VNU Journal of Science: Comp. Science & Com. Eng, Vol. 37, No. 2 (2021) 47-59

VNU Journal of Science:
Computer Science and Communication Engineering

Journal homepage: http://www.jcsce.vnu.edu.vn/index.php/jcsce

Original Article

Automatic Building of a Large and Straightforward Dataset for Image-Based Table Structure Recognition

Vinh Quang Tran, Diep Nguyen Thi Ngoc*

VNU University of Engineering and Technology, 144 Xuan Thuy, Cau Giay, Hanoi, Vietnam

Received 14 May 2021
Revised 27 August 2021; Accepted 1 November 2021

Abstract: Table is one of the most common ways to represent structured data in documents. Existing researches on image-based table structure recognition often rely on limited datasets with the largest amount of 3,789 human-labeled tables as ICDAR 19 Track B dataset. A recent Table Bank dataset for table structures contains 145K tables, however, the tables are labeled in an HTML tag sequence format, which impedes the development of image-based recognition methods. In this paper, we propose several processing methods that automatically convert an HTML tag sequence annotation into bounding box annotation for table cells in one table image. By assembling these methods, we could convert 42,028 tables with high correctness, which is 11 times larger than the largest existing dataset (ICDAR 19). We then demonstrate that using these bounding box annotations, a straightforward representation of objects in images, we can achieve much higher F1-scores of table structure recognition at many high IoU thresholds using only off-the-shelf deep learning models: F1-score of 0.66 compared to the state-of-the-art of 0.44 for ICDAR19 dataset. A further experiment on using explicit bounding box annotation for image-based table structure recognition results in higher accuracy (70.6%) than implicit text sequence annotation (only 33.8%). The experiments show the effectiveness of our largest-to-date dataset to open up opportunities to generalize on real-world applications. Our dataset and experimental models are publicly available at shorturl.at/hwHY3.

Keywords: table detection, table structure recognition, deep learning, dataset.

1. Introduction

Table is one of the most commonly used methods to represent information in documents, thanks to its intuitiveness of representing structured data. 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 process them. However, since most of these documents are either in image or exchange format like PDF, they do not contain

* Corresponding author.
E-mail address: ngocdiep@vnu.edu.vn

https://doi.org/10.25073/2588-1086/vnucsce.230

the layout coding of the documents, making it hard for users to use these files efficiently. Based on the valuable information contained in tables, many researchers have been focusing on the problem of processing and recognizing the structures of tables from image input.

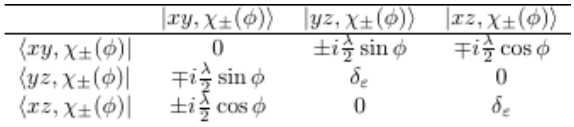
Table 1. Number of tables (as images) and table cells (as bounding boxes) in several popular table structure recognition datasets compared to our dataset

Dataset	# of Tables	# Cells
TableBank [5]	145,463	(n/a)
ICDAR 13 [6]	154	14,114
ICDAR 19 [7]	3,789	371,045
TabStructDB[4]	1,081	53,174
Ours	42,028	1,234,518

Based on previous research [1–4], we define two important tasks related to table processing as below:

- *Table Detection:* Detect the coordinates or regions of all tables inside a document.
- *Table Structure Recognition:* For each located table, its structural information needs to be recovered. The structural information is defined by rows', columns' or cells' location which form the layout of the table. The common and simple representation of this structure is via the bounding boxes of all cells in the table as shown in 1(c).

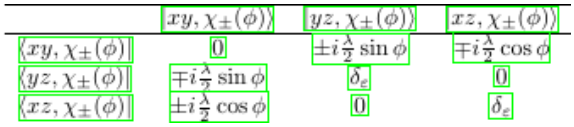
While it is important to solve table structure recognition, only a few datasets for this task are available and many of which have a modest number of tables. The most popular datasets are shown in table 1 and the largest dataset with bounding box annotation is ICDAR 19 (3,789 table images). Although the TableBank dataset has over 417K document images whose tables are labeled in bounding box format, its separated set for table structure recognition contains only 145k images, of which all are annotated using sequential HTML tags, providing no information about the location of table cells in the table images. An example of such ground truth for the table image in Figure 1(a) is shown in Figure 1(b). Compared to the intuitive bounding box annotation in Figure 1(c), this kind of annotation is not suitable for the task of image-based table structure recognition. It is because even if we can successfully predict the HTML tags of a table image, we don't have any information of the tag's coordinate to infer the location of the corresponding table cell. Figure 1 also shows that the table structure recognition task requires a fine-grain detection of bounding boxes for all table cells in the table, however, the labor cost for annotating this kind of information is undeniably high.



