Segmentation of News Items from Newspaper Images

Thesis submitted in partial fulfillment of requirements
For the degree of
Master of Computer Application
of
Computer Science and Engineering Department
of
Jadavpur University

by

Vishal Kumar Chauhan
Regn. No. - 154257 of 2020-2021
Exam Roll No. - MCA2360032

under the supervision of

Dr. Mahantaps Kundu
Professor

Department of Computer Science and Engineering
Jadavpur University
Kolkata, West Bengal, India
2023

Certificate from the Supervisor

This is to certify that the work embodied in this thesis entitled **"Segmentation of News Items from Newspaper Images"** has been satisfactorily completed by **Vishal Kumar Chauhan** (Registration Number 154257 of $2020 - 21$; Class Roll No. 002010503049; Examination Roll No. $MCA2360032$). It is a bona-fide piece of work carried out under my supervision and guidance at Jadavpur University, Kolkata for partial fulfilment of the requirements for the awarding of the **Master of Computer Application** degree of the Department of Computer Science and Engineering, Faculty of Engineering and Technology, Jadavpur University, during the academic year $2022 - 23$.

Dr. Mahantapas Kundu,
Professor,
Department of Computer Science and Engineering,
Jadavpur University.
(Supervisor)

Forwarded By:

Prof. Nandini Mukherjee,
Head,
Department of Computer Science and Engineering,
Jadavpur University.

Prof. Ardhendu Ghoshal,
Dean,
Faculty of Engineering & Technology,
Jadavpur University.

Department of Computer Science and Engineering
Faculty of Engineering And Technology
Jadavpur University, Kolkata - 700 032

Certificate of Approval

This is to certify that the thesis entitled **"Segmentation of News Items from Newspaper Images"** is a bona-fide record of work carried out by **Vishal Kumar Chauhan** (Registration Number 154257 of $2020 - 21$; Class Roll No. 002010503049; Examination Roll No. $MCA2360032$) in partial fulfilment of the requirements for the award of the degree of **Master of Computer Application** in the **Department of Computer Science and Engineering, Jadavpur University**, during the period of January 2023 to May 2023. It is understood that by this approval, the undersigned do not necessarily endorse or approve any statement made, opinion expressed or conclusion drawn therein but approve the thesis only for the purpose of which it has been submitted.

Examiners:

(Signature of the Examiner)

(Signature of the Supervisor)

Department of Computer Science and Engineering
Faculty of Engineering And Technology
Jadavpur University, Kolkata - 700 032

Declaration of Originality and Compliance of Academic Ethics

I hereby declare that the thesis entitled **"Segmentation of News Items from Newspaper Images"** contains literature survey and original research work by the undersigned candidate, as a part of his degree of **Master of Computer Application** in the **Department of Computer Science and Engineering, Jadavpur University**. All information have been obtained and presented in accordance with academic rules and ethical conduct.

I also declare that, as required by these rules and conduct, I have fully cited and referenced all materials and results that are not original to this work.

Name: Vishal Kumar Chauhan

Examination Roll No.: MCA2360032

Registration No.: 154257 of $2020 - 21$

Thesis Title: "Segmentation of News Items from Newspaper Images"

Signature of the Candidate:

ACKNOWLEDGEMENTS

I am pleased to express my gratitude and regards towards my Project Guides **Dr. Mahantapas Kundu & Dr. Nibaran Das**, Professor, Department of Computer Science and Engineering, Jadavpur University, without their valuable guidance, inspiration and attention towards me, pursuing my project would have been impossible.

Last but not the least, I express my regards towards my friends and family for bearing with me and for being a source of constant motivation during the entire term of the work.

Vishal Kumar Chauhan
MCA Final Year
Exam Roll No. - MCA2360032
Regn. No. - 154257 of 2020 – 21
Department of Computer Science and Engineering,
Jadavpur University.

Contents

List of Figures

Abstract

Segmenting news items from newspaper images is a challenging task due to the diverse layouts and overlapping content. In this paper, we propose an algorithm that utilizes the lines separating articles as key features for accurate segmentation. The algorithm focuses on identifying the rectangle points formed by vertical and horizontal lines. To provide a user-friendly interface, we develop a Web-API using Flask. Users can easily upload newspaper images and obtain segmented news items as output. The experimental results demonstrate the effectiveness of our approach in accurately segmenting news items from newspaper images.The algorithm is tested on a set of complex English newspaper images such as The Hindu, The Telegraph and The Statement.

Chapter 1

Introduction

Newspaper analysis plays a crucial role in various fields such as journalism, information retrieval, and data mining. Extracting relevant information from newspaper articles requires accurate segmentation of the text regions. However, segmenting news articles from newspaper images is a complex task due to the diverse layouts, varying font sizes, and overlapping content. Analyzing newspaper images is particularly difficult because the lines may not be connected, aligned perpendicular to the page edge, or even straight. Additionally, the spacing between columns can be smaller than the spaces between words, and sometimes separate components may appear connected.

There are many publications that address document analysis. The segmentation methods may be broadly divided into bottom-up[15, 21, 11, 16] and top-down[12, 14] methods.

Bottom-up approaches start with low-level components, such as connected components, and the goal is to merge and classify them accurately as document regions. One common technique is to use projection methods to locate lines within the document, and then group these regions to form paragraphs[11].

On the other hand, top-down approaches take a more holistic approach. They begin by identifying high-level structures, such as paragraphs or columns, and then further refine the segmentation by analyzing the content within these structures.

Both bottom-up and top-down methods have their advantages and challenges. The choice of approach depends on the specific requirements of the document analysis task and the characteristics of the documents being analyzed.

In this paper, we propose an algorithm to address these challenges and accurately segment news items from newspaper images. Our algorithm utilizes the lines that separate articles as key features for segmentation. By analyzing these lines, we aim to identify and extract individual news items from the newspaper images[18].For this the main algorithm focuses on addressing these challenges by finding the rectangle points formed by vertical and horizontal lines.We find the line using canny and Probabilistic HoughLine algorithm. After then finding the points of rectangle using these vertical line is main task [Chapter-3]. After the successful development of our algorithm for newspaper item segmentation we proceeded to create a Web-API[Chapter-4]. To provide a user-friendly interface for this segmentation process, we develop a Web-API using Flask, a popular Python web framework.

The experimental setup involves the implementation of the custom algorithm and the development of the Web-API. Through the Web-API, users can easily upload newspaper images and obtain segmented news items as output.

The algorithm is tested on a set of complex English newspaper images such as The Hindu, The Telegraph and The Statesman [Chapter-2].

Below figure show the Final images return by our Web-API: Below figure show the annotated images.

Figure 1.1: Result images obtain by applying the Segmentation algorithm that is used in Web-API

This work contributes to the field of newspaper analysis by presenting a custom algorithm and a user-friendly Web-API for accurate document paragraph segmentation. The experimental results highlight the effectiveness of our approach and its potential for enhancing newspaper analysis and information extraction tasks.

Chapter 2

Dataset and Preparation

This section describes how the necessary data was collected for the study. The research focused on gathering data from The Hindu, The Telegraph and The Statesman. The data collection process involved accessing the digital e-newspapers and using a data creation instrument to draw rectangles around the lines of interest by labelme annotation tool[4]. The annotations were then stored in a JSON file.

2.1 Data Sources

The main sources of data were The Hindu[2],The Telegraph[7] and The Statesman[6] newspapers. These newspapers were chosen because they have a lines that separate the articles and We accessed their digital archives with the necessary permissions from newspaper.

We collected data from The Hindu newspaper for a period of approximately two months, ranging from 01-04-2023 to 28-05-2023. This dataset consists of newspaper pages from various dates within this time frame.

Similarly, we collected data from The Telegraph newspaper for a period of approximately one month, ranging from 01-02-2023 to 21-02-2023. This dataset includes newspaper pages from different dates within this one-month duration.

Additionally, we collected data from The Statesman newspaper for a period of approximately one month, ranging from 01-02-2023 to 22-02-2023. This dataset comprises newspaper pages from different dates within this one-month timeframe.

In total, we gathered more than 50 images of each newspaper, carefully selecting them from various dates to ensure diversity and representativeness in our dataset. These images serve as valuable resources for our research and analysis. After then We Annotate theses Dataset.

2.2 Data Collection Instrument

To create the dataset, a data creation instrument was used. This instrument involved the use of the labelme annotation tool, which allowed annotators to draw rectangles or any type of polygon around paragraph images and articles. The rectangles defined the position and size of each annotated line.

The Labelme annotation tool is a popular and widely used software tool designed for image annotation and labeling tasks. It provides a user-friendly interface that allows users to manually annotate objects or regions of interest within an image.

