CHAPTER 6

Table OCR Extraction Framework

In this chapter, This chapter contains five (5) sections explaining the experiment: Section 5.1 Table OCR Extraction Framework, Section 5.2 Table OCR Extraction System Block Diagram, Section 5.3 Table OCR Extraction Data Flow Diagram, Section 5.4 Flowchart, Section 5.5 Execute Section, 5.6 Evaluation and Discussion and Section, 5.7 Highlight of Contribution and Conclusion.

5.1 Table OCR Extraction Framework

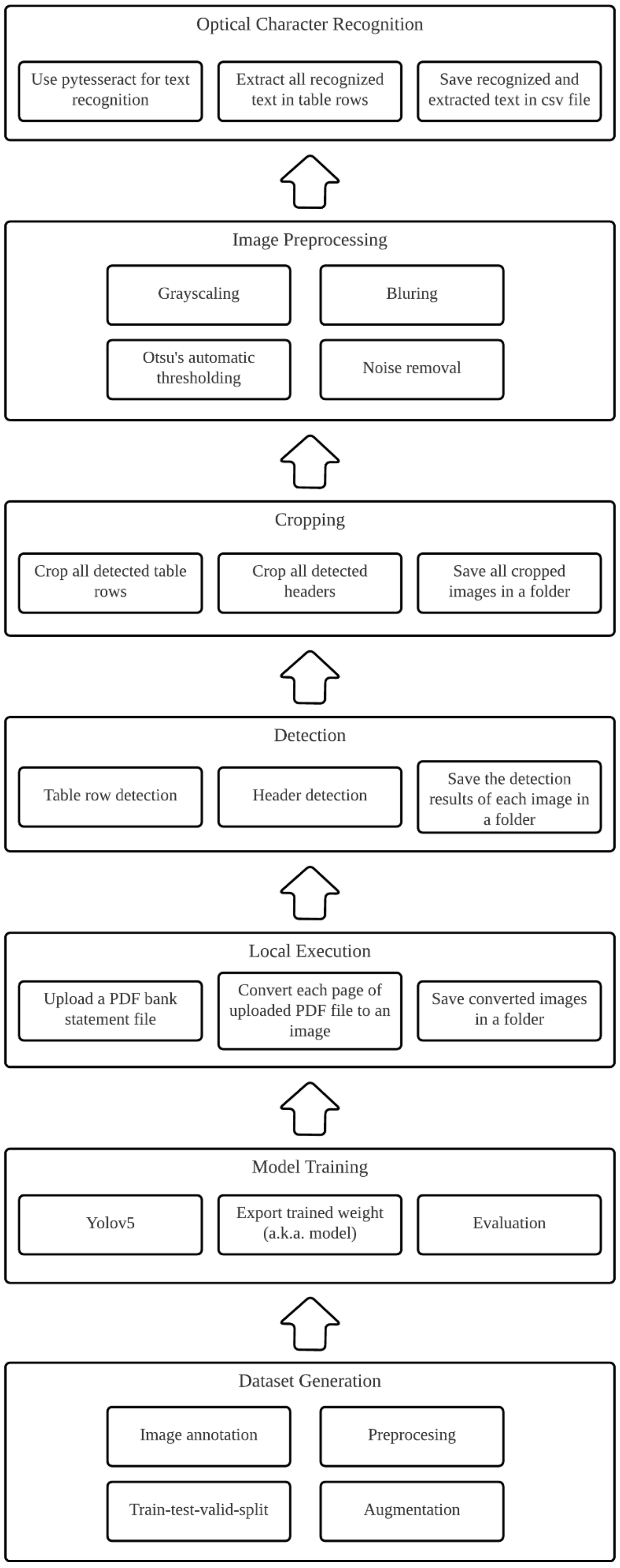

Optical Character Recognition
Use pytesseract for text recognition
Extract all recognized text in table rows
Save recognized and extracted text in csv file
Image Preprocessing
Grayscaling
Bluring
Otsu's automatic thresholding
Noise removal
Cropping
Crop all detected table rows
Crop all detected headers
Save all cropped images in a folder
Detection
Table row detection
Header detection
Save the detection results of each image in a folder
Local Execution
Upload a PDF bank statement file
Convert each page of uploaded PDF file to an image
Save converted images in a folder
Model Training
Yolov5
Export trained weight (a.k.a. model)
Evaluation
Dataset Generation
Image annotation
Preprocesing
Train-test-valid-split
Augmentation

Figure 5.1: Table OCR Extraction Layered Framework

Figure 5.1 presents the design of the Table OCR Extraction layered framework that contains 7 layers. The 7 layers include Dataset Generation, Model Training, Local Execution, Detection, Cropping, Image Preprocessing and Optical Character Recognition. Each layer of this framework will handle different tasks that ensure the final results are achieved.

5.1.1 Dataset Generation

First of all, the design flow of the framework begins by generating a dataset using the Roboflow website. I will convert all the PDF Bank Statement files I have to images first and then upload them to Roboflow. After done uploading, the uploader is required to annotate each uploaded image manually where this project is having "Header", "HeaderRow" and "Row" annotations and then followed by the process of train-test-valid images split with 70% training, 20% validation and 10% testing. Subsequently, three (3) preprocessing actions are applied which are auto orient, resize and grayscale. Auto-orient preprocessing makes sure your images are suitable for viewing (i.e. top-left orientation). Resize preprocessing helps the user to resize all his uploaded images where the size is set to the width of 416 and height of 416 in order to ensure that all images are having the same size. After that, Grayscale preprocessing is to convert the images to black and white or grey monochrome which carries only intensity information. In the augmentation step, two augmentation methods are applied which are grayscale and brightness of the bounding box. Grayscale augmentation randomly makes an image converted to a single-channel image where 25 % of variability is applied for uploaded images in this project. Meanwhile, the variability of the brightness of the bounding box is set to -40% and +40% to help the trained model be more resilient to lighting and camera setting changes.

5.1.2 Model Training

Secondly, YOLOv5 deep learning is applied in this project where the training of the YOLOv5 deep learning model is implemented in the Google Collaboratory. The first step is to clone the YOLOv5 repository from the GitHub page and then install all the required libraries and dependencies and then followed by the personal API key of Roboflow must be written and executed in order to retrieve the generated dataset on the Roboflow website previously. After that, we can start training our YOLOv5 model as our dataset is ready. In order to train the deep learning model, there are a few arguments that need to be passed in before instantiating the model including img, batch, epochs, data, weights, cfg and cache. The training losses and model performance metrics will be saved and loaded on Tensorboard upon done training. After the deep learning model training model is done, the trained weight (a.k.a model) is exported for local use.

5.1.3 Local Execution

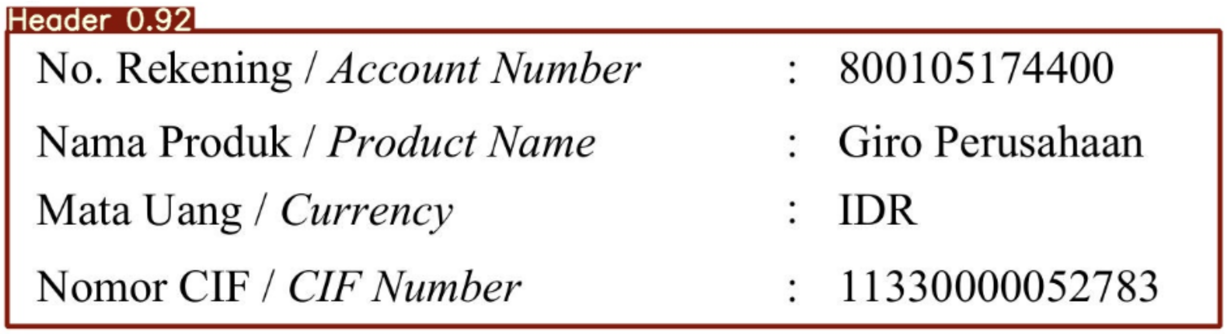
In this phase, a Streamlit web app is developed to ease users to upload their PDF bank statement files and the trained weight (a.k.a model) has been exported from Google Colab and put into this system. Any PDF bank statement files from local storage can be uploaded to perform Region of Interest (ROI) Detection, and Optical Character Recognition for text extraction purposes. After uploading, the system will convert every page of the uploaded PDF file into an image in JPEG format using a python library called pdf2image where all the converted images will be saved in a folder.

5.1.4 Detection

In this phase, the Streamlit web app or web system will utilise the trained weights (a.k.a model) that we put in into the system just now to perform "Header", "Row" and "HeaderRow" detection on every converted image. When the model detects the targeted object, the object or the region will be wrapped by a bounding box indicating the object name like "Row" and a confidence value like 0.87. Different types of objects will be wrapped in a unique colour of bounding box so that users can easily classify them. Each page (has been converted to an image) of detection results will be saved into a folder.

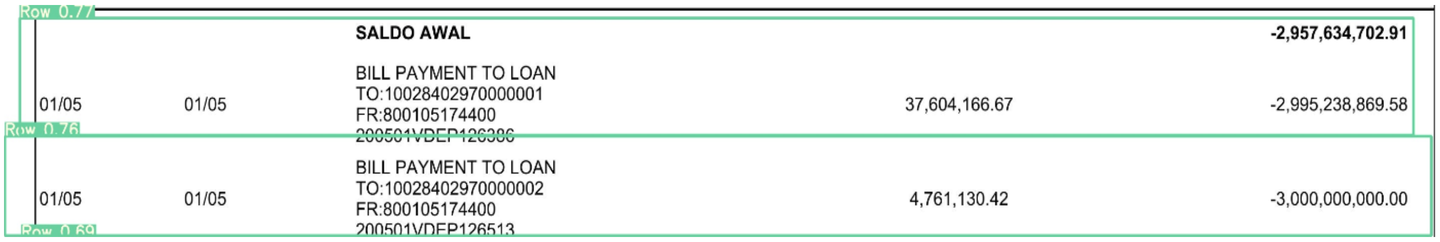
5.1.5 Cropping

After the detection process, the system will continue by cropping all the annotated regions (wrapped with a bounding box) in every image and saving all the cropped images into a folder. All the crop images are the parts that we are interested in extracting its text in an image where the parts that we are interested in are "Header", "HeaderRow" and "Row". Examples are shown below:



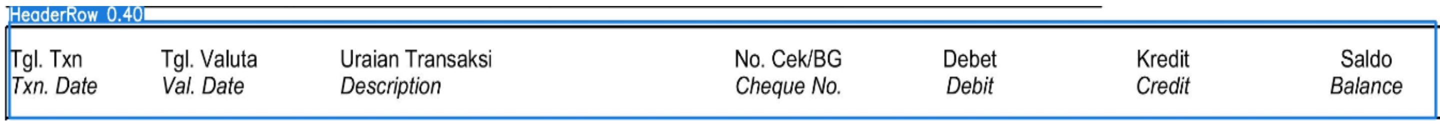
No. Rekening / *Account Number*	:	800105174400
Nama Produk / *Product Name*	:	Giro Perusahaan
Mata Uang / *Currency*	:	IDR
Nomor CIF / *CIF Number*	:	11330000052783

Figure 5.1.5a: Header



		SALDO AWAL			**-2,957,634,702.91**
01/05	01/05	BILL PAYMENT TO LOAN TO:10028402970000001 FR:800105174400 200501VDEP126386	37,604,166.67		-2,995,238,869.58
01/05	01/05	BILL PAYMENT TO LOAN TO:10028402970000002 FR:800105174400 200501VDEP126513	4,761,130.42		-3,000,000,000.00

Figure 5.1.5b: Row



Tgl. Txn *Txn. Date*	Tgl. Valuta *Val. Date*	Uraian Transaksi *Description*	No. Cek/BG *Cheque No.*	Debet *Debit*	Kredit *Credit*	Saldo *Balance*

Figure 5.1.5c: HeaderRow

5.1.6 Image Preprocessing

At this layer, the program will preprocess all the cropped images where the system will do some image preprocessing operations including, grayscaling, blurring, thresholding and noise removal. Grayscale preprocessing is to convert the images to black and white or grey monochrome which carries only intensity information while the blurring method applied is GaussianBlur, where it helps to average out rapid changes in pixel intensity in the meantime makes the stroke itself bold so more features may be detected during optical character recognition. Then, the thresholding method applied is called Otsu's thresholding which will automatically determine the best threshold value and followed by noise removal to remove some noises in the image like black spots.

