



UTM
UNIVERSITI TEKNOLOGI MALAYSIA

SMJE 4383 ADVANCE PROGRAMMING

20222023/1

ASSIGNMENT 2

SCREEN SCRAPING & OCR TEXT RECOGNITION USING PYTHON SCRIPT

AP. IR. DR. ZOOL HILMI ISMAIL

Link : <https://github.com/maathirah/ASSIGNMENT>

AMIRAH BATRISYIA BINTI AMINUDDIN A19MJ0011

MA ATHIRAH BINTI SAPAWI A19MJ3030

SECTION 01

INTRODUCTION

Screen scraping and OCR recognition are both techniques used to extract data from images or text that are displayed on a computer screen.

Screen scraping refers to the process of extracting data from a website or application by automatically simulating the actions of a human user, such as clicking buttons or filling in forms, and then parsing the resulting HTML or XML code to extract the desired information. Screen scraping can be done using specialized software or programming libraries.

OCR (Optical Character Recognition) recognition, on the other hand, is a technology that enables computers to recognize printed or handwritten text that is contained within an image or a scanned document. OCR software can analyse the shapes of individual characters and convert them into machine-readable text that can be edited, searched, or analysed.

While screen scraping is used to extract information from web pages or applications, OCR recognition is typically used to extract text from scanned documents, such as invoices, receipts, or contracts, or from images containing text, such as product labels or street signs.

Both screen scraping and OCR recognition can be powerful tools for automating data entry, data analysis, and other tasks that require the extraction of information from digital sources. However, it is important to use these techniques in a responsible and ethical manner, respecting any relevant copyright or privacy laws, and avoiding any activity that could be seen as malicious or abusive.

OBJECTIVE

The objective of the assignment is to develop a solution using Screen Scraping and OCR Text Recognition in Python to extract the required information from the images.

METHODOLOGY

This code is an example of how to extract text from an image and save it in a text file using the Tesseract OCR engine, OpenCV, and Python. The steps involved in this code are:

1. The code starts by importing necessary libraries/modules including the Python Imaging Library (PIL), pytesseract, OpenCV (cv2), and numpy.
2. It reads the image using OpenCV imread() function and converts it into grayscale image using cvtColor() function.
3. Then it converts the grayscale image into a binary image using thresholding.
4. The binary image is then displayed using the cv2.imshow() function.
5. The code then configures the parameters for the Tesseract OCR engine and feeds the binary image to pytesseract.image_to_data() function to obtain information about the text within the image.
6. The OCR output information is stored in a dictionary named 'details'. The keys of this dictionary represent different attributes of the text detected by the OCR engine.
7. A loop is run over the total number of text boxes in the OCR output. A text box is essentially a region in the image where the OCR engine has detected some text.
8. The loop checks for a confidence score for the text box (represented by the 'conf' key in the 'details' dictionary) and if the score is greater than 30, then it extracts the coordinates of the text box from the 'left', 'top', 'width', and 'height' keys of the 'details' dictionary.
9. A rectangle is then drawn around the text box in the binary image using the cv2.rectangle() function.
10. The binary image with the rectangle around the text is then displayed using the cv2.imshow() function.
11. The code then extracts the text from the OCR output and stores it in a list called 'parse_text'. It does so by looping over the 'text' key of the 'details' dictionary and appending each word to a list called 'word_list' until it encounters a space (' ') or the last word. It then appends the 'word_list' to 'parse_text' and starts a new 'word_list' for the next set of words.

12. Finally, the text extracted from the image is written to a text file named 'result_text.txt' using the `csv.writer()` function.

This code uses the PIL and cv2 libraries to read and process an image, pytesseract to perform optical character recognition (OCR) on the image, and csv to write the recognized text to a CSV file.

First, the Image module from PIL and the Output module from pytesseract are imported.

Then, an image is read using the `cv2.imread()` function, and converted to grayscale using `cv2.cvtColor()` with `cv2.COLOR_BGR2GRAY` as the second argument. The resulting grayscale image is then thresholded to obtain a binary image using `cv2.threshold()`.

The resulting thresholded image is then displayed using `cv2.imshow()`, and the user is prompted to press any key to continue using `cv2.waitKey()`. Finally, the window displaying the image is closed using `cv2.destroyAllWindows()`.

