Digital Image Processing Group Assignment

Due date: 24th October 2022

By

Name	**Programme**	**Student ID**	**Student Imail**
Yong Lik Kent	BSc. (Hons) Computer Science	19098219	19098219@imail.sunway.edu.my
Wong Chai Hoong	BSc. (Hons) Computer Science	19118751	19118751@imail.sunway.edu.my
Lee Boon Bing	BSc. (Hons) Computer Science	18109082	18109082@imail.sunway.edu.my
Ng Jia Shiang	BSc. (Hons) Computer Science	20042800	20042800@imail.sunway.edu.my

Table of Content

Task A

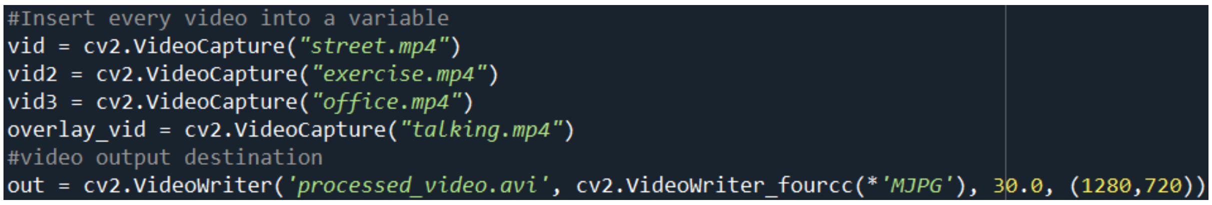
Introduction

Recently, one of our friends had some issues after becoming a YouTuber. In order to prevent the film from being stolen, she had to obscure the faces on the footage she captured and superimpose a video of her speaking over it with various watermarks. The primary objective of the group project is to apply what we have learned about digital image processing to assist her in overcoming obstacles. Under the methodology, all of the steps of the code will be shown with explanations for the programs. And in the result, an output with an explanation will also be included below.

Methodology

To tackle the first issue, we employ four approaches: the cascade Classifier of the face detector, the if-else of adding the overlay video, copyMakeBorder for the border of the overlay video, and the for loop of watermarks on the video. The four techniques will be discussed in further depth below.

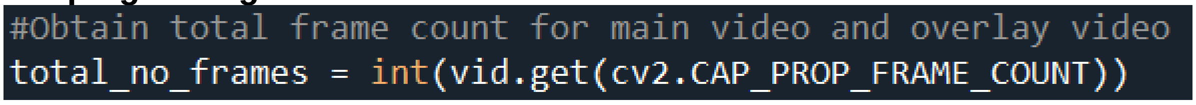
First Problem – Importing video and Blurring faces that appeared within.



```
#Insert every video into a variable
vid = cv2.VideoCapture("street.mp4")
vid2 = cv2.VideoCapture("exercise.mp4")
vid3 = cv2.VideoCapture("office.mp4")
overlay_vid = cv2.VideoCapture("talking.mp4")
#video output destination
out = cv2.VideoWriter('processed_video.avi', cv2.VideoWriter_fourcc(*'MJPG'), 30.0, (1280,720))
```

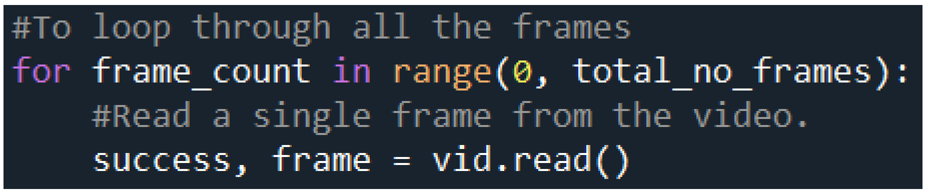
At the start of our code, we first need to input the main video using cv2.VideoCapture for street.mp4, exercise.mp4, and office.mp4 files. Next, it is to input the overlay talking video (talking.mp4) of our friend.
Then the out variable using the function of cv2.VideoWriter is to specify the expected file name where the output would be exported to, the other elements in cv2.VideoWriter is to define the video codec, frame per second, and video width and height.

Looping through each frame of the main video



```
#Obtain total frame count for main video and overlay video
total_no_frames = int(vid.get(cv2.CAP_PROP_FRAME_COUNT))
```

The code above is to import both watermarks for the video by preventing the video getting stolen. Next is, use the total frame count to calculate the process when the video is out writing. This is to show the progress when exporting the video, since it will take quite some time to write it.

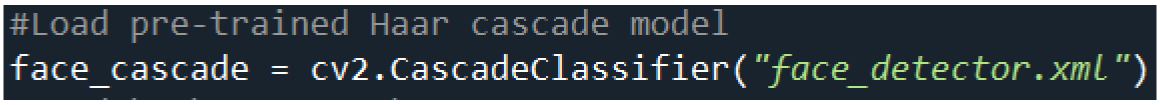


```
#To loop through all the frames
for frame_count in range(0, total_no_frames):
    #Read a single frame from the video.
    success, frame = vid.read()
```

In this first for loop, we will loop through each frame. If it's successful then read the video frame. We also print out the progress every time the for loop is triggered

While looping through each of the frames, we then increase 10 of the video brightness for each frame.

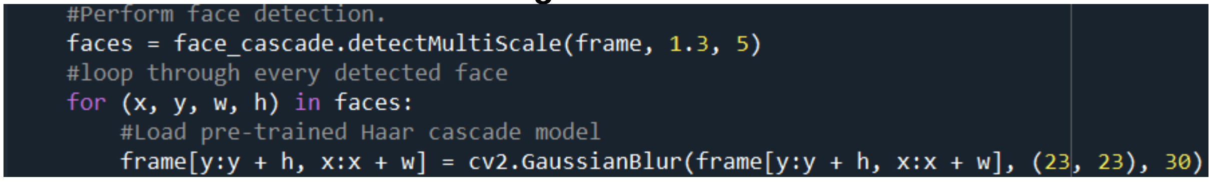
Cascade Classifier



```
#Load pre-trained Haar cascade model
face_cascade = cv2.CascadeClassifier("face_detector.xml")
```

We need to import the xml file of face_detector.xml to use its CascadeClassifier function to use face detection in our program.

Face Detection and Face Blurring

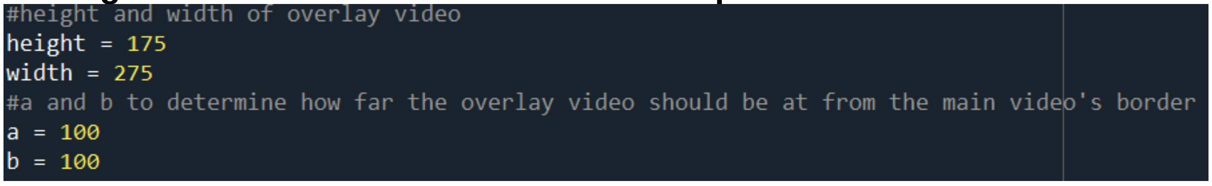


```
    #Perform face detection.
    faces = face_cascade.detectMultiScale(frame, 1.3, 5)
    #loop through every detected face
    for (x, y, w, h) in faces:
        #Load pre-trained Haar cascade model
        frame[y:y + h, x:x + w] = cv2.GaussianBlur(frame[y:y + h, x:x + w], (23, 23), 30)
```

After importing the face detector of the Cascade Classifier, we then use the detect Multiscale of the cascade to perform the face detection for each of the frames. After detecting the faces for each frame, we then loop through all the detected faces and use the Gaussian Blur function to blur the faces, since it is used to reduce noise and details.

Second Problem – Resize and Overlay Talking.mp4 over every 3 of the videos

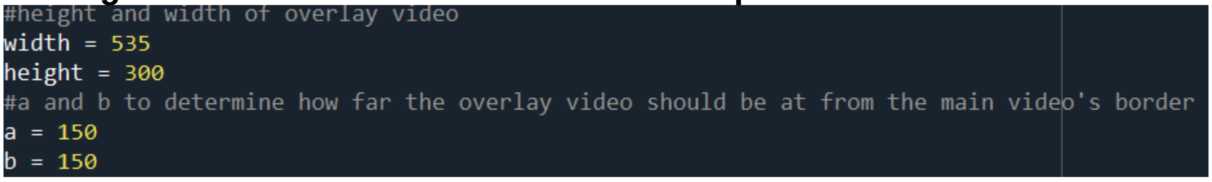
Settings for main video that are 1080 x 720 pixels



```
#height and width of overlay video
height = 175
width = 275
#a and b to determine how far the overlay video should be at from the main video's border
a = 100
b = 100
```

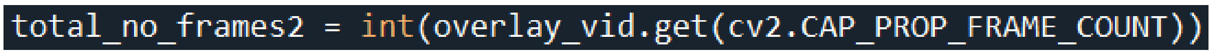
We need to set the height and width of the talking video position under the main video, then also to set the size of the talking video. In this, we set 175 heights and 275 widths for the size of the talking video, and the position of the overlay talking video is 100 pixels away from the top of the border of the main video, 100 pixels away from the left of the border of the main video.

