Shanney Suhendra
Role: Data Science Intern at Colearn
Duration: May 30 to Aug 26, 2022

Table of Contents

OCR - Optical Character Recognition

Definition
Process of converting an image of texts into machine-readable text format.

How it Works

1. *Image acquisition*
   Image analyzed for dark and light areas. and the dark areas are identified as characters that need to be recognized, while light areas are identified as background.
2. *Preprocessing*
   Clean image data for model input to reduce complexity.
3. *Text recognition*
   Pattern recognition → Each character compared with various examples of characters in different fonts and formats that are fed to the program.
   Feature detection → program applies rules regarding features of a specific letter/number to recognize characters; num of curves, angled lines. For example, the capital letter “A” is stored as two diagonal lines that meet with a horizontal line across the middle.

Benefits/Advantages
Automate workflows, turn read-only files to editable text, create audible files, translate foreign languages.

Use in CoLearn
Question image uploaded and processed, explanation videos with similar question/topic given as result.

Engines Experimented

1. Tesseract OCR
2. Easy OCR

Language/Environment: Google Colab, Python

Tesseract OCR

Objective

Extract texts from 100 sample images using Tesseract. Explore different methods of preprocessing images and comparing results with the raw image. For this project, only 1 image was used initially for the exploration of preprocessing methods and the best model found is then used for all other images.

What is Tesseract OCR

It is an open source OCR engine, compatible with many programming languages and frameworks through wrappers. Its algorithm utilizes the LSTM (long short term memory) model, which is a form of RNN (recurrent neural network). The model supports deep learning, a subset of machine learning based on neural networks that permit machines to train itself to perform a task.

*CNN (convolutional neural network) is typically used to recognize a single character in an image but problems when faced with an image containing text of arbitrary length are solved using RNNs.

Tesseract OCR has its own limitations, such as it can't recognize handwritten text and they're not always good at analyzing the natural reading order of documents.

Steps

1. Import pytesseract and necessary packages
2. Extract text from raw image
3. Preprocess image and compare to raw image
4. Extract texts from different preprocessing image
5. Compare results and conclude best preprocessing method (or none)
6. Extract text from all 100 image using best model found

Step 1&2

Install pytesseract and import necessary packages. All code is implemented in Google Colaboratory.

```
!pip install pytesseract
!sudo apt install tesseract-ocr

# import the necessary packages
from PIL import Image
import pytesseract
import argparse
import cv2
import os
from skimage import io
import numpy as np
```

Image used is in URL and will be read using the skimage package. Before preprocessing the image, we extract text from the raw image itself (fig 1). Extracted text is shown on table 1.

```
from pytesseract import Output
from google.colab.patches import cv2_imshow

img = io.imread('https://coln-prd-sg-s3-ads-pub.s3-ap-southeast-1.amazonaws.com/images/questions/M0280781P018.png')
d = pytesseract.image_to_data(img, output_type=Output.DICT)
print(img.shape)
cv2_imshow(img)
print(d["text"])
```

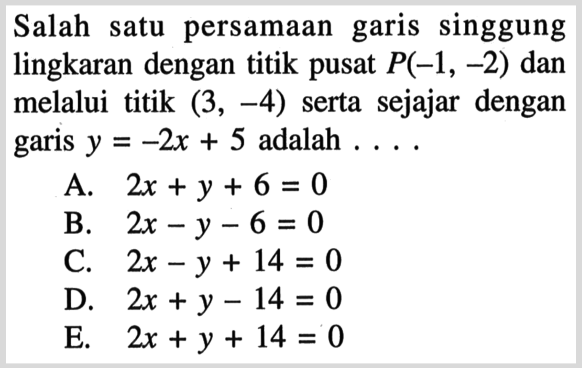
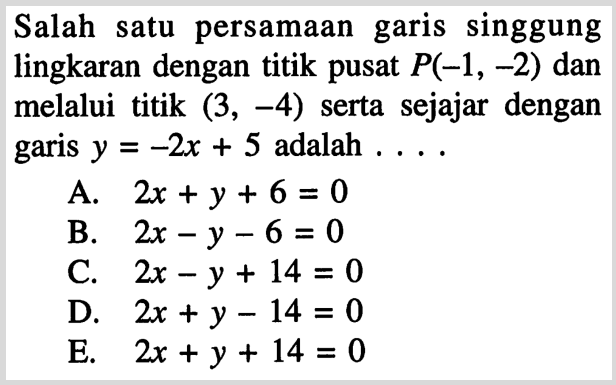


Fig 1. raw image

Step 3

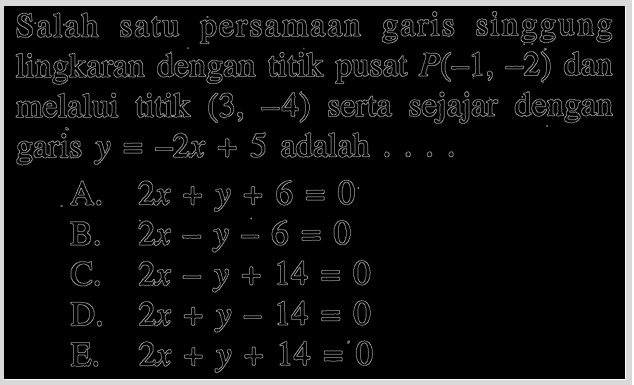
Preprocessing is used to increase the accuracy of the model and reduce its complexity. Multiple preprocessing functions were created, as shown below, for exploration to see which methods appropriately pre-process the image. Resulting images from the following preprocessing functions were plotted as shown on fig 2-7. Extracted text from each of the pre-processed images are shown on table 1.

```
# get grayscale image
def get_grayscale(image):
    return cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

#thresholding
def thresholding(image):
    return cv2.threshold(image, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)[1]

#canny edge detection
def canny(image):
    return cv2.Canny(image, 100, 200)

#opening - erosion followed by dilation
def opening(image):
    kernel = np.ones((5,5),np.uint8)
    return cv2.morphologyEx(image, cv2.MORPH_OPEN, kernel)

#dilation
def dilate(image):
    kernel = np.ones((5,5),np.uint8)
    return cv2.dilate(image, kernel, iterations = 1)

#erosion
def erode(image):
    kernel = np.ones((5,5),np.uint8)
    return cv2.erode(image, kernel, iterations = 1)

# noise removal
def remove_noise(image):
    return cv2.medianBlur(image,5)
```


