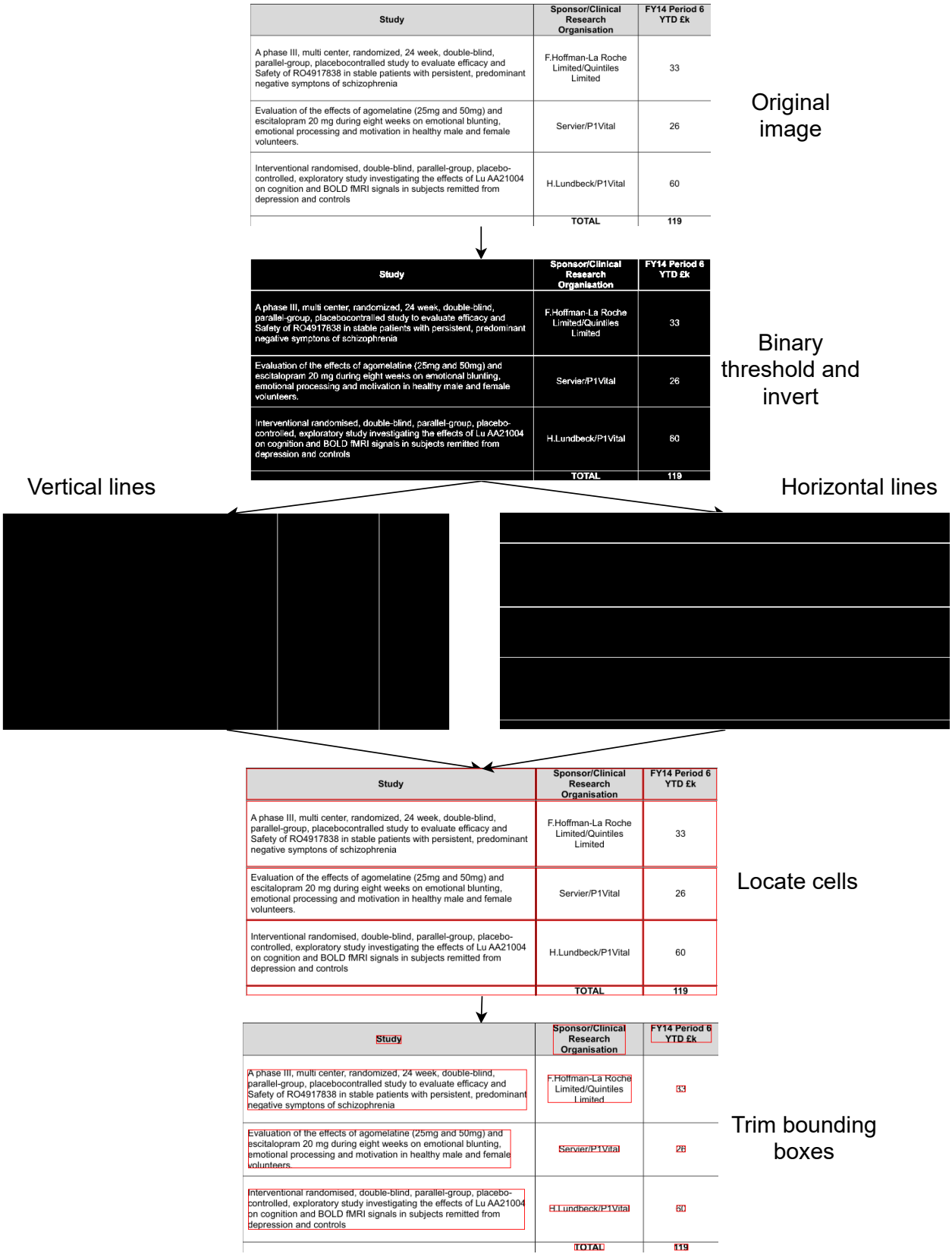


Figure 4.10: A working example of method 3.

Unlike [22, 17, 21] which do text block projection to recover table cells, this method will project horizontal and vertical lines instead. It works by finding out rows or columns of a table that contain a full line, which means that those rows or columns can have only a unique pixel value. Since the image was thresholded earlier, the unique value need to be 0. If not, it means that there might be a cell somewhere in the middle. After finding out the rows and columns containing full lines, the areas between them are marked with an index number. The coordinates of these areas are used to construct bounding boxes of table cells by using each pair of horizontal and vertical areas. Since these boxes are too large, they are trimmed so that the boxes cover just enough of the characters of table cells. For cases where a table cell is empty, their bounding boxes are removed as this research only concerns cells with visible content. Also, since a table contains uneven lines, the coordinates detected can be wrong because the row or column contains that line may be marked as not containing lines. To deal with this problem, before the trimming step, any extra borderlines not detected earlier are trimmed to avoid bounding boxes becoming too large because of them.

