# Text Detection from Image based on Tesseract OCR

## 0.Introduction

Text detection in computer vision is truly an important and also, useful technique nowadays. It is wildly used no matter in our daily life or public services like plate licenses detection in traffic videos. But today we’re only talking about text detection from image, since it’s kind of like the basic one in all the related techniques.

There are three parts while we're trying to deploy this technique. Firstly, if it’s the first time that you are trying to write your own detection, the environment you use need to be setup from the very beginning. And for me, I used JupiterLab alongside with libraries such as openCV and Tesseract. After the environment setup, There is a necessary step called pre-processing, which is because that among all the images, there always be something else in the picture that will effect the final result of text detection, so it is very important to transfer the image you would like to proceed with it the right form. In this step, the basic processes are turning RGB images into greyscale. And then, you have to use threshold statement to get the binary form. And finally, using filters like median filter or Gaussian filter can help to reduce noise.

## 1.Pre-processing

As we can see that the pre-processing of the image can directly affect the result of text detection, so I do put a lot of effort in this step. Threshold statement has so many choices for us to transfer greyscale images into binary images. In the threshold statement, you can use cv2. threshold（scr, thresh\_kernel, maxval,thresh\_type) to complete the process above. And in this statement, scr stands for the picture you would like to process with; The kernel is usually a number between 0 and 255, which is the threshold you setup for the process. Maxval is the value for the binary process, which is usually 255. For example, if you set the kernel to 80 and maxval to 255 and you proceed it in the most basic threshold method, the value of pixel above 80 would be replaced with 255, and the rest of them would be 0, that’s how a binary image being created from its greyscale form. And finally, threshold\_type is the method you use for binary processing. Here are the details for each of them.



pic 1.1 result of Mean Adaptive Threshold pic 1.2 result of Gaussian Adaptie Threshold

For different method of blur, I chose Median Filter and Gaussian Filter. Both of them are able to smooth the image and reduce the noise, but for Median Filter, it is a non-linear digital filtering technique used to remove noise from an image, and it works by moving a window (usually square) over the image and replacing the center pixel value with the median value of the pixel values within the window. And as for Gaussian Filter, is a linear filter that uses a Gaussian function to weigh the pixel values within a neighborhood. This filter smooths the image by averaging the pixel values, with the weights decreasing according to the Gaussian function as the distance from the center pixel increases.Here are the details for their difference.



pic 1.3 result of Median Filter pic 1.4 result of Gaussian Filter

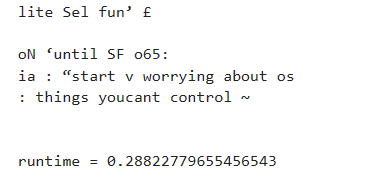
During the pre-processing, I found that after the procedure above, there are still many noises left in the image, and thus, the result is not so satisfactory. So with the help of the hint, I added dilate statement to help improving noise reduction.The statement cv2.dilate(src,k,iterations = \*) is using the certain image src, and choose the neighbourhood k, usually (3,3), to iterate \* times to dilate the image. Here's the difference.



pic 1.5 pic proceeded with Mean method of Adaptive Threshold



pic 1.6 pic proceeded with median blur without dilate



pic1.7 the text detection result of the blured pic above

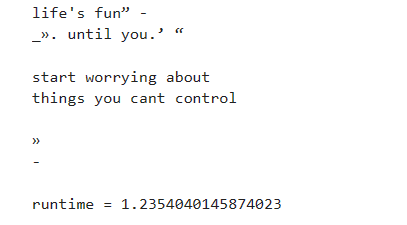
Here we can see that there are errors with lots of noise in the result of text detection, but the same binary picture with only one more step, the dilate process, we can see that the result is correct without error. However, the runtime and the amount of noise are both increased.