Salah satu persamaan garis singgung lingkaran dengan titik pusat $P(-1, -2)$ dan melalui titik $(3, -4)$ serta sejajar dengan garis $y = -2x + 5$ adalah . . . .

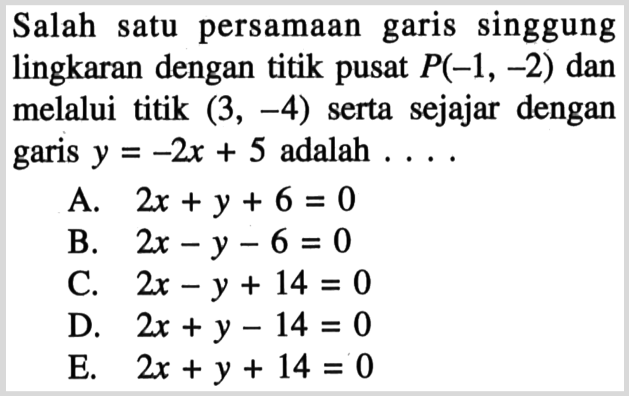
A. $2x + y + 6 = 0$
B. $2x - y - 6 = 0$
C. $2x - y + 14 = 0$
D. $2x + y - 14 = 0$
E. $2x + y + 14 = 0$

Fig 2. thresholding (get_grayscale → thresholding)


Salah satu persamaan garis singgung lingkaran dengan titik pusat $P(-1, -2)$ dan melalui titik $(3, -4)$ serta sejajar dengan garis $y = -2x + 5$ adalah . . . .

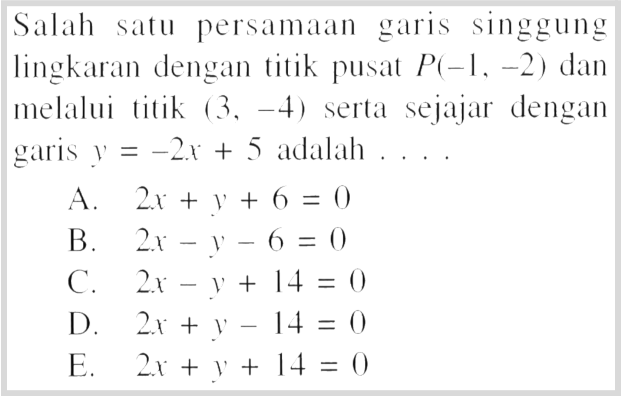
A. $2x + y + 6 = 0$
B. $2x - y - 6 = 0$
C. $2x - y + 14 = 0$
D. $2x + y - 14 = 0$
E. $2x + y + 14 = 0$

Fig 3. canny-edge


Salah satu persamaan garis singgung lingkaran dengan titik pusat $P(-1, -2)$ dan melalui titik $(3, -4)$ serta sejajar dengan garis $y = -2x + 5$ adalah . . . .

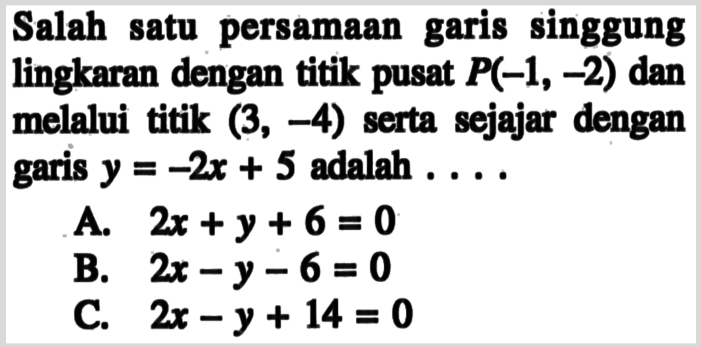
A. $2x + y + 6 = 0$
B. $2x - y - 6 = 0$
C. $2x - y + 14 = 0$
D. $2x + y - 14 = 0$
E. $2x + y + 14 = 0$

Fig 4. opening


Salah satu persamaan garis singgung lingkaran dengan titik pusat $P(-1, -2)$ dan melalui titik $(3, -4)$ serta sejajar dengan garis $y = -2x + 5$ adalah . . . .