Settings for main video that are 1920 x 1080 pixels



```
#height and width of overlay video
width = 535
height = 300
#a and b to determine how far the overlay video should be at from the main video's border
a = 150
b = 150
```

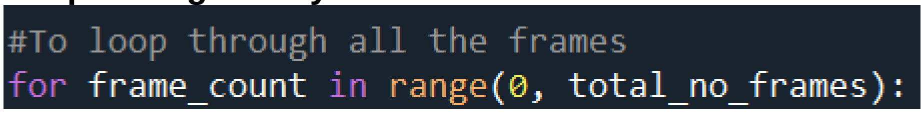
The width and height of the overlay video are increased, while the distance from the border of the main video is also further since the overall video pixel size has increased to accommodate the bigger main video pixel size.



```
total_no_frames2 = int(overlay_vid.get(cv2.CAP_PROP_FRAME_COUNT))
```

Obtain the total number of frames for the overlay video.

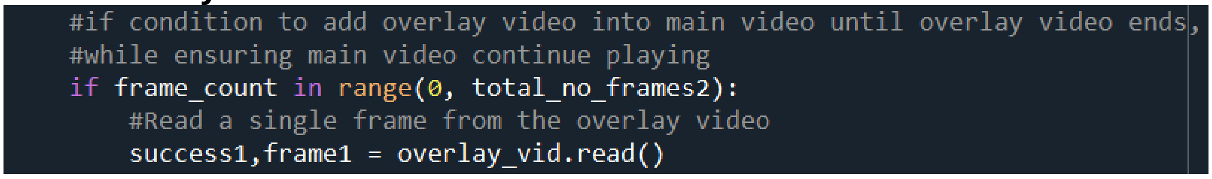
Loop through every frame of main video



```python
#To loop through all the frames
for frame_count in range(0, total_no_frames):
```

We need to first loop through every frame of the main video, read the frame of the video, perform face blurring

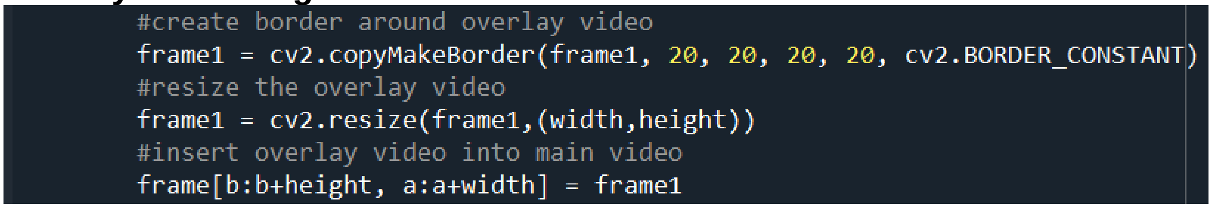
Read overlay video



```python
    #if condition to add overlay video into main video until overlay video ends,
    #while ensuring main video continue playing
    if frame_count in range(0, total_no_frames2):
        #Read a single frame from the overlay video
        success1,frame1 = overlay_vid.read()
```

We use if condition to add the overlay video into the main video. This is to ensure that when the overlay video ends, the main video will continue playing until the end. We also used frame count to calculate the progress when we out write the video, and when it is successful, then read each video frame of overlay_vid. The rationale is while main video runs, it could use the main video frame count to determine when it should start and end.

Overlay the talking video on to the main video

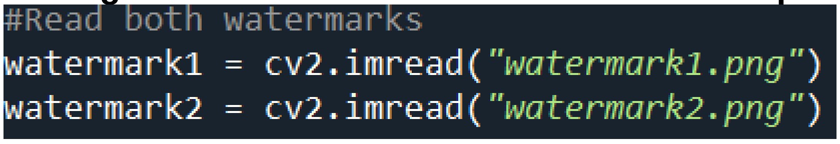


```python
        #create border around overlay video
        frame1 = cv2.copyMakeBorder(frame1, 20, 20, 20, 20, cv2.BORDER_CONSTANT)
        #resize the overlay video
        frame1 = cv2.resize(frame1,(width,height))
        #insert overlay video into main video
        frame[b:b+height, a:a+width] = frame1
```

After adding the overlay video above, we then need to create the border around the overlay video using the cv2.copyMakeBorder function to do it, then resize the overlay video and insert the overlay video into the main video.

Third Problem – Add watermark to the videos

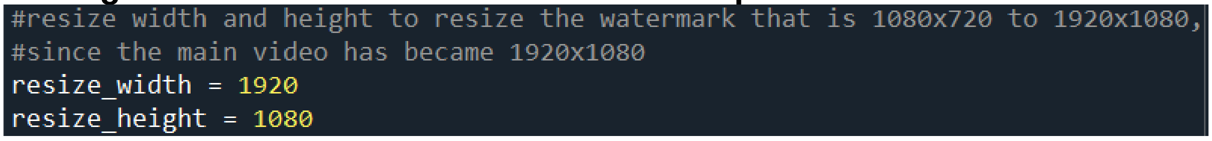
Settings for the main video that are 1080 x 720 pixels



```python
#Read both watermarks
watermark1 = cv2.imread("watermark1.png")
watermark2 = cv2.imread("watermark2.png")
```

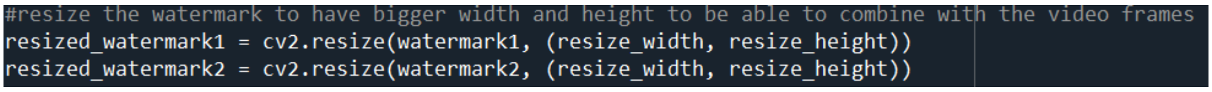
Importing both watermarks

Settings for main video that are 1920 x 1080 pixels



```python
#resize width and height to resize the watermark that is 1080x720 to 1920x1080,
#since the main video has became 1920x1080
resize_width = 1920
resize_height = 1080
```

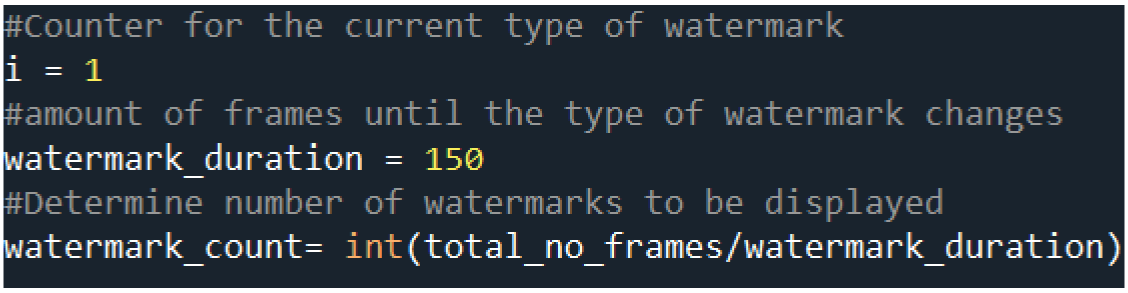
Define the new width and height for resizing of the watermark, since the main video and watermark has to be the same size to be combined together.



```
#resize the watermark to have bigger width and height to be able to combine with the video frames
resized_watermark1 = cv2.resize(watermark1, (resize_width, resize_height))
resized_watermark2 = cv2.resize(watermark2, (resize_width, resize_height))
```

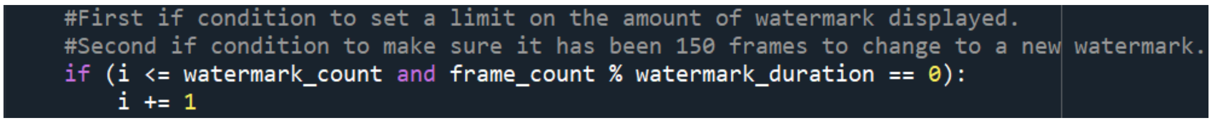
Resize both watermarks with the defined height and width.