pic 1.8 pic after dilate process



pic 1.9 median blur result of pic after dilate process



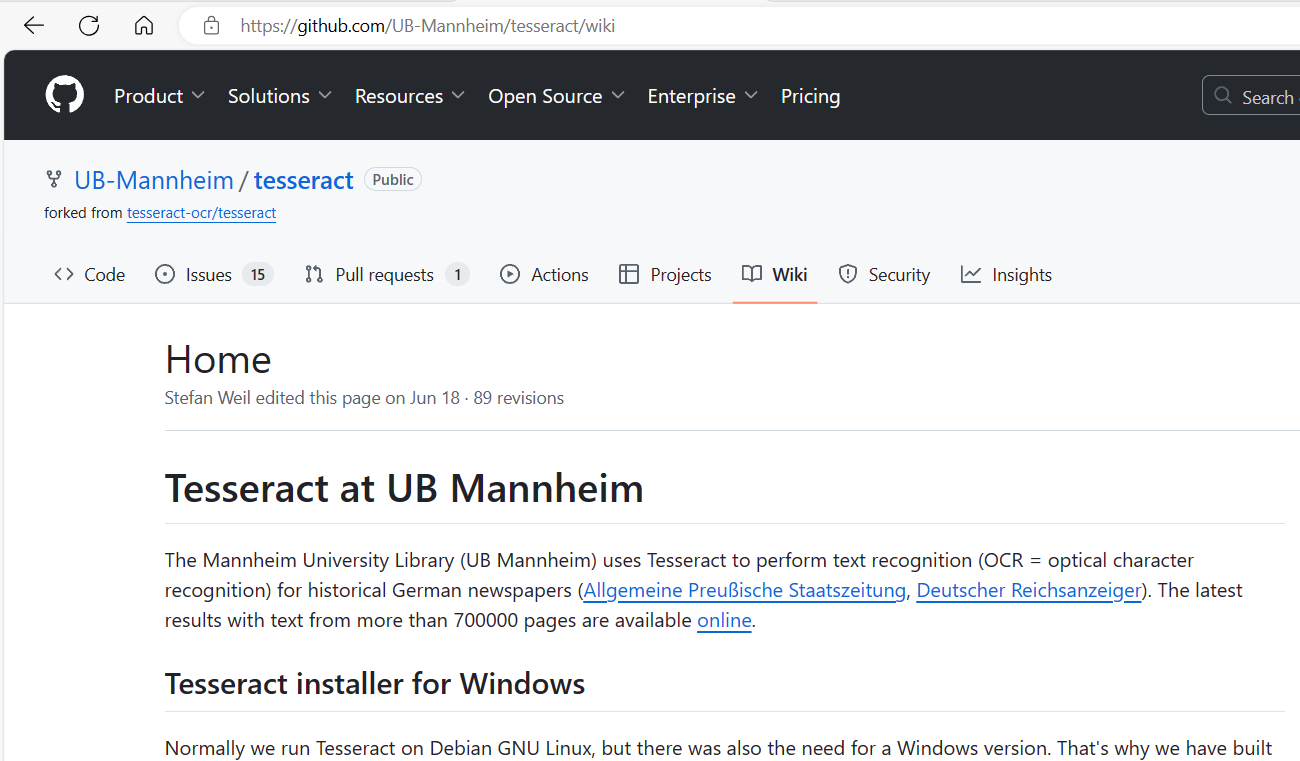
pic 1.10 text detection result of the result being proceeded with dilate process