5.1.7 Optical Character Recognition

In this last phase, the program or the system will use Pytesseract to extract the text within every preprocessed cropped image. Particularly, the system will recognize and extract the text from the preprocessed cropped images which are detected or annotated as "Row". The system will use the image_to_sring function to recognize and extract the text within the image and finally save the results as a CSV file since the images classified as "Row" are actually cropped from a table in a bank statement PDF file.

5.2 System Block Diagram

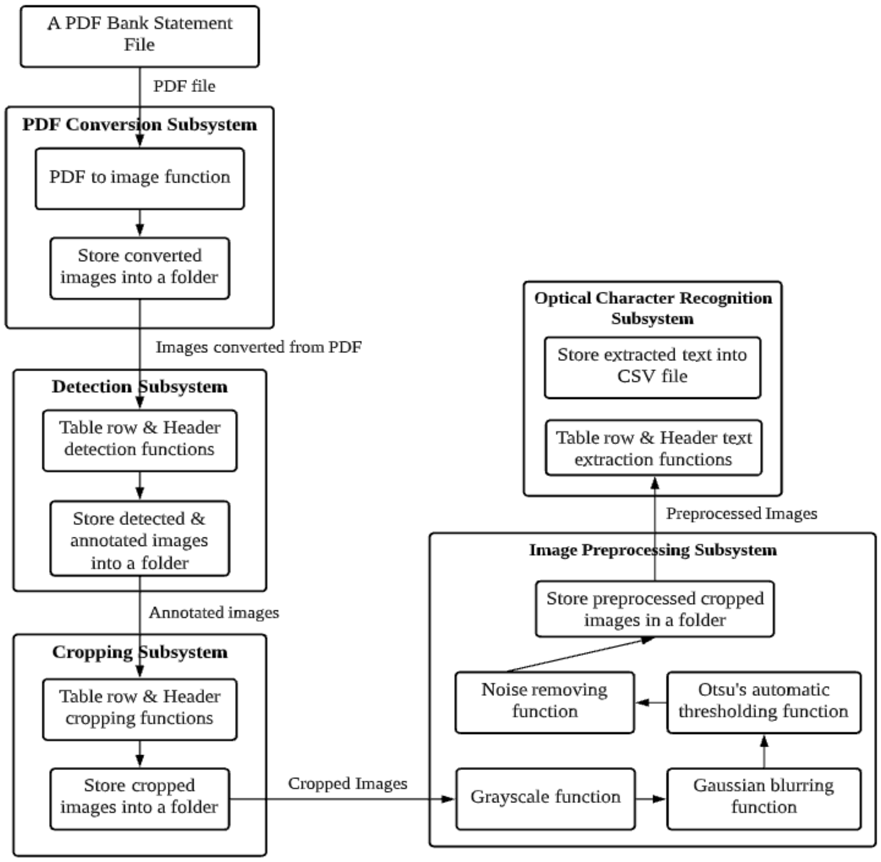

A PDF Bank Statement File
PDF file
PDF Conversion Subsystem
PDF to image function
Store converted images into a folder
Images converted from PDF
Detection Subsystem
Table row & Header detection functions
Store detected & annotated images into a folder
Annotated images
Cropping Subsystem
Table row & Header cropping functions
Store cropped images into a folder
Cropped Images
Image Preprocessing Subsystem
Grayscale function
Gaussian blurring function
Otsu's automatic thresholding function
Noise removing function
Store preprocessed cropped images in a folder
Preprocessed Images
Optical Character Recognition Subsystem
Table row & Header text extraction functions
Store extracted text into CSV file

Figure 5.2: System Block Diagram

Figure 5.2 presents the system block diagram for the Table OCR Extraction Framework. From the diagram above, we can see that there are a total of 5 subsystems to form this framework. The subsystems are *PDF Conversion Subsystem*, *Detection Subsystem*, *Cropping Subsystem, Image Preprocessing Subsystem* and *Optical Character Recognition Subsystem*.

5.2.1 PDF Conversion Subsystem

First of all, this subsystem will accept a PDF file uploaded from the user and then the system will utilise the function from the python library called pdf2image to convert each page of the uploaded PDF file to an image in jpeg format. Upon converting to image, the converted image will be stored inside a folder for later use.

5.2.2 Detection Subsystem

After that, the detection subsystem will retrieve all converted images that have just been stored inside a folder to do detection on each of these images. In this process, the system will utilise the Yolov5 trained weight or so called the Yolov5 trained deep learning model to do classification on the image where the system will draw a bounding on the part that the model could classify and then annotate it whether it is “Row”, “Header” or “HeaderRow". Upon detecting and annotating an image, the results will be stored inside a folder for display purposes in the Streamlit web app as well as later use.

5.2.3 Cropping Subsystem

Later, the cropping subsystem will retrieve all detection results (images with annotations) that have just been gone through and saved inside a folder. In this process, the system will crop all the parts that have been annotated (means wrapped by bounding box) within an image and save it inside a folder. The interested parts within an image based on priority are “Row”, “HeaderRow”, “Header”. So, the system will crop all these annotations and save it as a single image in JPEG format inside a folder.

5.2.4 Image Preprocessing Subsystem

After that, the image preprocessing subsystem will preprocess all the cropped images. The system will retrieve all the cropped images that have just been saved in the folder and carry out the below image preprocessing operations using opencv functions:

1. **Grayscaling:** Convert the images to black and white or grey monochrome which carries only intensity information. Example code:

```
image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
```

2. **Blurring:** Helps to average out rapid changes in pixel intensity in the meantime makes the stroke itself bold so more features may be detected during optical character recognition. Example code:

```
image = cv2.GaussianBlur(image, (5, 5), 0)
```

3. **Thresholding:** Apply Otsu's thresholding which will automatically determine the best threshold value. Example code:

```
threshImage = cv2.threshold(image, 0, 255, cv2.THRESH_BINARY |
cv2.THRESH_OTSU)
```

4. **Noise Removal:** Helps to remove some noises in the image such as black spots in the image. Example code:

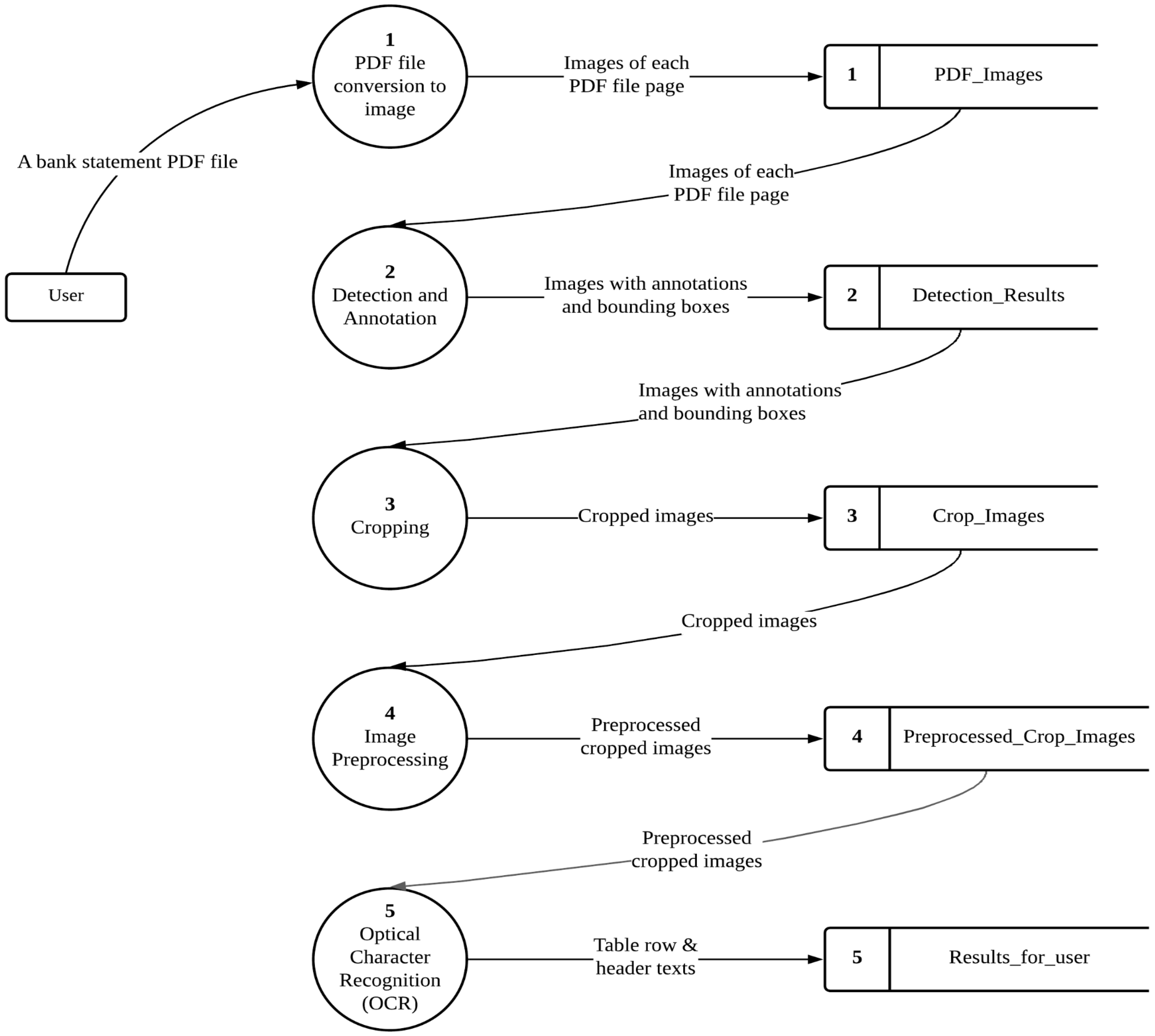
```
def noise_removal(image):
    kernel = np.ones((1, 1), np.uint8)
    image = cv2.dilate(image, kernel, iterations=1)
    kernel = np.ones((1, 1), np.uint8)
    image = cv2.erode(image, kernel, iterations=1)
    image = cv2.morphologyEx(image, cv2.MORPH_CLOSE, kernel)
    image = cv2.medianBlur(image, 3)
    return image
cleaned_image = noise_removal(threshImage)
```

After going through all these preprocessings, the system will save the preprocessed cropped images into a folder for later used.