The tool offers a range of annotation capabilities, such as drawing bounding boxes, polygons, and keypoints around objects, as well as labeling them with corresponding class labels. Users can easily navigate through images, zoom in and out, and precisely annotate objects using various tools provided by Labelme.

One of the key advantages of Labelme is its versatility in handling different types of annotation tasks. It supports the annotation of both single-object and multi-object images, making it suitable for a wide range of applications, including object detection, image segmentation, and instance segmentation.

Labelme also supports the import and export of annotations in popular formats such as JSON (JavaScript Object Notation). This allows for easy integration of the annotated data with other tools and frameworks used in computer vision and machine learning pipelines.

Moreover, Labelme is an open-source tool, which means that its source code is freely available for customization and extension. This makes it highly adaptable to specific annotation requirements and facilitates the development of new features by the community.

In summary, the Labelme annotation tool is a versatile and user-friendly software solution that enables efficient and accurate annotation of objects and regions in images. Its rich feature set, support for various annotation tasks, and open-source nature make it a valuable asset in the field of computer vision and image analysis.

2.3 Ground Truth Annotation

We established guidelines to ensure consistency and accuracy during the annotation process. These guidelines provided clear instructions on how to identify and annotate the lines within the articles. Multiple annotators were involved to minimize biases and errors. We measured the agreement between annotators to assess the reliability of the annotations.

2.3.1 Annotation Process

To begin the annotation process, the labelme annotation tool needs to be installed. This can be done by visiting the labelme website and downloading the repository, ensuring the required dependencies are installed, and running the installation command. Once labelme is installed, images of newspaper articles can be annotated using polygons. Launching labelme opens the interface in the app,where the image file of the newspaper article is opened for annotation. The polygon annotation mode is selected, and polygons are drawn around the lines of interest within the article. After annotating the polygons, the annotations are saved as a JSON file, including the image path, annotations, and labels. This process is repeated for each image in the dataset.

The annotation process involved the following steps:

1. **Install labelme:**
   Your system must have installed Anaconda, then run the below code in terminal.

```
conda create --name=labelme python=3
source activate labelme
pip install labelme
```

2. **Run labelme:**
   For running labelme make sure your are in right environment or you just run the commands given bleow.

```
source activate labelme
labelme
```

   The interface look like below image:

Figure 2.1: User Interface of labelme

3. **Create Polygons (Rectangles):**
   Onces labelme is opened then click on the "Open" button and select the image file of the newspaper article you want to annotate. The image will be displayed in the labelme interface.

   On the top you get a edit option click on it and select the rectangle option.

Figure 2.2: Editing using labelme

After then you are able to annotate the rectangles in the image.Start drawing the rectangle around the lines of interest within the newspaper article. Click to create vertices and continue clicking to create the shape of the polygon. Adjust the shape by dragging the vertices as needed. You can create multiple polygons if there are multiple lines to annotate.

After annotating a rectangles, you can assign a label to it by selecting the label from the drop-down menu on the left-hand side.

Repeat the process of drawing polygons and assigning labels for all the lines you want to annotate in the image.

Figure 2.3: Ground truth image formed by labelme

Save the annotations by clicking on the "Save" button.

Figure 2.4: Annotated example images

Below figure show the annotated images.

Figure 2.5: Annotated example images

Chapter 3

Proposed Methods

This chapter explains the technical details of the work done. It explain the methods that proposed to get the result. Below flow chart show the all methods step by step that used in this paper.

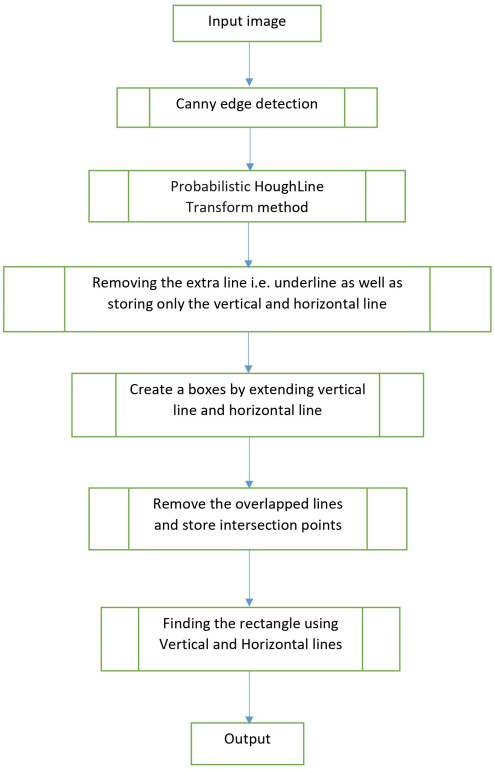


Figure 3.1: Flow chart for Proposed Methods

3.1 Canny edge detection:

Canny edge detection is an image processing technique used to detect edges in an image [19] [20] [10]. It was developed by John F. Canny in 1986 and has since become a widely used algorithm for edge detection.
The Canny edge detection algorithm involves several steps:

1. Noise reduction
2. Gradient calculation
3. Non-Max suppression
4. Double Thresholding
5. Hysteresis

3.1.1 Noise reduction:

Since edge detection is a susceptible to noise in the image, first step to remove the noise in the image with a Gaussian filter and get the smoothed image.

3.1.2 Gradient calculation:

Smoothed image is the filtered with a Sobel kernel in both **x** and **y** direction to get derivatives **Gx** and **Gy**. From this we get Gradient Intensity matrix and Gradient Angles.

$$\textbf{Edge Gradient(G)} = \sqrt{Gx2 + Gy2} \tag{3.1}$$

$$\textbf{Angle (theta)} = tan - 1(Gy/Gx) \tag{3.2}$$

3.1.3 Non-Max suppression:

The final image should have thin edges. Thus, we must perform non-maximum suppression to thin out the edges.
Algorithm: The algorithm goes through all the pixel on the image. For each pixel in the image, non-maximum suppression examines its gradient magnitude and compares it with the gradient magnitudes of its neighboring pixels along the gradient direction. The objective is to identify pixels that possess the maximum gradient magnitude among their immediate neighbors.

If the gradient magnitude of a pixel is greater than the gradient magnitudes of its neighbors in the gradient direction, it is preserved as a potential edge pixel. Otherwise, if the gradient magnitude is smaller than its neighbors, it is suppressed or discarded, assuming it is not a local maximum in the edge strength.

Let's take an easy example:

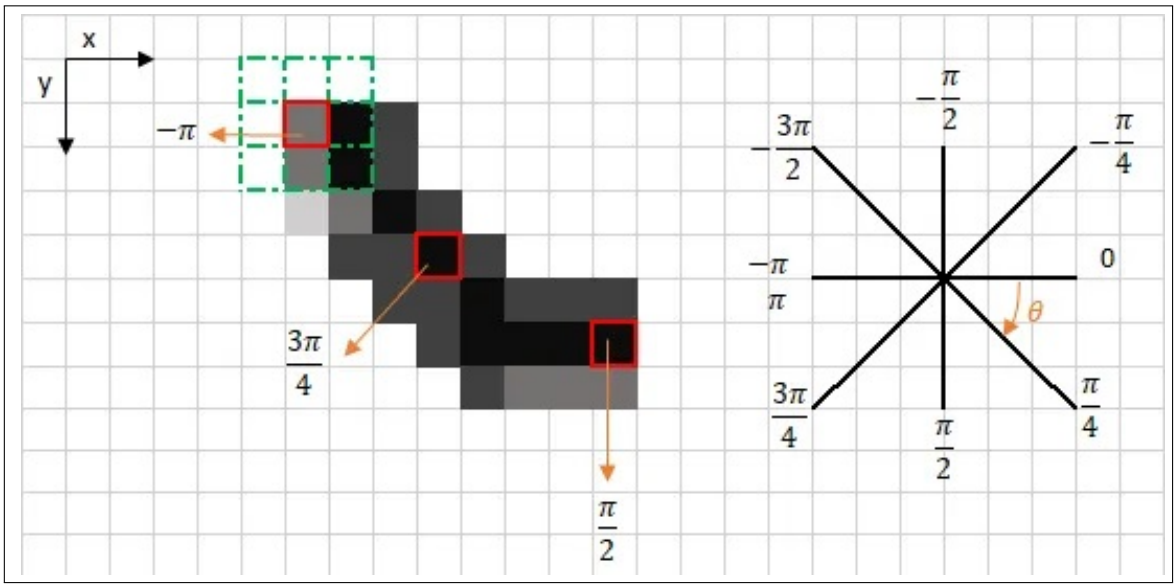


Figure 3.2: Example Fig for Non-Maximum suppression (Source[1])

The upper left corner red box present on the above image, represents an intensity pixel of the Gradient Intensity matrix being processed. The corresponding edge direction is represented by the orange arrow with an angle of -pi radians (+/-180 degrees).