The OCR process begins by defining a custom configuration for Tesseract using the `custom_config` variable. Then, `pytesseract.image_to_data()` is called with the thresholded image, `Output.DICT` as the `output_type` argument, the custom configuration, and 'eng' as the `lang` argument. This returns a dictionary containing information about the recognized text.

The code then extracts the total number of bounding boxes from the dictionary, and loops over each bounding box. If the confidence level of the bounding box is greater than 30, the code retrieves the coordinates and dimensions of the bounding box, and draws a rectangle around it on the thresholded image using `cv2.rectangle()`.

After all the bounding boxes have been processed, the code extracts the recognized text from the dictionary and puts it into a 2D list called `parse_text`, where each inner list represents a line of text and each element in the inner list represents a word.

Finally, the code uses `csv.writer()` to write the contents of `parse_text` to a CSV file named `result_text.txt`, with spaces as the delimiter. The resulting file is automatically closed when the `with` block is exited.

RESULT AND DISCUSSION

from PIL import Image

from pytesseract import Output

import numpy as np

import cv2

import pytesseract

The above lines of code import the necessary libraries for performing the OCR (Optical Character Recognition) task on an image.

image = cv2.imread('receipt_image.png')

This line reads the input image, which is assumed to be in PNG format and saved in the same directory as the script.

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

This line converts the input image to grayscale, which is a requirement for many OCR algorithms to work properly.

threshold_img = cv2.threshold(gray_image, 0, 255, cv2.THRESH_BINARY | cv2.THRESH_OTSU)[1]

This line applies a binary threshold to the grayscale image, which converts the image into a black and white format. This step is necessary because if you have a colored image, then Tesseract (OCR engine used here) won't be able to detect text correctly and this will give an incorrect result.

cv2.imshow('threshold image', threshold_img)

This line displays the thresholded image on the screen.

cv2.waitKey(0)

This line waits for a key event and keeps the output window open until the user presses a key.

cv2.destroyAllWindows()

This line closes all open windows.

custom_config = r'--oem 3 --psm 6'

This line sets the custom configuration options for Tesseract, which include the OCR engine mode (--oem) and page segmentation mode (--psm).

details = pytesseract.image_to_data(threshold_img, output_type=Output.DICT, config=custom_config, lang='eng')

This line extracts text from the thresholded image using Tesseract OCR engine. The output is returned in the form of a dictionary where each word is assigned a key-value pair, with the corresponding coordinates and size of the bounding box, as well as the confidence score for the OCR engine.

print(details.keys())

This line prints the keys of the dictionary returned by Tesseract OCR, which includes level, page_num, block_num, par_num, line_num, word_num, left, top, width, height, conf, and text.

total_boxes = len(details['text'])

This line calculates the total number of text boxes extracted from the image.

for sequence_number in range(total_boxes):

if int(details['conf'][sequence_number]) > 30:

(x, y, w, h) = (details['left'][sequence_number], details['top'][sequence_number], details['width'][sequence_number], details['height'][sequence_number])

This code loops through each text box and extracts its coordinates and dimensions only if the confidence score of the OCR engine is greater than 30.

threshold_img = cv2.rectangle(threshold_img, (x, y), (x + w, y + h), (0, 255, 0), 2)

This code adds a green rectangle around each detected text box to highlight it in the image.

cv2.imshow('captured text', threshold_img)

This line displays the image with bounding boxes around the detected text.