Watermarks



```
#Counter for the current type of watermark
i = 1
#amount of frames until the type of watermark changes
watermark_duration = 150
#Determine number of watermarks to be displayed
watermark_count= int(total_no_frames/watermark_duration)
```

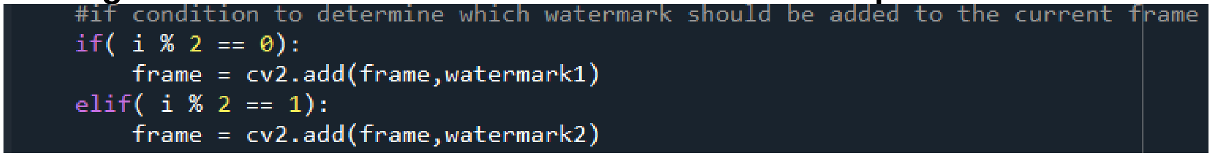
The “i” is to determine what type of watermark should be added to the main video. The watermark duration is 150 frames, which is only 5 seconds since there are 30 frames in a second for each video. Watermark count is to determine what is the max amount of times different watermark is displayed, to avoid an infinite loop.



```
    #First if condition to set a limit on the amount of watermark displayed.
    #Second if condition to make sure it has been 150 frames to change to a new watermark.
    if (i <= watermark_count and frame_count % watermark_duration == 0):
        i += 1
```

Conditions to increase the counter that determines the current type of watermark to be added are ensuring the counter is lower or equal to the watermark count or the max amount of times different watermark is displayed, another condition is the current frame count has to be the modulus of watermark duration, or 150 and have no remainder, to trigger an increment of “i”.

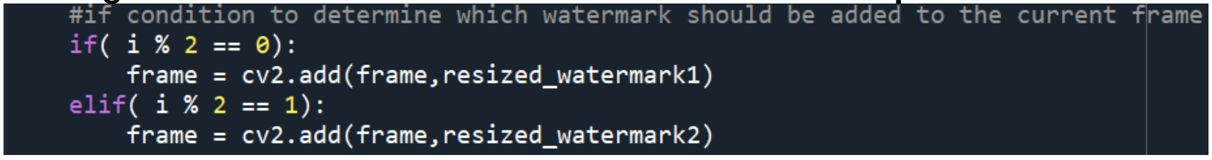
Adding watermark into main video that is 1080 x 720 pixels



```
    #if condition to determine which watermark should be added to the current frame
    if( i % 2 == 0):
        frame = cv2.add(frame,watermark1)
    elif( i % 2 == 1):
        frame = cv2.add(frame,watermark2)
```

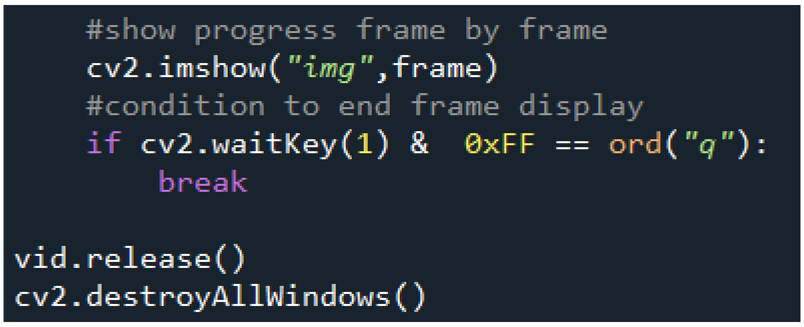
The value of “i” determines which watermark would be added on to the main video. If its value modulus by 2 has no remainder, watermark1 is added. If its value modulus by 2 has the remainder of 1, watermark2 is added,

Adding watermark into main video that is 1920 x 1080 pixels



```
    #if condition to determine which watermark should be added to the current frame
    if( i % 2 == 0):
        frame = cv2.add(frame,resized_watermark1)
    elif( i % 2 == 1):
        frame = cv2.add(frame,resized_watermark2)
```

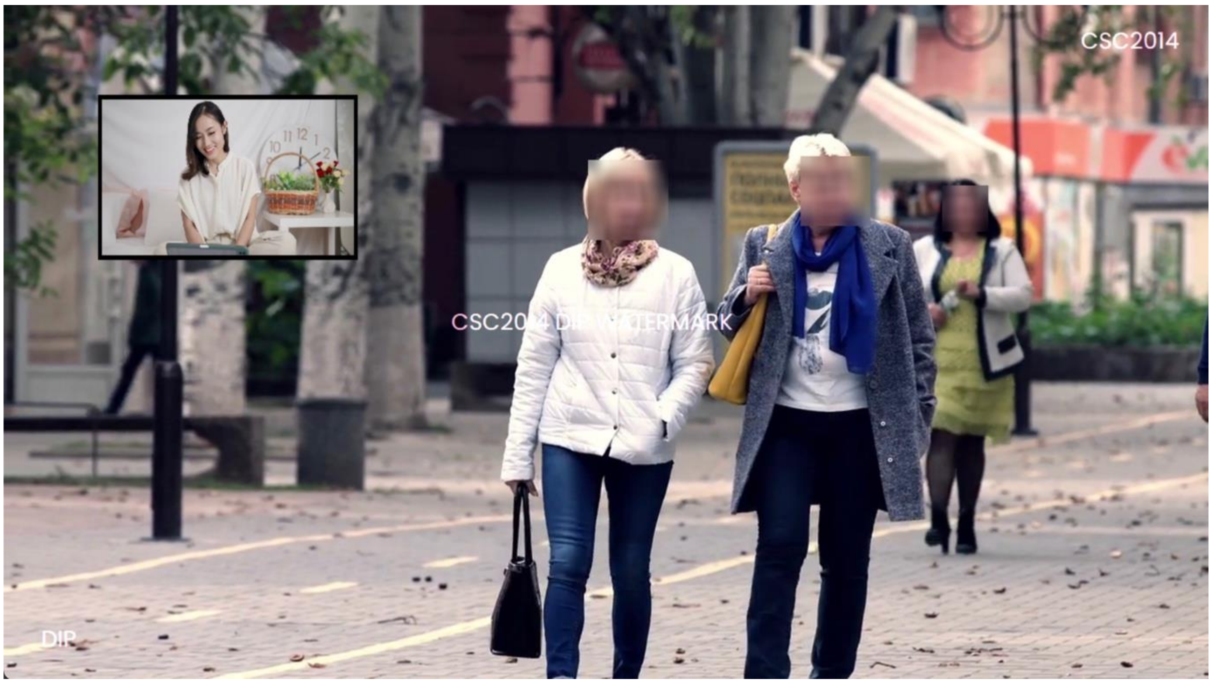
The value of “i” determines which watermark would be added on to the main video. If its value modulus by 2 has no remainder, resized watermark1 is added. If its value modulus by 2 has remainder of 1, resized watermark2 is added,