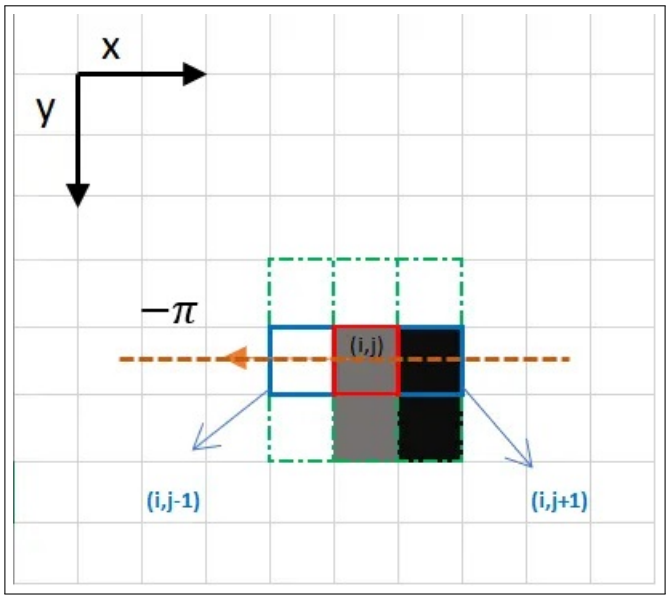


Figure 3.3: Applying Non-maximum suppression (Source[1])

The edge direction is the orange dotted line (horizontal from left to right). The purpose of the algorithm is to check if the pixels on the same direction are more or less intense than the ones being processed. In the example above, the pixel **(i, j)** is being processed, and the pixels on the same direction are highlighted in blue **(i, j-1)** and **(i, j+1)**. If one those two pixels are more intense than the one being processed, then only the more intense one is kept. Pixel **(i, j-1)** seems to be more intense, because it is white (value of 255). Hence, the intensity value of the current pixel **(i, j)** is set to 0. If there are no pixels in the edge direction having more intense values, then the value of the current pixel is kept. Similarly we check for each and every pixels.

By applying non-maximum suppression, the resulting edge map will have only thin, well-defined edges with maximum response along the gradient direction. This process helps to avoid thickening of edges and preserve the most salient edge pixels.

Non-maximum suppression is an essential step in many edge detection algorithms as it helps improve the accuracy and localization of detected edges.

3.1.4 Double Thresholding:

For this step we need two threshold values value, **minVal** and **maxVal**. Any edge with intensity gradient more than **maxVal** are sure to be strong edges and those below **minVal** are non-edges, so discarded. Those who lie between these two thresholds are classified edges or non-edges based on their connectivity. And these edges are considered as weak edges.

3.1.5 Hysteresis:

The weak edge pixels that are connected to strong edges are considered as part of the edge. This step helps to link fragmented edges and suppress noise-induced weak edges.

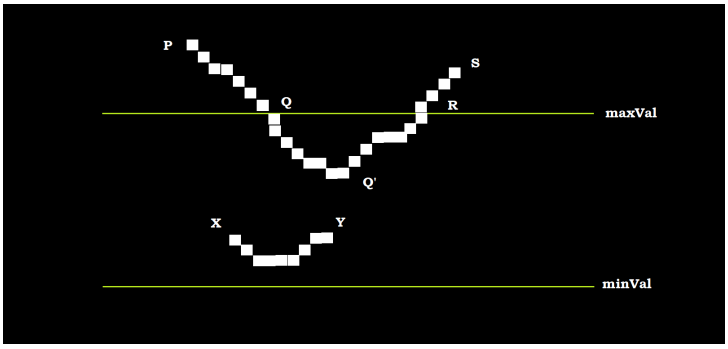


Figure 3.4: Hysteresis visualization

In above image we see that PQ and RS curve are part of strong pixel and QQ'R is part of weak pixel and it is connected with strong pixels so they are now part of strong pixel. But XY curve is not connected to any strong pixels so it is discarded.

By applying these steps, you can obtain a binary image where the edges are highlighted as white pixels and the non-edge regions are represented as black pixels.
Fig no 3.1 show the whole process of canny edge detection:

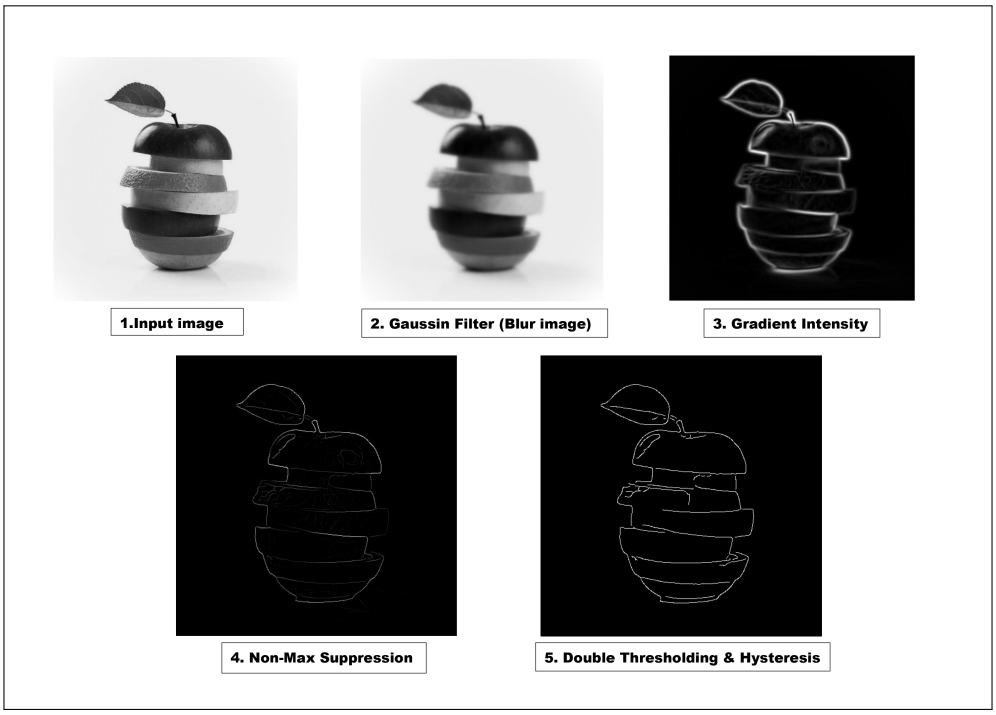


Figure 3.5: The output of different steps of canny algorithm executed on an image

3.2 Probabilistic HoughLine Transform method:

For understanding Probabilistic transform we need to know how the HoughLine transform works.Probabilistic HoughLine Transform method [17] [9] is one type of type of HoughLine transform [8] [13] and Advance then Standard HoughLine Transform.

3.2.1 HoughLine Transform method:

The Hough Transform is a method that is used in image processing to detect any shape, if that shape can be represented in mathematical form. It can detect the shape even if it is broken or distorted a little bit. Let's now understand how Hough transform works for line detection using the HoughLine transform method. As a first step, to apply the Houghline method, an edge detection of the specific image is desirable.
A line can be represented as

$$y = mx + c$$

Where m is a slope and b is the intercept.

Or in parametric form, as

$$r = x\cos\theta + y\sin\theta$$

Where r is the distance from the origin to the closest point on the straight line. (r, θ) corresponding to the half space representation of a line.