5.2.5 Optical Character Recognition Subsystem

Finally, the optical character recognition subsystem comes into the role. This subsystem will use the python library package called pytesseract to extract the text within every preprocessed cropped image. Particularly, the system will recognize and extract the text from the preprocessed cropped images which are detected and annotated as “Row”. The system will use the image_to_sring function to recognize and extract the text within the image and finally save the results as a CSV file since the images classified as “Row” are actually cropped from a table in a bank statement PDF file.

5.3 Data Flow Diagram (DFD)



User
A bank statement PDF file
1 PDF file conversion to image
Images of each PDF file page
1 PDF_Images
Images of each PDF file page
2 Detection and Annotation
Images with annotations and bounding boxes
2 Detection_Results
Images with annotations and bounding boxes
3 Cropping
Cropped images
3 Crop_Images
Cropped images
4 Image Preprocessing
Preprocessed cropped images
4 Preprocessed_Crop_Images
Preprocessed cropped images
5 Optical Character Recognition (OCR)
Table row & header texts
5 Results_for_user

Figure 5.3: Table OCR Extraction Framework: Data Flow Diagram

Figure 5.3 presents the data flow diagram for the FoCUS web forum crawler. From the diagram above, we can see that there is one external entity, 5 processes and 5 data stores. Below is the diagram that describes the meaning of each shape in the data flow diagram.

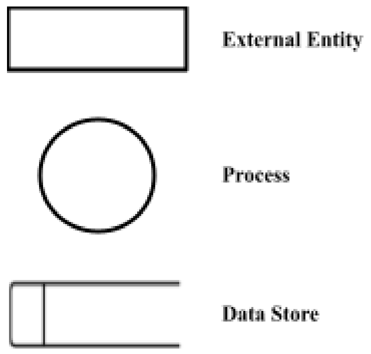


Figure 5.4: Shape meaning in Data Flow Diagram

5.3.1 External Entity: User

Moving forward, we can see that there is an **external entity** called **user** in the data flow diagram. User is the one who uses this program or system to extract the table row text and header text from the PDF file. First of all, the user is required to upload a PDF bank statement file that he/she wants to extract the text inside to the program, which the PDF file will be passed to the **process 1**.

5.3.2 Process 1: PDF File Conversion

In **Process 1**, the program will receive the uploaded PDF file provided by the user. It will then start converting each page of the PDF file to an image in JPEG format.

5.3.3 Data Store 1: PDF_Images

After the **process 1** is done, the **first data store** comes into play, where all converted images will be stored in a folder, namely PDF_Images.

5.3.4 Process 2: Detection and Annotation

In **Process 2**, the program will retrieve the results of process 1 from the data store 1 which are the converted images from PDF. Upon retrieving each converted image, the program will start to detect and annotate the image by using the trained weight (a.k.a. deep learning model) where the program will do “Header”, “Row” and “HeaderRow” detection. If the program detects there is a part in the image as “Row”, the program will draw a bounding box to wrap that part and then annotate it with “Row”.

5.3.5 Data Store 2: Detection_Results

After the **process 2** is done, the **second data store** comes into play, where all the annotated images will be stored in a folder, namely *Detection_results*.

5.3.6 Process 3: Cropping

In **Process 3**, the program will retrieve all the annotated images from data store 2. Then, it will start cropping each annotated image, which it will crop all every part that is wrapped by a bounding box in the image.

5.3.7 Data Store 3: Crop_Images

After the **process 3** is done, the **third data store** comes into play, where all the cropped images will be stored in a folder called *Crop_Images*.

5.3.8 Process 4: Image Preprocessing

In **process 4**, the program will retrieve all the cropped images from data store 3. Then, it will start preprocessing all cropped images by going through the operations of grayscaling, blurring, thresholding, and noise removal.

5.3.8 Data Store 4: Preprocessed_Crop_Images

After the **process 4** is done, the **forth data store** comes into play, where all the preprocessed cropped images will be stored in a folder called *Preprocessed_Crop_Images*.

5.3.9 Process 5: Optical Character Recognition (OCR)

In **Process 4**, the program will retrieve all the preprocessed cropped images from data store 4. Then, the system will use the python library package called pytesseract to extract the text within every preprocessed cropped image. Particularly, the system will recognize and extract the text from the preprocessed cropped images which are detected and annotated as “Row” by using the image_to_sring function to recognize and extract the text within the image.

5.3.9 Data Store 5: Results_for_user

After the **process 5** is done, the **fifth data store** comes into play, where all the extracted texts annotated as “Row” will be saved in a CSV file inside the folder called *Results_for_user*.

5.4 Flowchart

In this project, the FoCUS crawler is made up of 5 python scripts. The 5 python scripts are:

1. **main.py**
   Python script that gathers the main functions in other python scripts.
2. **upload_label_crop.py**
   Python script that accept user’s uploaded PDF, detect region of interests and crop region of interests
3. **Preprocess_ocr.py**
   Python script that preprocess all cropped images and do OCR for text extraction)

5.4.1 main.py

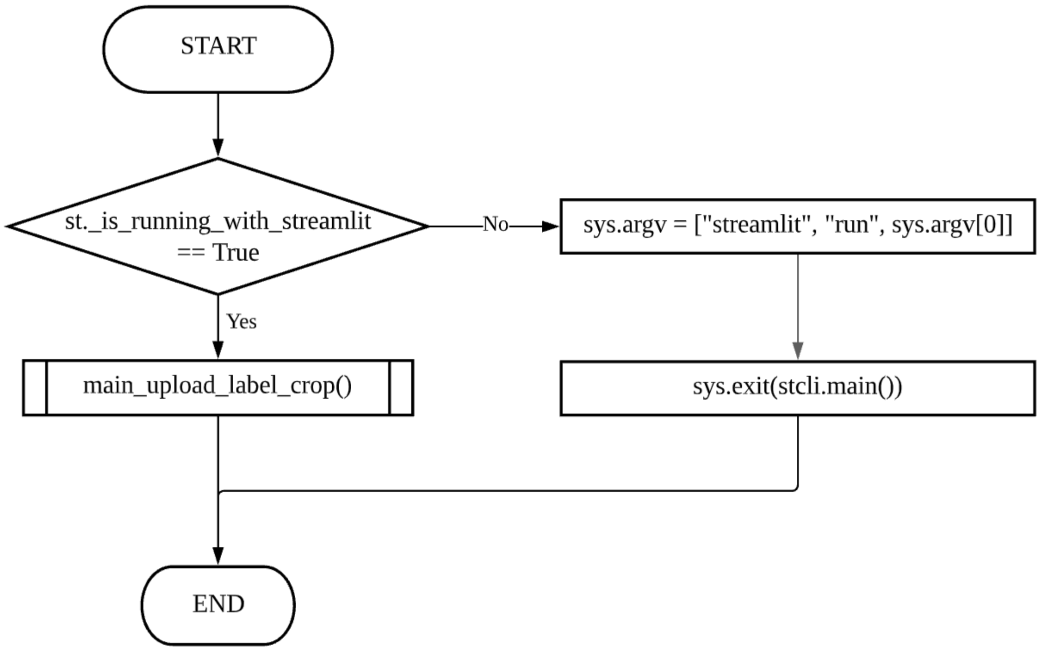

START
st._is_running_with_streamlit == True
No
sys.argv = ["streamlit", "run", sys.argv[0]]
Yes
main_upload_label_crop()
sys.exit(stcli.main())
END

Figure 5.5: Flowchart of main.py

The above figure shows the program flow of the *main.py* script. We can see that the *main.py* calls the main function that exists in other python scripts where the *main.py* script calls the `main_upload_label_crop()` function in the beginning. The if else statement shown in the figure above is used to ease the user to run the program as this is a Streamlit web program in which by default the user is required to type the command of `streamlit run main.py` in the terminal in order to run the program. However, the user can just run this python program as usual, such as clicking the **run icon** in the IDE after adding the if else statement shown in the figure above. Once you click the **run icon**, the if statement at first will check if your streamlit program is currently running, it will proceed to the `main_upload_label_crop()` function if yes. Else, the program will run the code block inside the else statement to activate or run your streamlit program. `main_upload_label_crop()` is the main function of the *upload_label_crop.py* script.

5.4.1.1 Functions used in main.py

1. main_upload_label_crop()

Description	Accept PDF file uploaded by user, convert the PDF file to images, do detection and annotations and lastly do cropping on the annotated images.
Method Type	static
Argument	Do not require argument
Return	Do not return anything
Example	`main_upload_label_crop()`