With this method, I was able to generate annotations for 29,285 images, which account for 20.13% of the dataset. An example is shown in **Figure 4.10**. Since this approach relies on both horizontal and vertical line information to locate table cells, it

Number of plants	ROI (%)			Payback time (years)		
	Low	Medium	High	Low	Medium	High
0	-1.22	2.19	36.28	-0.82	0.46	0.03
14	-0.81	1.06	19.68	-1.24	0.95	0.05
21	-0.65	0.63	13.44	-1.53	1.59	0.07
28	-0.57	0.41	10.17	-1.75	2.46	0.10
35	-0.52	0.27	8.16	-1.92	3.73	0.12
42	-0.49	0.18	6.79	-2.05	5.71	0.15
49	-0.46	0.11	5.81	-2.16	9.28	0.17
56	-0.44	0.06	5.06	-2.25	17.60	0.20
63	-0.43	0.02	4.48	-2.33	58.95	0.22
70	-0.42	-0.02	4.01	-2.40	-66.33	0.25

Figure 4.11: Method 3 fails to recognize correctly table with spanning cells.

only works with fully bordered tables, not semi-bordered and borderless ones. Moreover, this method makes assumptions about the tables' and texts' colors similar to method 1 so it shares the same weakness with this approach. Furthermore, this method will not work on any tables with spanning cells because it only registers lines that are fully drawn across the table, which does not apply to tables with spanning cells. Therefore, separators near the spanning cells will not be detected, making the approach detect wrong structure of the table. **Figure 4.11** is an example of this.

4.4 Method 4

To overcome previous approach's weakness regarding spanning cell, this method is designed to work with any fully-bordered tables with spanning cells. The overall process is shown in **Figure 4.12**.

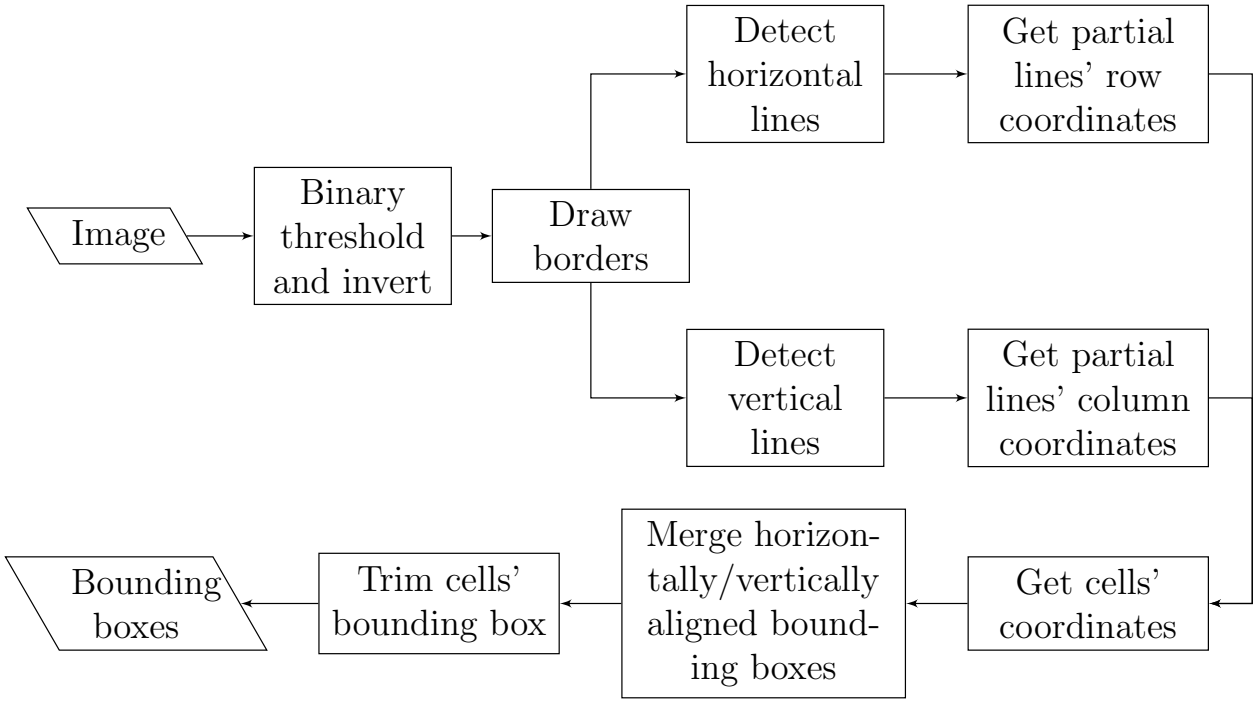


Figure 4.12: Process of method 4.

The beginning of this approach are exactly the same as method 3 until the getting rows and columns coordinates step. If a table contains spanning cell, any lines stay

next to spanning cells are not recognized in previous method. For example, in **Figure 4.11**, the line between the first and second rows is not registered in the horizontal projection so the first and second rows are recognized as one. To solve this problem, instead of finding rows and columns containing a full line of the image, this method scans for any rows and columns that have white lines in their respective mask and marks their coordinates. Unlike previous method, these coordinates will not be marked with index numbers because it is complex when it comes to indexing spanning cells. After finding the coordinates, the cells are recovered similarly to method 3.

Number of plants	ROI (%)		
	Low	Medium	High
7	-1.22	2.19	36.28
14	-0.81	1.06	19.68

Figure 4.13: Method 4 generates redundant bounding boxes. This part was taken from the top left of the table in Figure 4.11.