(a) A sample table image

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

(b) Groundtruth annotation in TableBank [5]



(c) Expected output of table structure recognition

Figure 1. An example of table image, groundtruth annotation in TableBank dataset [5], and expected output of table structure recognition which is also our model's result.

In this paper, we propose several methods to automatically convert the HTML tag sequences into bounding box locations in the table images

using the visual information and the HTML tags. Ensembling the proposed converting methods, we obtain a very large dataset for table structure recognition task, which includes over 42K table images and 1.2M table cells with relatively high correctness (see Table 1). Our dataset is **11 times larger** than the current largest dataset (ICDAR 19) for the same task with table cell level annotation. To demonstrate the effectiveness of our new dataset, we use an off-the-shelf and also a state-of-the-art model on table detection and table structure recognition model, CascadeTableNet [2], to train a table structure recognition model called CellNet. The experiments show that our large dataset helps generalize well on both tasks. Especially when testing on other existing datasets (such as ICDAR 19), it outperforms the CascadeTableNet [2] with a large margin of F1 score on all IoU thresholds. Additionally, we show that using explicit bounding box annotation yields higher accuracy than implicit text sequence annotation.

In summary, our contributions in this paper are:

- Converting methods from HTML tag sequence format to bounding box coordinate format for table images;
- Generating a very large dataset (42K images) for table structure recognition task;
- Training a table structure recognition model (CellNet) that outperforms the state-of-the-art models on ICDAR 19 and TableBank datasets.

The paper is organized as follows: Section 2 discusses some related works. Section 3 describes our proposed converting methods. Section 4 describes some experiments and results. Section 5 concludes the paper.

2. Related Works

Most table structure recognition methods can be divided into two main categories: rule-based and data-driven approaches. Rule-based methods are those that rely on some contents in tables such as text content, graphical lines, and arrangements to make a set of predefined rules to find the structure of the tables. On the other hand, data-driven primarily rely on machine learning techniques to recognize the structures of tables directly from the images.

Rule-based approaches: In the early 1990s, Chandran and Kasturi [8] proposed a method to detect tables from binary document images using sets of predefined conditions for table lines. Itonori [9] used text-block arrangement and ruled lines to detect tables in document images. Smith [10] proposed using Tesseract OCR with layout analysis to locate table regions. Kieninger and Dengel [11] demonstrated a bottom-up method to recognize logical structure of tables without relying on visual clues like lines. Van Nguyen et al. [12] proposed a method for extracting table structure from scanned images using CRAFT to detect text areas.

Data-driven approaches: Cesarini et al. [13] first used Modified X-Y tree as a hierarchical representation of a document and then used supervised learning to find different classes of tables in the tree. Kasar et al. [14] proposed finding intersecting points and using SVM classifiers to locate tables. Hao et al. [15] was the first to try localizing tables in PDF files using deep learning technique, while still relying on predefined rules. Gilani et al. [16], Siddiqui et al. [17] proposed the use of Faster R-CNN to locate tables in document images, completely eliminate the need for heuristics rules. Siddiqui et al. [4] also used Faster R-CNN to locate tables' rows and columns. Similarly, Schreiber et al. [3] also used Faster R-CNN but treated structure recognition task as rows and columns segmentation. Prasad et al. [2] proposed CascadeTabNet which used Cascade Mask R-CNN to locate tables and their cells.

Comparing those two approaches, rule-based is often limited in well-crafted logic (rules) whereas data-driven requires a large

amount of data in order to show effectiveness. Regarding the table structure recognition problem, the largest existing dataset (ICDAR 19) contains about 3.7K images. In this paper, we firstly build a larger dataset and then we use CascadeTabNet as our off-the-shelf model of choice to show how a large dataset can significantly improve the model performance.