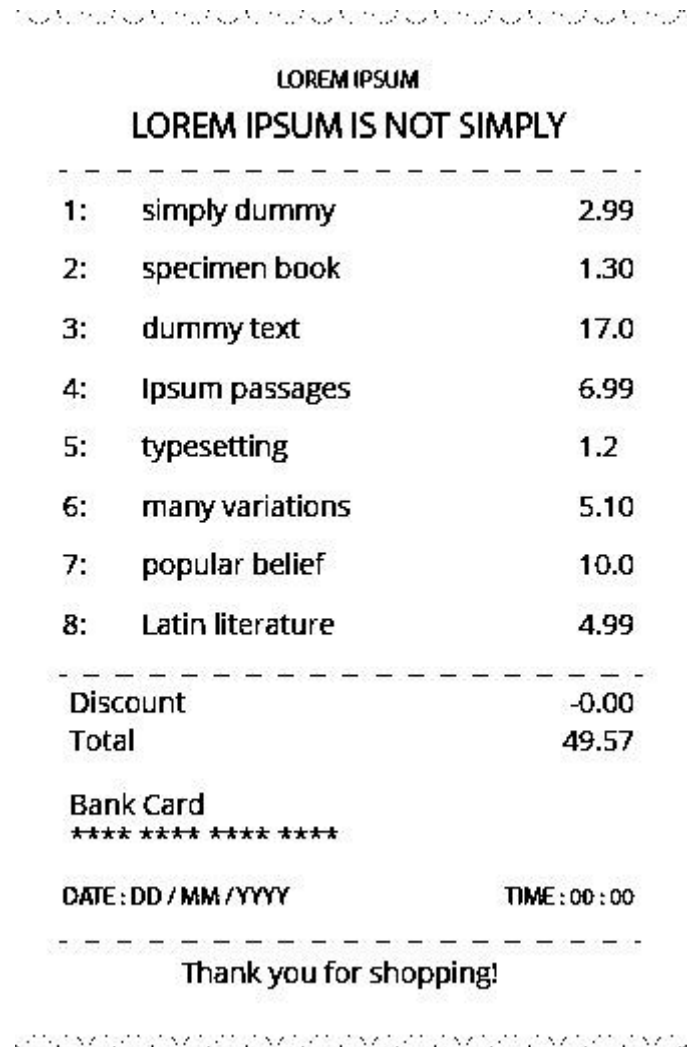
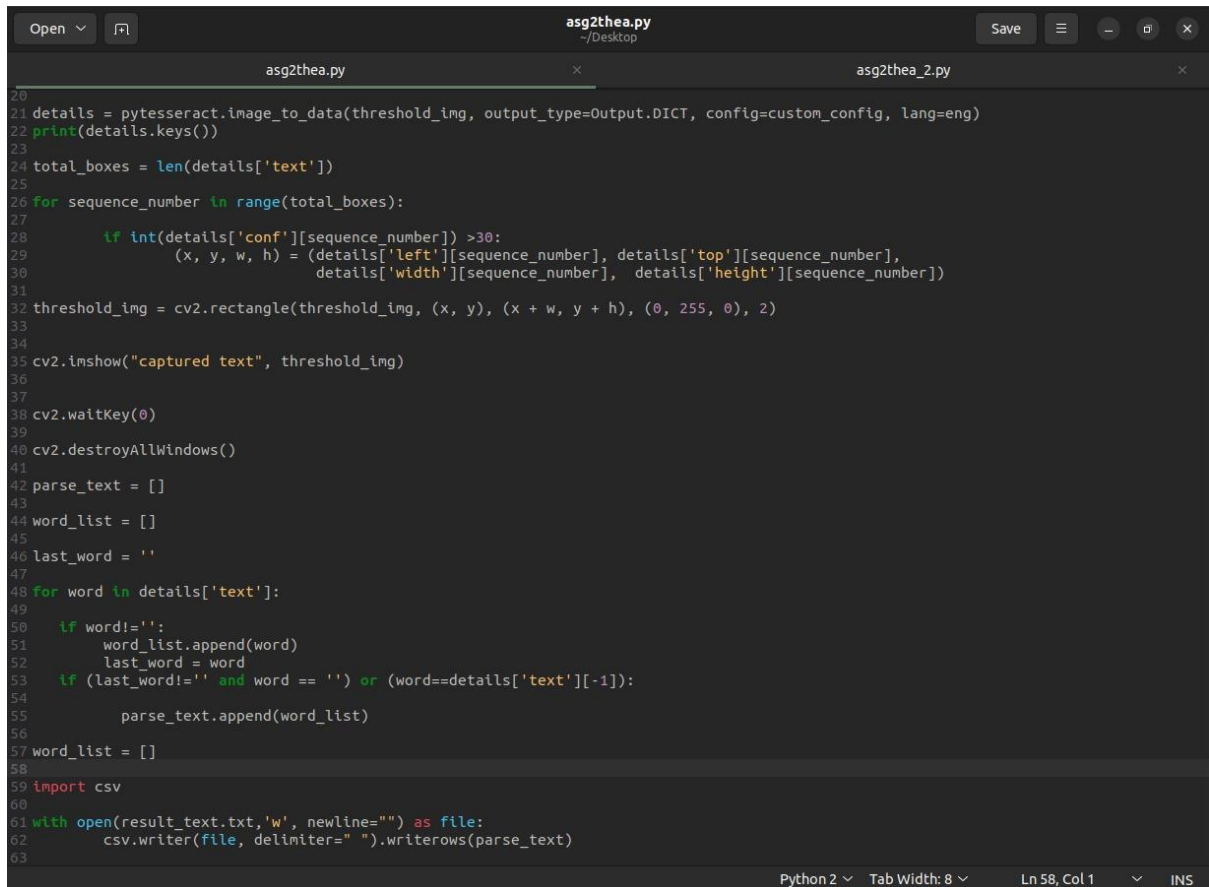