However, this step recovers redundant cells inside spanning ones because it does not take into account the absence of lines in the latter ones. An example is shown in **Figure 4.13**. Since this structure is incorrect, these cells need to be merged together as one in the next step. For every pair of table cells, this step check whether if they are horizontally, vertically or not aligned at all. If they are not aligned in any direction at all, they will not be merged. If they are either vertically or horizontally aligned, the mask of the area between them are checked for the existence of lines by checking its number of unique pixels, as well as their value. These boxes will be merged if the mask has only 0 as its unique value, which indicates that there are no lines between these two boxes. After merging, all bounding boxes are trimmed similar to method 3 to get the final result.

By using this approach, 41,256 annotations were generated (28.36% of the dataset), making this method the most effective one. An example of this method is shown in **Figure 4.14**. While this approach can tackle the problem regarding spanning cells of

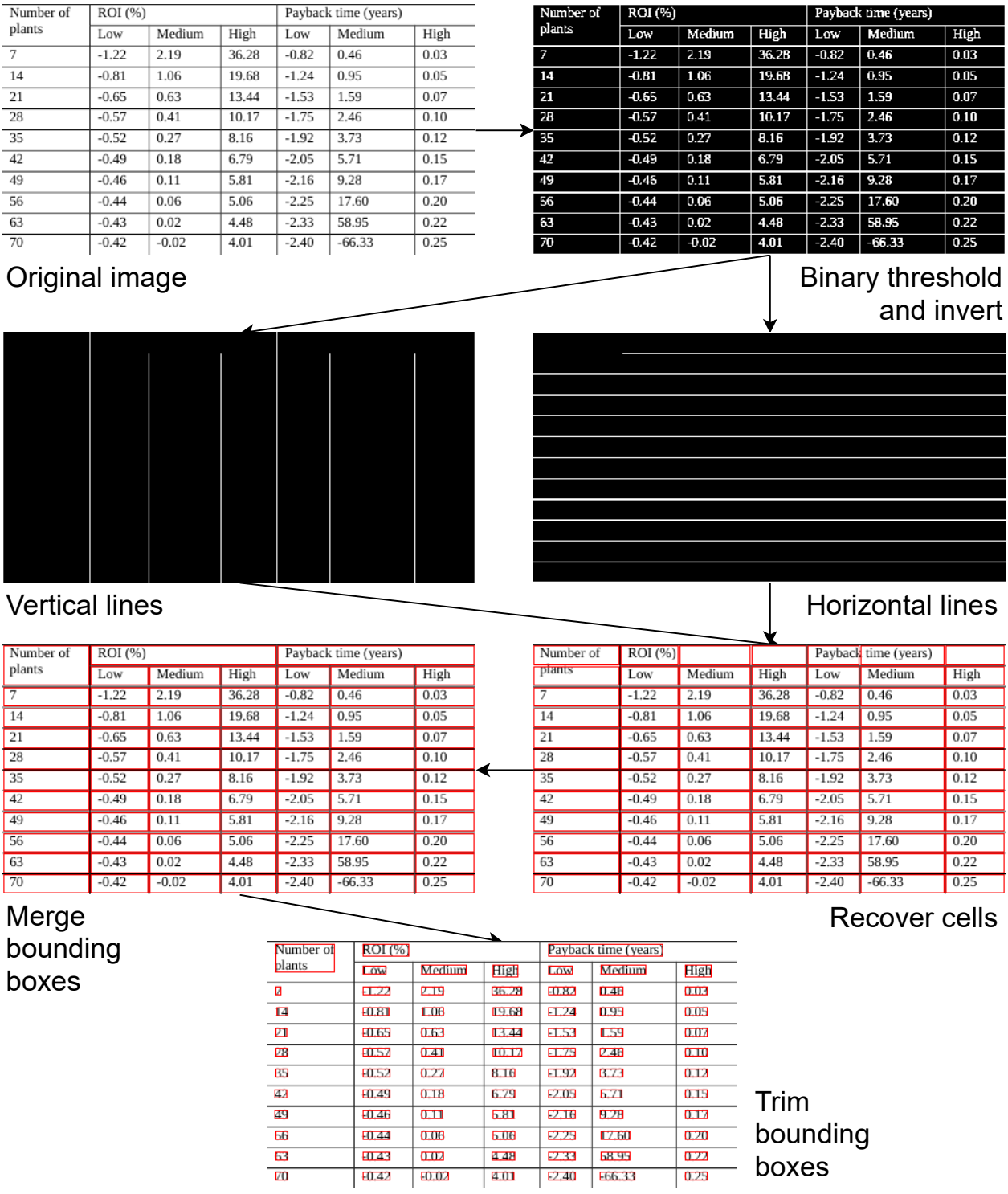


Figure 4.14: An example of method 4.

method 3, it still shares the same other weaknesses with the previous approach due to its reliance on lines.

4.5 Checking the annotations

Since the bounding boxes are generated automatically by 4 method, they can still be wrong if not checked. Because the only annotation available for this dataset is the HTML sequence tags, they will be used to check if the bounding boxes match the structure of the table (or the HTML tags). There are two methods used to verify this.