3. Methods

As we have discussed in Section 1, the TableBank Recognition dataset [5] is large but limited by the HTML tag annotation. In this section, we will describe several methods that we use to convert the HTML tag annotation to bounding box annotation for table images. The conversion principle is based on finding the location of each cell in a table image. We utilize the HMTL tag annotation as additional information to check whether the new annotations are correct or not. Moreover, due to the variability of table structure and appearance, we propose four different methods which leverage Digital Image Processing and Deep Learning techniques to find the table cells. Each method is hypothetically designed to cover others' weaknesses. The parameters or thresholds used in each method were chosen by experiments.

3.1. Method 1: For Fully-bordered Tables

Since table lines usually provide valuable information about table structure, this method uses them to find table cells of fully-bordered tables. The overall process is shown in Figure 6.

In the first method, the image is thresholded into a binary image and then inverted. In our experiments, the threshold value of 192 is chosen because it can help generate the most correct annotations. After the image is inverted, a border surrounding the image is drawn to ensure consistency between examples in the dataset.

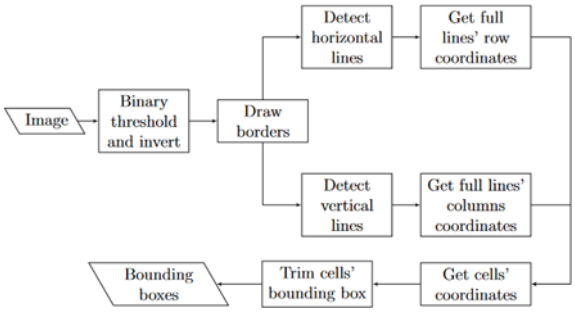


Figure 2. Pipeline for fully-bordered tables (method 1).

To generate masks for table 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 $(hor\_krn\_len, 1)$. For vertical lines, the kernel's shape is $(1, vert\_krn\_len)$. The hor_krn_len and $vert\_krn\_len$ are calculated as follows:

$$hor\_krn\_len = \left\lfloor \left\lfloor \frac{image\ width}{\max num.of\ cols.} \right\rfloor * \frac{1}{4} \right\rfloor \quad (1)$$

$$vert\_krn\_len = \left\lfloor \left\lfloor \frac{image\ height}{num.of\ rows} \right\rfloor * \frac{1}{4} \right\rfloor * 3 \quad (2)$$

with $\max num.of\ columns$ is the maximum number of cells of all table rows and $num.of\ rows$ is the number of table rows taken from original annotation files.

Unlike existing methods in [15, 18, 19] which do text block projection to recover table cells, our method will do horizontal and vertical lines projection 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 needs 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.

Figure 3. A typical example of false conversion by Method 1, which happens when a table has spanning cells

Number of plants	ROI (%)			Payback time (years)		
	Low	Medium	High	Low	Medium	High
7	-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

With this method, we were able to generate annotations for 29,285 images, which account for 20.13% of the dataset. As one observation, we see that this method does not generalize well with tables having spanning cells. An example is shown in Figure 3.

3.2. Method 2: For Fully-bordered Tables with Spanning Cells

This method is designed to overcome the weakness of the first method regarding spanning cells. It works with any fully-bordered tables with multiple spanning cells. The overall process is shown in Figure 4.

The beginning steps of this approach are the same as the first method until the getting rows' and columns' coordinates step. If a table contains one or more spanning cells, any lines staying next to spanning cells are not recognized in the previous method.

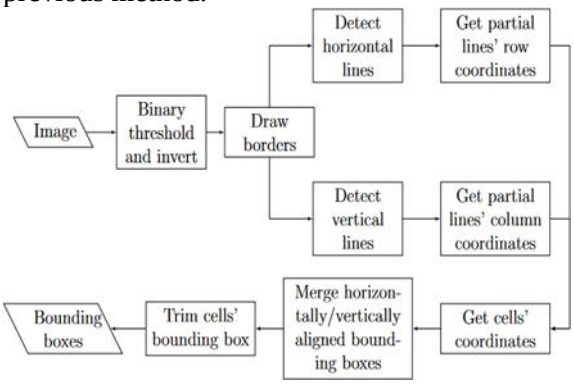


Figure 4. Converting pipeline for fully-bordered tables with spanning cells (method 2).