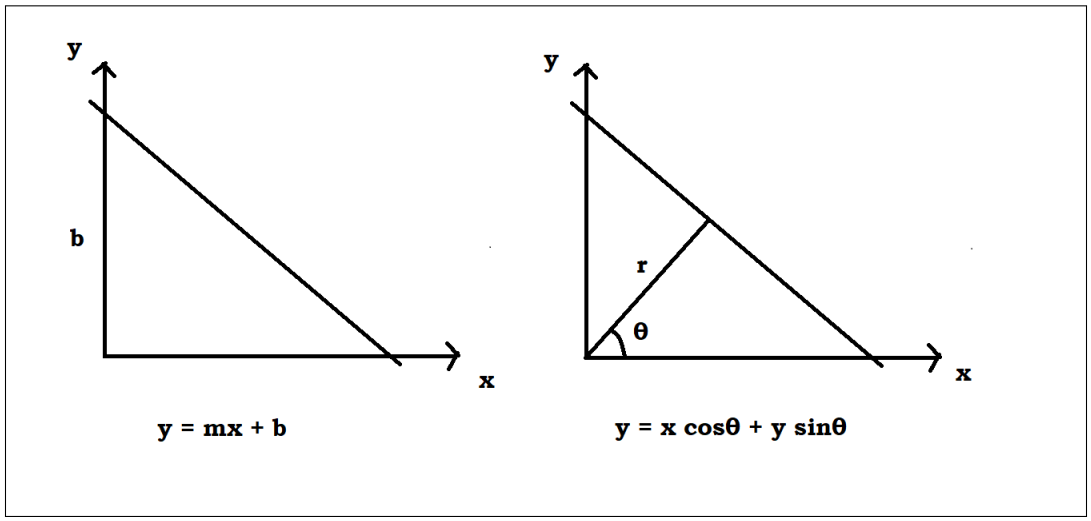
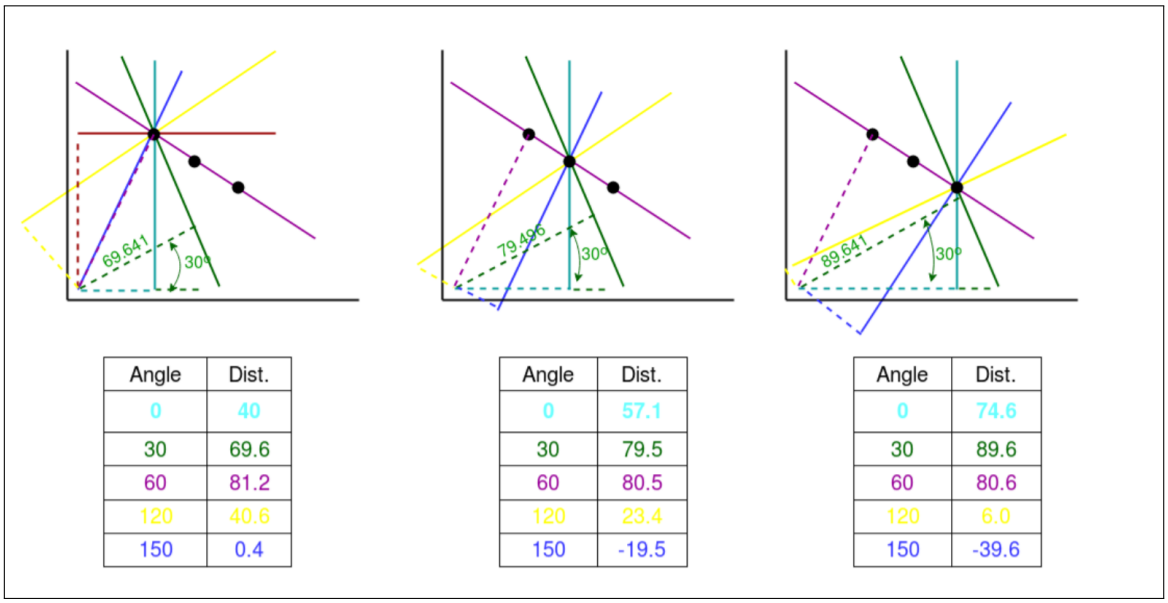


Figure 3.6: Line Representation

To start, we create a 2D array called the accumulator, which will store values for two parameters. Initially, all the values in the accumulator are set to zero. The rows of the array represent the parameter r, while the columns represent the parameter θ (theta). The size of the array depends on the desired accuracy. For example, if we want an accuracy of 1 degree for the angles, we would need 180 columns since the maximum angle for a straight line is 180 degrees. Similarly, for the parameter r, the maximum distance possible is the diagonal length of the image. By considering one-pixel accuracy, the number of rows in the array would match the diagonal length of the image. Let's explore this concept further using the example provided below.
For example,

Let's us consider three point on a line represented as black dots in below figure. Take the first point of the line. Now in the line equation, put the value θ (theta) = 0, 1, 2,. . . ., 180 and check the r you get. Collect all r and θ values for first point.
Do the same for second and third point.
After then check which pairs of r and θ are same among these three point collected values. If a line goes through these data point r and θ must be same .Because a line is represented by a single r and θ.
Here in below figure, All r values for θ=60 are approximately same



Angle	Dist.
0	40
30	69.6
60	81.2
120	40.6
150	0.4

Angle	Dist.
0	57.1
30	79.5
60	80.5
120	23.4
150	-19.5

Angle	Dist.
0	74.6
30	89.6
60	80.6
120	6.0
150	-39.6

Figure 3.7: HoughLine Transform method working (Source[3])

3.2.2 Type of HoughLine Transform Method:

There are two main types of HoughLine Transform methods:

Standard HoughLine Transform: This method works by representing each line in the image as a point in the parameter space called the Hough space. In this space, lines are represented by their parameters, such as the slope and intercept. The method then performs an accumulator voting process to find the most significant lines in the image.The process show in section[3.2] is a Standard HoughLine Transform method.

Probabilistic HoughLine Transform: The Probabilistic HoughLine Transform is an extension of the Standard HoughLine Transform that aims to improve efficiency and speed. Instead of considering all possible combinations of parameters, it randomly selects a subset of points from the edge image and performs the line detection only on these points. This significantly reduces the computational complexity, making it faster and more suitable for real-time applications.

3.2.3 Why we choose Probabilistic HoughLine Transform?

The Probabilistic HoughLine Transform is preferred in scenarios where we want to detect lines in an image quickly and efficiently. It is particularly useful when dealing with images that contain a large number of lines or when real-time performance is crucial.

The main advantage of the Probabilistic HoughLine Transform is its ability to handle partial or fragmented lines in an image. It can detect line segments with gaps and does not require a continuous line structure. This makes it robust to noisy or cluttered images where traditional line detection methods may fail.

By using the Probabilistic HoughLine Transform, we can accurately and efficiently detect lines in an image, which is essential for tasks such as line-based object detection, lane detection in

autonomous driving, and line segment extraction in image analysis.

Overall, the Probabilistic HoughLine Transform method provides a faster and more robust approach to line detection, making it a preferred choice in many computer vision applications.

3.2.4 Discussion on Probabilistic HoughLine Transform function

In my algorithm I use **cv2.HoughLinesP()** function and the elaboration of function **cv2.HoughLinesP()**
cv2.HoughLinesP(dst, rho, theta, threshold, minLineLength, maxLineGap)

dst: Input image should be a binary image, so apply threshold edge detection before finding applying hough transform
rho : The resolution of the parameter r in pixels. We use 1 pixel
theta: The resolution of the parameter θ in radians. We use 1 degree (CV_PI/180)
threshold : The minimum number of intersections to detect a line.
Threshold value means minimum vote it should get for it to be considered as a line. We need to remember here that number of votes depend upon number of points on the line. So it represents the minimum length of line that should be detected
minLineLength: The minimum number of points that can form a line. Lines with less than this number of points are disregarded
maxLineGap: The maximum gap between two points to be considered in the same line

3.3 Removing the extra line i.e. underline as well as storing only the vertical and horizontal line

The algorithm take a collection of line points and remove the extra lines that is not perfectly vertical and horizontal as well as this algorithm remove the underline if there exist any.
Algorithm: Remove underline and store only vertical and horizontal lines:
Input:

1. Image
2. ***lines***(take collection of line determine by HoughLinep() function and store in 2d variable named as "lines")

Output:

1. return ***lines_list*** that contains vertical and horizontal lines only.
2. Save the change image

Step: Description:

1. Populate 2D array ***lines_list***[][]
2. Set ***line_length*** as the length of the lines
3. For $i = 0$ to ***line_length*** $- 1$:
   (a) Set ***x1*** $=$ ***lines***$[i][0]$

(b) Set ***y1*** = ***lines***$[i][1]$

(c) Set ***x2*** = ***lines***$[i][2]$

(d) Set ***y2*** = ***lines***$[i][3]$

(e) If ***y1*** is equal to ***y2***, that means horizontal lines are taken:

i. Set ***flag1*** = True

ii. For i = ***x1*** to ***x2***:

A. Now check if the line is touching any letter or not just by looking for any pixel that is black in the grayscale image and also checking for any pixel that is white in the canny-edged image.

B. If yes, then set ***flag1*** = False

C. And break

iii. If ***flag1*** = True:

A. Draw the line with green color on the original image

B. Append the point [***x1***, ***y1***, ***x2***, ***y2***] to ***lines_list***

(f) If ***x1*** is equal to ***x2***, that means vertical lines are taken:

i. Draw the line with green color on the original image

ii. Append the point [***x1***, ***y1***, ***x2***, ***y2***] to ***lines_list***

4. Return ***lines_list*** and ***image***

3.4 Create a boxes by extending vertical line and horizontal line

This algorithm operates on the image generated by the previous algorithm and a list of lines. It begins by placing a box around the image, with a 15-pixel margin on each side. Then, it extends each vertical line upwards and downwards, and each horizontal line to the right and left, until they intersect with another line that has a green color. This process ensures that the lines reach and connect with other green lines present in the image.

The output of the algorithm includes the modified image, where the lines have been extended and connected, as well as two separate 2D arrays. One array contains the coordinates of the vertical lines, including their start and end points. The other array holds the coordinates of the horizontal lines, again with their respective start and end points. This separation allows for further analysis or utilization of the vertical and horizontal lines independently.