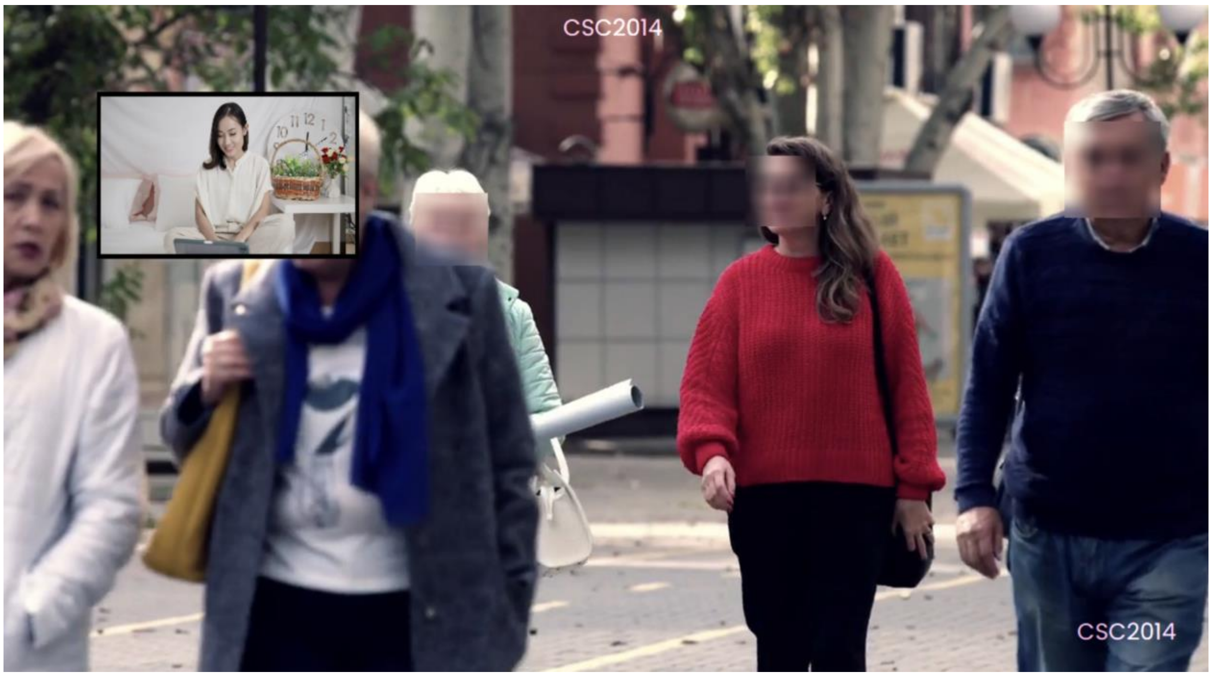
```
    #show progress frame by frame
    cv2.imshow("img",frame)
    #condition to end frame display
    if cv2.waitKey(1) &  0xFF == ord("q"):
        break

vid.release()
cv2.destroyAllWindows()
```

Finally, imshow the frame to show the final result of the outwrite video. Then release the video and destroy All Windows.

Result



The above image shows the first second of the output video. The blurring of people's faces is well shown by using Gaussian Blur. Secondly, the talking video with a black border is overlaying on the upper left side of the original video. Lastly, the watermark shown is watermark 1 where the watermark is middle, top right and bottom left.



This section demonstrates how our code works by changing the watermarks every 5 seconds while the blurring of people's faces and overlaying video remains. Watermark 2 will be used this time where watermark at bottom right and on top in the middle.

The image above is to demonstrate of our If else of overlaying the talking video is working in our code. As the talking video is only 15 seconds and the background total is 33 seconds. After the talking video end, it will disappear while the background video will continue.

Task B

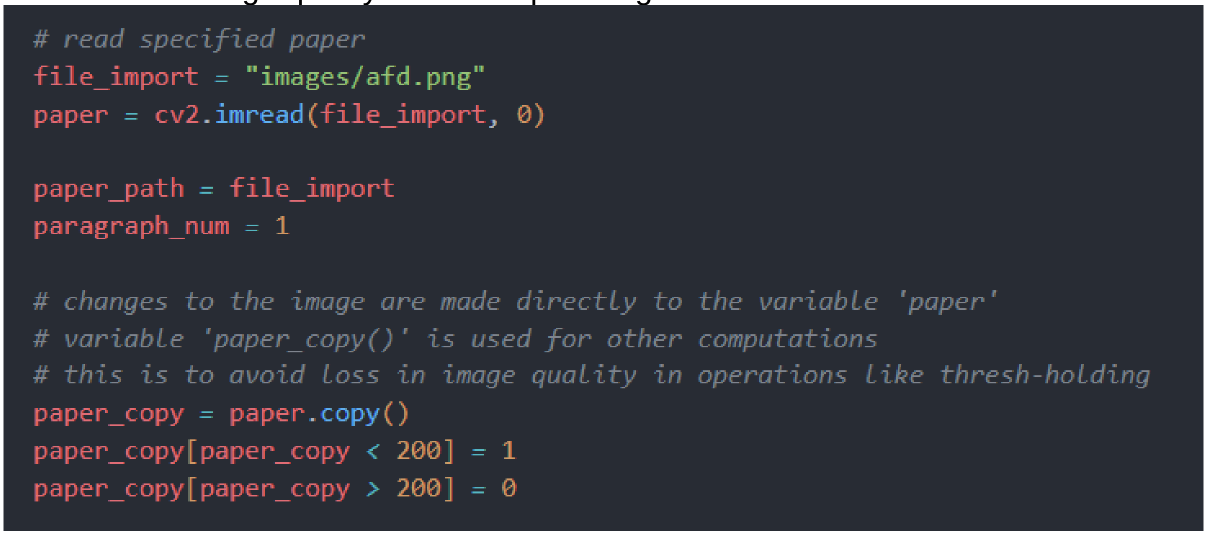
Introduction

A senior is currently working on his literature review for his Final Year Project (FYP) He has a few scientific papers that are in different formatting and he has to go through the papers to analyse the writing styles of different authors. He has requested help from us in order to extract all the paragraphs from the paper in the correct order and also ignore any sort of tables in the paper

Methodology

First Part – Ignore tables from images

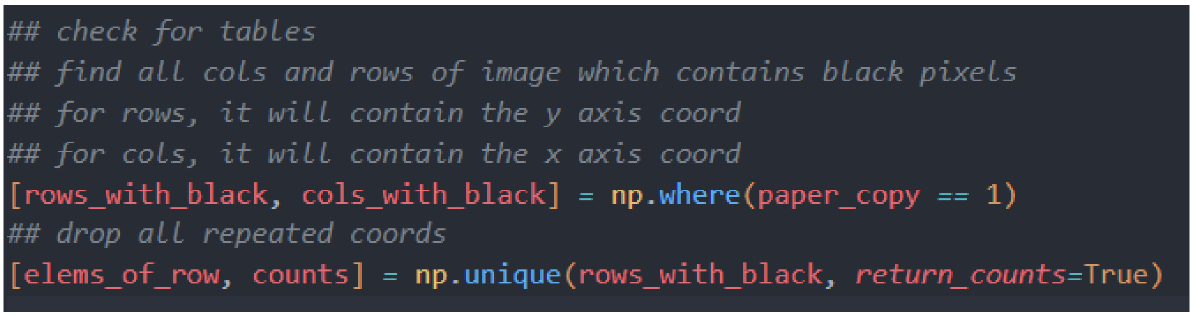
First, we our plan is to have 2 copies of the input image. One is for output related purposes, the other is for computation of coordinates and other operations. This is to avoid loss of image quality on the output image.



```python
# read specified paper
file_import = "images/afd.png"
paper = cv2.imread(file_import, 0)

paper_path = file_import
paragraph_num = 1

# changes to the image are made directly to the variable 'paper'
# variable 'paper_copy()' is used for other computations
# this is to avoid loss in image quality in operations like thresh-holding
paper_copy = paper.copy()
paper_copy[paper_copy < 200] = 1
paper_copy[paper_copy > 200] = 0
```

After that, we plan to remove table(s) from the image first. To remove a table, we have to identify the location of the table. One distinct feature of a table is that it will have straight black lines as borders.

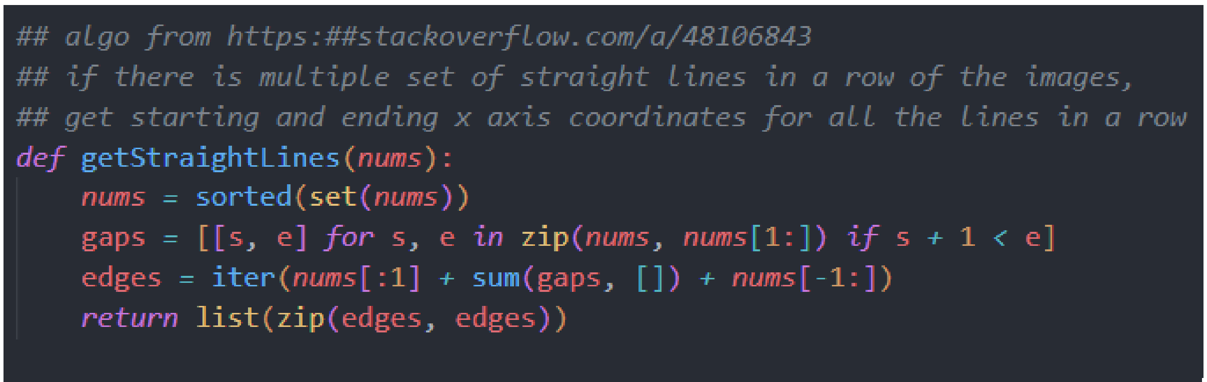
We get the rows of image which contains black pixels to narrow down our scope of search.