For example, in Figure 3, 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 the 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 1.

Number of plants	ROI (%)			Payback time (years)		
	Low	Medium	High	Low	Medium	High
7	-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 5. A typical example error of method 2 due to redundant bounding boxes on spanning cells after cell recovery step.

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 5. Since

this structure is incorrect, these cells need to be merged as one in the next step. For every pair of table cells, this step checks 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 is 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 1 to get the final result.

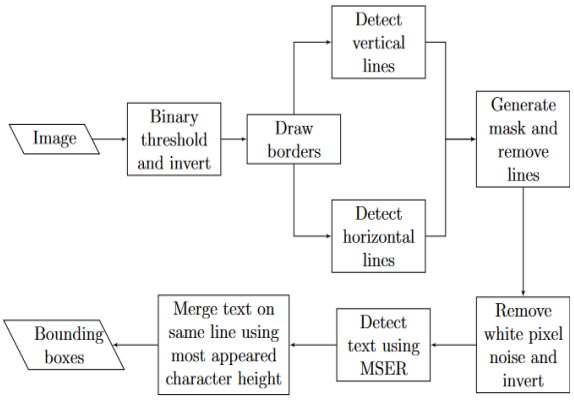


Figure 6. Converting pipeline for general tables (method 3).

By using this approach, 41,256 annotations were generated (28.36% of the dataset), making this method the most effective one. While this method can be considered an upgrade version of method 1, we decided to keep both methods because in some cases, a table cell can contain characters standing near each other or equations that form a line. In such cases, this method can mistake them as a table borderline, while the previous method does not. Therefore, it is necessary to keep method 1 to counter such cases.

3.3. Method 3: For General Tables

Since previous methods only work on fully-bordered tables, this method is designed to work with all types of tables, regardless of how they are bordered. It works by using Maximally Stable Extremal Regions (MSER) to detect table cells. The steps are shown in Figure 6.

Similar to the first two methods, the first step involves binary thresholding and inverting the image. The chosen thresholding value for this approach is 225 because it can generate the most correct annotations. Then, the border surrounding the image is also drawn. Like the previous methods, we use a vertical and a horizontal kernel with the same size calculated in Equation 1 and 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.

After generating horizontal and vertical line masks, they are combined using bitwise OR operation to generate a table line mask. Subtracting the inverted image earlier with this mask should remove the table lines from the image, keeping only the text. Because some images may have some noisy small pixels near the table lines, a structuring element with the size of $(1,1)$ is used to remove these noise using opening operation, which erodes and then dilates the image using the provided kernel.

Then, the image is inverted and MSER [20] is used to detect text areas in the image. Any areas smaller than 4 pixels are set to be pruned to ignore any noises 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 [21], we use most appeared bounding boxes' height to merge them. The structuring element used for merging bounding boxes has the shape of $(most\ appeared\ height * 1.3, 1)$. Finally, connected components in the image are

extracted as rectangles and used as the final bounding boxes result.

By using this method, we were able to generate annotations for about 26.66% of the whole dataset (38,783/145,463 images).

3.4. Method 4: For Colored Tables

Previous methods do not work correctly if the text areas have pixel values higher than the threshold value and vice versa for background areas. Therefore, method 4 is designed to work with these types of tables. Similar to [12], this method also uses CRAFT (Character-Region Awareness For Text detection) [22] to detect text areas and then locate the cell location of the table. CRAFT is particularly helpful in cases where table cells and text are colored as it is capable of detecting them, regardless of their colors.

First, the image is passed to the CRAFT model to generate text region proposals. Since the results of CRAFT are polygons that have angles, they are converted to rectangle bounding boxes instead because all images in the dataset are computer-generated, meaning that the text is 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 7. 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 do not satisfy this condition do 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 using the following equation:

$$thresh = \left\lfloor \frac{\text{min. height of 2 bboxes}}{n} \right\rfloor \quad (3)$$

In method 4a, $n = 4$ for both horizontal and vertical distance. In method 4b, $n = 3$ for vertical distance and $n = 2$ for horizontal distance. The purpose of two sets of values is to generate correct annotations as many as it can with this method.