## 2.Setup of Tesseract OCR

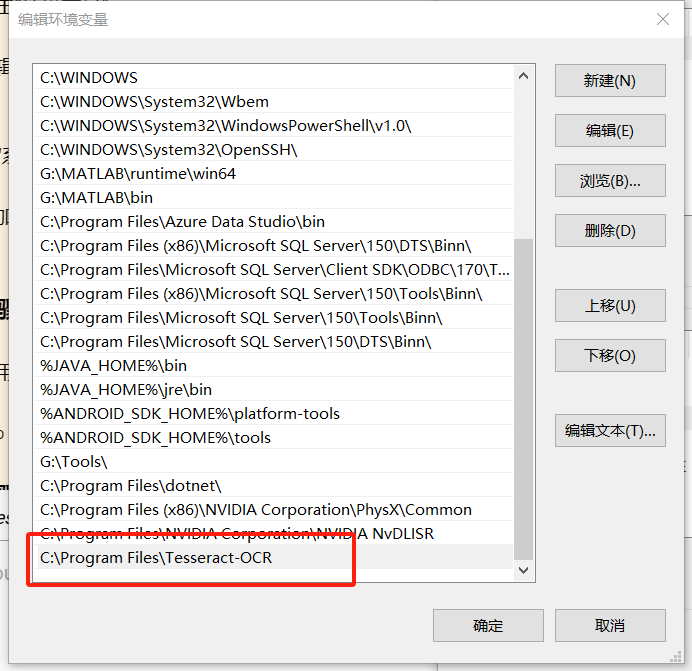
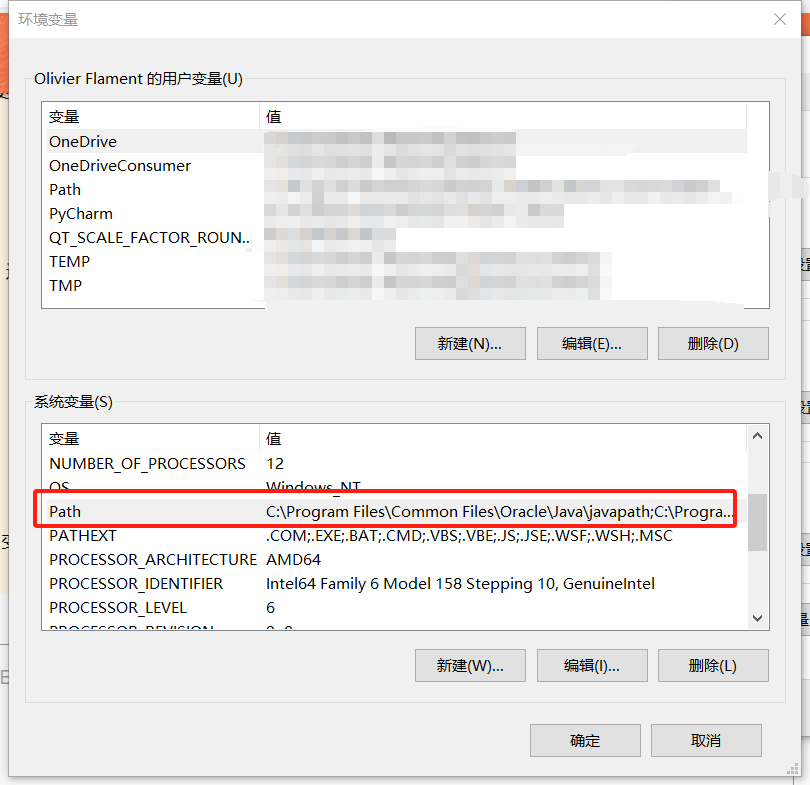
After that is the final step of this whole project, text detection. Usually we can use libraries such as Easy OCR and Tesseract OCR, but here we will take Tesseract OCR as an example.

Tesseract OCR is a library that you have to download the file from GitHub first, and here are some steps you need to do before using it.