5.4.2 upload_label_crop.py

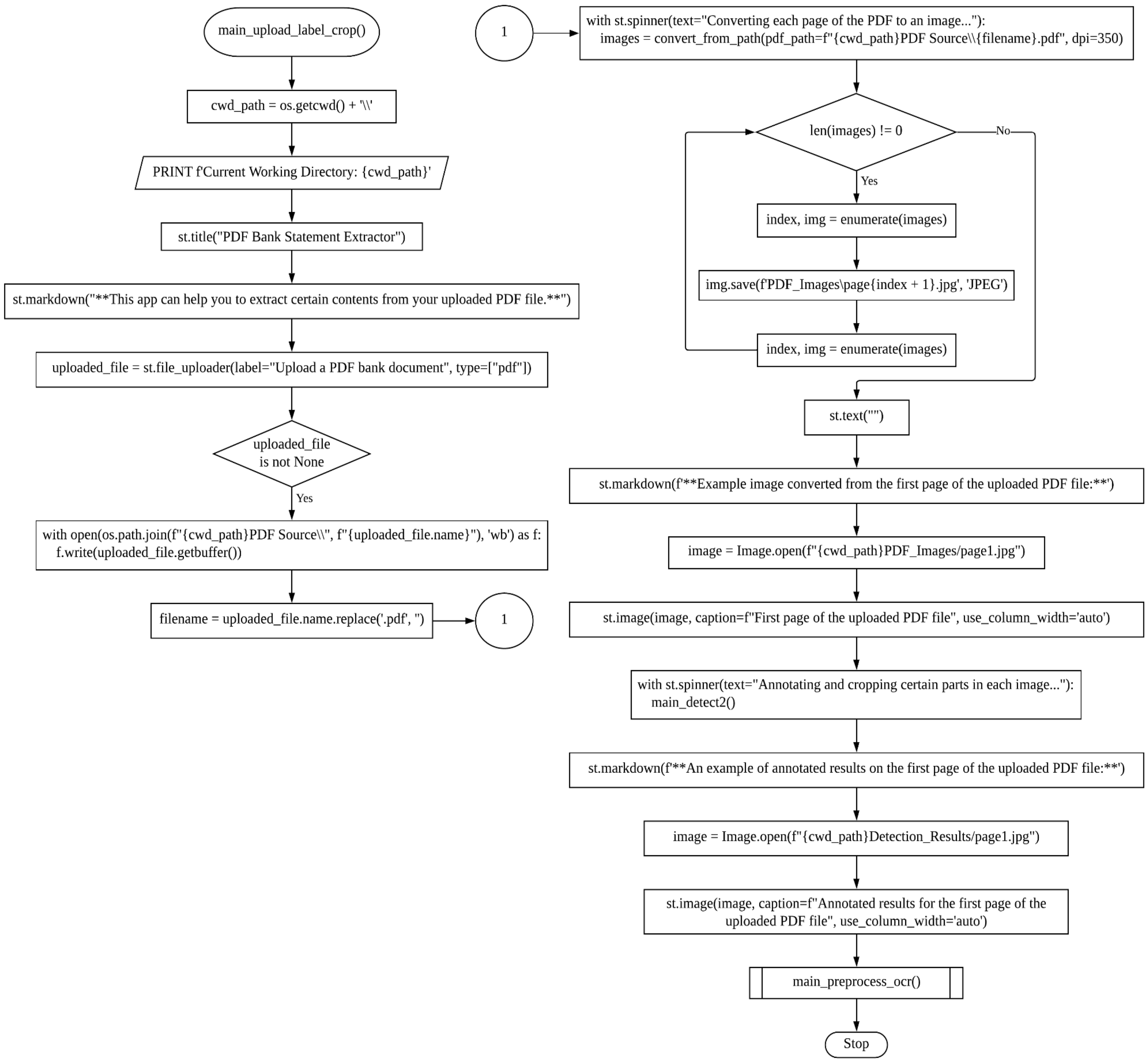

main_upload_label_crop()
cwd_path = os.getcwd() + '\\'
PRINT f'Current Working Directory: {cwd_path}'
st.title("PDF Bank Statement Extractor")
st.markdown("**This app can help you to extract certain contents from your uploaded PDF file.**")
uploaded_file = st.file_uploader(label="Upload a PDF bank document", type=["pdf"])
uploaded_file is not None
Yes
with open(os.path.join(f"{cwd_path}PDF Source\\", f"{uploaded_file.name}"), 'wb') as f: f.write(uploaded_file.getbuffer())
filename = uploaded_file.name.replace('.pdf', '')
1
1
with st.spinner(text="Converting each page of the PDF to an image..."): images = convert_from_path(pdf_path=f"{cwd_path}PDF Source\\{filename}.pdf", dpi=350)
len(images) != 0
No
Yes
index, img = enumerate(images)
img.save(f'PDF_Images\page{index + 1}.jpg', 'JPEG')
index, img = enumerate(images)
st.text("")
st.markdown(f'**Example image converted from the first page of the uploaded PDF file:**')
image = Image.open(f"{cwd_path}PDF_Images/page1.jpg")
st.image(image, caption=f"First page of the uploaded PDF file", use_column_width='auto')
with st.spinner(text="Annotating and cropping certain parts in each image..."): main_detect2()
st.markdown(f'**An example of annotated results on the first page of the uploaded PDF file:**')
image = Image.open(f"{cwd_path}Detection_Results/page1.jpg")
st.image(image, caption=f"Annotated results for the first page of the uploaded PDF file", use_column_width='auto')
main_preprocess_ocr()
Stop

Figure 5.6: Flowchart of upload_label_crop.py

5.4.2.1 Functions used in upload_label_crop.py

1. main_preprocess_ocr_()

Description	Retrieve all the images that have been detected and annotated from the folder and then do image preprocessing on retrieved images followed by OCR to extract the required text.
Method Type	static
Argument	Do not require argument
Return	Do not return anything
Example	`main_upreprocess_ocr()`

5.4.3 preprocess_ocr.py

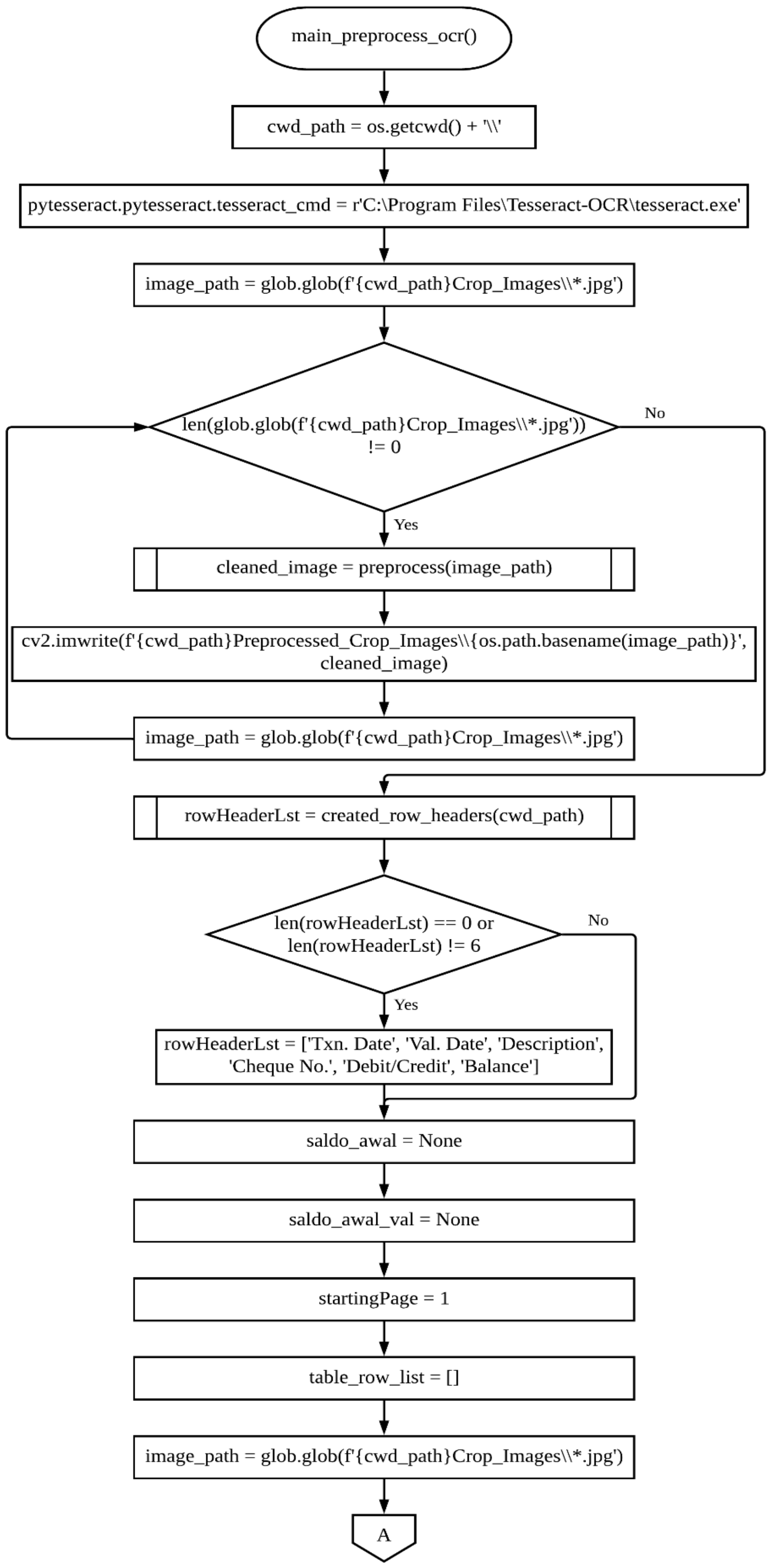

main_preprocess_ocr()
cwd_path = os.getcwd() + '\\'
pytesseract.pytesseract.tesseract_cmd = r'C:\Program Files\Tesseract-OCR\tesseract.exe'
image_path = glob.glob(f'{cwd_path}Crop_Images*.jpg')
len(glob.glob(f'{cwd_path}Crop_Images*.jpg')) != 0
No
Yes
cleaned_image = preprocess(image_path)
cv2.imwrite(f'{cwd_path}Preprocessed_Crop_Images\\{os.path.basename(image_path)}', cleaned_image)
image_path = glob.glob(f'{cwd_path}Crop_Images*.jpg')
rowHeaderLst = created_row_headers(cwd_path)
len(rowHeaderLst) == 0 or len(rowHeaderLst) != 6
No
Yes
rowHeaderLst = ['Txn. Date', 'Val. Date', 'Description', 'Cheque No.', 'Debit/Credit', 'Balance']
saldo_awal = None
saldo_awal_val = None
startingPage = 1
table_row_list = []
image_path = glob.glob(f'{cwd_path}Crop_Images*.jpg')
A

Figure 5.7: Flowchart of preprocess_ocr.py

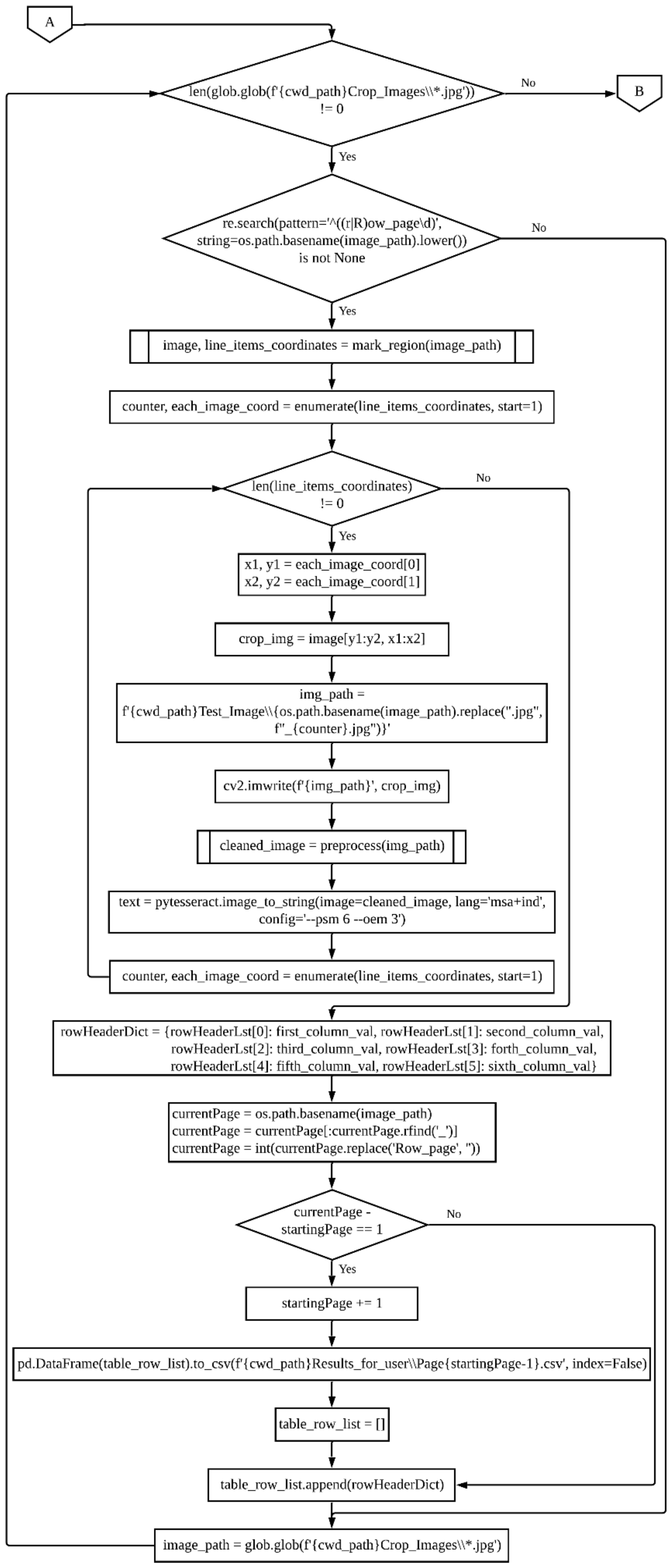

A
len(glob.glob(f'{cwd_path}Crop_Images*.jpg')) != 0
No
B
Yes
re.search(pattern='^((r|R)ow_page\d)', string=os.path.basename(image_path).lower()) is not None
No
Yes
image, line_items_coordinates = mark_region(image_path)
counter, each_image_coord = enumerate(line_items_coordinates, start=1)
len(line_items_coordinates) != 0
No
Yes
x1, y1 = each_image_coord[0]
x2, y2 = each_image_coord[1]
crop_img = image[y1:y2, x1:x2]
img_path = f'{cwd_path}Test_Image\\{os.path.basename(image_path).replace(".jpg", f"_{counter}.jpg")}'
cv2.imwrite(f'{img_path}', crop_img)
cleaned_image = preprocess(img_path)
text = pytesseract.image_to_string(image=cleaned_image, lang='msa+ind', config='--psm 6 --oem 3')
counter, each_image_coord = enumerate(line_items_coordinates, start=1)
rowHeaderDict = {rowHeaderLst[0]: first_column_val, rowHeaderLst[1]: second_column_val, rowHeaderLst[2]: third_column_val, rowHeaderLst[3]: forth_column_val, rowHeaderLst[4]: fifth_column_val, rowHeaderLst[5]: sixth_column_val}
currentPage = os.path.basename(image_path)
currentPage = currentPage[:currentPage.rfind('_')]
currentPage = int(currentPage.replace('Row_page', ''))
currentPage - startingPage == 1
No
Yes
startingPage += 1
pd.DataFrame(table_row_list).to_csv(f'{cwd_path}Results_for_user\\Page{startingPage-1}.csv', index=False)
table_row_list = []
table_row_list.append(rowHeaderDict)
image_path = glob.glob(f'{cwd_path}Crop_Images*.jpg')