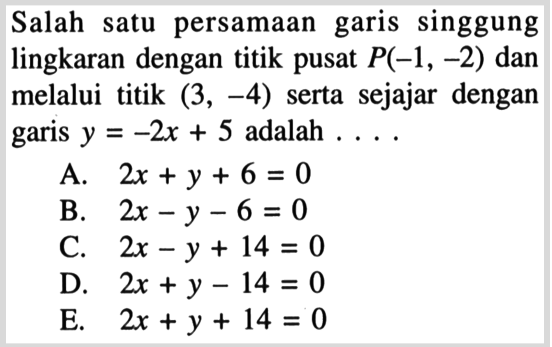
A. $2x + y + 6 = 0$
B. $2x - y - 6 = 0$
C. $2x - y + 14 = 0$
D. $2x + y - 14 = 0$
E. $2x + y + 14 = 0$

Fig 5. dilate


Salah satu persamaan garis singgung lingkaran dengan titik pusat $P(-1, -2)$ dan melalui titik $(3, -4)$ serta sejajar dengan garis $y = -2x + 5$ adalah . . . .

A. $2x + y + 6 = 0$
B. $2x - y - 6 = 0$
C. $2x - y + 14 = 0$

Fig 6. erode


Salah satu persamaan garis singgung lingkaran dengan titik pusat $P(-1, -2)$ dan melalui titik $(3, -4)$ serta sejajar dengan garis $y = -2x + 5$ adalah . . . .

A. $2x + y + 6 = 0$
B. $2x - y - 6 = 0$
C. $2x - y + 14 = 0$
D. $2x + y - 14 = 0$
E. $2x + y + 14 = 0$

Fig 7. noise removal

**Short Analysis on Pre-processed Images

Fig 2 → Texts a little bolder than raw image, almost no change.

Fig 3 → Edges of each alphabet, numbers, and punctuations detected.

Fig 4 → Blur applied on image. We can zoom in and further compare it with the raw image (fig 8).

Fig 5 → boundaries of regions of foreground pixels (white pixels/area) Enlarged. Text now looks less bold and outline of text less clear.

Fig 6 → essentially the opposite of fig 5, boundaries of regions of foreground pixels (white pixels/area) reduced. Text now looks more bold.

Fig 7 → image is blurred, very small change.

Fig 8.

Step 4&5

pre-processing functions	extracted text
raw image	Salah satu persamaan garis singgung lingkaran dengan titik pusat P(—1, —2) dan melalui titik (3, —4) serta sejajar dengan garis y = —2x + 5 adalah.... A. 2x+y+6=0 B. 2x-y-6=0 C. 2x-y+14=0 D. 2x+y-14=0 E. 2x+y+14=0
thresholding	Salah satu persamaan garis singgung lingkaran dengan titik pusat P(—1, —2) dan melalui titik (3, —4) serta sejajar dengan garis y = —2x + 5 adalah.... A. 2x+y+6=0 B. 2x-y-6=0 C. 2x-y+14=0 D. 2x+y-14=0 E. 2x+y+14=0
canny-edge	Salah satu persamaan garis singgung lingkaren dengan titik pusat P(-1, -—2) dem melalui titik (3, —4) serta sejajar dengan garis y= 2x +5 adalah.... a On) a Ora) 2x-yrl4s0 ax + y= l4=0 a Ae) ikon.
opening	Salah satu persamaan garis singgung lingkaran dengan titik pusat P(—1, —2) dan melalui titik (3, —4) serta sejajar dengan garis y= —2x + 5 adalah.... 2x+y+6=0 2x-y-6=0 2x-y+14=0 2x+y-14=0 2x+y+14=0 MOO >
dilate	Salah satu persamaan garis singgung lingkaran dengan tik pusat P(—1, —2) dan melalur Gtk (3, serta sejajar dengan garis y = —2x + 5 adalah... . vt yt+60=0 2v-y-6=0 2vx-yt 14 =0 d2v+y- 14 =0 ov +y+ 14 =0 TOO YS - S
erode	Salah satu persamaan garis singgung lingkaran dengan titik pusat P(—1, -2) dan melalui titik (3, —4) serta sejajar dengan garis y = —2x + 5 adalah.... 2ax+y+6=0 2x-y-6=0 2x-y+14=0 2x+y-—14=0 2x+y+14=0 HOW p>
noise removal	Salah satu persamaan garis singgung lingkaran dengan titik pusat P(—1, —2) dan melalui titik (3, —4) serta sejajar dengan garis y= —2x + 5 adalah.... A. 2x+y+6=0 B. 2x-y-6=0 C. 2x-y+14=0 D. 2x+y-14=0 E. 2x+y+14=0

Table 1. texts extracted

**Analysis

Comparing the results manually, we can see that the text extracted from the raw image, thresholding image, and noise removal image produced the same output and is most accurate to the text read from the raw image. As a result, I decided to use the raw image as the input to the OCR engine for text extraction, with no pre-processing done.

Step 6

After the best model is found, I extracted text from all 100 images using a loop as shown below (links is a list of the 100 URL images). Afterwards, I paired each of the extracted text with their corresponding image link using the zip function.

```
img = []
for i in links:
  img.append(io.imread(i))

text = []
for j in img:
  t = pytesseract.image_to_data(j, output_type=Output.DICT)
  text.append(t["text"])
```

```
result = list(zip(links, text))
```

Utilizing pandas, I created a table for the result, with the first column being the URL link images and the second column being the respective extracted texts. The table was then downloaded to a csv file.

```
import pandas as pd
resultTable = pd.DataFrame(result)
resultTable
```

