**Report of text detecting system**

In this assignment, i am going to **use camera on PC to detect and recognize text on image in real time.** Green rectangles are drawn around the text and the chosen texts are successful recognized and print on the image. However, some technical problem remain to be solved. This system can only be used for English text detection mission. Detection accuracy is easily affect by many factors, such as text clarity and ambient lighting. Limited by computing power, sometimes this system can only run in a low speed.

Here is the outline of the system code:

**1.create new virtual environment and activate it**

**2.pip install related package and set resource path**

**3.read image from camera**

**4.preprocess the image for both EAST model and pytesseract**

**5.create blob object and implement forward propagation**

**6.get output value from model**

**7.calculate the text border coordinate and filtrate with threshold**

**8.draw border on original image**

**9.use pytesseract to recognize text in chosen border(try on different image)**

**10.show image**

**11.shut down and release resource**

**Stage1&2: create new virtual environment and activate it**

During mylearning, i found different mission may rely on different version of package. pytesseract can only run on python3.7+, but there is only python3.6 in my anaconda. I create a new virtual environment in anaconda prompt and install python3.7\pytesseract\jupter notebook. Activate myenv and open jupyter notebook in command line.

**Stage3&4: read and preprocess image**

After trying detecting text on 1 piece of image successfully, i wonder if i can use PC camera for real time text detecting system since all i need is just a while loop. Preprocessing the image is relatively complex since using EAST model need to resize the image to fit the output format and gray image\ image blurred by gaussian blur\binary image may perform better in text recognition in pytesseract.

EAST model only receives square images and must be a multiple of 32 in size. So i record the size of original image and then resize it in 512\*512. Calculate the height-width ratio so i can modify the size of text border.

To improve the accuracy of text recognition, i turn original image to gray image and use gaussian blur to reduce noise and finally turn it into binary image. I try to adjust parameter like the size of gaussian core and threshold in binary image to pursue the best accuracy.

But quickly i found EAST model only accept image with 3 channel, and EAST so image processing above are not that useful in this project.

**Stage5: create blob object and implement forward propagation**

EAST is a DNN model used for text detection. The first thing i need to do is get the name of the two output layers. I use function to get the index of output layers and then use list comprehension to get the name.

Then i create blob object to convert original image to required format. Normalization can help improve the speed of the system. (123.68, 116.78, 103.94): These are the mean values used for mean normalization of image data. These values are typically calculated based on the dataset of the pre-trained model and are used to subtract the mean from the images for normalization.

Use blob as model input and implement forward propagation, model will return geometry and scores, put them into a tuple.

**Stage6&7: get output value and calculate the text border coordinate**

These two stage may be the most confusing part of the whole system code.

The output of EAST algorithm contain two different layers.feature\_fusion/Conv\_7/Sigmoid give the possibility of each pixel indicating if it is part of text region. feature\_fusion/concat\_3 give us the distance of each pixel to the bound and the angle. What i need to do is calculate the coordinate of bound so that i can draw the rectangle on the origin image.

First of all, i use for loop on rows and on column to get the output of east, for each pixel in characteristic pattern, we have its possibility and 4 distance and its angle.

If the possibility of a pixel is less than 0.8, which means that it is not a part of text region and it will be ignored. For the rest pixel, the coordinates on the feature map need to be converted back to the coordinates of the original image. In EAST, The feature map is a quarter of the size of the original image so offset(x,y)=4\*x , 4\*y.