4.5.1 Method 1: Checking the structure

The first method works by sorting the bounding boxes and then comparing the structure of the annotations. First, all bounding boxes are assigned to their corresponding rows. In many cases, bounding boxes belonging to the same table row may not have the same top and bottom coordinates To deal such cases, all boxes are horizontally projected. A row index number will be assigned to consecutive rows of pixels that have bounding boxes and this number will show that which row in the table a bounding box belongs to. Then, all bounding boxes in a same row are sorted from left to right based on their most left coordinates.

After sorting , this approach will check if the number of rows and the maximum number of visible columns match those in original XML annotations. If the numbers do not match, the bounding boxes will not be checked further because the generated annotation is wrong due to the lack or excess of rows or columns. Then, for each row, the number of visible cells (or the `<tdy>` tags) is compared to the one in XML file to make sure the number of cells are equals for the same row in both annotations. If the number of cells in a row do not match, the new annotation is considered wrong. However, upon checking a few annotation files, it seems that TableBank annotations can have errors where cells that are visible in the image are marked as non-visible (or the `<tdn>` tags). Therefore, the number of detected cells are also compared to the

total number of cells of that row in the original file as well. If either conditions stands correct, it means the row has same number of cells in both files. If all rows pass the conditions, the XML file will be generated.

This approach is used for method 1, 2, and 3.

4.5.2 Method 2: Checking the number of cells

The second method simply involves comparing the number of bounding boxes detected by a method to the number of visible cells in the provided XML files. If the numbers match, it can be considered as a correct annotation. If they do not, the bounding boxes may not represent the structure of the table. To avoid wrong annotations from original files mentioned earlier, the number of detected bounding boxes is also compared to the total number of cells, regardless of their visibility. Therefore, an annotation is deemed correct if the number of bounding boxes either match the number of visible cells or the total number of cells in the XML file. This method is only used for method 4 because method 1 requires sorting the structure, which is complex to implement for table with spanning cells.

4.6 Dataset preparation

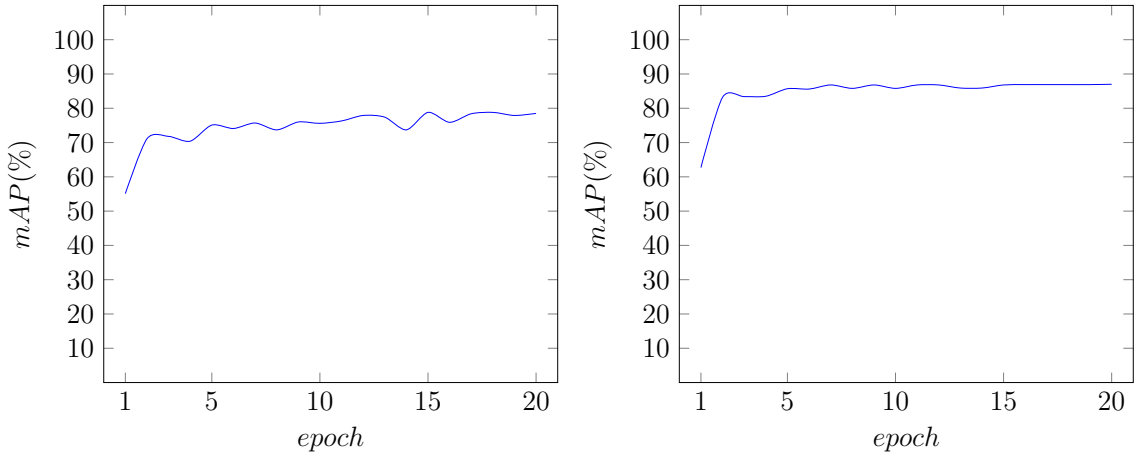
Table 4.1: Final result on annotation generation.

Method	Number of correct annotations
1	41,021 (28.20%)
2a	3,301 (2.26%)
2b	3,373 (2.31%)
3	29,285 (20.13%)
4	41,256 (28.36%)
Total (unique)	73,140 (50.28%)

Combining the number of annotations generated from various methods, a total

of 73,140 unique annotations were generated, which accounted for over half of the dataset. Detailed result is shown in **Table 4.1**. Since the number of bounding boxes was huge (2,888,543 table cells), in order to verify the them without having to check every single file, I sampled a few images from the newly generated annotations, manually inspected and used them to train a cell recognition model for checking the new annotations. The new sampled dataset consists of 4,500 files having correct annotations and 500 files having incorrect ones. Since TableBank dataset had inconsistency in annotating multi-row cells, these annotations were also fixed even though they matched the original annotation. After correcting the wrong annotations, the dataset was split 4,500 for training and 500 for test.

To check the annotations, a deep learning model was trained to generate the annotations, which were used to check against the previously generated ones. CascadeTabNet, which is Cascade mask R-CNN with HRNetV2p as the backbone network to extract high-resolution features and generate initial bounding boxes, was used to for this task. It was trained on the sampled dataset for 20 epochs as the mAP across multiple threshold from 0.5 to 0.95 fluctuated between 70 and 78% (**Figure 4.15a**). On test set, the model reached 87% mAP at IoU threshold of 0.75 (**Figure 4.15b**), which can be considered good enough for checking the annotations.