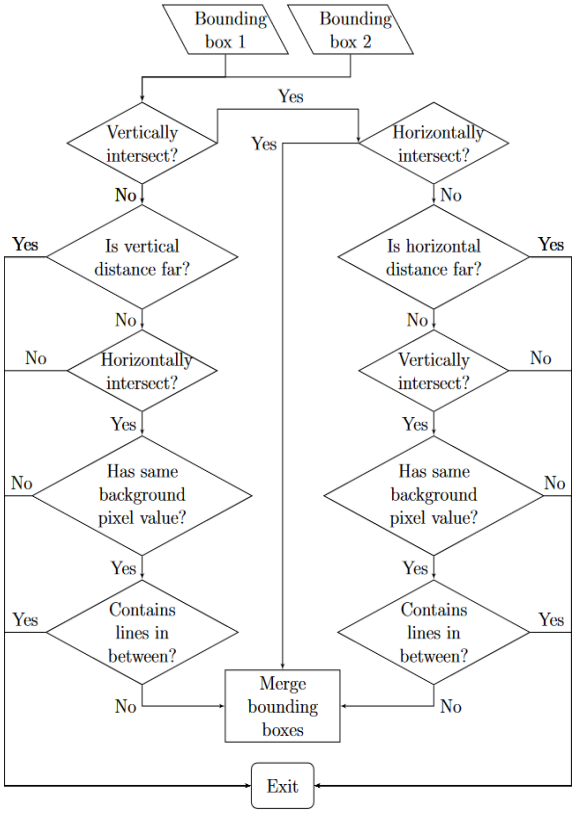


Figure 7. Merging deep learning-based detected bounding boxes in method 4 for colored tables.

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. We 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. 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 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, we 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, we were only able to generate annotations for 3,301 (2.26%) images using 4a's configuration and 3,374 (2.31%) images using 4b's. The main culprit for its weak performance is the CRAFT model, which cannot detect all the text areas in most images and thus, did not generate all cells needed to represent the structure of the tables.

3.5. Verifying Bounding Box Annotations

The bounding box annotations generated automatically by the above-described methods still need further verification. Because the only annotation available (ground truth) for this dataset is the HTML sequence tags, they will be used to verify if the bounding boxes match the structure of the table (or the HTML tags). We propose two methods used to verify this.

3.5.1. 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 with 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 which row in the table a bounding box belongs to. Then, all bounding boxes in the 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 is equal for the same row in both annotations. If the number of cells in a row does not match, the new annotation is considered wrong. The number of detected cells is also compared to the total number of cells of that row in the original file as well to avoid wrong annotations from the original file. If either condition stands correct, it means the row has the 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, 3, and 4.

3.5.2. 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. 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 matches the number of visible cells or the total number of cells in the XML file. This method is only used for method 2 because the previous approach requires sorting the structure from the bounding boxes, which is complex to implement for tables with spanning cells.

3.6. Dataset Generation

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. Since the number of bounding boxes was very large (2,888,543 table cells), in order to verify them without having to check every single file, we define a set of rules to filter the highly correct table images.

1. The number of detected bounding boxes and the number of tag-based cells in the HTML annotation files are equal.

2. Every bounding box must intersect with a bounding box generated from a sufficiently good table structure recognition model.

3. IoU of these bounding boxes must be greater than or equal to 0.5.

In order to complete this, we trained a table structure recognition model. First, we 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.

The model that we used had the same architecture with CascadeTabNet [2]. It has a Cascade mask R-CNN network with HRNetV2p as the backbone layer to extract high-resolution features and generate initial bounding boxes. It was trained on the sampled 4,500 table images for 20 epochs as the mAP across multiple thresholds from 0.5 to 0.95 fluctuated between 70 and 78%. On test set, the model reached 87% mAP at IoU threshold of 0.75, which can be considered good enough for checking the annotations.

After obtaining this model, we applied the above-described rule set to 73,140 images whose annotations had been created earlier. 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, that has 42,028 table images with bounding box annotations of 1,234,518 cells in total. This dataset is split into 37,825 for training and 4,203 for test.