	0	1
0	https://coln-prd-sg-s3-ads-pub.s3-ap-southeast...	Salah satu persamaan garis singgung lingkaran...
1	https://coln-prd-sg-s3-ads-pub.s3-ap-southeast...	Tentukan persamaan garis yang bergradien 3 da...
2	https://coln-prd-sg-s3-ads-pub.s3-ap-southeast...	Tentukan himpunan penyelesaian persamaan 25% ...
3	https://coln-prd-sg-s3-ads-pub.s3-ap-southeast...	Hitunglah setiap limit berikut. im ———— x90 ...
4	https://coln-prd-sg-s3-ads-pub.s3-ap-southeast...	Nilai lim (vox? + 18x -— 2.017 + V4x? — 20x +...
...	...	...
95	https://coln-prd-sg-s3-ads-pub.s3-ap-southeast...	Dua buah koin dilempar bersama sebanyak 60 ka...
96	https://coln-prd-sg-s3-ads-pub.s3-ap-southeast...	Perhatikan gambar berikut. AABC dan ADBE seban...
97	https://coln-prd-sg-s3-ads-pub.s3-ap-southeast...	Sebelum pelajaran olahraga dimulai, Pak Guru m...
98	https://coln-prd-sg-s3-ads-pub.s3-ap-southeast...	Nyatakan dalam bentuk pecahan campuran. 420 22
99	https://coln-prd-sg-s3-ads-pub.s3-ap-southeast...	Pak Rahman memiliki kebun sayur yang berbentu...

100 rows × 2 columns

```
resultTable.to_csv("linkstext.ipynb", index=False)
```

Easy OCR

Objective

Extract texts from 100 sample images using Easy OCR. Explore different methods of preprocessing images and comparing results with the raw image. For this project, only 1 image was used initially for the exploration of preprocessing methods and the best model found is then used for all other images.

Steps

1. Install easyocr and necessary packages
2. Extract text from raw image
3. Preprocess image and compare to raw image
4. Extract texts from different preprocessing image
5. Compare results and conclude best preprocessing method (or none)
6. Extract text from all 100 image using best model found
7. Additional experimentation - reader language
8. Post-processing of text - bounding box (part 1)
9. Post-processing of text - bounding box (part 2)

Step 1

Install pytesseract and import necessary packages. All code is implemented in Google Colaboratory.

```
pip install easyocr

import easyocr
from easyocr import Reader
import argparse
import cv2
```

Before preprocessing the image, we extract text from the raw image itself. Extracted text is shown on table 1.

```
link = "https://coln-prd-sg-s3-ads-pub.s3-ap-southeast-1.amazonaws.com/images/questions/M0280781P018.png"

img = cv2.imread(link)
reader = easyocr.Reader(['id'])
result = reader.readtext(link)

text = []
for i in result:
  text.append(i[1])
text = ' '.join(text)
text
```

Step 2&3

These two steps are the exact steps done in tesseract OCR above (page 2-3), since the preprocessing methods used are the same for both OCR.

Step 4&5

pre-processing functions	extracted text
raw image	Salah satu persamaan singgung lingkaran dengan titik pusat P(-I, ~2) dan melalui titik (3, -4) serta sejajar dengan y =-2x + 5 adalah A 2x + y + 6 = 0 B 2x -y - 6 = 0 C. 2x = y + 14 = 0 D 2x + y - 14 = 0 E 2x + y + 14 = 0 garis garis
thresholding	Salah satu persamaan garis singgung lingkaran dengan titik pusat P(-l, ~2) dan melalui titik (3, -4) serta sejajar dengan garis y = -2x + 5 adalah A 2x + y + 6 = 0 B 2x -y - 6 = 0 C. 2x - y + 14 = 0 D 2x + y - 14 = 0 E. 2x + y + 14 = 0
canny-edge	Salah satu persama@n sInggung Jingkaran dengan (itik puset P(-l, ~2) dan melalui (ilík (3, =4) serta sejaj|f dengan Y = -2x + S adalah A. 2x # y + 6 = 0 B. 2x ~y = 6 = C 2x = y + 14 = 0 D 2x + Y = 14 = 0 E. 22 + y # 14 =' 0 garis garis
opening	Salah satu persamaan singgung lingkaran dengan titik pusat P(-l, ~2) dan melalui titik (3, -4) serta sejajar dengan y = -2x + 5 adalah A 2x + y + 6 = 0 B 2x = y - 6 = 0 C. 2x = y + 14 = 0 D 2x + y = 14 = 0 E 2x + y + 14 = 0 garis garis

dilate	Salah satu persamaan singgung lingkaran dengan titik pusat P(-1. -2) dan melalui titik (3 , -4) serta sejajar dengan } = ~2r + 5 adalah A 2r + v + 6 = 0 B 21 =) = 6 = 0 C. 2r _ v + 14 = D. 2r + V _ 14 = 0 E. 21 +) + 14 = 0 garis garis
erode	Salah satu persamaan singgung lingkaran dengan titik pusat P(-1, -2) dan melalui titik (3, -~4) serta sejajar dengan y = -2x + 5 adalah A 2x + y + 6 = 0 B 2x -y-6 =0 C. 2x - y + 14 = 0 D 2x + y - 14 = 0 E 2x + y + 14 =0 garis garis
noise removal	Salah satu persamaan singgung lingkaran dengan titik pusat P(-I, ~2) dan melalui titik (3, -4) serta sejajar dengan y =-2x + 5 adalah A 2x + y + 6 = 0 B 2x -y - 6 = 0 C. 2x - y + 14 = 0 D 2x + y = 14 = 0 E 2x + y + 14 = 0 garis garis

**Analysis

Comparing the results manually, we can see that the text extracted from the thresholding image produced the same output and is most accurate to the text read from the raw image. A key difference between the thresholding output and all the other output is the word “garis”, where in the thresholding output they were found on the right positions, whereas both of the word “garis” in other outputs were found at the very end. As a result, I decided to use the thresholding image as the input to the OCR engine for text extraction, using the get_graycale and thresholding as the preprocessing functions.