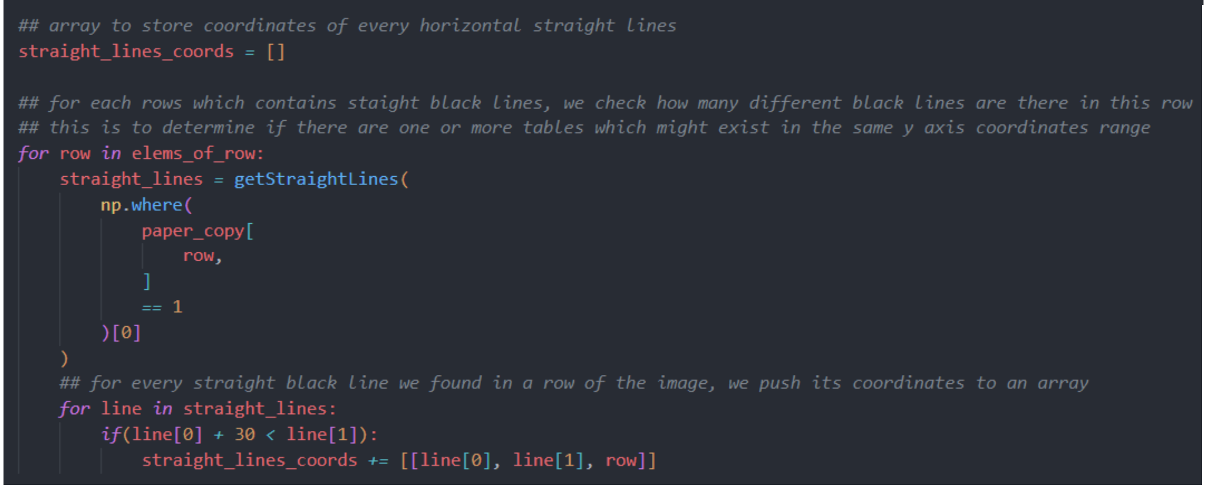
```python
## check for tables
## find all cols and rows of image which contains black pixels
## for rows, it will contain the y axis coord
## for cols, it will contain the x axis coord
[rows_with_black, cols_with_black] = np.where(paper_copy == 1)
## drop all repeated coords
[elems_of_row, counts] = np.unique(rows_with_black, return_counts=True)
```

Then, for each row with black pixels, we get all the starting x-coordinate, ending x-coordinate, for every horizontally continuous black pixel. If the length of the set of the continuous black pixel is more than 30, we push its x-to-x and y coordinates to an array.

The function shown in the image below is to find gaps within an array.

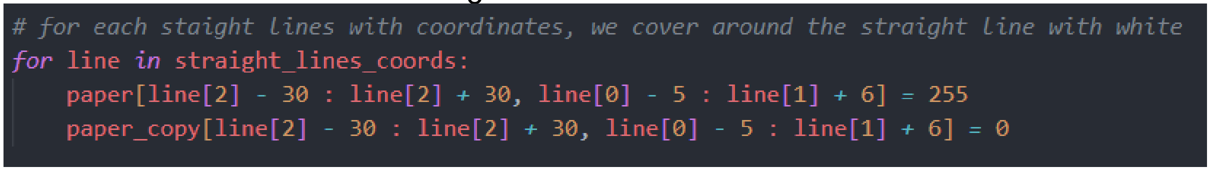


```python
## algo from https:##stackoverflow.com/a/48106843
## if there is multiple set of straight lines in a row of the images,
## get starting and ending x axis coordinates for all the lines in a row
def getStraightLines(nums):
    nums = sorted(set(nums))
    gaps = [[s, e] for s, e in zip(nums, nums[1:]) if s + 1 < e]
    edges = iter(nums[:1] + sum(gaps, []) + nums[-1:])
    return list(zip(edges, edges))
```



```python
## array to store coordinates of every horizontal straight lines
straight_lines_coords = []

## for each rows which contains staight black lines, we check how many different black lines are there in this row
## this is to determine if there are one or more tables which might exist in the same y axis coordinates range
for row in elems_of_row:
    straight_lines = getStraightLines(
        np.where(
            paper_copy[
                row,
            ]
            == 1
        )[0]
    )
    ## for every straight black line we found in a row of the image, we push its coordinates to an array
    for line in straight_lines:
        if(line[0] + 30 < line[1]):
            straight_lines_coords += [[line[0], line[1], row]]
```

Finally, we just replace a region around the border with white, so that the borders of the table would be removed along with the text the lies around it.

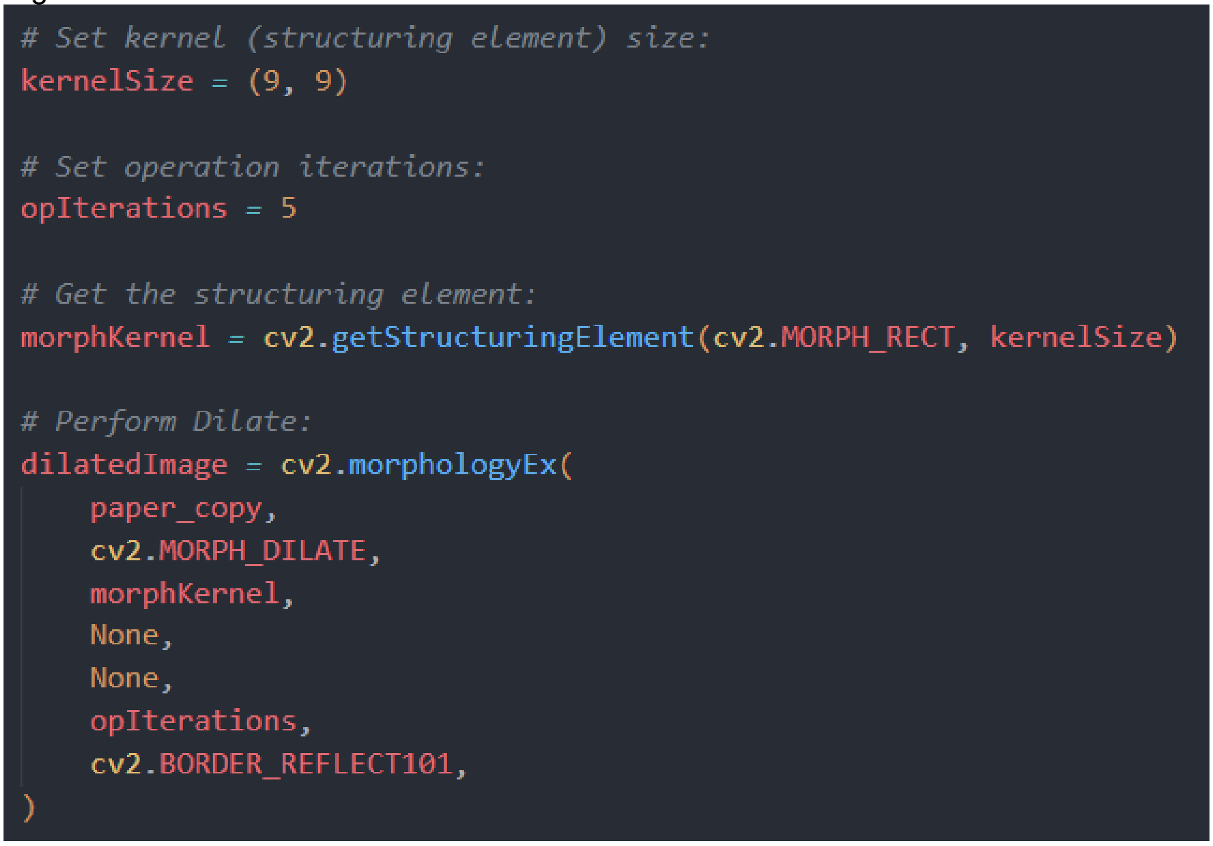


```python
# for each straight lines with coordinates, we cover around the straight line with white
for line in straight_lines_coords:
    paper[line[2] - 30 : line[2] + 30, line[0] - 5 : line[1] + 6] = 255
    paper_copy[line[2] - 30 : line[2] + 30, line[0] - 5 : line[1] + 6] = 0
```

Second Part – Export and show each paragraph

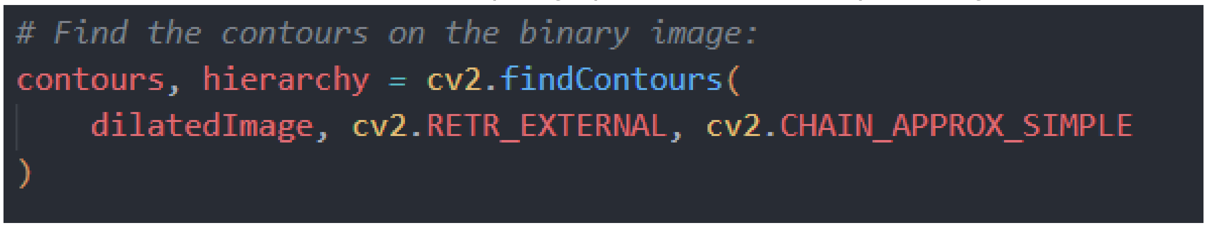
The following part is the part that is used to separate the image into individual paragraphs after removing the tables in the image.