(a) @0.5:0.95 (b) @0.75

Figure 4.15: mAP at different thresholds of table cell detection on sampled data.

Then, the model generated annotations for 73,140 images whose annotations

had been created earlier. An annotation was deemed as correct if they satisfied the following conditions:

- The number of both bounding boxes between both annotation files matches each other.
- Every bounding box must intersect with a bounding box generated from the model.
- IoU of these bounding boxes must be greater than or equal to 0.5.

The number of annotations matching all of the above conditions is 39,836, account for 54.46% of the earlier dataset or 27.38% of TableBank Structure Recognition dataset. These new annotations and 5,000 annotations sampled earlier were used as a new dataset, which was split 37,825 for training and 4,203 for test, for weakly supervised training the cell recognition model.

Chapter 5

Experiments and Results

5.1 Experiments

5.1.1 Table Detection

For Table Detection task, I also used CascadeTabNet model to locate table regions. The implementation was done using mmdetection. Due to the high VRAM usage of HRNetV2p, the training was done on Google Colaboratory. As the original dataset was split into Word and LaTeX, they were merged according to their set (training, validation and test set). Since the number of images in this dataset is huge, it took nearly a day to complete each epoch. Moreover, after the second epoch, the accuracy merely increased so I only trained them for 5 epochs. Evaluation was done on the merged TableBank dataset, as well as the TableBank LaTeX and ICDAR 2019 cTDaR Track A (Modern) datasets for comparison with other models.

5.1.2 Table Structure Recognition

For Table Structure Recognition task, CascadeTabNet architecture is used again. Unlike [33] that used transfer learning to train the previous model using smaller datasets for both table detection and structure recognition, I trained a separate model specif-

ically for table structure recognition only. The model was continued training from the model used to check annotations (**Section 4.6**). Since the model was already pretrained, it was only trained for another 10 epochs. The evaluation was done on ICDAR 2019 cTDaR Track B2 (Modern) for comparison with other approaches using toolkit by Padilla et al. [45]. Because this model can only recognize cells from table images, it is not directly comparable to other methods without some processing. To do that, evaluation was done by first taking the bounding boxes of tables with confidence over 0.7 from the Table Detection model. Then, for each bounding box, the image was cropped based on it and the cropped sections served as inputs to the Cell Recognition model. I also compared my model's performance in two additional cases. The first was with overlapping table cells removed by keeping only those with the highest confidence score. The second was using the ground truth table annotations to crop the images and use them as input instead.

5.2 Results

5.2.1 Table Detection

In Table Detection task, my model is able to achieve better results on both TableBank Detection dataset when compared to [33]. This is expected as the authors only trained the model on 3,000 images sampled from the dataset, while mine was trained on the whole dataset. Since the number of images is huge, their augmentation techniques can be considered unnecessary to achieve better result. The result is reported in **Table 5.1**.

Table 5.1: Results on combined TableBank dataset. Original authors' result is taken from [33].

Model	Precision	Recall
CascadeTabNet [33]	92.99%	95.71 %
CascadeTabNet (proposed)	**97.00%**	**98.50%**

Furthermore, I compared my model to those trained by Casado-García et al. [37] as well. Since they only evaluated their models on TableBank Detection (LaTeX) dataset, I also did the same with my trained model (using both Word and LaTeX sets). The result is shown in **Table 5.2**. It indicates that when trained on both datasets, CascadeTabNet can still achieve the best results when compared to other methods on the LaTeX dataset at various IoU thresholds. Moreover, at a higher threshold value, both precision and recall of my model only witness a slight decrease, unlike other models in the paper, which experience a significant drop in both measures. At the highest IoU threshold (0.9), CascadeTabNet is comparable to Mask R-CNN and SSD at the lower threshold (0.6). When lowering the threshold to 0.8, CascadeTabNet outperforms all other models at the lowest threshold of the paper. These results have shown that the model can generate tighter bounding boxes, close to those annotated automatically by Li et al. [6].

Table 5.2: Results on LaTeX TableBank dataset. Other models' results were taken from [37].

	IoU@0.6		IoU@0.7		IoU@0.8		IoU@0.9	
Model	P	R	P	R	P	R	P	R
CascadeTabNet	**98.7%**	**99.6%**	**98.7%**	**99.4%**	**98.5%**	**99.0%**	**95.9%**	**97.2%**
Mask R-CNN	94%	98%	94%	97%	93%	96%	84%	87%
RetinaNet	98%	86%	98%	86%	97%	85%	94%	82%
SSD	96%	97%	94%	95%	92%	92%	82%	82%
YOLO	98%	99%	98%	99%	96%	97%	74%	75%