Figure 5.8: Flowchart of preprocess_ocr.py (continue)

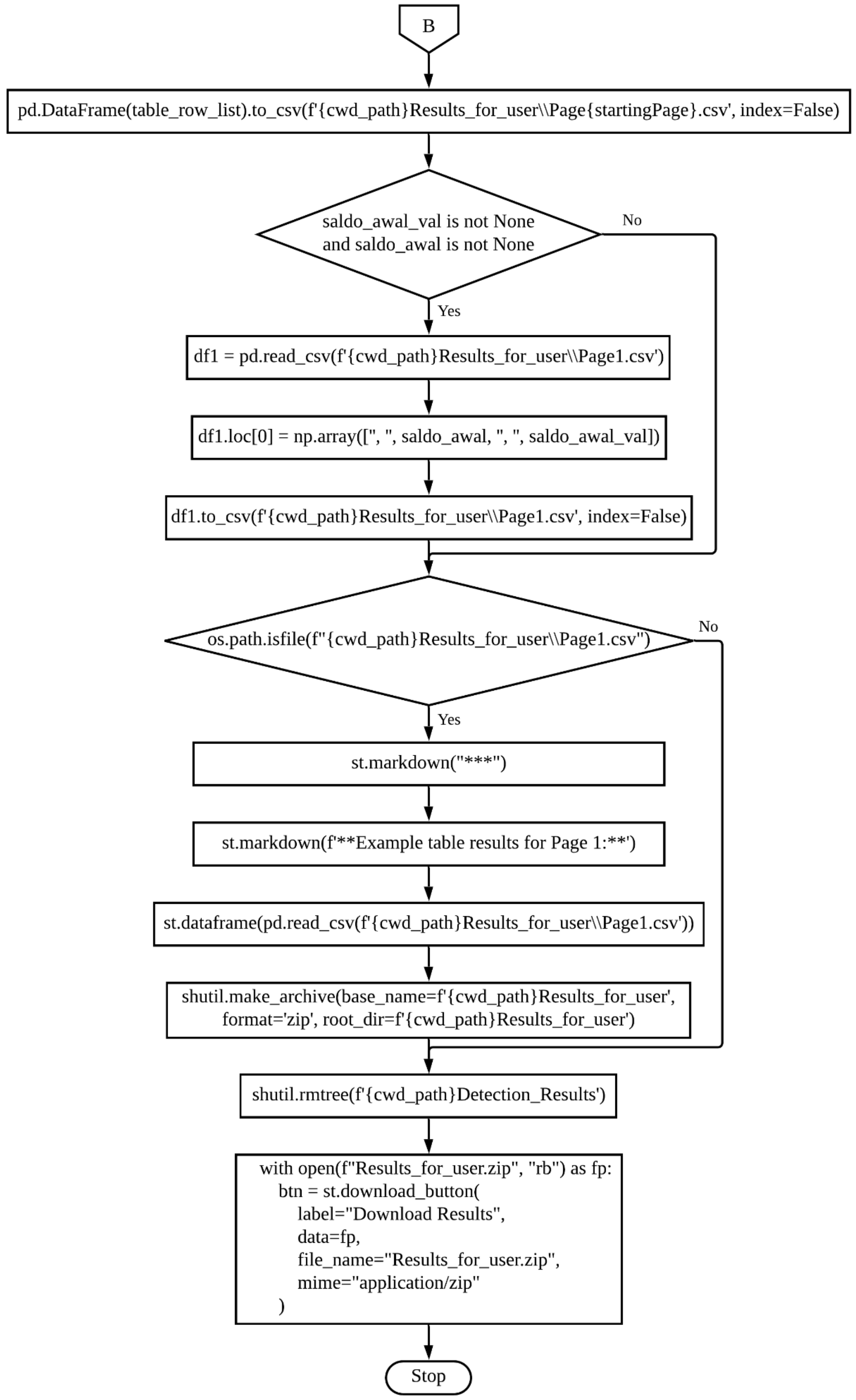

B
pd.DataFrame(table_row_list).to_csv(f'{cwd_path}Results_for_user\\Page{startingPage}.csv', index=False)
saldo_awal_val is not None and saldo_awal is not None
No
Yes
df1 = pd.read_csv(f'{cwd_path}Results_for_user\\Page1.csv')
df1.loc[0] = np.array(['', '', saldo_awal, '', '', saldo_awal_val])
df1.to_csv(f'{cwd_path}Results_for_user\\Page1.csv', index=False)
os.path.isfile(f"{cwd_path}Results_for_user\\Page1.csv")
No
Yes
st.markdown("***")
st.markdown(f'**Example table results for Page 1:**')
st.dataframe(pd.read_csv(f'{cwd_path}Results_for_user\\Page1.csv'))
shutil.make_archive(base_name=f'{cwd_path}Results_for_user', format='zip', root_dir=f'{cwd_path}Results_for_user')
shutil.rmtree(f'{cwd_path}Detection_Results')
with open(f"Results_for_user.zip", "rb") as fp:
btn = st.download_button(
label="Download Results",
data=fp,
file_name="Results_for_user.zip",
mime="application/zip"
)
Stop

Figure 5.9: Flowchart of preprocess_ocr.py (continue)

5.4.3.1 Functions used in preprocess_ocr.py

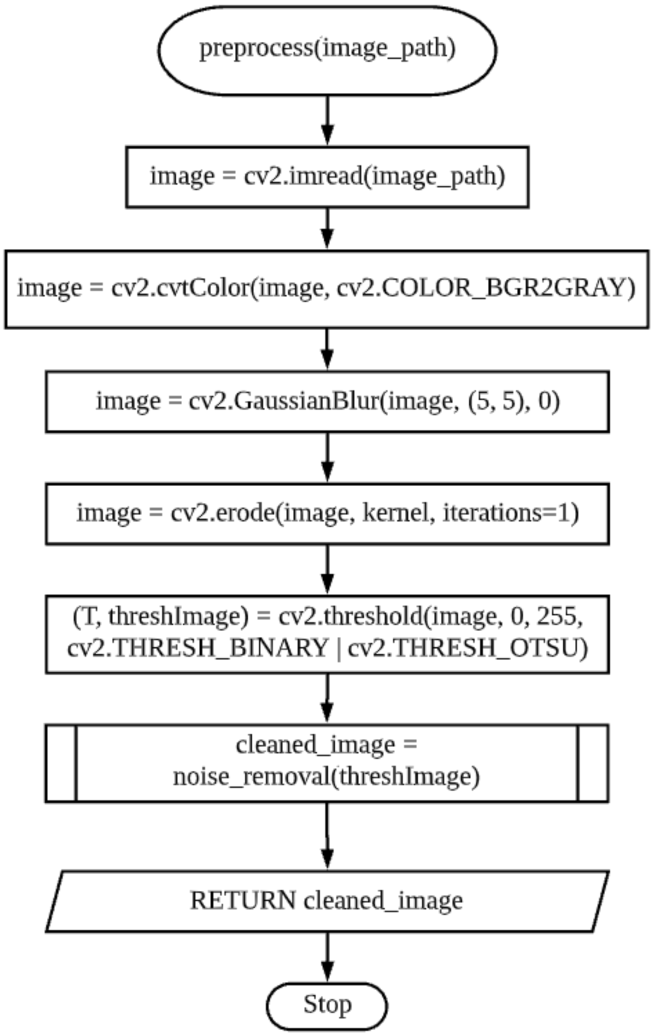

preprocess(image_path)
image = cv2.imread(image_path)
image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
image = cv2.GaussianBlur(image, (5, 5), 0)
image = cv2.erode(image, kernel, iterations=1)
(T, threshImage) = cv2.threshold(image, 0, 255, cv2.THRESH_BINARY | cv2.THRESH_OTSU)
cleaned_image = noise_removal(threshImage)
RETURN cleaned_image
Stop

Figure 5.10: preprocess(image_path) function

1. preprocess(image_path)

Description	Preprocess the received image by going through the operations of grayscaling, blurring, thresholding and noise removal.
Method Type	static
Argument	`image_path` = Path to the image source in your computer.
Return (`numpy.ndarray`)	Return a cleaned or preprocessed image which is in numpy.ndarray type.
Example	**`cleaned_image = preprocess('Crop_Images\\Header_page1_1.jpg')`**

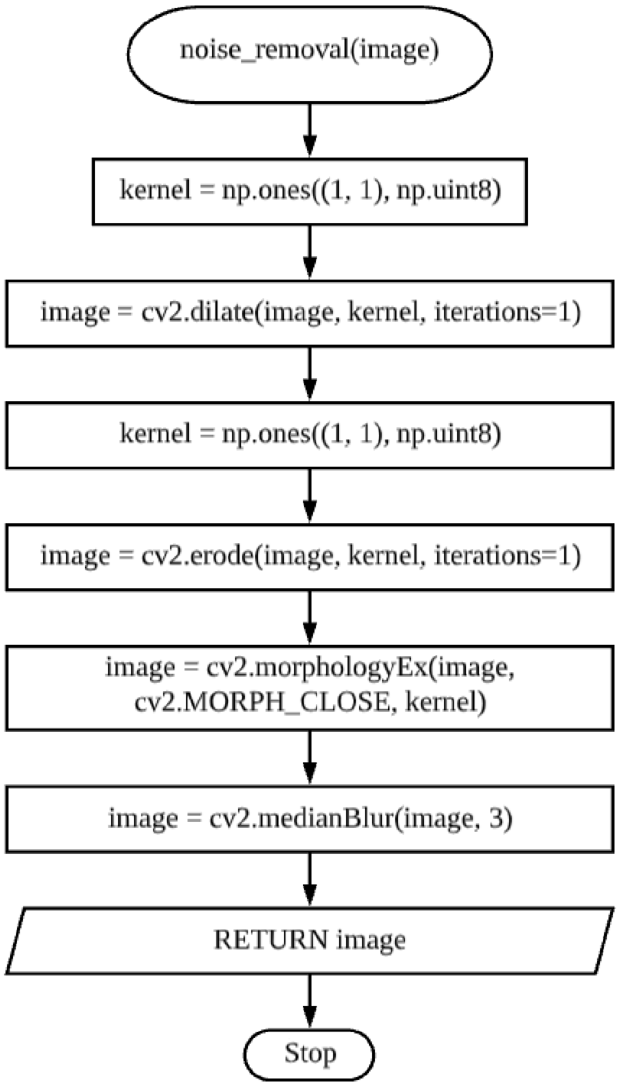

noise_removal(image)
kernel = np.ones((1, 1), np.uint8)
image = cv2.dilate(image, kernel, iterations=1)
kernel = np.ones((1, 1), np.uint8)
image = cv2.erode(image, kernel, iterations=1)
image = cv2.morphologyEx(image, cv2.MORPH_CLOSE, kernel)
image = cv2.medianBlur(image, 3)
RETURN image
Stop