Step 6

After the best model is found, I extracted text from all 100 images using a loop as shown below (links is a list of the 100 URL images). Afterwards, I paired each of the extracted text with their corresponding image link using the zip function.

```
img = []
for i in links:
  img.append(io.imread(i))

final = []

for j in range(len(img)):
  try:
    gray = get_grayscale(img[j])
    thresh = thresholding(gray)
    result = reader.readtext(thresh)
    text = []
    for k in result:
      text.append(k[1])
    text = ' '.join(text)
    final.append(text)
  except:
    print(links[j])
    final.append("error")
```

```
end = list(zip(links, final))
```

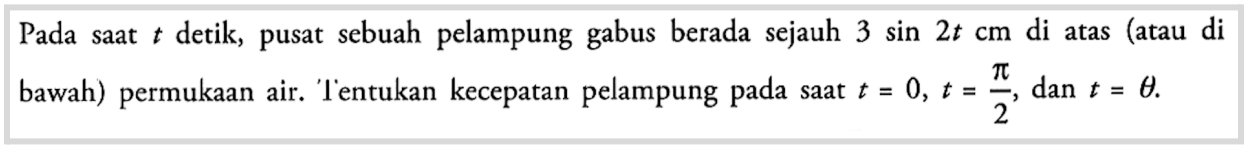
Utilizing pandas, I created a table for the result, with the first column being the URL link images and the second column being the respective extracted texts. The table was then downloaded to a csv file.

Step 7

Easy OCR has access to over 70+ languages. Looking at the code line below, reader = easyocr.Reader([‘id’]), ‘id’ means that we want to read the Indonesian language. If we want to detect English, we simply replace ‘id’ with ‘en’, and etc.

```
reader = easyocr.Reader(['id'])
result = reader.readtext(link)
```

The objective of this additional experiment is to see if the languages, ‘id’ or ‘en’, will produce different outputs, and ultimately, which language produces the better outputs. This experiment was done manually, wherein I used 5 sample images, and for each image I extracted its outputs using both easyocr.Reader([‘id’]) and easyocr.Reader([‘en’]), and comparing them. Below is an example of the comparison of 1 image:



‘id’

Pada saat detik, sebuah pelampung berada sejauh 3 sin 2t cm di atas (atau di T bawah) permukaan air. '[entukan kecepatan pelampung saat t = 0, t = dan t = 0. 2 gabus pusat pada

‘en’

Pada saat detik, sebuah pelampung berada sejauh 3 sin 2t cm di atas (atau di TC bawah) permukaan air: 'Tentukan kecepatan pelampung saat t = 0, t = dan t = 0. 2 gabus pusat pada

**Result Analysis

Overall there’s almost no difference, multiple different images used and all words were detected. Only minor differences mostly in spacing, capitalizations, etc.

Step 8

From the result that we found in step 5, we analyzed that there were many cases where words are extracted in the wrong order. We can fix this by utilizing what we call bounding boxes, which is an imaginary rectangle that is used to outline the object in a box. The reader.readtext(link) function produces 3 components for each bounding box: the (x, y) coordinates for each corner, the text itself, and its confidence.

```
result = reader.readtext(link)
print('')
result
```

```
Progress: |                                                  | 101.2% Complete
[([[24, 14], [232, 14], [232, 92], [24, 92]], 'Salah', 0.9999376247128333),
 ([[265, 23], [421, 23], [421, 91], [265, 91]], 'satu', 0.999876856803894),
 ([[447, 7], [849, 7], [849, 111], [447, 111]],
  'persamaan',
  0.7646343911269066),
 ([[1092, 6], [1443, 6], [1443, 119], [1092, 119]],
  'singgung',
  0.9999851642637401),
 ([[17, 101], [1439, 101], [1439, 210], [17, 210]],
  'lingkaran dengan titik pusat P(-I, ~2) dan',
  0.6208696135225676),
```

Salah satu persamaan garis singgung lingkaran dengan titik pusat $P(-1, -2)$ dan melalui titik $(3, -4)$ serta sejajar dengan garis $y = -2x + 5$ adalah . . . .

A. $2x + y + 6 = 0$
B. $2x - y - 6 = 0$
C. $2x - y + 14 = 0$
D. $2x + y - 14 = 0$
E. $2x + y + 14 = 0$

Hence, the objective of this post-processing of text is to sort the order of the words based on their bounding box coordinates.

```
def extract_text(img, y_thresh):
    result = ['Nothing']
    g, t, c = ['Nothing'], ['Nothing'], ['Nothing']
    try:
        result = reader.readtext(img, detail=1)
        g, t, c = extract_bb_text_confidence(result, y_thresh)
    except:
        pass
    return g, t, c

def bounding_box_sorting(boxes, y_thresh):
    num_boxes = len(boxes)
    # sort from top to bottom and left to right
    sorted_boxes = sorted(boxes, key=lambda x: (x[0][1], x[0][0]))
    _boxes = list(sorted_boxes)
    # check if the next neighgour box x coordinates > current box x coordinates, if not swap them.
    # repeat the swaping process to a threshold iteration and also select the threshold
    for i in range(5):
        for i in range(num_boxes - 1):
            if abs(_boxes[i + 1][0][1] - _boxes[i][0][1]) < y_thresh and (
                    _boxes[i + 1][0][0] < _boxes[i][0][0]):
                tmp = _boxes[i]
                _boxes[i] = _boxes[i + 1]
                _boxes[i + 1] = tmp
    return _boxes

def extract_bb_text_confidence(geometry_text_confidence, y_thresh):
    my_dict = dict(zip(np.arange(len(geometry_text_confidence)), [i[0] for i in geometry_text_confidence]))
    sorted_geometry = [list(my_dict.keys())[list(my_dict.values()).index(i)] for i in
                       bounding_box_sorting([i[0] for i in geometry_text_confidence], y_thresh)]
    result = [geometry_text_confidence[i] for i in sorted_geometry]
    g, t, c = [i[0] for i in result], [i[1] for i in result], [i[2] for i in result]
    return [g, t, c]
```