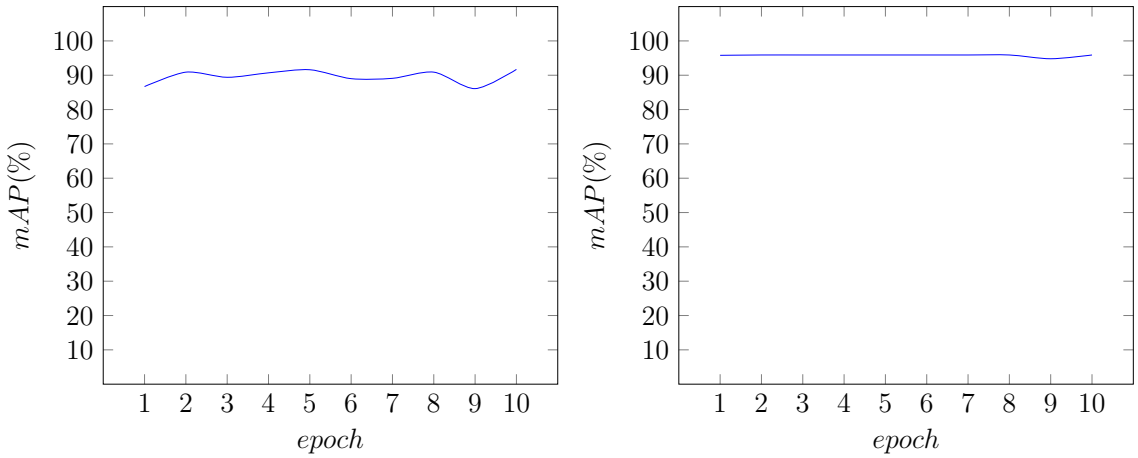
However, when compared to other models on ICDAR 19 Track A (Modern) dataset, my model performed the worst out of all. The result is shown in **Table 5.3**. The main reason for this underperformance is that my model were trained only on TableBank, while other models are trained on ICDAR19 and other dataset. Also, the lack of post-processing, which exists on TableRadar and NLPR-PAL, also hindered the performance of my model as well. Therefore, my model did not detect all the region of many tables in this dataset.

Table 5.3: F1-scores on ICDAR19 Track A (Modern) dataset. Other models' results were taken from [33].

Model	IoU				WAvg.
	@0.6	@0.7	@0.8	@0.9	
TableRadar	96.9%	95.7%	**95.1%**	89.7%	**94.0%**
NLPR-PAL	**97.9%**	**96.6%**	93.9%	85.0%	92.7%
CascadeTabNet [33]	94.3%	93.4%	92.5%	**90.1%**	90.1%
CascadeTabNet (proposed)	80.9%	76.5%	72.1%	64.9%	72.7%

5.2.2 Table Structure Recognition

Compared to pretrained model, the new cell recognition model's average mAP at various threshold from 0.5 to 0.95 increased significantly from 78.5% to 91.7%. At IoU threshold of 0.75, the mAP also increased noticeably from 87.0% to 95.9% and barely changed during the training process (**Figure ??**).



(a) @0.5:0.95 (b) @0.75

Figure 5.1: mAP at different thresholds of table cell detection on new dataset.

On ICDAR 2019 Track B2 (Modern) dataset, my model was able to achieve the best performance when compared with the original CascadeTabNet model and NLPR-PAL method. The result is shown in **Table 5.3**. Without the need for post-processing

like in original CascadeTabNet, my model outperformed the others significantly when the IoU threshold is low and slightly better when it is high. The result is further improved if any overlapped bounding boxes were removed. When using table ground truth as the input for the model, it only outperformed other models at the highest IoU threshold (0.9). Detailed result is shown in **Figure 5.4**.

Table 5.4: F1-scores on ICDAR19 Track B2 (Modern) dataset. Other models' results were taken from [33].

Model	IoU				WAvg.
	@0.6	@0.7	@0.8	@0.9	
CascadeTabNet [33]	0.438	0.354	0.190	0.036	0.232
NLPR-PAL	0.365	0.305	0.195	0.035	0.206
CascadeTabNet (proposed)	**0.645**	**0.477**	**0.220**	**0.040**	**0.310**
CascadeTabNet (non-overlap)	0.656	0.483	0.219	0.040	0.314
CascadeTabNet (with GT table)	0.579	0.436	0.204	0.044	0.285

Chapter 6

Conclusion

This thesis has described several approaches that utilize Digital Image Processing and Deep Learning techniques to generate annotations from TableBank dataset for table structure recognition task. Also, two methods for checking these annotations have been proposed. A cell recognition model has been trained on this new dataset in a weakly supervised manner. Experiments have shown that using the new dataset can slightly improve the performance of the model in table structure recognition task.

The main contributions of this thesis are:

1. Proposed several methods to generate annotations for TableBank dataset.
2. Proposed strategies for checking the generated annotations with the original ones of the dataset.
3. Trained a model using that dataset and achieved better results.

In conclusion, this research has provided a way to convert TableBank dataset's annotations into a new format, allowing this research as well as future works to have a huge dataset for training cell recognition models.

However, the quality of the generated annotations is still somewhat noisy. Also, the number of annotations generated is still low. Therefore, future research direction will be figuring out how to generate more annotations with higher quality.