Figure 5.11: noise_removal(image) function

2. noise_removal(image)

Description	Remove the noises contained in an image such as black spots in an image
Method Type	static
Argument	`image` = An image that is in numpy.ndarray type.
Return (`numpy.ndarray`)	Return a preprocessed image with lesser noise which is in numpy.ndarray type.
Example	`cleaned_image = noise_removal('Crop_Images\\Header_page1_1.jpg')`

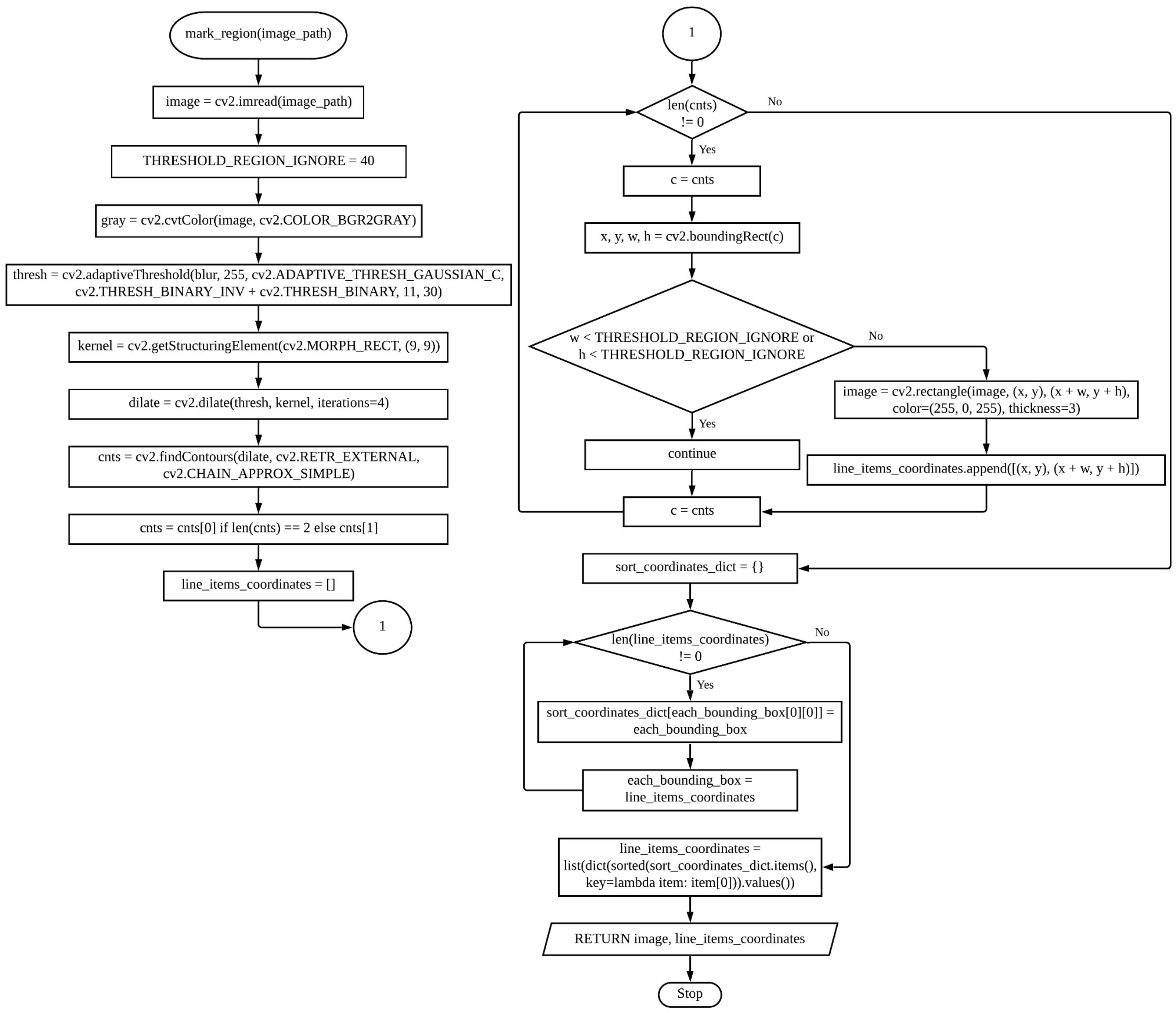

mark_region(image_path)
image = cv2.imread(image_path)
THRESHOLD_REGION_IGNORE = 40
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
thresh = cv2.adaptiveThreshold(blur, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY_INV + cv2.THRESH_BINARY, 11, 30)
kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (9, 9))
dilate = cv2.dilate(thresh, kernel, iterations=4)
cnts = cv2.findContours(dilate, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
cnts = cnts[0] if len(cnts) == 2 else cnts[1]
line_items_coordinates = []
1
1
len(cnts) != 0
No
Yes
c = cnts
x, y, w, h = cv2.boundingRect(c)
w < THRESHOLD_REGION_IGNORE or h < THRESHOLD_REGION_IGNORE
No
image = cv2.rectangle(image, (x, y), (x + w, y + h), color=(255, 0, 255), thickness=3)
Yes
continue
line_items_coordinates.append([(x, y), (x + w, y + h)])
c = cnts
sort_coordinates_dict = {}
len(line_items_coordinates) != 0
No
Yes
sort_coordinates_dict[each_bounding_box[0][0]] = each_bounding_box
each_bounding_box = line_items_coordinates
line_items_coordinates = list(dict(sorted(sort_coordinates_dict.items(), key=lambda item: item[0])).values())
RETURN image, line_items_coordinates
Stop

Figure 5.12: mark_region(image_path) function

3. mark_region(image_path)

Description	This function will draw a bounding box on some parts of an image if there are a bunch of texts on that part.
Method Type	static
Argument	`image path` = Path to the image source in your computer.

Return (`numpy.ndarray` & `list`)	Return an image with bounding boxes wrapped on each part of the image that has a bunch of texts. It also returns the coordinates of each bounding box drawn in an image.
Example	`image, line_items_coordinates = mark_region('Crop_Images\\Header_page1_1.jpg')`

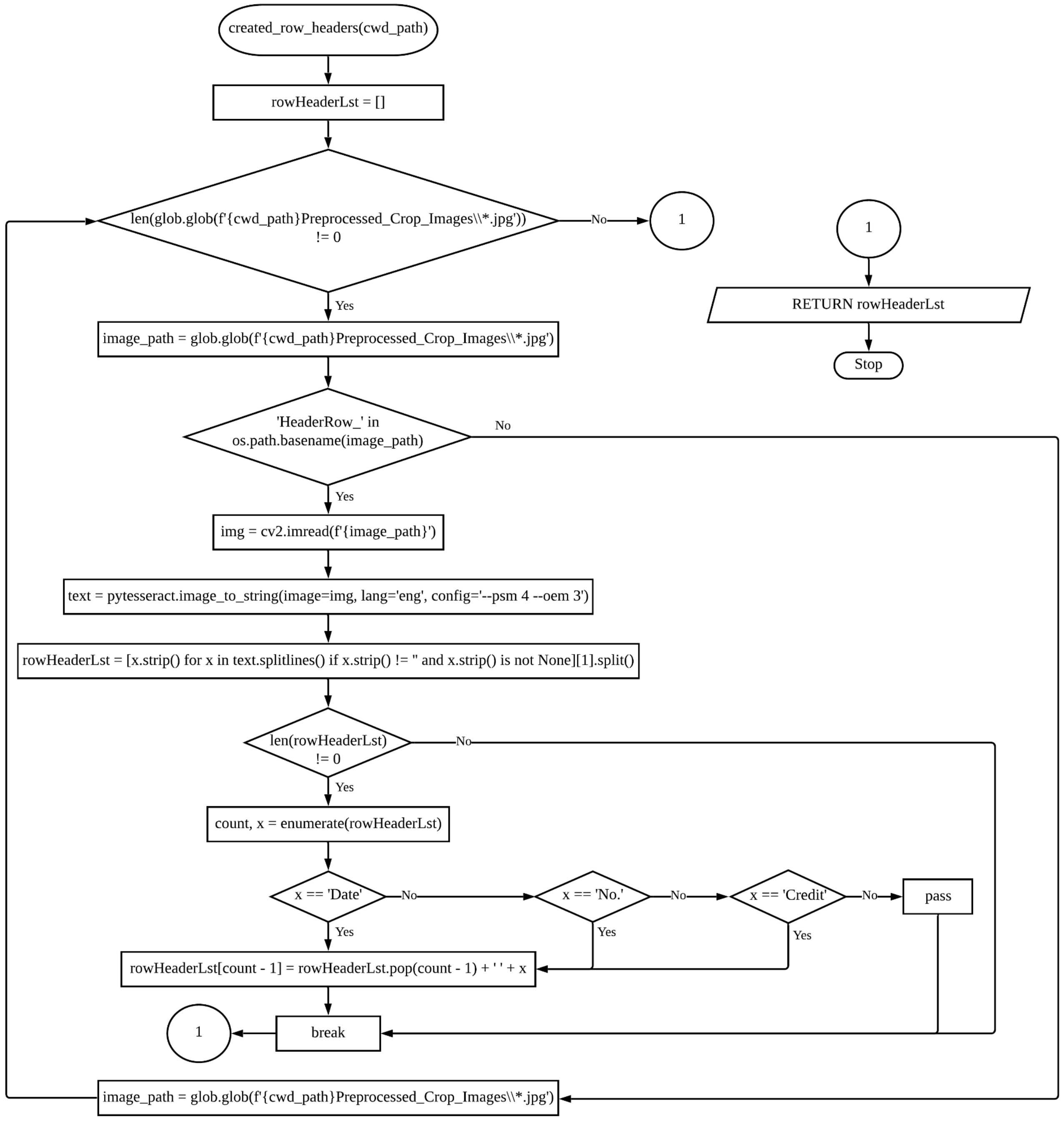

created_row_headers(cwd_path)
rowHeaderLst = []
len(glob.glob(f'{cwd_path}Preprocessed_Crop_Images\\*.jpg')) != 0
No
1
Yes
image_path = glob.glob(f'{cwd_path}Preprocessed_Crop_Images\\*.jpg')
RETURN rowHeaderLst
Stop
'HeaderRow_' in os.path.basename(image_path)
img = cv2.imread(f'{image_path}')
text = pytesseract.image_to_string(image=img, lang='eng', config='--psm 4 --oem 3')
rowHeaderLst = [x.strip() for x in text.splitlines() if x.strip() != '' and x.strip() is not None][1].split()
len(rowHeaderLst) != 0
count, x = enumerate(rowHeaderLst)
x == 'Date'
x == 'No.'
x == 'Credit'
pass
rowHeaderLst[count - 1] = rowHeaderLst.pop(count - 1) + ' ' + x
break