Finally, to check the accuracy of generated ground truth, we sample 1,000 images from the 42k image dataset and manually check them. Out of 1,000 images, 995 tables have correct annotations whereas only 5 tables have slightly incorrect annotations. This implies that our new dataset is highly reliable (99.5% accuracy).

4. Experiments and Results

4.1. Experiments

For the table detection task, we trained CascadeTabNet model on TableBank Detection dataset 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.

Model	P	R
CascadeTabNet [2]	92.99%	95.71%
CascadeTabNet (ours)	97.90%	99.20%

Table 2. Results on CascadeTabNet's TableBank dataset [2]. P: Precision; R: Recall

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 the model was trained for only 5 epochs. Evaluation was done on the merged TableBank dataset from [2], as well as the TableBank LaTeX dataset for comparison with other models from [23].

For the table structure recognition task, a similar CascadeTabNet model, which we called CellNet model is used. Unlike original CascadeTabNet [2] that do both table detection and structure recognition using a single model, we used CascadeTabNet for table detection and CellNet for table structure detection. CellNet was continued training from the model used to verify annotations in Section 3.6 on the new dataset. Since it was already pretrained, mAP did not change much during training so it was only trained for another 10 epochs. The evaluation was done on ICDAR 19 cTDaR Track B2 (Modern) for comparison with other approaches using toolkit by [24]. Because CellNet can only recognize cells from table images, it is not directly comparable to other methods without some processing. To do that, first, we passed the document images to CascadeTabNet to detect tables in the document. Then, we kept any tables with confidence score ≥ 0.7 and passed the cropped images to CellNet to detect table cells. We also compared our model's performance in two additional cases:

- CellNet (no overlap): Since CascadeTabNet can detect a table with different bounding boxes with high confident score, this can make our CellNet detect a table cell multiple times. Because of that, after getting the result from CellNet, we removed any bounding boxes that overlap each other, while keeping only the one with the highest confident score to evaluate the model.

- CellNet (with GT table): In this case, we use the ground truth tables of ICDAR 19 as input for our CellNet. This is to understand how effective the CellNet can be in recognizing table cells when table detection is performed perfectly.

4.2. Results

4.2.1. Table Detection

In the table detection task, our model is able to achieve better results on both TableBank Detection dataset when compared to the original CascadeTabNet model [2]. This is as expected since the authors only trained their model on 3,000 images sampled from the dataset, while ours was trained on the whole dataset. Since the number of images is large, their augmentation techniques can be considered unnecessary to achieve better results. The result is reported in Table 2.

Table 3. Results on LaTeX TableBank dataset. P: Precision. R: Recall.

Model	IoU@0.6		IoU@0.7		IoU@0.8		IoU@0.9	
	P	R	P	R	P	R	P	R
Mask R-CNN [23]	94%	98%	94%	97%	93%	96%	84%	87%
RetinaNet [23]	98%	86%	98%	86%	97%	85%	94%	82%
SSD [23]	96%	97%	94%	95%	92%	92%	82%	82%
YOLO [23]	98%	99%	98%	99%	96%	97%	74%	75%
CascadeTabNet (ours)	**98.7%**	**99.6%**	**98.7%**	**99.4%**	**98.5%**	**99.0%**	**95.9%**	**97.2%**

Table 4. F1-scores on ICDAR 19 Track B2 (Modern) dataset. WAvg.: Weighted Average:

Model	F1-score at IoU threshold				WAvg.
	@0.6	@0.7	@0.8	@0.9	
CascadeTabNet [2]	0.438	0.354	0.190	0.036	0.232
NLPR-PAL [2]	0.365	0.305	0.195	0.035	0.206
CellNet (ours)	0.645	0.477	**0.220**	0.040	0.310
CellNet (ours – no overlap)	**0.656**	**0.483**	0.219	0.040	**0.314**
CellNet (ours – with GT table)	0.579	0.436	0.204	**0.044**	0.285

Furthermore, we compared our model to those trained in [23]. Since they only evaluated their models on TableBank Detection (LaTeX) dataset, we also did the same with our trained model (which was trained with both Word and LaTeX sets). The result is shown in Table 3. 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 our 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 that are close to the ground truth.