Bibliography

[1] Junwei Bao, Duyu Tang, Nan Duan, Zhao Yan, Yuanhua Lv, Ming Zhou, and Tiejun Zhao. Table-to-text: Describing table region with natural language. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[2] Jonathan Herzig, Paweł Krzysztof Nowak, Thomas Müller, Francesco Piccinno, and Julian Martin Eisenschlos. Tapas: Weakly supervised table parsing via pre-training. *arXiv preprint arXiv:2004.02349*, 2020.

[3] Panupong Pasupat and Percy Liang. Compositional semantic parsing on semi-structured tables. *arXiv preprint arXiv:1508.00305*, 2015.

[4] Sujay Kumar Jauhar, Peter Turney, and Eduard Hovy. Tables as semi-structured knowledge for question answering. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 474–483, 2016.

[5] Parag Jain, Anirban Laha, Karthik Sankaranarayanan, Preksha Nema, Mitesh M Khapra, and Shreyas Shetty. A mixed hierarchical attention based encoder-decoder approach for standard table summarization. *arXiv preprint arXiv:1804.07790*, 2018.

[6] Minghao Li, Lei Cui, Shaohan Huang, Furu Wei, Ming Zhou, and Zhoujun Li. Tablebank: A benchmark dataset for table detection and recognition. *arXiv e-prints*, pages arXiv–1903, 2019.

[7] Surekha Chandran and Rangachar Kasturi. Structural recognition of tabulated

data. In *Proceedings of 2nd International Conference on Document Analysis and Recognition (ICDAR'93)*, pages 516–519. IEEE, 1993.

[8] Katsuhiko Itonori. Table structure recognition based on textblock arrangement and ruled line position. In *Proceedings of 2nd International Conference on Document Analysis and Recognition (ICDAR'93)*, pages 765–768. IEEE, 1993.

[9] Yuki Hirayama. A method for table structure analysis using dp matching. In *Proceedings of 3rd International Conference on Document Analysis and Recognition*, volume 2, pages 583–586. IEEE, 1995.

[10] Faisal Shafait and Ray Smith. Table detection in heterogeneous documents. In *Proceedings of the 9th IAPR International Workshop on Document Analysis Systems*, pages 65–72, 2010.

[11] Basilios Gatos, Dimitrios Danatsas, Ioannis Pratikakis, and Stavros J Perantonis. Automatic table detection in document images. In *International Conference on Pattern Recognition and Image Analysis*, pages 609–618. Springer, 2005.

[12] Thomas Kieninger and Andreas Dengel. The t-recs table recognition and analysis system. In *International Workshop on Document Analysis Systems*, pages 255–270. Springer, 1998.

[13] Nam Van Nguyen, Hanh Vu, Arthur Zucker, Younes Belkada, Hai Van Do, Doanh Ngoc-Nguyen, Thanh Tuan Nguyen Le, and Dong Van Hoang. Table structure recognition in scanned images using a clustering method. In *International Conference on Industrial Networks and Intelligent Systems*, pages 150–162. Springer, 2020.

[14] J-Y Ramel, Michel Crucianu, Nicole Vincent, and Claudie Faure. Detection, extraction and representation of tables. In *Seventh International Conference on Document Analysis and Recognition, 2003. Proceedings.*, pages 374–378. IEEE, 2003.

[15] Tamir Hassan and Robert Baumgartner. Table recognition and understanding

from pdf files. In *Ninth International Conference on Document Analysis and Recognition (ICDAR 2007)*, volume 2, pages 1143–1147. IEEE, 2007.

[16] Ying Liu, Kun Bai, Prasenjit Mitra, and C Lee Giles. Tableseer: automatic table metadata extraction and searching in digital libraries. In *Proceedings of the 7th ACM/IEEE-CS joint conference on Digital libraries*, pages 91–100, 2007.

[17] Alexey Shigarov, Andrey Mikhailov, and Andrey Altaev. Configurable table structure recognition in untagged pdf documents. In *Proceedings of the 2016 ACM Symposium on Document Engineering*, pages 119–122, 2016.

[18] Ermelinda Oro and Massimo Ruffolo. Trex: An approach for recognizing and extracting tables from pdf documents. In *2009 10th International Conference on Document Analysis and Recognition*, pages 906–910. IEEE, 2009.

[19] Francesca Cesarini, Simone Marinai, L Sarti, and Giovanni Soda. Trainable table location in document images. In *Object recognition supported by user interaction for service robots*, volume 3, pages 236–240. IEEE, 2002.

[20] Thotreingam Kasar, Philippine Barlas, Sebastien Adam, Clément Chatelain, and Thierry Paquet. Learning to detect tables in scanned document images using line information. In *2013 12th International Conference on Document Analysis and Recognition*, pages 1185–1189. IEEE, 2013.

[21] Yalin Wang, Ihsin T Phillips, and Robert M Haralick. Table structure understanding and its performance evaluation. *Pattern recognition*, 37(7):1479–1497, 2004.

[22] Leipeng Hao, Liangcai Gao, Xiaohan Yi, and Zhi Tang. A table detection method for pdf documents based on convolutional neural networks. In *2016 12th IAPR Workshop on Document Analysis Systems (DAS)*, pages 287–292. IEEE, 2016.