Working on top of the previously edited input image where the tables are removed, we perform a dilation on the remaining black pixels (inverted into white in context of our code) which will be the texts, we expand the texts until texts in a paragraph is joined together.



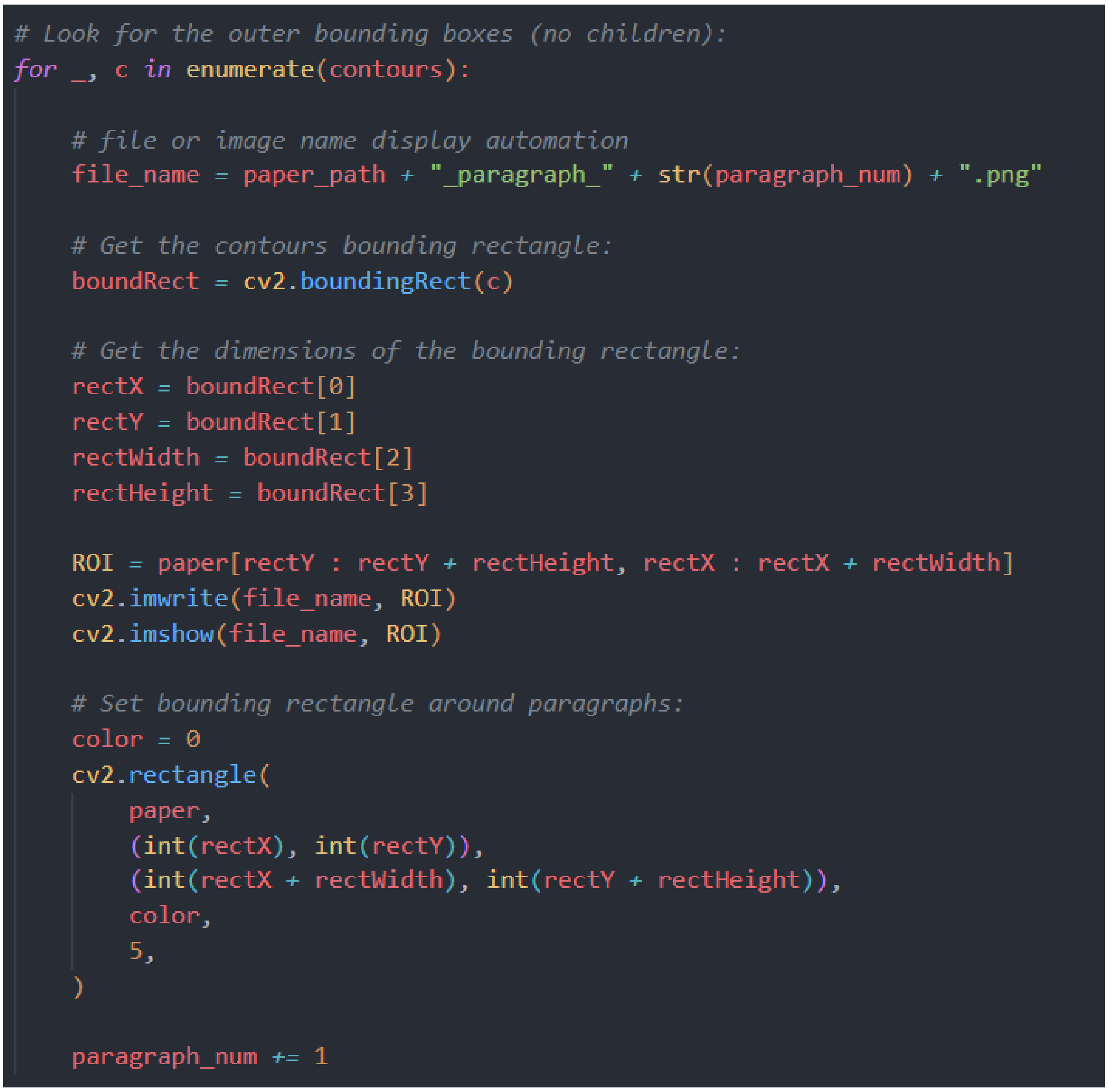
```
# Set kernel (structuring element) size:
kernelSize = (9, 9)

# Set operation iterations:
opIterations = 5

# Get the structuring element:
morphKernel = cv2.getStructuringElement(cv2.MORPH_RECT, kernelSize)

# Perform Dilate:
dilatedImage = cv2.morphologyEx(
    paper_copy,
    cv2.MORPH_DILATE,
    morphKernel,
    None,
    None,
    opIterations,
    cv2.BORDER_REFLECT101,
)
```

After that, we find the contours of the paragraphs which has been joined together.



```
# Find the contours on the binary image:
contours, hierarchy = cv2.findContours(
    dilatedImage, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE
)
```

After getting the contour coordinates, which will be our region of interest, the paragraphs, foreach of the paragraphs we draw a rectangle on the original image, and we export a cropped image based on the original image based on the coordinates of the contours.



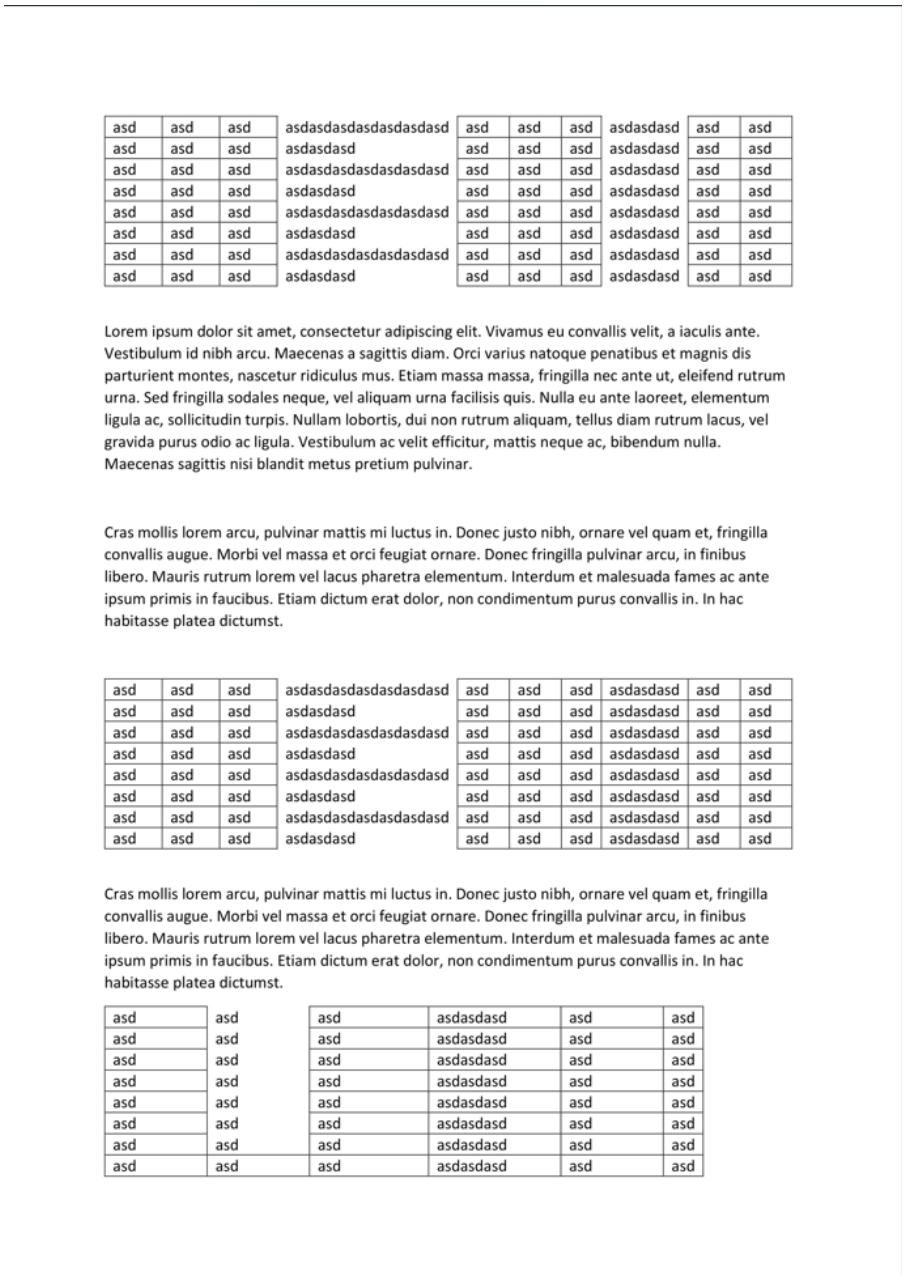
```python
# Look for the outer bounding boxes (no children):
for _, c in enumerate(contours):

    # file or image name display automation
    file_name = paper_path + "_paragraph_" + str(paragraph_num) + ".png"

    # Get the contours bounding rectangle:
    boundRect = cv2.boundingRect(c)

    # Get the dimensions of the bounding rectangle:
    rectX = boundRect[0]
    rectY = boundRect[1]
    rectWidth = boundRect[2]
    rectHeight = boundRect[3]

    ROI = paper[rectY : rectY + rectHeight, rectX : rectX + rectWidth]
    cv2.imwrite(file_name, ROI)
    cv2.imshow(file_name, ROI)

    # Set bounding rectangle around paragraphs:
    color = 0
    cv2.rectangle(
        paper,
        (int(rectX), int(rectY)),
        (int(rectX + rectWidth), int(rectY + rectHeight)),
        color,
        5,
    )

    paragraph_num += 1
```