4.2.2. Table Structure Recognition

On ICDAR 19 Track B2 (Modern) dataset, our model was able to achieve the best performance when compared with the original CascadeTabNet model and NLPR-PAL method. Detailed result is shown in Table 4. Without the need for post-processing like in CascadeTabNet, CellNet outperformed the others significantly when the IoU threshold was low and slightly better when it was high. The result was further improved if any overlapped bounding boxes were removed after the detection. When using table ground truth as the input for the model, it only outperformed CascadeTabNet and NLPR-PAL. But at the highest IoU threshold (0.9), it outperformed all other models.

Table 5. Table structure recognition results on 1,000 sampled table images from TableBank dataset, in which we manually compare our CellNet results with the ground truth HTML tags and visual bounding boxes

	Length	0-20	21-40	41-60	61-80	>80	All
	#Total	32	293	252	145	278	1,000
TableBank [5]	#Exact match	15	169	102	28	24	338
	Ratio	0.469	0.577	0.405	0.193	0.086	0.338
CellNet (ours – tags GT)	#Exact match	21	237	192	89	103	642
	Ratio	0.656	0.809	0.762	0.614	0.371	0.642
CellNet (ours – bounding boxes)	Correct	23	250	207	101	125	706
	Ratio	0.719	0.853	0.821	0.697	0.450	0.706

We also evaluated the effect of having explicit bounding box annotation on the model's performance by comparing the result to TableBank's image-to-text model. Since the result is not directly comparable, we evaluated using two methods:

- Comparing the bounding boxes' arrangement to the ground truth HTML annotation.
- Check if bounding boxes are annotated correctly according to the table.

The results (see Table 5) showed that in both comparisons, our method performed significantly better than that of the proposed method in TableBank [5]. This proved that explicit bounding box annotation is more effective than implicit text sequence annotation.

5. Conclusions

In this paper, we have proposed an automated approach to create a large dataset for table structure recognition with cell-level bounding box annotation. The dataset contains more than 42K table images with a reliable accuracy (99.5%). The dataset surpasses the largest existing dataset for the recognition task by 11 times. Our experiments have also shown that the table structure recognition model trained on this new dataset outperforms all existing solutions on several benchmark datasets.

Our contributions in this paper are summarized as follows:

- Proposed several methods to automatically generate bounding box annotations from HTML tag annotation of TableBank dataset.
- Generated a new large dataset with over 42K table images for structure recognition task.
- Developed a table structure recognition model using the newly generated dataset resulting in significant improvement over previous works.

Our upcoming works will involve using the trained table structure recognition model to supervise the conversion process from HTML tag sequences to coordinate annotation, retraining the recognition models, and optimizing recognition performance with other deep learning techniques. We may also combine several datasets to increase the generalization of the new dataset.

Acknowledgments

This work has been supported by VNU University of Engineering and Technology under the project number CN19.10.

References

[1] S. S. Paliwal, D. Vishwanath, R. Rahul, M. Sharma, L. 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), IEEE, 2019, pp. 128–133.

[2] D. Prasad, A. Gadpal, K. Kapadni, M. Visave, K. 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, 2020, pp. 572–573.

[3] S. Schreiber, S. Agne, I. Wolf, A. Dengel, S. Ahmed, Deepdesrt: Deep Learning for Detection and Structure Recognition of Rables in Document Images, in: 2017 14th IAPR international conference on document analysis and recognition (ICDAR), Vol. 1,

[4] S. A. Siddiqui, I. A. Fateh, S. T. R. Rizvi, A. Dengel, S. Ahmed, Deeptabstr: Deep Learning Based Table Structure Recognition, in: 2019 International Conference on Document Analysis and Recognition (ICDAR), IEEE, 2019, pp. 1403–1409.

[5] M. Li, L. Cui, S. Huang, F. Wei, M. Zhou, Z. Li, Tablebank: A Benchmark Dataset for Table Detection and Recognition, arXiv e-prints (2019) arXiv–1903.

[6] M. Göbel, T. Hassan, E. Oro, G. Orsi, Icdar 2013 Table Competition, in: 2013 12th International Conference on Document Analysis and Recognition, IEEE, 2013, pp. 1449–1453.