[23] Azka Gilani, Shah Rukh Qasim, Imran Malik, and Faisal Shafait. Table detection using deep learning. In *2017 14th IAPR international conference on document analysis and recognition (ICDAR)*, volume 1, pages 771–776. IEEE, 2017.

[24] Ningning Sun, Yuanping Zhu, and Xiaoming Hu. Faster r-cnn based table detection combining corner locating. In *2019 International Conference on Document Analysis and Recognition (ICDAR)*, pages 1314–1319. IEEE, 2019.

[25] Shoaib Ahmed Siddiqui, Muhammad Imran Malik, Stefan Agne, Andreas Dengel, and Sheraz Ahmed. Decnt: Deep deformable cnn for table detection. *IEEE Access*, 6:74151–74161, 2018.

[26] Yilun Huang, Qinqin Yan, Yibo Li, Yifan Chen, Xiong Wang, Liangcai Gao, and Zhi Tang. A yolo-based table detection method. In *2019 International Conference on Document Analysis and Recognition (ICDAR)*, pages 813–818. IEEE, 2019.

[27] Madhav Agarwal, Ajoy Mondal, and CV Jawahar. Cdec-net: Composite deformable cascade network for table detection in document images. *arXiv preprint arXiv:2008.10831*, 2020.

[28] Shoaib Ahmed Siddiqui, Imran Ali Fateh, Syed Tahseen Raza Rizvi, Andreas Dengel, and Sheraz Ahmed. Deeptabstr: Deep learning based table structure recognition. In *2019 International Conference on Document Analysis and Recognition (ICDAR)*, pages 1403–1409. IEEE, 2019.

[29] Sebastian Schreiber, Stefan Agne, Ivo Wolf, Andreas Dengel, and Sheraz Ahmed. Deepdesrt: Deep learning for detection and structure recognition of tables in document images. In *2017 14th IAPR international conference on document analysis and recognition (ICDAR)*, volume 1, pages 1162–1167. IEEE, 2017.

[30] Shubham Singh Paliwal, D Vishwanath, Rohit Rahul, Monika Sharma, and Lovekesh Vig. Tablenet: Deep learning model for end-to-end table detection and tabular data extraction from scanned document images. In *2019 International Conference on Document Analysis and Recognition (ICDAR)*, pages 128–133. IEEE, 2019.

[31] Darshan Adiga, Shabir Ahmad Bhat, Muzaffar Bashir Shah, and Viveka Vyeth. Table structure recognition based on cell relationship, a bottom-up approach. In *Proceedings of the International Conference on Recent Advances in Natural Language Processing (RANLP 2019)*, pages 1–8, 2019.

[32] Sachin Raja, Ajoy Mondal, and CV Jawahar. Table structure recognition using top-down and bottom-up cues. In *European Conference on Computer Vision*, pages 70–86. Springer, 2020.

[33] Devashish Prasad, Ayan Gadpal, Kshitij Kapadni, Manish Visave, and Kavita Sultanpure. Cascadetabnet: An approach for end to end table detection and structure recognition from image-based documents. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 572–573, 2020.

[34] Xinyi Zheng, Douglas Burdick, Lucian Popa, Xu Zhong, and Nancy Xin Ru Wang. Global table extractor (gte): A framework for joint table identification and cell structure recognition using visual context. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 697–706, 2021.

[35] Shah Rukh Qasim, Hassan Mahmood, and Faisal Shafait. Rethinking table recognition using graph neural networks. In *2019 International Conference on Document Analysis and Recognition (ICDAR)*, pages 142–147. IEEE, 2019.

[36] Zewen Chi, Heyan Huang, Heng-Da Xu, Houjin Yu, Wanxuan Yin, and Xian-Ling Mao. Complicated table structure recognition. *arXiv preprint arXiv:1908.04729*, 2019.

[37] Ángela Casado-García, César Domínguez, Jónathan Heras, Eloy Mata, and Vico Pascual. The benefits of close-domain fine-tuning for table detection in document images. In *International Workshop on Document Analysis Systems*, pages 199–215. Springer, 2020.

[38] Max Göbel, Tamir Hassan, Ermelinda Oro, and Giorgio Orsi. Icdar 2013 table competition. In *2013 12th International Conference on Document Analysis and Recognition*, pages 1449–1453. IEEE, 2013.

[39] Richard Zanibbi, Dorothea Blostein, and James R Cordy. A survey of table recognition. *Document Analysis and Recognition*, 7(1):1–16, 2004.

[40] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.

[41] Ross Girshick. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015.

[42] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: towards real-time object detection with region proposal networks. *IEEE transactions on pattern analysis and machine intelligence*, 39(6):1137–1149, 2016.

[43] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017.

[44] Zhaowei Cai and Nuno Vasconcelos. Cascade r-cnn: high quality object detection and instance segmentation. *IEEE transactions on pattern analysis and machine intelligence*, 2019.

[45] Rafael Padilla, Wesley L Passos, Thadeu LB Dias, Sergio L Netto, and Eduardo AB da Silva. A comparative analysis of object detection metrics with a companion open-source toolkit. *Electronics*, 10(3):279, 2021.