Following the code above, to extract text with the sorted text, we simply have to call the extract_text(img, y_thresh) function. Within that function, it calls the extract_bb_text_confidence(result, y_thresh).

The result of reader.readtext(link) is a list of elements, with each element consisting of the 3 previously mentioned components. What extract_bb_text_confidence does is that it extracts the first coordinate (top left corner) of the first component of each element and pairs each coordinate with its corresponding current position. It then passes this list as it calls the bounding_box_sorting(boxes, y_thresh) function.

In this function, the algorithm for sorting the bounding box is to first sort from top to bottom and then left to right. Using the y-axis, we can find out which bounding boxes are in the same row. This is calculated by using the y_thresh value (to be explained more in step 9). Once each row is sorted, we use the x-axis to sort their positioning, wherein the bounding box with the lowest number comes first in the row.

**Result
→ Before
Salah satu persamaan singgung lingkaran dengan titik pusat P(-I, ~2) dan melalui titik (3, -4) serta sejajar dengan y =-2x + 5 adalah A 2x + y + 6 = 0 B 2x -y - 6 = 0 C. 2x = y + 14 = 0 D 2x + y - 14 = 0 E 2x + y + 14 = 0 garis garis
→ After
Salah persamaan singgung satu garis lingkaran dengan titik pusat P(-I, ~2) dan melalui titik (3, -4) serta sejajar dengan garis =-2x + 5 adalah y 2x + y + 6 = 0 A -y - 6 = 0 B 2x C. 2x = y + 14 0 = D 2x + y - 14 0 = E 2x + y + 14 = 0

**Analysis
As we can see, after calling the extract_text(img, y_thresh) function, while one word was misplaced, the words “garis” were extracted in the correct position, leading to an overall better performance.

Step 9

I did further analysis with the y_thresh value that I've briefly mentioned above. For more clarification, the y_thresh value determines which row the bounding box coordinate lies in, in which it is the gap between rows. For example, if the y_thresh value is 10, then bounding boxes with coordinates 0-10 are in the first row, 11-20 second row, and so on. The initial y_thresh value I used was 10.

```
thresholds = [10, 20, 25, 30, 40, 50]
tt = []

for i in thresholds:
  g,t,c = extract_text(link, i)
  tt.append(t)

ttt = []
for i in tt:
  ttt.append(' '.join(i))
```

For this experiment, I used 30 images and looped them through the values 10, 20, 25, 30, 30 and 50. For each image, I manually compared the 6 results and kept track of which y_thresh value resulted in the best output. Here is an example of the results of 1 image:

Y_thresh	
10	Diberikan titik A(2, ~l), B(4, k), dan C(6, 1). Tentukan nilai k agar vektor AB tegak lurus dengan vektor AC b Tentukan nilai y agar titik-titik A(-1, y), B(l, ~2) dan C(2, 0) segaris. 3 Nyatakan vektor 7 sebagai hasil kali skalar dengan vektor satuan yang searah dengan vektor 1 , misal 7 kë Tentukan juga nilai k yang memenuhi.
20	Diberikan titik A(2, ~l), B(4, k), dan C(6, 1). Tentukan nilai k agar vektor AB tegak lurus dengan vektor AC b Tentukan nilai y agar titik-titik A(-1, y), B(l, ~2) dan C(2, 0) segaris. 3 Nyatakan vektor 7 sebagai hasil kali skalar dengan vektor satuan yang searah dengan vektor 1 , misal 7 kë Tentukan juga nilai k yang memenuhi.
25	Diberikan titik A(2, ~l), B(4, k), dan C(6, 1). Tentukan nilai k agar vektor AB tegak lurus dengan vektor AC b Tentukan nilai y agar titik-titik A(-1, y), B(l, ~2) dan C(2, 0) segaris. Nyatakan vektor 3 7 sebagai hasil kali skalar dengan vektor satuan yang searah dengan vektor 1 , misal 7 kë Tentukan juga nilai k yang memenuhi.
30	Diberikan titik A(2, ~l), B(4, k), dan C(6, 1). Tentukan nilai k agar vektor AB tegak lurus dengan vektor AC b Tentukan nilai y agar titik-titik A(-1, y), B(l, ~2) dan C(2, 0) segaris. Nyatakan vektor 7 3 sebagai hasil kali skalar dengan vektor satuan yang searah dengan vektor 1 , misal 7 kë Tentukan juga nilai k yang memenuhi.
40	Diberikan titik A(2, ~l), B(4, k), dan C(6, 1). Tentukan nilai k agar vektor AB tegak lurus dengan vektor AC b Tentukan nilai y agar titik-titik A(-1, y), B(l, ~2) dan C(2, 0) segaris. Nyatakan vektor 7 3 sebagai hasil kali skalar dengan vektor satuan yang searah dengan vektor 1 , misal 7 kë Tentukan juga nilai k yang memenuhi.
50	Diberikan titik A(2, ~l), B(4, k), dan C(6, 1). Tentukan nilai k agar vektor AB tegak lurus dengan b Tentukan nilai y agar titik-titik A(-1, y), B(l, ~2) dan C(2, 0) vektor AC 3 segaris. Nyatakan vektor 7 sebagai hasil kali skalar dengan vektor satuan yang searah dengan vektor 1 , misal 7 kë Tentukan juga nilai k yang memenuhi.