Input:

1. ***Image***
2. ***lines_list*** (contains vertical and horizontal lines only)

Output:

1. Return the saved ***image***
2. Return ***hori_line*** (contains only horizontal lines)

3. Return ***vert_line*** (contains only vertical lines)

Step: Description:

1. Find the dimensions of the ***image***, i.e., height and width of the ***image***, by using ***image.shape()***
2. Set ***y_height*** = ***image.shape***[0]
3. Set ***x_width*** = ***image.shape***[1]
4. Draw a box over the ***image*** using ***cv.rectangle(image, (15,15), (x_width-15,y_height-15), (0,255,0), 3)***
5. Populate 2D array ***hori_line***[][]
6. Populate 2D array ***vert_line***[][]
7. Set l as the length of ***lines_list***
8. For $i = 0$ to $l - 1$:
   (a) Set ***xt1*** = ***x1*** = ***lines_list***[i][0]
   (b) Set ***yt1*** = ***y1*** = ***lines_list***[i][1]
   (c) Set ***xt2*** = ***x2*** = ***lines_list***[i][2]
   (d) Set ***yt2*** = ***y2*** = ***lines_list***[i][3]
   (e) If ***y1*** is equal to ***y2*** (for horizontal lines):
      i. While ***x1*** is greater than 15 (expanding line to the left side):
         A. If any pixel is green:
         B. Break
         C. Decrement the value of ***x1*** by 1
      ii. Draw the extended left side of the line
      iii. While ***x2*** is less than ***x_width***-15 (expanding line to the right side):
         A. If any pixel is green:
         B. Break
         C. Increment the value of ***x2*** by 1
      iv. Draw the extended right side of the line
      v. Append the extended horizontal line to ***hori_line***
   (f) If ***x1*** is equal to ***x2*** (for vertical lines):
      i. While ***y1*** is greater than 15 (expanding line upward):

A. If any pixel is green:

B. Break

C. Decrement the value of ***y1*** by 1

ii. Draw the extended upward part of the line

iii. While ***y2*** is less than ***y_height***-15 (expanding line downward):

A. If any pixel is green:

B. Break

C. Increment the value of ***y2*** by 1

iv. Draw the extended downward part of the line

v. Append the extended vertical line to ***vert_line***

9. Return ***image***, ***vert_line***, ***hori_line***

3.5 Remove the overlapped lines and store intersection points

The given algorithm processes two sets of lines: horizontal lines and vertical lines. Its purpose is to eliminate any overlapping lines and determine the intersection points between the extended vertical and horizontal lines. The algorithm accomplishes this by sorting the lines, creating arrays to store the intersection points, and iterative selecting non-overlapping lines for the final outputs. The steps involve sorting the lines based on their coordinates, adding boundary lines, identifying unique lines, appending relevant points to the arrays, and ultimately returning the non-overlapping horizontal lines, the non-overlapping vertical lines, and the intersection points. The algorithm essentially organizes the lines, removes overlaps, and identifies the points where the lines intersect when extended.

Input:

1. ***hori_line*** (contains horizontal lines only)
2. ***vert_line*** (contains vertical lines only)

Output:

1. Return ***hori_final*** (contains horizontal lines without any overlapping)
2. Return ***vert_final*** (contains vertical lines without any overlapping)
3. Return ***inter_point*** (contains all intersection points created by extending vertical lines upward and horizontal lines to the right side)

Step: Description:

1. Populate 2D array ***inter_point***[][]
2. Sort ***hori_line*** with respect to ***y1*** and for every equal ***y1***, then sort with respect to ***x1***
3. Sort ***vert_line*** with respect to ***x1*** and for every equal ***x1***, then sort with respect to ***y1***
4. Insert a line $[15, 15, x\_width - 15, 15]$ at the front of ***hori_line***
5. Insert a line $[x\_width - 15, 15, x\_width - 15, y\_height - 15]$ at the end of ***vert_line***
6. Populate 2D array ***hori_final***[][]
7. Append ***hori_line[0]*** to ***hori_final***
8. Append [***hori_line[0][2]***, ***hori_line[0][3]***] to ***inter_point***
9. Set ***hl*** as the length of ***hori_line***
10. For $i = 1$ to ***hl***-1:
    (a) If the previous line in ***hori_line*** is equal to the current line:
        i. Continue
    (b) Else:
        i. Append the current line to ***hori_final***
        ii. Append the right point (i.e., (***x2***, ***y2***)) of the line to ***inter_point***
11. Populate 2D array ***vert_final***[][]
12. Append ***vert_line[0]*** to ***vert_final***
13. Append [***vert_line[0][0]***, ***vert_line[0][1]***] to ***inter_point***
14. Set ***vl*** as the length of ***vert_line***
15. For $i = 1$ to ***vl***-1:
    (a) If the previous line in ***vert_line*** is equal to the current line:
        i. Continue
    (b) Else:
        i. Append the current line to ***vert_final***
        ii. If i is not equal to ***vl***-1:
            A. Append the upward point (i.e., (***x1***, ***y1***)) of the line to ***inter_point***
16. Insert the point $[15, y\_height - 15, x\_width - 15, y\_height - 15]$ at the end of ***hori_final***
17. Return ***hori_final***, ***vert_final***, ***inter_point***

3.6 Finding the rectangle using Vertical and Horizontal lines

This algorithm take a newspaper images, horizontal lines points, vertical lines points and intersection point on these lines and generates rectangles boxes over image or store these point in rect variable.

Algorithm: Find the intersection point on a particular horizontal line, function name: find_inter_pt_on_h_line()

This function take horizontal line and intersection points and return the intersection point on the horizontal line as we now in horizontal line hy1 = hy2 so we return only hx1 as res.

Input:

1. ***Horizontal line point*** $(\boldsymbol{hx1}, \boldsymbol{hy1}, \boldsymbol{hx2}, \boldsymbol{hy2})$
2. ***inter_point*** (contains intersection points on the horizontal line made by the vertical line when extended upward and intersection points on the vertical line made by the horizontal line extended to the right side)

Output:

1. Return ***res***

Step: Description:

1. Populate array ***temp***[]
2. Set ***il*** as the length of ***inter_point***
3. For $i = 0$ to ***il***-1:
   (a) If ***inter_point[i][1]*** is equal to ***hy1***:
      i. Append ***inter_point[i][0]*** to ***temp***
4. Sort ***temp*** in increasing order
5. Set ***l*** as the length of ***temp***
6. Set ***res*** as -1
7. For $i = 0$ to ***l***-1:
   (a) If ***temp[i]*** is greater than ***hx1***:
      i. Set ***res*** as ***temp[i]***
      ii. break
8. Return ***res***

Algorithm: Find the intersection point on a particular vertical line, function name: find_inter_pt_on_v_line()

This function is designed to process a vertical line and a collection of intersection point. Given that the vertical line has identical x-coordinates for its starting and ending points (vx1 = vx2), we can identify it as a vertical line. The primary objective of the function is to extract the y-coordinate of the starting point (vy1) and return it as the resulting value. By focusing solely on the vertical component, we disregard the variation in the x-coordinate, as it remains constant along the vertical line.

Input:

1. ***Vertical line point*** $(\boldsymbol{vx1}, \boldsymbol{vy1}, \boldsymbol{vx2}, \boldsymbol{vy2})$
2. ***inter_point*** (contains intersection points on the horizontal line made by the vertical line when extended upward and intersection points on the vertical line made by the horizontal line extended to the right side)

Output:

1. Return ***res***

Step: Description:

1. Populate array ***temp***[]
2. Set ***il*** as the length of ***inter_point***
3. For $i = 0$ to ***il***-1:
   (a) If ***inter_point[i][1]*** is equal to ***vx1***:
      i. Append ***inter_point[i][1]*** to ***temp***
4. Sort ***temp*** in increasing order
5. Set ***l*** as the length of ***temp***
6. Set ***res*** as -1
7. For $i = 0$ to ***l***-1:
   (a) If ***temp[i]*** is greater than ***vy1***:
      i. Set ***res*** as ***temp[i]***
      ii. Return ***res***
8. Return ***res***

Algorithm: Rectangle detection

The algorithm takes horizontal lines, vertical lines, and intersection points as input. It creates an empty array to store rectangle points. It iterates through the horizontal lines and finds the intersection point on each line. Then it searches for a matching vertical line based on the intersection point. If found, it creates a rectangle using the line points and appends it to the array. The algorithm updates the line points based on the intersection and repeats the process. Finally, it returns the array of rectangle points.