[7] L. Gao, Y. Huang, H. Déjean, J.-L. Meunier, Q. Yan, Y. Fang, F. Kleber, E. Lang, Icdar 2019 Competition on Table Detection and Recognition (ctdar), in: 2019 International Conference on

Document Analysis and Recognition (ICDAR), IEEE, 2019, pp. 1510–1515.

[8] S. Chandran, R. Kasturi, Structural Recognition of Tabulated Data, in: Proceedings of 2nd International Conference on Document Analysis and Recognition (ICDAR'93), IEEE, 1993, pp. 516–519.

[9] K. 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), IEEE, 1993, pp. 765–768.

[10] F. Shafait, R. Smith, Table Detection in Heterogeneous Documents, in: Proceedings of the 9th IAPR International Workshop on Document Analysis Systems, 2010, pp. 65–72.

[11] T. Kieninger, A. Dengel, The t-recs Table Recognition and Analysis System, in: International Workshop on Document Analysis Systems, Springer, 1998, pp. 255–270.

[12] N. V. Nguyen, H. Vu, A. Zucker, Y. Belkada, H. V. Do, D. N. Nguyen, T. T. N. Le, D. V. Hoang, Table Structure Recognition in Scanned Images Using a Clustering Method, in: International Conference on Industrial Networks and Intelligent Systems, Springer, 2020, pp. 150–162.

[13] F. Cesarini, S. Marinai, L. Sarti, G. Soda, Trainable Table Location in Document Images, in: Object Recognition Supported By User Interaction for Service robots, Vol. 3, 2002, pp. 236–240.

[14] T. Kasar, P. Barlas, S. Adam, C. Chatelain, T. Paquet, Learning to Detect Tables in Scanned Document Images Using Line Information, in: 2013 12th International Conference on Document Analysis and Recognition, IEEE, 2013, pp. 1185–1189.

[15] L. Hao, L. Gao, X. Yi, Z. Tang, A Table Detection Method for Pdf Documents based on Convolutional Neural Networks, in: 2016 12th IAPR Workshop on Document Analysis Systems (DAS), IEEE, 2016, pp. 287–292.

[16] Gilani, S. R. Qasim, I. Malik, F. Shafait, Table Detection Using Deep Learning, in: 2017 14th IAPR international conference on document analysis and recognition (ICDAR), Vol. 1, 2017, pp. 771–776.

[17] S. A. Siddiqui, M. I. Malik, S. Agne, A. Dengel, S. Ahmed, Decnt: Deep Deformable Cnn For Table Detection, IEEE Access, Vol. 6, 2018. Pp. 74151–74161.

[18] Shigarov, A. Mikhailov, A. Altaev, Configurable Table Structure Recognition in Untagged Pdf Documents, in: Proceedings of the 2016 ACM Symposium on Document Engineering, 2016, pp. 119–122.

[19] Y. Wang, I. T. Phillips, R. M. Haralick, Table Structure Understanding and Its Performance Evaluation, Pattern recognition, Vol. 37, 2004, pp. 1479–1497.

[20] M. Donoser, H. Bischof, Efficient Maximally Stable Extremal Region (Mser) Tracking, in: 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06), Vol. 1, Ieee, 2006, pp. 553–560.

[21] B. Gatos, D. Danatsas, I. Pratikakis, S. J. Perantonis, Automatic Table Detection In Document Images, in: International Conference on Pattern Recognition and Image Analysis, Springer, 2005, pp. 609–618.

[22] Y. Baek, B. Lee, D. Han, S. Yun, H. Lee, Character Region Awareness for Text Detection, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2019, pp. 9365–9374.

[23] Á. Casado-Garc´ıa, C. Domínguez, J. Heras, E. Mata, V. Pascual, The Benefits of Close-Domain Fine-Tuning for Table Detection in Document Images, in: International Workshop on Document Analysis Systems, Springer, 2020, pp. 199–215.

[24] R. Padilla, W. L. Passos, T. L. Dias, S. L. Netto, E. A. da Silva, A Comparative Analysis of Object Detection Metrics with a Companion Open-Source Toolkit, Electronics Vol. 10, 2021, pp. 279.