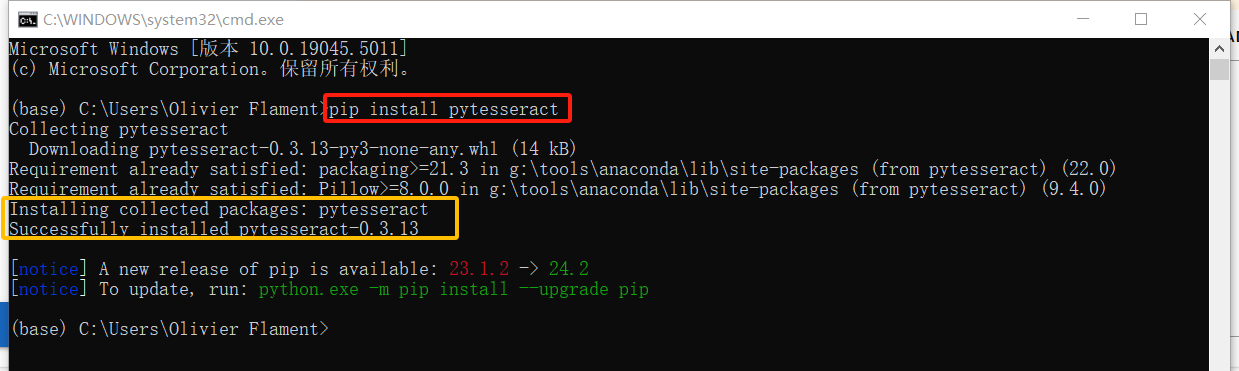
Step 1, search the GitHub page of Tesseract, and download the install package.



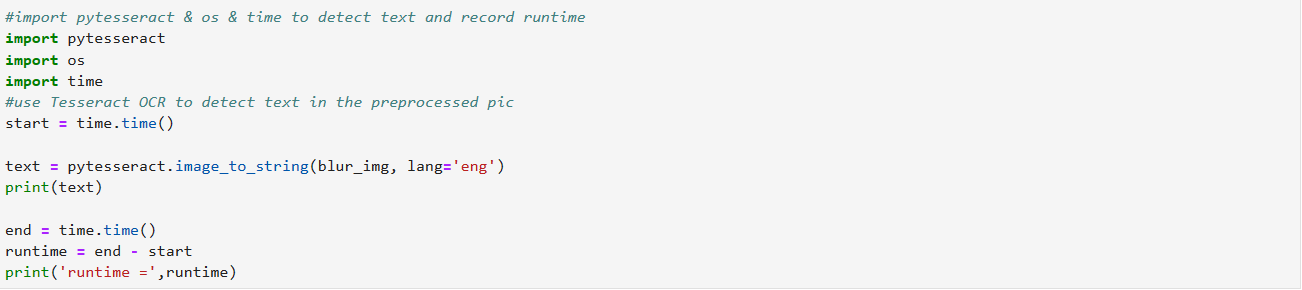
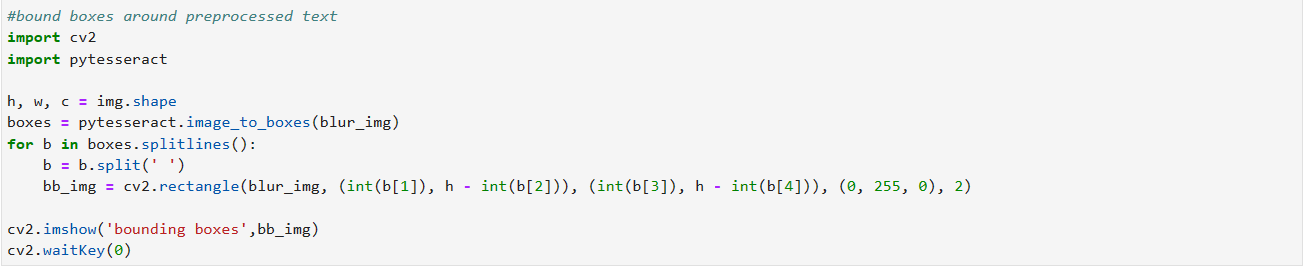
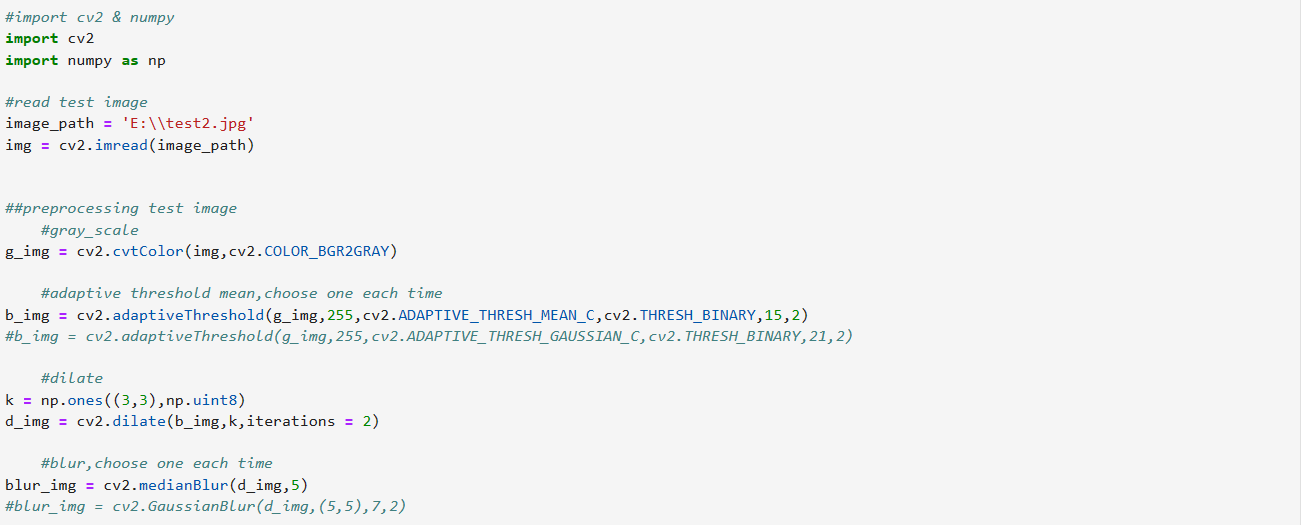
Step 2, install the package and set up the environment