Input:

1. ***hori_final*** (Contains start and end points of all horizontal lines)
2. ***vert_final*** (Contains start and end points of all vertical lines)
3. ***inter_point*** (contains intersection points on the horizontal line made by the vertical line when extended upward and intersection points on the vertical line made by the horizontal line extended to the right side)

Output:

1. ***rect*** (contains the rectangle points)

Step: Description:

1. Populate 2D array ***rect***[]
2. Set ***hl*** as the length of ***hori_final***
3. For $i = 0$ to ***hl***-2:
   (a) While ***hori_final[i][0]*** $\neq$ ***hori_final[i][2]***:
      i. Set ***hx1*** as ***hori_final[i][0]***
      ii. Set ***hy1*** as ***hori_final[i][1]***
      iii. Set ***hx2*** as ***hori_final[i][2]***
      iv. Set ***hy2*** as ***hori_final[i][3]***
      v. Set ***ix*** as ***find_inter_pt_on_h_line(hx1, hy1, hx2, hy2, inter_point)***
      vi. Set ***iy*** as ***hy1***
      vii. Set ***index*** as -1
      viii. Set ***vl*** as the length of ***vert_final***
      ix. For $j = 0$ to ***vl***-1:
         A. If $|$***vert_final[j][0]*** $-$ ***ix***$| < 8$ and $|$***vert_final[j][1]*** $-$ ***iy***$| < 8$:
         B. Set ***index*** as j

C. Break

x. Set ***vx1*** as ***vert_final[index][0]***

xi. Set ***vy1*** as ***vert_final[index][1]***

xii. Set ***vx2*** as ***vert_final[index][2]***

xiii. Set ***vy2*** as ***vert_final[index][3]***

xiv. Set ***jy*** as ***find_inter_pt_on_v_line(vx1, vy1, vx2, vy2, inter_point)***

xv. Set ***jx*** as ***vx1***

xvi. If ***jy*** is equal to -1:

A. Append the point [***hx1***, ***hy1***, ***vx2***, ***vy2***] to ***rect***

B. Call the ***cv2.rectangle()*** to draw this rectangle on the image

C. Set ***hori_final[1][0]*** as ***ix*** #

D. Set ***hori_final[i][1]*** as ***iy*** #

xvii. Else:

A. Append the point [***hx1***, ***hy1***, ***jx***, ***jy***] to ***rect***

B. Call the ***cv2.rectangle()*** to draw this rectangle on the image

C. Set ***hori_final[i][0]*** as ***ix*** #

D. Set ***hori_final[i][1]*** as ***iy*** #

E. Set ***vert_final[index][0]*** as ***jx*** #

F. Set ***vert_final[index][1]*** as ***jy*** #

4. Return ***rect***

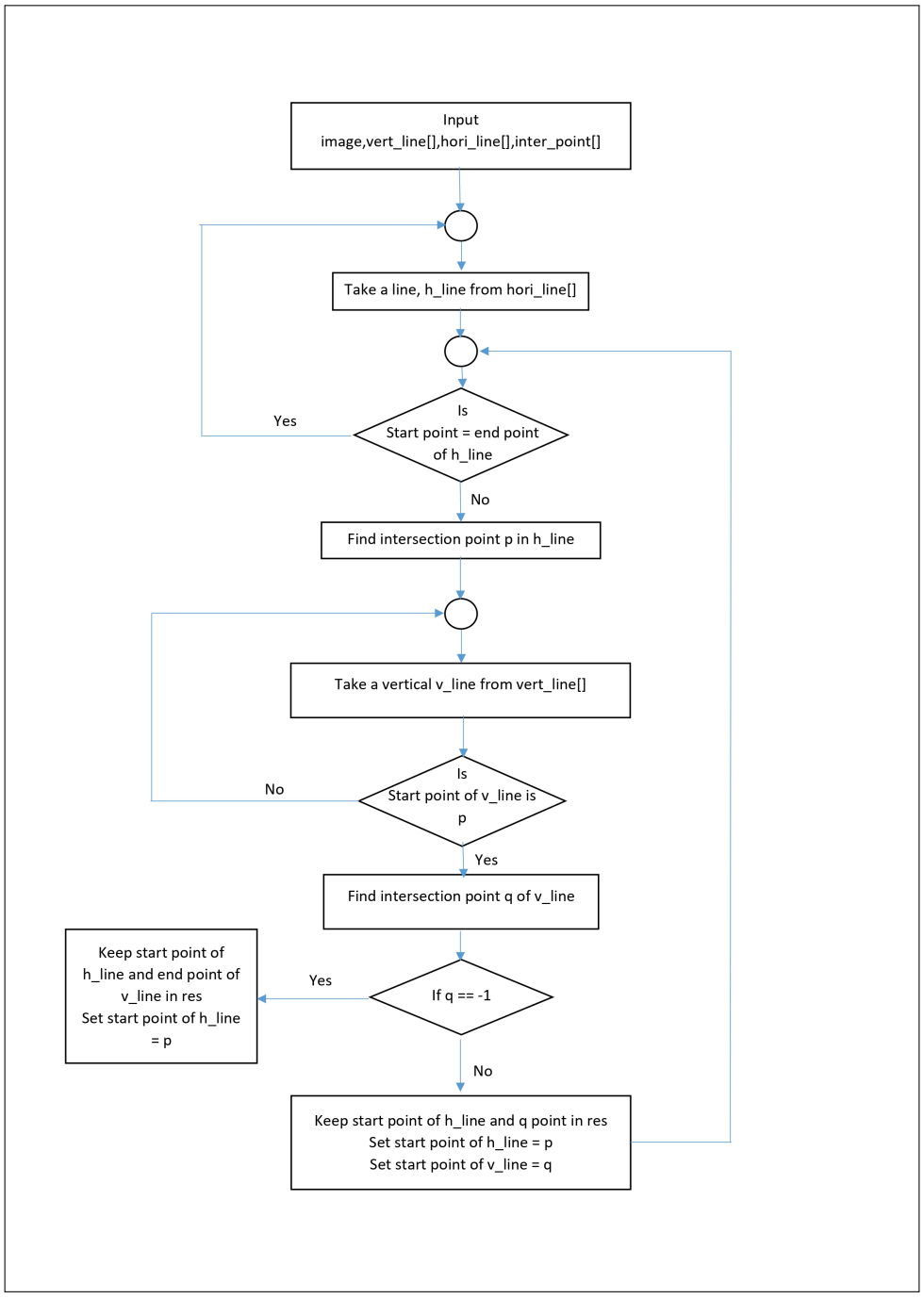


Figure 3.8: Rectangle Detection Algorithm Flow Chart

Chapter 4

Experimental Result and Analysis

In this chapter, our focus revolves around the experiments conducted using the Layout Parser library experiments and the development of the Web-API and there experiments.

Initially, the experiments involved using the Layout Parser library[5] with various pre-trained models such as PubLayNet and PrimaLayout. While these models demonstrated promising results in paragraph segmentation of documents, it encountered difficulties in accurately handling congested data and articles with multiple paragraphs that is new type of newspapers. This prompted the need for a customized approach to improve segmentation accuracy.

To address this challenge, a custom algorithm was developed and for the algorithm see Chapter - 3, which formed the foundation of our Web-API. The Web-API was implemented using Flask, a popular Python web framework, allowing for easy integration and deployment. Through the Web-API, users could upload newspaper images and receive segmented news items as output.

4.1 Experiment & Result Layoutparser

This section explains the installation and usage of the layout parser models. It also highlights the limitations of the parser when applied to newspapers. The parser faces challenges in accurately processing newspapers due to factors such as narrow spacing between paragraphs, connected text, and variations in text sizes. These factors contribute to the parser's failure in producing accurate results for newspaper layouts.

4.1.1 Experimental Setup:

To conduct the experiments, the Layout Parser library was installed and utilized within a Google Colab environment. Here is a step-by-step breakdown of the experimental setup:

1. **Installation of Layout Parser:** The Layout Parser library was installed in the Google Colab environment by running the necessary installation commands. This included installing the required dependencies and ensuring compatibility with the Python version used.

   Code:

   ```
   !pip install layoutparser
   ```

```
!pip install "layoutparser[effdet]"
!pip install layoutparser torchvision && pip install

    "git+https://github.com/facebookresearch/detectron2.git@v0.5
     #egg=detectron2"
!pip install "layoutparser[paddledetection]"
```

2. **Import needed library:** Import the layeout parser and and needed library.
   Code:

```
import layoutparser as lp
import cv2
```

3. **Dataset and Model Selection:** Two different datasets were utilized for the experiments: PubLayNet, PrimaLayout. For each dataset, a specific pre-trained model was employed: "faster_rcnn_R_50_FPN_3x" for PubLayNet and "mask_rcnn_R_50_FPN_3x" for PrimaLayout.