**Calculate the width and height of the text box:**

**xData0：**The distance from the top boundary of the text box to the current pixel.

**xData1：**The distance from the right boundary of the text box to the current pixel.。

**xData2**：The distance from the bottom boundary of the text box to the current pixel.

**xData3：**The distance from the left boundary of the text box to the current pixel.

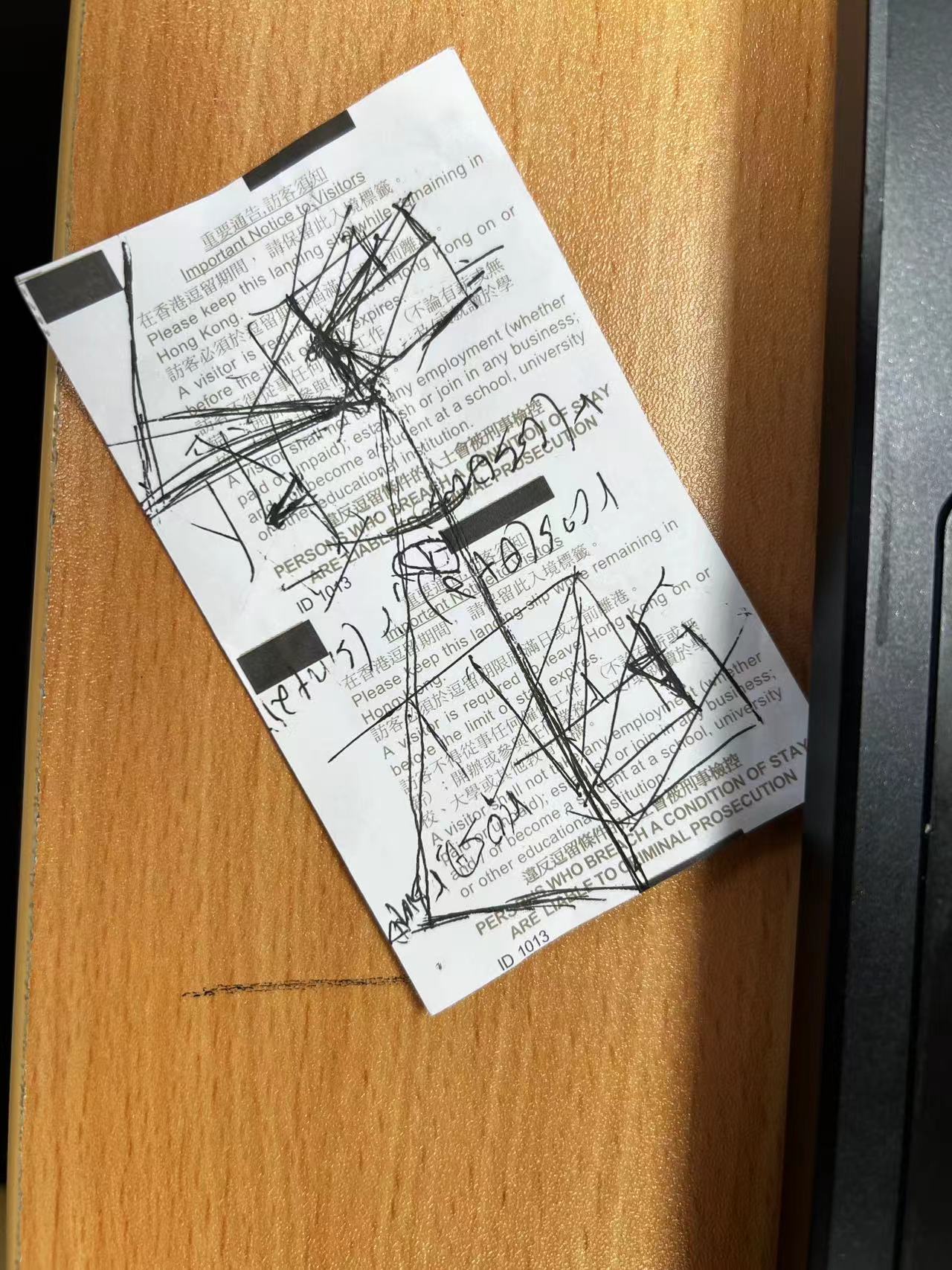
**anglesData：**spin angel of the text bounding

**box\_h:**  The height of the text box, calculated by xData0 and xData2.

**box\_w:**  The width of the text box, calculated by xData1 and xData3.

**Calculate the four coordinates of the text box:**

Use cosine and sine transformations to handle the rotation of the text box, and calculate the start and end coordinates of the text box in the original image (startX, startY, endX, endY).





Xdata2

Xdata[1]



**When angel is 0 :**

endX = offsetX + xdata1[x] and endY = offsetX + xdata2[x]

**When angel is not 0 :**

endX = int(offsetX + (cos \* xData1[x]) + (sin \* xData2[x]))  
 endY = int(offsetY - (sin \* xData1[x]) + (cos \* xData2[x]))

**Add Coordinates and Confidence:**

Add the calculated text box coordinates and the corresponding confidence score to the list.

**Non-maximum suppression (non\_max\_suppression):**

Use non-maximum suppression to filter out overlapping text boxes, leaving only those that are most likely to contain text.

**Traverse Final Text Boxes (for (startX, startY, endX, endY) in boxes):**

For each finalized text box, scales its coordinates proportionally back to the original image size and draws a rectangular box on top of the image.

**Stage8&9: draw text border and implement text recognition in the border**

For each finalized text box, it is plotted on the original image, and the area within the text box is text recognized using pytesseract. A processed image is displayed on the screen with a drawn text box and recognized text. In order for the printed text not to obscure the text box itself, I reduced the print coordinates of the text by 10 units along the y-axis, which means that the position of the text in the image will be shifted up by 10 units

**Stage10&11: show images and shut down**

Finally, I set the rollout condition to exit the loop if the user presses "q". Then free up the camera resource and close all OpenCV created windows.

**In summary, this text detection and recognition system is basically able to meet the needs, but I found that there are still problems in actual use:**

First of all, the operation of this system is very unsmooth, and the picture of the laptop camera is very stuck, and it is difficult to read the recognized text.

Secondly, the calculation of coordinates and extraction is very loose, and the meaning of the data output by the model is ambiguous, which leads to my lack of clarity in understanding the text box coordinates required for calculation, and can only use the method of calculating the length and width of the text box and the coordinates of the lower right corner of the text box. The combination of the symbols of this part of the formula is not the only correct combination.