Step 3, use statement ‘pip install pytesseract’ in the CMD.exe Prompt to install the library



## Code Presentation and Result



So in this code, firstly we pre-process the picture that we want to detect the text within with grayscale, threshold, dilate and filter process, then we bound boxes around text in preprocessed image to see the result of the text detection, and finally, we use the pytesseract.image\_to\_string statement to print the result into text.

And to figure out the best detection method, I tried different parametres, and here are the details.

So firstly, we can try out when using Mean Adaptive Threshold, using (3,3) neighbourhood to dilate the image 2 times or just dilate it 1 time with (5,5) neighbourhood; And for different filter, we can try to see the difference between Median Filter with value 5 and Gaussian Filter with (3,3) neighbourhood. And so with using Gaussian Adaptive Threshold.

Group Mean Adaptive Threshold(Group MAT)

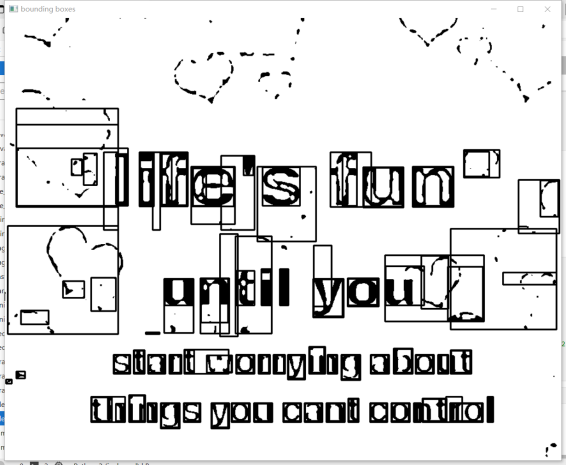
MAT 1: dilate 2 times with (3,3) neighbourhood, Median Filter.