   Label Map of corresponding Datasets

```
PubLayNet    {0: "Text", 1: "Title", 2: "List", 3:"Table", 4:"Figure"}
PrimaLayout  {1:"TextRegion", 2:"ImageRegion", 3:"TableRegion",
              4:"MathsRegion",5:"SeparatorRegion",6:"OtherRegion"}
```

4. **Load the data model:** Load the deep layout model from the layoutparser API and store it in model variable. And Detect the layout of the input image and store it in layout.

   Code for loading API:

```
model = lp.models.Detectron2LayoutModel(
          'lp://PubLayNet/faster_rcnn_R_50_FPN_3x/config',
          extra_config=["MODEL.ROI_HEADS.SCORE_THRESH_TEST", 0.2],
          label_map={0:"Text",1:"Title",2:"List",3:"Table",4:"Figure"}
            )
```

   Code for Detect layout in image:

```
layout = model.detect(image)
```

   Similarly, we can do for PrimaLayout.

4.1.2 Result and Analysis

In this section, we see the performance of the layout parser model on both a simple document and a newspaper dataset.

Using PubLayNet:

Figure 4.1: Result of simple document and Newspaper using Layout parser

4.2 Experiment & Result of Web-API

This section going to represent about how I developed the Web-API using flask and how flask connected to python programming. And how it works and compare between the ground truth and result.

4.2.1 Experimental Setup:

1. Install Flask by running the following command in your command prompt or terminal:

```
pip install flask
```

2. Create a folder named "Segment-News-paper-project" (or any name you prefer) to serve as the root folder for your project.All folder that must be created if we are going to create the Web-API using flask.

3. Within the "Segment-News-paper-project" folder, create the following sub-folders and files:

   - **"app.py":** This file will serve as the main application file that handles route requests and renders the appropriate HTML pages.
   - **"segment.py":** This file will contain the algorithmic logic for the segmentation process.
   - **"labelme.py":** This file will be responsible for creating a JSON file and image that can be used with the Labelme annotation tool. It should provide an editable segmentation output.
   - **"static" folder:** Create this folder to store static files such as CSS, JavaScript, and media files.
   - **"templates" folder:** Create this folder to store HTML files that will be rendered by Flask templates.

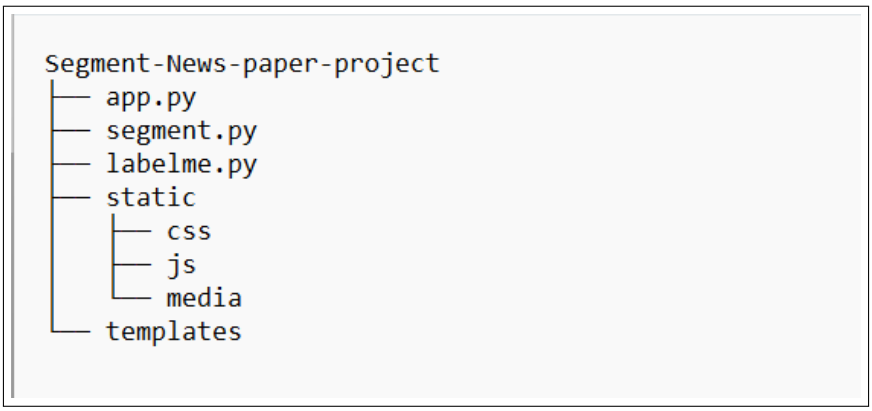


Figure 4.2: Show the folder structure from root

4.2.2 User interface functionality

Web API interface look like:

Figure 4.3: User Interface

Functionality of each button:

- **Process Button:** This button triggers the execution of the processing steps defined in the segment.py file. It initiates the algorithm to perform the segmentation on the uploaded image and generate the segmented output.
- **Segment Button:** Clicking on the Segment button redirects the user to a new page where they can select the segmented items of the newspaper. This allows the user to interactively choose the specific regions or sections they want to focus on.
- **Download Button:** By clicking on the Download button, the user can download the segmented image. This button enables the user to save the image with the rectangle boxes highlighting the segmented areas. The image can be used for further analysis or reference.
- **Lbm-image and Lbm-json Buttons:** These buttons allow the user to download the segmented image and its corresponding JSON file. The JSON file is compatible with the Labelme annotation tool. By using these buttons, the user can open the image and JSON file in Labelme to perform additional functionalities such as editing, annotating, or extracting information from the segmented regions.
- **Reset Button:** The Reset button resets the page, clearing any uploaded images or previously performed segmentation. It provides a convenient option for the user to start fresh or upload a new image for segmentation.

This image show how it look after click on segment Button:

Figure 4.4: After click on Segment button

After downloading the Lbm-image and Lbm-json we can open the image and edit,delete and create the rectangle. Below figure show that how it look like after open it in Labelme.

Figure 4.5: Open the image in Lableme

4.2.3 Result and Analysis

This field show code and the result comparison of the Lableme tool ground truth and the newspaper segmentation using Web-API for The Hindu Newspaper and we also use the same code for The Statesman and The Telegraph. It calculates the percentage of matching of the segmented item with the ground truth.

To use the code, make sure to place the folder containing the ground truth images and corresponding JSON files created by Labelme, as well as the folder containing the images and JSON files generated by the Web-API, in the same parent folder.And make sure that the images and their corresponding JSON files in the ground truth folder and the Web-API folder have the same names. It matches the files based on their names and compares the segmented items accordingly.

The code iterates through each pair of corresponding files from the two folders and compares the segmented items in the JSON files. It counts the number of matching items and calculates the percentage of matching segmented items for each file. Finally, it computes the average matching percentage across all files.

By running the code, you will obtain the matching percentage for each file, indicating the similarity between the ground truth and the Web-API segmentation. Additionally, the overall average matching percentage will provide an assessment of the agreement between the two methods across the entire dataset.

Code:

```
import os
import json
# import matplotlib.pyplot as plt

labelme_folder = 'Labelme_the_hindu'
webapi_folder = 'MyWeb_the_hindu'

# Get the current working directory
current_directory = os.getcwd()

# Get the absolute paths of the folders
labelme_folder_path = os.path.join(current_directory, labelme_folder)
webapi_folder_path = os.path.join(current_directory, webapi_folder)

# Get the list of JSON files in both folders
labelme_files = [file for file in os.listdir
                 (labelme_folder) if file.endswith('.json')]
webapi_files = [file for file in os.listdir
                 (webapi_folder) if file.endswith('.json')]

total_file = 0
sum_persentage = 0
#Compare each file in labelme_folder with its corresponding
#file in webapi_folder
for labelme_file in labelme_files:
    labelme_filepath = os.path.join(labelme_folder, labelme_file)
    webapi_file = labelme_file
    webapi_filepath = os.path.join(webapi_folder, webapi_file)

    # Read the JSON data from both files
    with open(labelme_filepath, 'r') as file:
        labelme_data = json.load(file)

    with open(webapi_filepath, 'r') as file:
        webapi_data = json.load(file)


    # Extract points from "labelme_data.json"
    labelme_points = []
    for shape in labelme_data['shapes']:
        points = shape['points']
        labelme_points.append(points)
```

```
    # Extract points from "webapi_data.json"
    webapi_points = []
    for shape in webapi_data['shapes']:
        points = shape['points']
        webapi_points.append(points)

    count = 0
    threshold = 40
    for point in labelme_points:
        x1=point[0][0]
        y1=point[0][1]
        x2=point[1][0]
        y2=point[1][1]
        for pt in webapi_points:
            a1=pt[0][0]
            b1=pt[0][1]
            a2=pt[1][0]
            b2=pt[1][1]
            if abs(a1-x1)<threshold and abs(b1-y1)<threshold
            and abs(a2-x2)<threshold and abs(b2-y2)<threshold:
                count = count + 1
                break

    persentage = (count / len(labelme_points)) * 100
    sum_persentage = sum_persentage + persentage
    total_file= total_file+1

avg_per = sum_persentage/total_file
print("persentage: ", avg_per)
```

Output:

```
E:\000_6th_sem_final\chapter5>python result.py
persentage:  80.22280183612364
```

Figure 4.6: Output of Code for Result

The accuracy result of the The Hindu Newspaper is 80.22% as we see in output and this result based on the 100 selected images of Hindu Newspaper that have the lines with in the articles. Similarly the accuracy percentages of the segmentation results obtained from the Web-API compared to the ground truth for two other newspapers, The Statesman and The Telegraph. The accuracy percentages for The Statesman and The Telegraph were found to be 66.32% and 59.15% respectively.

To visualize these results, we created a bar chart. On the x-axis, we labeled the newspapers (Labelme, The Hindu, The Statesman, and The Telegraph), and on the y-axis, we represented the

accuracy percentage. Each bar in the chart represents the accuracy percentage for the corresponding newspaper. The chart allows us to compare the accuracy of the Web-API segmentation results with the ground truth for different newspapers.