Figure 5.13: created_row_headers(cwd_path) function

4. created_row_headers(cwd_path)

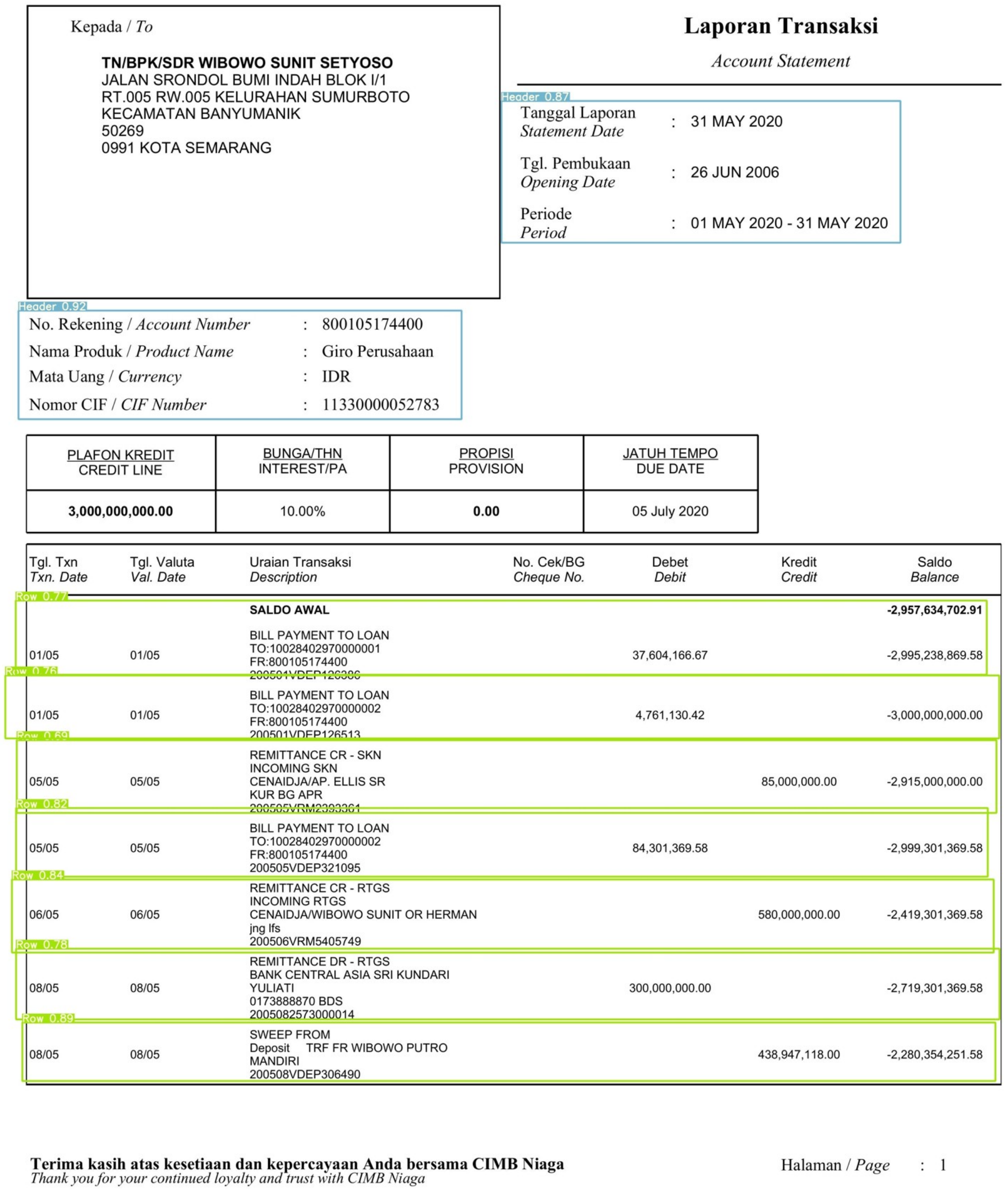
Description	This function will loop through all the images in the *Preprocessed_Crop_Images* folder to find an image that contains the header of the table only in order to extract the table header contents.
Method Type	static
Argument	`cwd_path` = Current working directory path
Return (`list`)	Return a list that contains each header text of a table in an image.
Example	`rowHeaderLst = created_row_headers('.\FYP_2')`

5.5 Execution Session

In this section, the CIMB PDF bank statement files have been selected for experiments. One is using a native CIMB PDF file for **table row detection**, while another one is using a non-native CIMB PDF file for **table row detection**. The purpose is to compare the differences between the number of rows content could be detected while using the similar trained deep learning model. Meanwhile, I also will compare the model is able to detect the **Table Header**, as well as the **Header** within the PDF file. To maximise the fairness of the experiment, I will only choose the first page of both selected native and non-native selected PDF files.

Activities/PDF Files	**Experiment 1**		**Experiment 2**	
	Native PDF		**Non-Native PDF**	
	Actual Numbers	**Detected Numbers**	**Actual Numbers**	**Detected Numbers**
Table Rows	8	7	12	4
Table Header	1	0	1	0
Header	2	2	2	2

Table 5.5.1: Detection and extraction results


Header 0.87
Header 0.92
Row 0.77
Row 0.76
Row 0.69
Row 0.82
Row 0.84
Row 0.78
Row 0.89

Kepada / *To*

TN/BPK/SDR WIBOWO SUNIT SETYOSO
JALAN SRONDOL BUMI INDAH BLOK I/1
RT.005 RW.005 KELURAHAN SUMURBOTO
KECAMATAN BANYUMANIK
50269
0991 KOTA SEMARANG

Laporan Transaksi
Account Statement

Tanggal Laporan *Statement Date*	:	31 MAY 2020
Tgl. Pembukaan *Opening Date*	:	26 JUN 2006
Periode *Period*	:	01 MAY 2020 - 31 MAY 2020

No. Rekening / *Account Number*	:	800105174400
Nama Produk / *Product Name*	:	Giro Perusahaan
Mata Uang / *Currency*	:	IDR
Nomor CIF / *CIF Number*	:	11330000052783

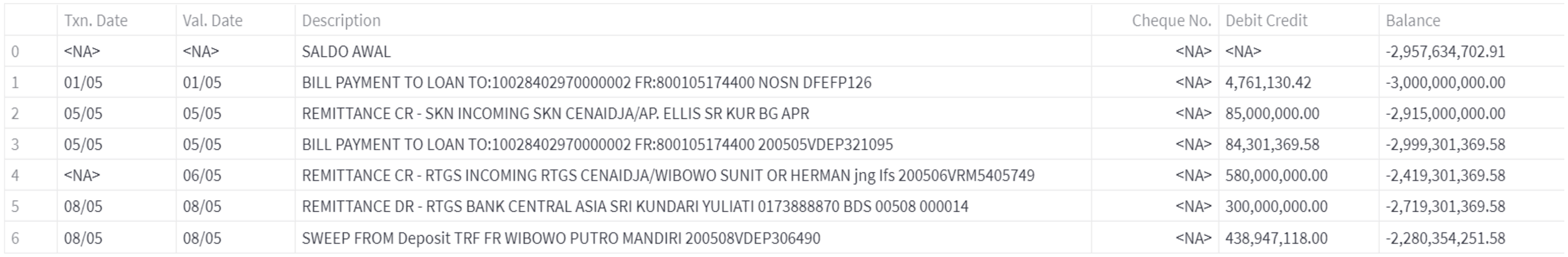
PLAFON KREDIT CREDIT LINE	BUNGA/THN INTEREST/PA	PROPISI PROVISION	JATUH TEMPO DUE DATE
3,000,000,000.00	10.00%	**0.00**	05 July 2020

Tgl. Txn *Txn. Date*	Tgl. Valuta *Val. Date*	Uraian Transaksi *Description*	No. Cek/BG *Cheque No.*	Debet *Debit*	Kredit *Credit*	Saldo *Balance*
		SALDO AWAL				**-2,957,634,702.91**
01/05	01/05	BILL PAYMENT TO LOAN TO:10028402970000001 FR:800105174400 200501VDEP126386		37,604,166.67		-2,995,238,869.58
01/05	01/05	BILL PAYMENT TO LOAN TO:10028402970000002 FR:800105174400 200501VDEP126513		4,761,130.42		-3,000,000,000.00
05/05	05/05	REMITTANCE CR - SKN INCOMING SKN CENAIDJA/AP. ELLIS SR KUR BG APR 200505VRM2393361			85,000,000.00	-2,915,000,000.00
05/05	05/05	BILL PAYMENT TO LOAN TO:10028402970000002 FR:800105174400 200505VDEP321095		84,301,369.58		-2,999,301,369.58
06/05	06/05	REMITTANCE CR - RTGS INCOMING RTGS CENAIDJA/WIBOWO SUNIT OR HERMAN jng lfs 200506VRM5405749			580,000,000.00	-2,419,301,369.58
08/05	08/05	REMITTANCE DR - RTGS BANK CENTRAL ASIA SRI KUNDARI YULIATI 0173888870 BDS 2005082573000014		300,000,000.00		-2,719,301,369.58
08/05	08/05	SWEEP FROM Deposit TRF FR WIBOWO PUTRO MANDIRI 200508VDEP306490			438,947,118.00	-2,280,354,251.58

Terima kasih atas kesetiaan dan kepercayaan Anda bersama CIMB Niaga
Thank you for your continued loyalty and trust with CIMB Niaga