-Image Pre-processing



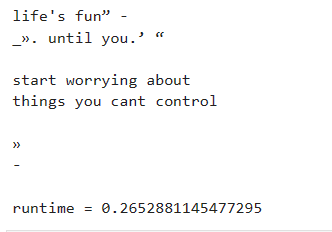
MAT 1.1 Mean Adaptive Threshold MAT 1.2 Dilate MAT 1.3 Median Filter

-Bound Boxes



MAT 1.4 Box bounding result

-Text Detection and runtime

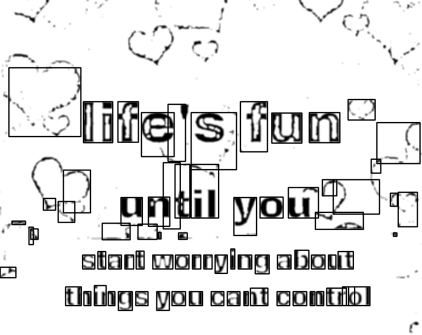


MAT 1.5 Text Detection result and run time

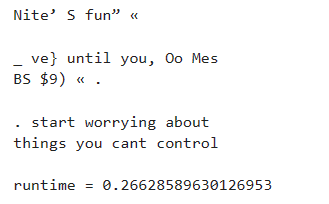
-MAT 2: dilate 2 times with (3,3) neighbourhood, Gaussian Filter.



MAT 2.1 Gaussian Filter



MAT 2.2 Box bounding result



MAT 2.3 Text Detection result and run time

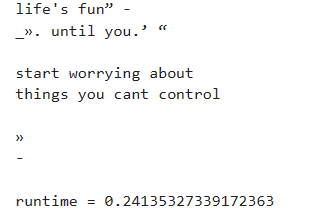
-MAT 3: dilate 1 time with (5,5) neighbourhood, Median Filter.



MAT 3.1 Dilate MAT 3.2 Median Filter



MAT 3.3 Box bounding result



MAT 3.4 Box bounding result

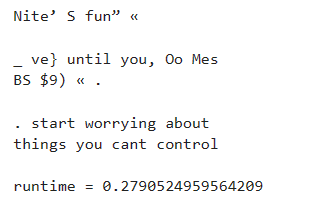
-MAT 4: dilate 1 times with (5,5) neighbourhood, Gaussian Filter.



MAT 4.1 Gaussian Filter



MAT4.2 Box bounding result



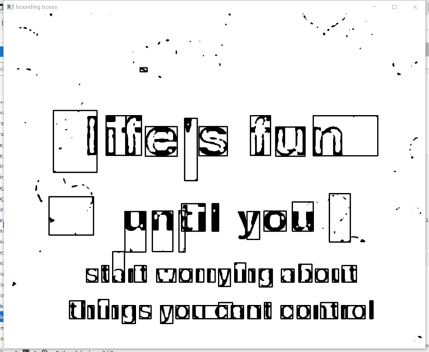
MAT 4.3 Text Detection result and run time

Group Gaussian Adaptive Threshold(Group GAT)

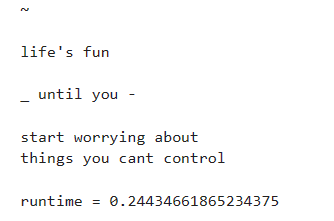
-GAT 1: dilate 2 times with (3,3) neighbourhood, Median Filter.



GAT 1.1 Gaussian Adaptive Threshold GAT 1.2 Dilate GAT 1.3 Median Filter



GAT1.4 Box bounding result



GAT 1.5 Text Detection result and run time

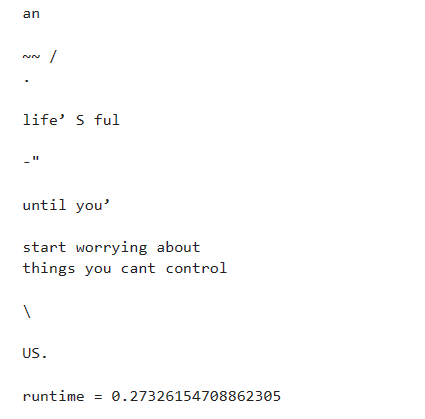
-GAT 2: dilate 2 times with (3,3) neighbourhood, Gaussian Filter.