Here is Final Bar-Chart figure:

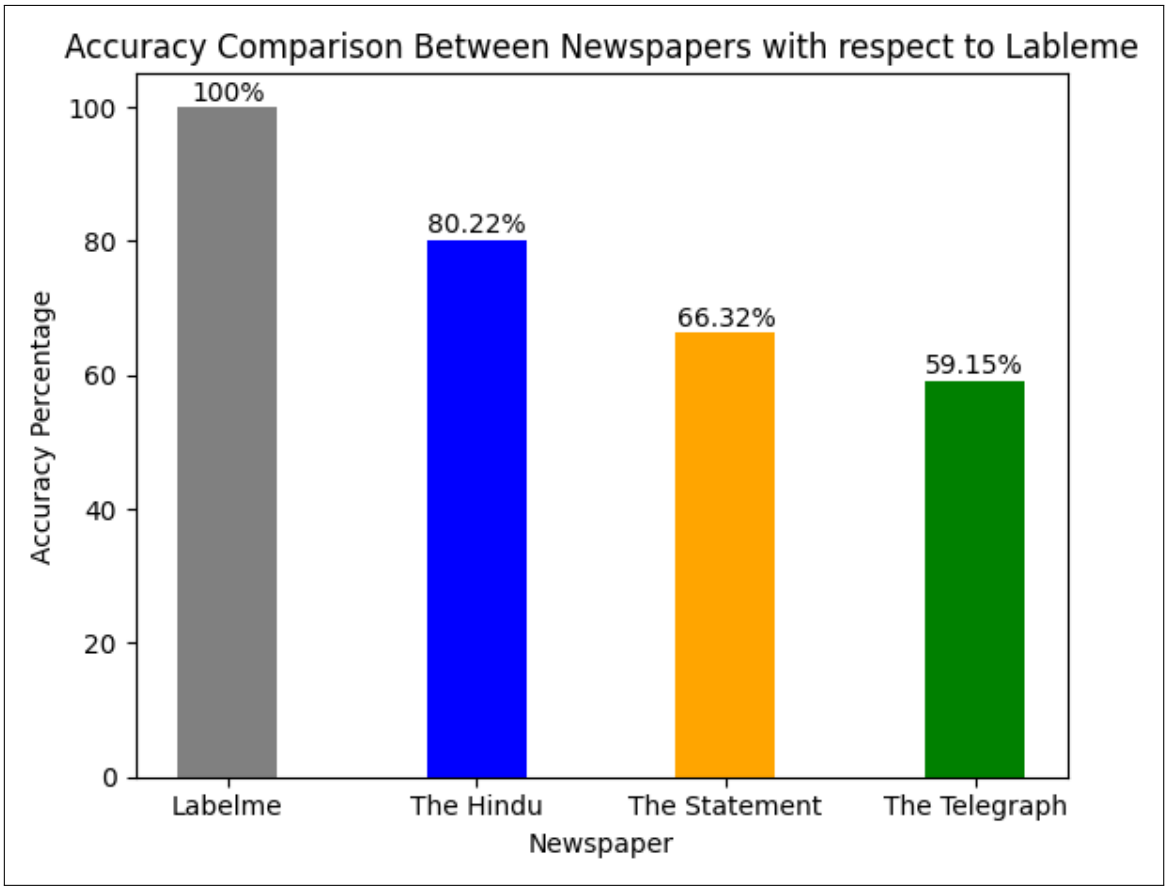


Figure 4.7: Bar-Char result of three different newspaper comparing with the Labelme ground truthm

Chapter 5

Conclusion

This paper discusses an algorithm that performs segmentation of news items from newspaper images that contain line in between there articles and focused on the development of a web-based API for news item segmentation from newspaper images. The experiment involved creating a custom algorithm and integrating it into a Flask-based web application. The API allows users to upload newspaper images and obtain segmented results, facilitating the extraction of individual news items.

Genuine newspaper such as The Hindu, The Statesman and The Telegraph images were used to test the algorithm. The algorithm scored 80.22% in The Hindu newspaper and 66.32% and 59.15% in The Statesman and The Telegraph respectively.

Through experimentation and analysis, it was observed that the API performed well on simple documents and newspapers with clear separation line between news items. However, challenges arose when dealing with complex layouts or newspapers with densely packed content and the newspaper that not contain any lines then algorithm fails. To address this, further improvements and enhancements can be made to the segmentation algorithm to handle such cases effectively.

The developed web API provides a user-friendly interface with buttons for processing, segmenting, and downloading the results. Additionally, the option to download images and JSON files compatible with the labelme annotation tool enables further annotation and analysis of the segmented regions.

The outcome of this project has the potential to significantly streamline the extraction of news items from newspaper images, enabling researchers, journalists, and data analysts to access and analyze valuable information more efficiently. The subsequent sections will delve into the dataset preparation, the proposed segmentation method, the experimental results, and the conclusions drawn from this endeavor.

References

[1] Canny edge detection step by step in python. https://towardsdatascience.com/canny-edge-detection-step-by-step-in-python-computer-vision-b49c3a2d8123. Accessed: 2023-05-29.

[2] The hindu enewspaepr. https://epaper.thehindu.com/login?utm_source=google&utm_medium=cpc&utm_campaign=TH_ePaper_BrandSearchExact&tpcc=THEPGS&gad=1&gclid=Cj0KCQjwmtGjBhDhARIsAEqfDEcqj717JIhzTQSJ4YupmF1odCE_alboAf0nBVe6IG6g-pMJKaXC2AwaAl9JEALw_wcB. Accessed: 2023-05-29.

[3] Houghline detection method. https://indiantechwarrior.com/houghline-method-line-detection-with-opencv/. Accessed: 2023-05-29.

[4] Labelme annotation tool. https://github.com/wkentaro/labelme. Accessed: 2023-05-29.

[5] Layout parser models zoo. https://layout-parser.readthedocs.io/en/latest/notes/modelzoo.html. Accessed: 2023-05-29.

[6] The statesman. https://epaper.thestatesman.com/. Accessed: 2023-05-29.

[7] The telegraph enewspaepr. https://www.telegraphindia.com/. Accessed: 2023-05-29.

[8] N. Aggarwal and W. C. Karl. Line detection in images through regularized hough transform. *IEEE transactions on image processing*, 15(3):582–591, 2006.

[9] A. Bandara and P. Giragama. A retinal image enhancement technique for blood vessel segmentation algorithm. In *2017 IEEE international conference on industrial and information systems (ICIIS)*, pages 1–5. IEEE, 2017.

[10] L. Ding and A. Goshtasby. On the canny edge detector. *Pattern recognition*, 34(3):721–725, 2001.

[11] B. Gatos and N. Papamarkos. Applying fast segmentation techniques at a binary image represented by a set of non-overlapping blocks. In *Proceedings of Sixth International Conference on Document Analysis and Recognition*, pages 1147–1151. IEEE, 2001.

[12] K. Hadjar, O. Hitz, and R. Ingold. Newspaper page decomposition using a split and merge approach. In *Proceedings of sixth international conference on document analysis and recognition*, pages 1186–1189. IEEE, 2001.

[13] J. Illingworth and J. Kittler. A survey of the hough transform. *Computer vision, graphics, and image processing*, 44(1):87–116, 1988.

[14] J. Liu, Y. Y. Tang, and C. Y. Suen. Chinese document layout analysis based on adaptive split-and-merge and qualitative spatial reasoning. *Pattern Recognition*, 30(8):1265–1278, 1997.

[15] P. E. Mitchell and H. Yan. Document page segmentation based on pattern spread analysis. *Optical Engineering*, 39(3):724–734, 2000.

[16] P. E. Mitchell and H. Yan. Newspaper layout analysis incorporating connected component separation. *Image and Vision Computing*, 22(4):307–317, 2004.

[17] Z. Mu and Z. Li. A novel shi-tomasi corner detection algorithm based on progressive probabilistic hough transform. In *2018 Chinese Automation Congress (CAC)*, pages 2918–2922. IEEE, 2018.

[18] T. Palfray, D. Hebert, S. Nicolas, P. Tranouez, and T. Paquet. Logical segmentation for article extraction in digitized old newspapers. In *Proceedings of the 2012 ACM symposium on Document engineering*, pages 129–132, 2012.

[19] W. Rong, Z. Li, W. Zhang, and L. Sun. An improved canny edge detection algorithm. In *2014 IEEE international conference on mechatronics and automation*, pages 577–582. IEEE, 2014.

[20] R. Song, Z. Zhang, and H. Liu. Edge connection based canny edge detection algorithm. *Pattern Recognition and Image Analysis*, 27:740–747, 2017.

[21] J. Xi, J. Hu, and L. Wu. Page segmentation of chinese newspapers. *Pattern recognition*, 35(12):2695–2704, 2002.