Figure 1 : input receipt image



```
20
21 details = pytesseract.image_to_data(threshold_img, output_type=Output.DICT, config=custom_config, lang=eng)
22 print(details.keys())
23
24 total_boxes = len(details['text'])
25
26 for sequence_number in range(total_boxes):
27     if int(details['conf'][sequence_number]) > 30:
28         (x, y, w, h) = (details['left'][sequence_number], details['top'][sequence_number],
29                         details['width'][sequence_number], details['height'][sequence_number])
30
31 threshold_img = cv2.rectangle(threshold_img, (x, y), (x + w, y + h), (0, 255, 0), 2)
32
33
34
35 cv2.imshow("captured text", threshold_img)
36
37
38 cv2.waitKey(0)
39
40 cv2.destroyAllWindows()
41
42 parse_text = []
43
44 word_list = []
45
46 last_word = ''
47
48 for word in details['text']:
49     if word != '':
50         word_list.append(word)
51         last_word = word
52     if (last_word != '' and word == '') or (word == details['text'][-1]):
53
54         parse_text.append(word_list)
55
56
57 word_list = []
58
59 import csv
60
61 with open(result_text.txt, 'w', newline='') as file:
62     csv.writer(file, delimiter=" ").writerows(parse_text)
63
```

Python 2 ▾ Tab Width: 8 ▾ Ln 58, Col 1 ▾ INS