Results

Let us put our code to the test with a sample input image which is relatively complex.

The test image is made up of 3 tables in arrow with paragraphs in between, paragraphs spanning the entire page, 2 tables in a row with a paragraph in between, and an uncommon table which has a paragraph between it and ends with a table row.

Input:



asd	asd	asd	asdasdasdasdasdasdasd	asd	asd	asd	asdasdasd	asd	asd
asd	asd	asd	asdasdasd	asd	asd	asd	asdasdasd	asd	asd
asd	asd	asd	asdasdasdasdasdasdasd	asd	asd	asd	asdasdasd	asd	asd
asd	asd	asd	asdasdasd	asd	asd	asd	asdasdasd	asd	asd
asd	asd	asd	asdasdasdasdasdasdasd	asd	asd	asd	asdasdasd	asd	asd
asd	asd	asd	asdasdasd	asd	asd	asd	asdasdasd	asd	asd
asd	asd	asd	asdasdasdasdasdasdasd	asd	asd	asd	asdasdasd	asd	asd
asd	asd	asd	asdasdasd	asd	asd	asd	asdasdasd	asd	asd

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Vivamus eu convallis velit, a iaculis ante. Vestibulum id nibh arcu. Maecenas a sagittis diam. Orci varius natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Etiam massa massa, fringilla nec ante ut, eleifend rutrum urna. Sed fringilla sodales neque, vel aliquam urna facilisis quis. Nulla eu ante laoreet, elementum ligula ac, sollicitudin turpis. Nullam lobortis, dui non rutrum aliquam, tellus diam rutrum lacus, vel gravida purus odio ac ligula. Vestibulum ac velit efficitur, mattis neque ac, bibendum nulla. Maecenas sagittis nisi blandit metus pretium pulvinar.

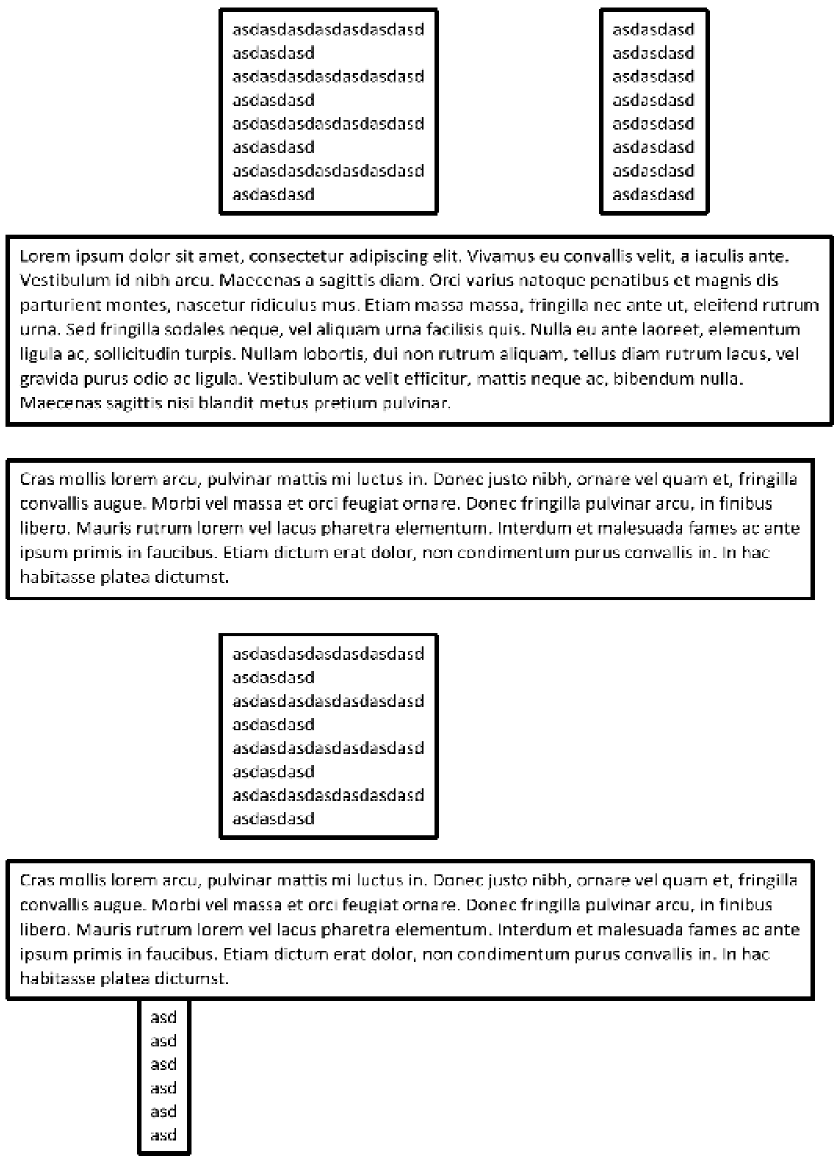
Cras mollis lorem arcu, pulvinar mattis mi luctus in. Donec justo nibh, ornare vel quam et, fringilla convallis augue. Morbi vel massa et orci feugiat ornare. Donec fringilla pulvinar arcu, in finibus libero. Mauris rutrum lorem vel lacus pharetra elementum. Interdum et malesuada fames ac ante ipsum primis in faucibus. Etiam dictum erat dolor, non condimentum purus convallis in. In hac habitasse platea dictumst.

asd	asd	asd	asdasdasdasdasdasdasd	asd	asd	asd	asdasdasd	asd	asd
asd	asd	asd	asdasdasd	asd	asd	asd	asdasdasd	asd	asd
asd	asd	asd	asdasdasdasdasdasdasd	asd	asd	asd	asdasdasd	asd	asd
asd	asd	asd	asdasdasd	asd	asd	asd	asdasdasd	asd	asd
asd	asd	asd	asdasdasdasdasdasdasd	asd	asd	asd	asdasdasd	asd	asd
asd	asd	asd	asdasdasd	asd	asd	asd	asdasdasd	asd	asd
asd	asd	asd	asdasdasdasdasdasdasd	asd	asd	asd	asdasdasd	asd	asd
asd	asd	asd	asdasdasd	asd	asd	asd	asdasdasd	asd	asd