**Analysis

Y_thresh = 25 received the most number of images whose image produces the best output at 25 compared to other values.

Tesseract VS Easy OCR

I did a simple comparison between both OCR. I Preprocessed 15 images using only the thresholding function and extracted text from them using the 2 OCR. I then manually compared the results.

Statistical Result

7/15 → Easy OCR

4/15 → Tesseract

4/15 → Equal

Easy OCR produced a better result overall.

NLP - Natural Language Processing

Definition

Subfield of AI, concerned with understanding the structure and meaning of human language
Goal → understand unstructured texts and retrieve meaningful pieces of information from it

Techniques

1. Sentiment analysis → determine whether data is positive, negative or neutral
2. Stemming & Lemmatization → normalize word
   Stemming: truncates word to its stem word: 'studies' → 'studi' and 'studying' → 'study'
   Lemmatization: finds dictionary word: 'studies' → 'study' and 'studying' → 'study'
3. Bag of words → concerned with frequency of each word. represent text in numbers for use in ML models
4. Term frequency - Inverse Document Frequency → computes "weights" that represent how relevant a word is to a document in a collection of documents
5. Word Cloud → identify keywords in text, the higher the frequency the larger and bolder the font

Experiment on Sentiment Analysis - preprocessing data

For the NLP experiment, I did sentiment analysis on 280 rating reviews. End goal is to have the code determine whether the review is positive, negative or neutral, and compare it to the actual correct ratings to see the percentage accuracy. The first step I did was to preprocess the reviews:

1. Clean data - remove punctuations and numbers, lower capitalization

```
import re
def clean(text):
# Removes all special characters and numericals leaving the alphabets
    text = re.sub('[^A-Za-z]+', ' ', text).lower()
    return text

cleanx_data = x_data.apply(clean)
```

2. Remove stop words - Stop words are common words. Instead of using the 'english' stop-word list, I made my own list. Reason for this is that the 'english' list consists of specific words that are crucial in creating the algorithm. An example of these words is 'not'. If 'not' was eliminated, sentences, for example, consisting of 'not good' will turn to 'good', thus eliminating the nuance that we're looking for in a sentence. Additionally, since punctuations were removed, words in the list like 'they're' should be changed to 'theyre'.

```
common = ['ourselves', 'hers', 'between', 'yourself', 'but', 'again', 'mother', 'father', 'blue', 'yellow'
          'there', 'about', 'once', 'during', 'out', 'very', 'having', 'america', 'house', 'paper', 'state'
          'with', 'they', 'own', 'an', 'be', 'some', 'for', 'do', 'plant', 'green', 'water''zero','one',
          'its', 'yours', 'such', 'into', 'of', 'most', 'itself', 'two','three','four','five','six','seven',
          'other', 'is', 's', 'am', 'or', 'who', 'as', 'from', 'eight', 'nine','ten',
          'him', 'each', 'the', 'themselves', 'until', 'below', 'are',
          'we', 'these', 'your', 'his', 'through', 'nor', 'me',
          'were', 'her', 'more', 'himself', 'this', 'down', 'should',
          'our', 'their', 'while', 'above', 'both', 'up', 'to', 'ours',
          'had', 'she', 'all', 'when', 'at', 'any', 'before', 'them',
          'same', 'and', 'been', 'have', 'in', 'will', 'on', 'does',
          'yourselves', 'then', 'that', 'because', 'what', 'over', 'why',
          'so', 'can', 'did', 'now', 'he', 'you', 'herself',
          'has', 'just', 'where', 'only', 'myself', 'which', 'those',
          'i', 'after', 'few', 'whom', 't', 'being', 'if', 'theirs', 'my',
          'against', 'a', 'by', 'doing', 'it', 'how', 'further', 'was', 'here',
          'than', 'theyre']
```

```
stop_data = []
for i in token:
  stop_data.append((" ").join(w for w in i if w not in common))
```

3. Tokenization & POS tagging - Tokenizing in breaking the sentence word for word. For example, the sentence " This is not good" will turn to ['This', 'is', 'not', 'good']. POS tagging comes after tokenization, where POS tagging is a process of converting each token into a tuple having the form (word, tag). This is essential to preserve the context of the word and is essential for Lemmatization. For example, ('This', 'DT'), ('is', 'VBZ')...

```
from nltk.tokenize import word_tokenize
from nltk.tag import pos_tag

stoptoken_data = []
for i in stop_data:
  stoptoken_data.append(word_tokenize(i))

stoptokenpos_data = []
for i in stop_data:
  stoptokenpos_data.append(pos_tag(word_tokenize(i)))
```