Figure 2: Code

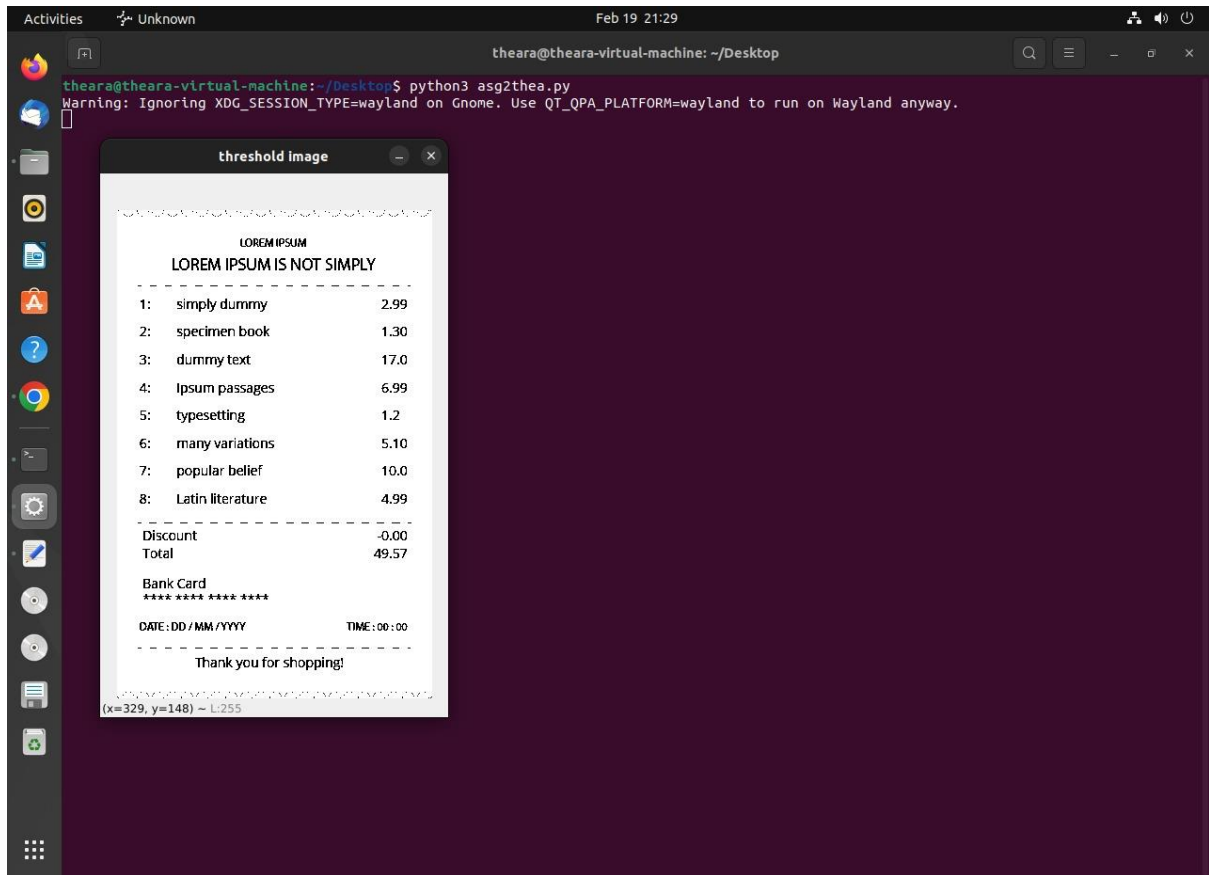


Figure 3 : When run

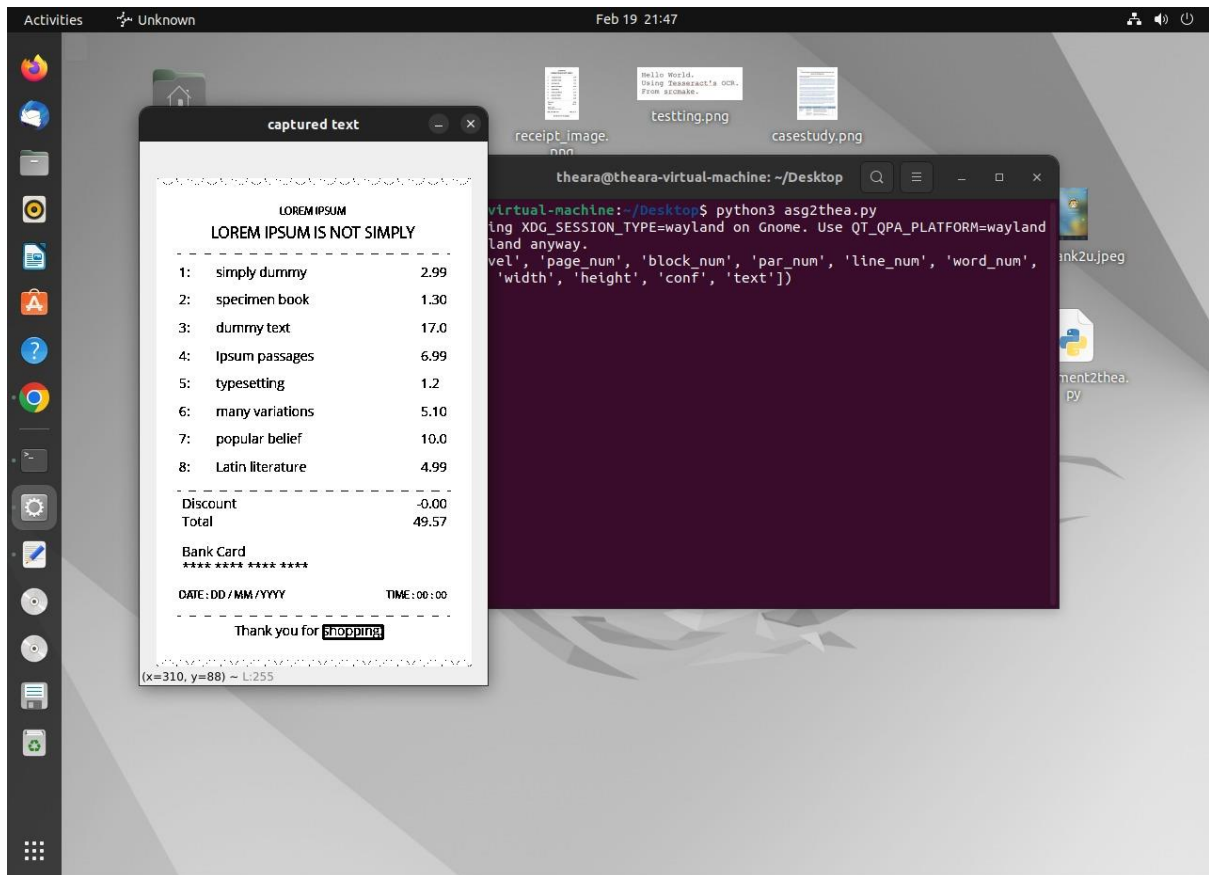


Figure 4 : Captured text

LOREM PSUM

LOREM PSU MIS NOT SIMPLY

1

simply dummy

2.99

2

specimen book

1.30

3

dummy text

17.0

4

psum passages

6.99

5

typesetting

1.2

6

many variations

6.10

7

popular belief

10.0

8

latin literature

4.99

Discount

10.00

Total

49.57

Bar < Card

DATE: DD / MM / YYYY

TIME: 00:00

I thank you for shopping

Figure 5: Highlighted text and numbers

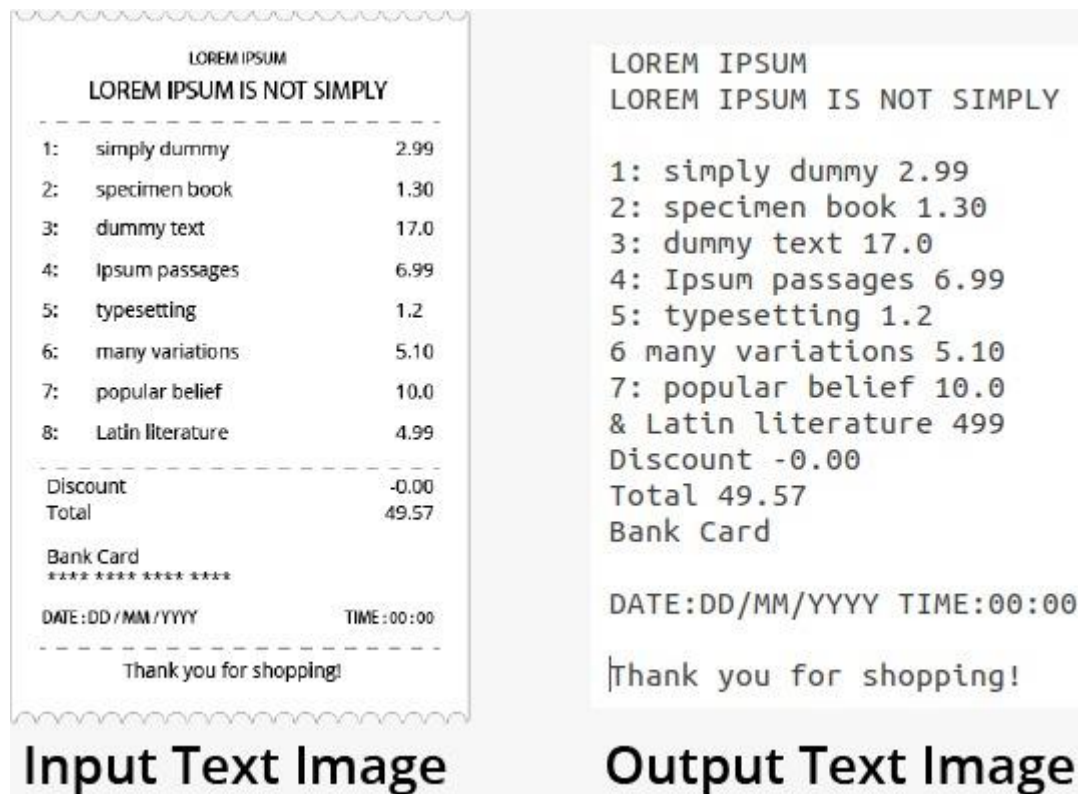


Figure 6: Input and Output image