Cras mollis lorem arcu, pulvinar mattis mi luctus in. Donec justo nibh, ornare vel quam et, fringilla convallis augue. Morbi vel massa et orci feugiat ornare. Donec fringilla pulvinar arcu, in finibus libero. Mauris rutrum lorem vel lacus pharetra elementum. Interdum et malesuada fames ac ante ipsum primis in faucibus. Etiam dictum erat dolor, non condimentum purus convallis in. In hac habitasse platea dictumst.

asd	asd	asd	asdasdasd	asd	asd
asd	asd	asd	asdasdasd	asd	asd
asd	asd	asd	asdasdasd	asd	asd
asd	asd	asd	asdasdasd	asd	asd
asd	asd	asd	asdasdasd	asd	asd
asd	asd	asd	asdasdasd	asd	asd
asd	asd	asd	asdasdasd	asd	asd
asd	asd	asd	asdasdasd	asd	asd

Output:



asdasdasdasdasdasdasd
asdasdasd
asdasdasdasdasdasdasd
asdasdasd
asdasdasdasdasdasdasd
asdasdasd
asdasdasdasdasdasdasd
asdasdasd

asdasdasd
asdasdasd
asdasdasd
asdasdasd
asdasdasd
asdasdasd
asdasdasd
asdasdasd

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Vivamus eu convallis velit, a iaculis ante. Vestibulum id nibh arcu. Maecenas a sagittis diam. Orci varius natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Etiam massa massa, fringilla nec ante ut, eleifend rutrum urna. Sed fringilla sodales neque, vel aliquam urna facilisis quis. Nulla eu ante laoreet, elementum ligula ac, sollicitudin turpis. Nullam lobortis, dui non rutrum aliquam, tellus diam rutrum lacus, vel gravida purus odio ac ligula. Vestibulum ac velit efficitur, mattis neque ac, bibendum nulla. Maecenas sagittis nisi blandit metus pretium pulvinar.

Cras mollis lorem arcu, pulvinar mattis mi luctus in. Donec justo nibh, ornare vel quam et, fringilla convallis augue. Morbi vel massa et orci feugiat ornare. Donec fringilla pulvinar arcu, in finibus libero. Mauris rutrum lorem vel lacus pharetra elementum. Interdum et malesuada fames ac ante ipsum primis in faucibus. Etiam dictum erat dolor, non condimentum purus convallis in. In hac habitasse platea dictumst.

asdasdasdasdasdasdasd
asdasdasd
asdasdasdasdasdasdasd
asdasdasd
asdasdasdasdasdasdasd
asdasdasd
asdasdasdasdasdasdasd
asdasdasd

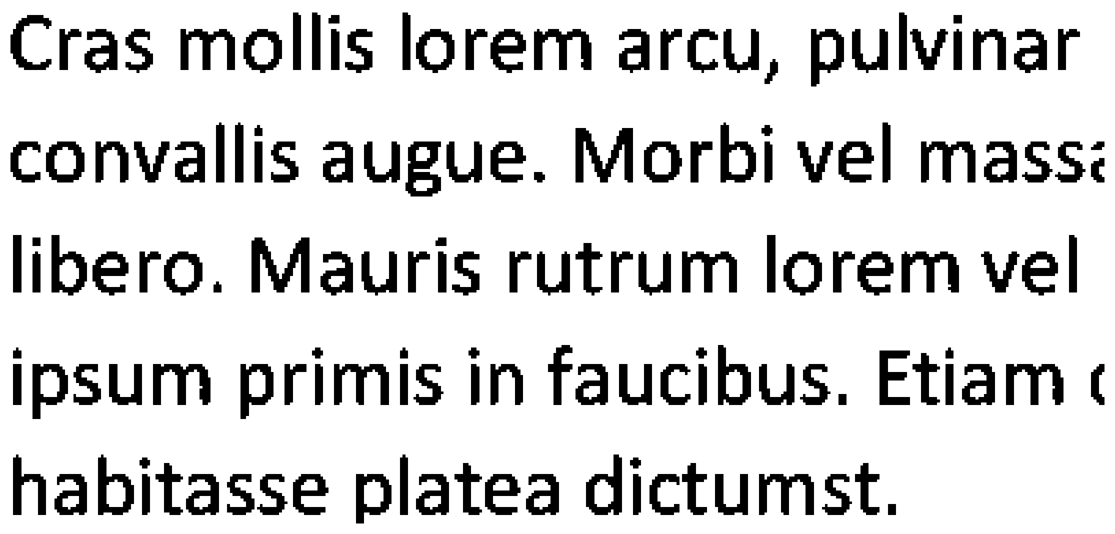
Cras mollis lorem arcu, pulvinar mattis mi luctus in. Donec justo nibh, ornare vel quam et, fringilla convallis augue. Morbi vel massa et orci feugiat ornare. Donec fringilla pulvinar arcu, in finibus libero. Mauris rutrum lorem vel lacus pharetra elementum. Interdum et malesuada fames ac ante ipsum primis in faucibus. Etiam dictum erat dolor, non condimentum purus convallis in. In hac habitasse platea dictumst.

asd
asd
asd
asd
asd
asd

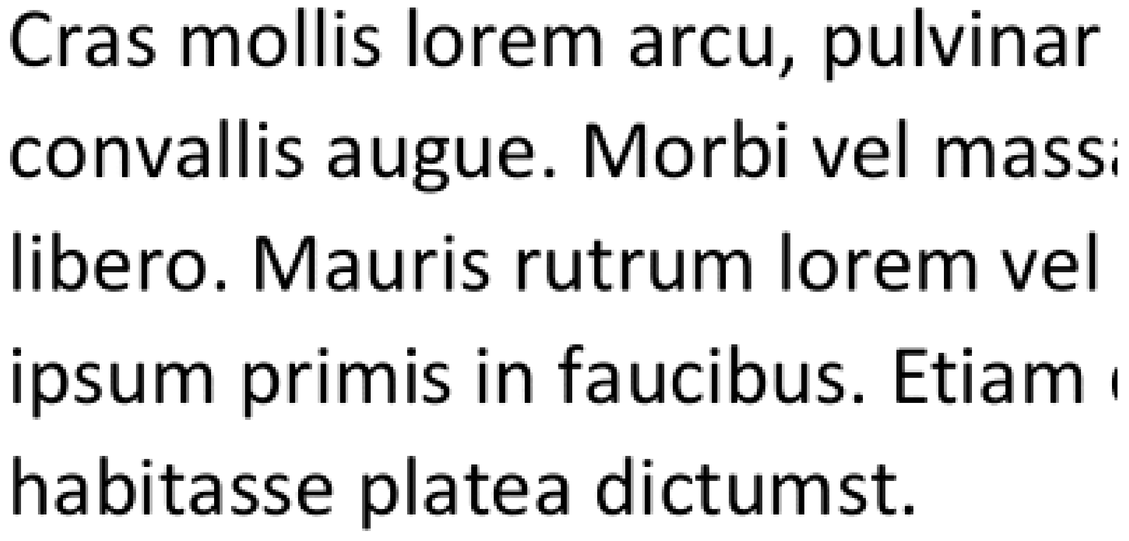
As we can see from the one of the output images, the region of interest which is our paragraphs has a rectangle around it, it works as intended with the tables out of the output image.

When we inspect one of the output images of a paragraph, we can see that the edges of the characters are smooth when compared with the output if we directly did thresh holding on the image.

Directly doing thresholding to the image:


Cras mollis lorem arcu, pulvinar
convallis augue. Morbi vel mass
libero. Mauris rutrum lorem vel
ipsum primis in faucibus. Etiam
habitasse platea dictumst.

Our output where we protect the image quality:


Cras mollis lorem arcu, pulvinar
convallis augue. Morbi vel mass
libero. Mauris rutrum lorem vel
ipsum primis in faucibus. Etiam
habitasse platea dictumst.

Limitations

Our code couldn’t retrieve each paragraph in reading order, it is believed that the hierarchy of cv2.findCoutours was the source of the problem, however, we couldn’t find any or develop our own solutions.

Other than that, given the limited information regarding the rules of determining what is a table, for example, “Is a paragraph surrounded by borders considered a table?” and another similar variation, our code would only hide a limited region around a border of a table, so if there would be edge cases like a paragraph which is surrounded by borders. Since our methodology is to cover a specific region of the border so that the text within the border would also be removed. If the intended region of interest within a table exceeds our pre-determined region, the output wouldn't be ideal