4. Stemming & lemmatization

```
stemmer = PorterStemmer()
stem_data = []

for e in stoptoken_data:
  stem = []
  for k in e:
    stem.append(stemmer.stem(k))
  stem_data.append(stem)
```

```
from nltk.corpus import wordnet
nltk.download('wordnet')
nltk.download('omw-1.4')
pos_dict = {'J':wordnet.ADJ, 'V':wordnet.VERB, 'N':wordnet.NOUN, 'R':wordnet.ADV}

#Instantiate WordNetLemmatizer
wordnet_lemmatizer = WordNetLemmatizer()

lemma_data = []
for i in stoptokenpos_data:
  lemma = []
  for ele, tag in i:
      tag = pos_dict.get(tag[0])
      if not tag:
          lemma.append(ele)
      else:
          lemma.append(wordnet_lemmatizer.lemmatize(ele, tag))
  lemma_data.append(lemma)
```

**preprocessing progress

On the screenshot below, you can see the process result from the steps done above.

```
preprocessedTable
```

	review	cleaned review	tokenized	POS tagged	lemma	stem
0	Another Midrange killer Smartphone by Xiaomi\n...	another midrange killer smartphone xiaomi majo...	[another, midrange, killer, smartphone, xiaomi...	[(another, DT), (midrange, NN), (killer, NN), ...	[another, midrange, killer, smartphone, xiaomi...	[anoth, midrang, killer, smartphon, xiaomi, ma...
1	All ok but vry small size mobile	ok vry small size mobile	[ok, vry, small, size, mobile]	[(ok, JJ), (vry, NN), (small, JJ), (size, NN),...	[ok, vry, small, size, mobile]	[ok, vri, small, size, mobil]
2	Quite good	quite good	[quite, good]	[(quite, RB), (good, JJ)]	[quite, good]	[quit, good]
3	Redmi has always have been the the king of bud...	redmi always king budget segment yet another g...	[redmi, always, king, budget, segment, yet, an...	[(redmi, NN), (always, RB), (king, VBG), (budg...	[redmi, always, king, budget, segment, yet, an...	[redmi, alway, king, budget, segment, yet, ano...
4	worst product from MI. I am a hardcore fan of ...	worst product mi hardcore fan mi really disapp...	[worst, product, mi, hardcore, fan, mi, really...	[(worst, JJS), (product, NN), (mi, NN), (hardc...	[bad, product, mi, hardcore, fan, mi, really, ...	[worst, product, mi, hardcor, fan, mi, realli,...
...	...	...	...	...	...	...

Experiment on Sentiment Analysis - TextBlob

A Python library for processing textual data. The main line of code is TextBlob(text).sentiment, which gives us 2 measures:

Polarity - positivity of opinion (-1 to 1, 1 being more positive, 0 neutral, -1 more negative)

Subjectivity - talks about how subjective the opinion is (0 to 1, 0 being objective and 1 being subjective)

```
from textblob import TextBlob

# function to calculate subjectivity
def getSubjectivity(review):
    return TextBlob(review).sentiment.subjectivity

# function to calculate polarity
def getPolarity(review):
    return TextBlob(review).sentiment.polarity

# function to analyze the reviews
def analysis(score):
    if score < 0:
        return 'Negative'
    elif score == 0:
        return 'Neutral'
    else:
        return 'Positive'
```

**For this experiment, I want to see whether the lemmatized inputs or stemmed inputs will produce a better output. And so, the following code is done twice with different inputs and compared.

```
pol_stem = []
for i in stem_data:
  sentence = ""
  for k in i:
    sentence += k
  pol_stem.append(getPolarity(sentence))

analysis_stem = []
for p in pol_stem:
  analysis_stem.append(analysis(p))
```

```
comparison_stem = []
for i in range(len(y_data2)):
  if y_data2[i] == analysis_stem[i]:
    comparison_stem.append("true")
  else:
    comparison_stem.append("false")
```

```
pol_lemma = []
for i in lemma_data:
  sentence = ""
  for k in i:
    sentence += k
  pol_lemma.append(getPolarity(sentence))

pol_lemma

analysis_lemma = []
for p in pol_lemma:
  analysis_lemma.append(analysis(p))
```

```
comparison_lemma = []
for i in range(len(y_data2)):
  if y_data2[i] == analysis_lemma[i]:
    comparison_lemma.append("true")
  else:
    comparison_lemma.append("false")
```

**Analysis
Both outputs from the different inputs (stem and lemma) resulted in the same number of correctly predicted ratings. 61/280

<u>Experiment on Sentiment Analysis - Vader</u>
Stands for Valence Aware Dictionary and Sentiment Reasoner. The vader sentiment gives us 4 measures: negative, neutral, positive and compound. The negative, neutral and positive measure adds up to 1 and the compound is the metric used to draw the overall sentiment:
Positive – compound >= 0.5
Neutral – -0.5 < compound < 0.5
Negative – -0.5 >= compound

```
from nltk.sentiment.vader import SentimentIntensityAnalyzer
nltk.download('vader_lexicon')

analyzer = SentimentIntensityAnalyzer()

# function to calculate vader sentiment
def vadersentimentanalysis(review):
    vs = analyzer.polarity_scores(review)
    return vs['compound']

# function to analyse
def vader_analysis(compound):
    if compound >= 0.5:
        return 'Positive'
    elif compound <= -0.5 :
        return 'Negative'
    else:
        return 'Neutral'
```

Just like the TextBlob experiment, I want to see whether the lemmatized inputs or stemmed inputs will produce a better output. And so, the function is called twice for the inputs. As a result, the stemmed input resulted in a 30/280 accuracy while the lemma input resulted in a 31/280 accuracy

Result and Analysis

We can see that the TextBlob resulted in a higher accuracy for sentiment analysis. Specific to comparing lemmatization and stemming, we can deduce from the above vader result that the lemmatized input produced a higher accuracy.