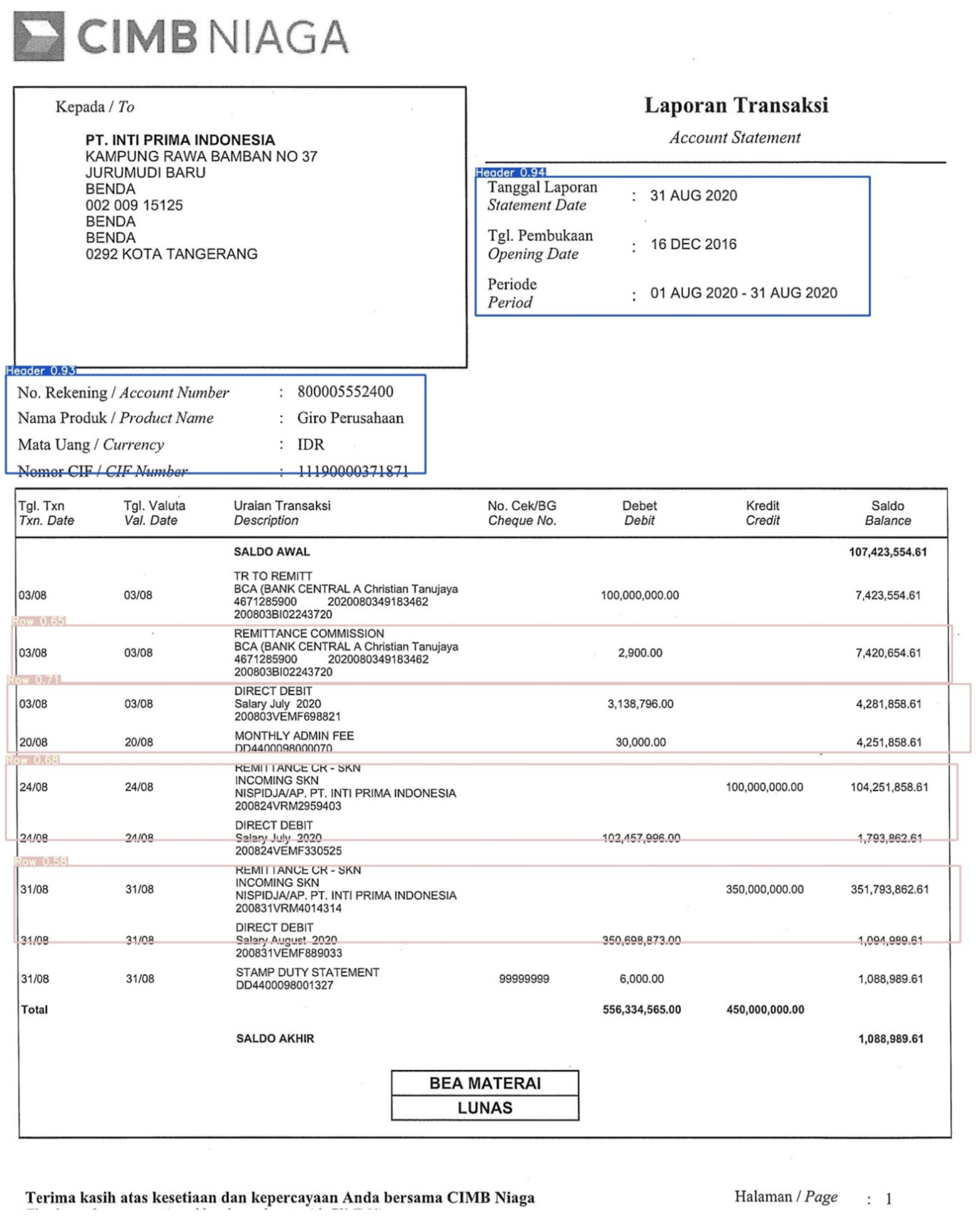
Halaman / *Page* : 1

Figure 5.14: Experiment 1 detection results



	Txn. Date	Val. Date	Description	Cheque No.	Debit Credit	Balance
0	<NA>	<NA>	SALDO AWAL	<NA>	<NA>	-2,957,634,702.91
1	01/05	01/05	BILL PAYMENT TO LOAN TO:10028402970000002 FR:800105174400 NOSN DFEFP126	<NA>	4,761,130.42	-3,000,000,000.00
2	05/05	05/05	REMITTANCE CR - SKN INCOMING SKN CENAIDJA/AP. ELLIS SR KUR BG APR	<NA>	85,000,000.00	-2,915,000,000.00
3	05/05	05/05	BILL PAYMENT TO LOAN TO:10028402970000002 FR:800105174400 200505VDEP321095	<NA>	84,301,369.58	-2,999,301,369.58
4	<NA>	06/05	REMITTANCE CR - RTGS INCOMING RTGS CENAIDJA/WIBOWO SUNIT OR HERMAN jng lfs 200506VRM5405749	<NA>	580,000,000.00	-2,419,301,369.58
5	08/05	08/05	REMITTANCE DR - RTGS BANK CENTRAL ASIA SRI KUNDARI YULIATI 0173888870 BDS 00508 000014	<NA>	300,000,000.00	-2,719,301,369.58
6	08/05	08/05	SWEEP FROM Deposit TRF FR WIBOWO PUTRO MANDIRI 200508VDEP306490	<NA>	438,947,118.00	-2,280,354,251.58

Figure 5.15: Experiment 1 extraction results


Header 0.94
Header 0.93
Row 0.65
Row 0.71
Row 0.68
Row 0.58

CIMB NIAGA

Kepada / *To*

PT. INTI PRIMA INDONESIA
KAMPUNG RAWA BAMBAN NO 37
JURUMUDI BARU
BENDA
002 009 15125
BENDA
BENDA
0292 KOTA TANGERANG

Laporan Transaksi
Account Statement

Tanggal Laporan *Statement Date*	:	31 AUG 2020
Tgl. Pembukaan *Opening Date*	:	16 DEC 2016
Periode *Period*	:	01 AUG 2020 - 31 AUG 2020

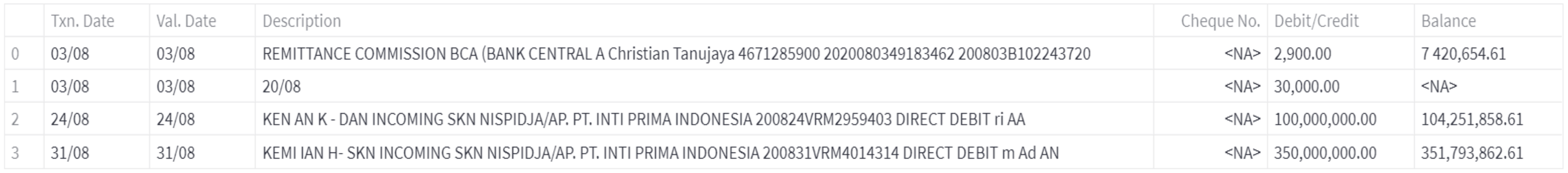
No. Rekening / *Account Number*	:	800005552400
Nama Produk / *Product Name*	:	Giro Perusahaan
Mata Uang / *Currency*	:	IDR
Nomor CIF / *CIF Number*	:	11190000371871

Tgl. Txn *Txn. Date*	Tgl. Valuta *Val. Date*	Uraian Transaksi *Description*	No. Cek/BG *Cheque No.*	Debet *Debit*	Kredit *Credit*	Saldo *Balance*
		SALDO AWAL				**107,423,554.61**
03/08	03/08	TR TO REMITT BCA (BANK CENTRAL A Christian Tanujaya 4671285900 2020080349183462 200803BI02243720		100,000,000.00		7,423,554.61
03/08	03/08	REMITTANCE COMMISSION BCA (BANK CENTRAL A Christian Tanujaya 4671285900 2020080349183462 200803BI02243720		2,900.00		7,420,654.61
03/08	03/08	DIRECT DEBIT Salary July 2020 200803VEMF698821		3,138,796.00		4,281,858.61
20/08	20/08	MONTHLY ADMIN FEE DD4400098000070		30,000.00		4,251,858.61
24/08	24/08	REMITTANCE CR - SKN INCOMING SKN NISPIDJA/AP. PT. INTI PRIMA INDONESIA 200824VRM2959403			100,000,000.00	104,251,858.61
24/08	24/08	DIRECT DEBIT Salary July 2020 200824VEMF330525		102,457,996.00		1,793,862.61
31/08	31/08	REMITTANCE CR - SKN INCOMING SKN NISPIDJA/AP. PT. INTI PRIMA INDONESIA 200831VRM4014314			350,000,000.00	351,793,862.61
31/08	31/08	DIRECT DEBIT Salary August 2020 200831VEMF889033		350,698,873.00		1,094,989.61
31/08	31/08	STAMP DUTY STATEMENT DD4400098001327	99999999	6,000.00		1,088,989.61
Total				**556,334,565.00**	**450,000,000.00**	
		SALDO AKHIR				**1,088,989.61**

BEA MATERAI
LUNAS

Terima kasih atas kesetiaan dan kepercayaan Anda bersama CIMB Niaga Halaman / *Page* : 1

Figure 5.16: Experiment 2 detection results



	Txn. Date	Val. Date	Description	Cheque No.	Debit/Credit	Balance
0	03/08	03/08	REMITTANCE COMMISSION BCA (BANK CENTRAL A Christian Tanujaya 4671285900 2020080349183462 200803B102243720	<NA>	2,900.00	7 420,654.61
1	03/08	03/08	20/08	<NA>	30,000.00	<NA>
2	24/08	24/08	KEN AN K - DAN INCOMING SKN NISPIDJA/AP. PT. INTI PRIMA INDONESIA 200824VRM2959403 DIRECT DEBIT ri AA	<NA>	100,000,000.00	104,251,858.61
3	31/08	31/08	KEMI IAN H- SKN INCOMING SKN NISPIDJA/AP. PT. INTI PRIMA INDONESIA 200831VRM4014314 DIRECT DEBIT m Ad AN	<NA>	350,000,000.00	351,793,862.61

Figure 5.17: Experiment 2 extraction results

<u>Table Row</u>

From the tables and figures above, we can see that there are more table rows can be detected in native PDF and most of the rows are also detected correctly. Meanwhile, the results of non-native PDF are weaker compared to native PDF as the detected rows are not many which does not even reach the half of the actual number and the detected rows are also not so accurate where there are some detected rows that have actually detected more than 1 row in original so it treats 2 rows as 1 row while detecting.

<u>Table Header</u>

From the tables and figures above, surprisingly we can see that there is no table header detected for both native and non-native PDFs. This may be due to the fact of insufficient training data.

<u>Header</u>

From the tables and figures above, we can see that both native and non-native PDFs perform quite well on the header detection where the system is able to locate the position of headers quite well in both native and non-native PDFs.

5.6 Evaluation and Discussion

As shown in Section 5.5, it is obvious that the current approach is capable of carrying out multiple detections on the region of interest within a PDF file. At least the framework is able to locate and draw a bounding box on the parts that the user wants to extract even though sometimes the accuracy is not so high but I believe that if we can acquire more quality data for training, the accuracy in detecting the content correctly will be improved highly.

However, there is 1 big limitation in using this approach to do extraction. This may cause you big trouble even though you achieved very good results in detection. The limitation is that it is very difficult to identify which column that the data in each row should belong to. This is because sometimes the number of data contained in different rows are different. For example, let say we are having a table columns of **date, description, chequeNo, debit, credit** and **balance**, the first row that we detected contains only 2 data which parks under **description** and **balance** while the second row contains 5 data which parks under **date, description, chequeNo, debit** and **balance**, we may put the first row's **description** data under **date column** as the **description** data is the first data in this row while detecting and the **balance** data may be put under **description** column. Meanwhile, the **balance** data in the second row may be put under the credit **column** as the second row does not have **credit** data which causes the fifth data to be replaced with the **balance** data that actually is sixth data.

In addition, this framework is workable for native and non-native PDF files. From section 5.5, we can see that the overall performance on native PDF is better than non-native PDF. The reason why the non-native PDFs do not always achieve better results is because they are required to pre-process before doing OCR. The non-native PDFs are likely to have a lot of noise and sometimes the contents are skewed. Therefore, the Gaussian Blur and Median Blur might be appropriate to be utilised to remove the noises as well as the deskew approach can be utilised to resolve the content offset and become uniform. As a result, these 2 limitations can be treated as part of my future work for the improvement to this framework to make it more reliable and applicable.

5.7 Highlight of Contribution and Conclusion

In this paper, I proposed and implemented a Table OCR Extraction Framework, a framework that is built using the Yolov5 deep learning model to be able to detect the desired contents in the PDF bank statement file. I reduced the row detection problem in a table and developed a system that is able to detect the rows in each table by using the trained deep learning model. After the detection, the system will crop all the detections out as a single image for later OCR purposes in which the process will help to extract the row data out as much as possible. The experiment in this paper mainly uses the CIMB bank statement PDF file for analysis and evaluation. Besides, the overall workflow is elucidated in this paper as well as several diagrams are provided for better understanding such as Layered Diagram, System Block Diagram, Data Flow Diagram and Flowcharts.