GAT 2.1 Gaussian Filter



GAT 2.2 Box bounding result

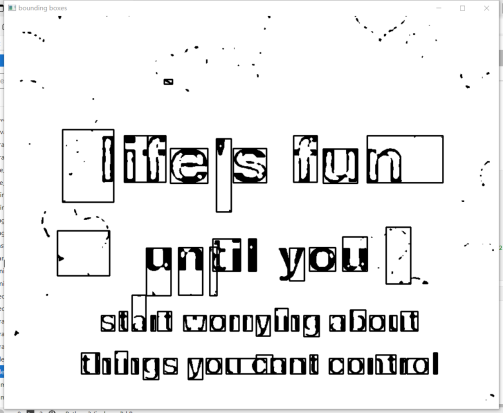


GAT 2.3 Text Detection result and run time

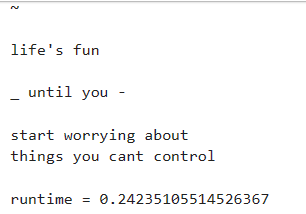
-GAT 3: dilate 1 time with (5,5) neighbourhood, Median Filter.



GAT 3.1 Dilate GAT 3.2 Median Filter



GAT 3.3 Box bounding result



GAT 3.4 Box bounding result

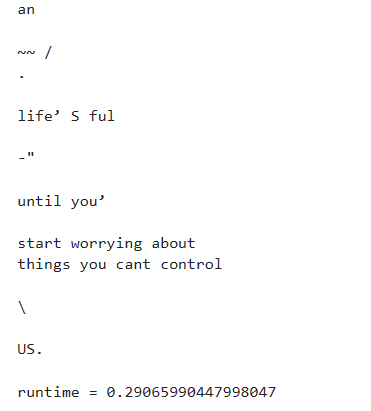
-GAT 3: dilate 1 time with (5,5) neighbourhood, Gaussian Filter.



GAT 4.1 Gaussian Filter



GAT4.2 Box bounding result



GAT 4.3 Text Detection result and run time

## Efficiency and Effectiveness Evaluation

So for the evaluation for each group, effectiveness is an important element to value, and for each result, correctness and noise are both needed to be considered.Correctness is the number of matching result between text detection result and the original text, and the noise is the string that not mentioned in the original text. The form below shows the evaluate method of effectiveness.

|  |  |  |
| --- | --- | --- |
| Element | Amount | Score |
| Correctness | 55 | 4 |
| 54 | 3 |
| 53 | 2 |
| below 53 | 0 |
| Noise: | 0-4 | 3 |
| 5-10 | 2 |
| above 10 | 0 |

And also, the runtime is the element that we can evaluate the efficiency. As the amount of data is not so large, I consider to rank them, and the quicker one gets a higher rank. The form below shows the evaluate method of efficiency.

|  |  |  |
| --- | --- | --- |
| Element | Rank | Score |
| Runtime | 1 | 4 |
| 2 | 3 |
| 3 | 2 |
| 4 | 0 |

## Result and Discussion

So with the evaluation method above, the score of each group in MAT and GAT is:

|  |  |  |  |
| --- | --- | --- | --- |
| Group | Effectiveness | Efficiency | Total Score |
| MAT1 | 4+2 | 3 | 9 |
| MAT2 | 0+0 | 2 | 2 |
| MAT3 | 4+0 | 4 | 8 |
| MAT4 | 0+0 | 0 | 0 |
| GAT1 | 4+3 | 3 | 10 |
| GAT2 | 2+0 | 2 | 4 |
| GAT3 | 4+3 | 4 | 11 |
| GAT4 | 2+0 | 0 | 2 |

From the form of score above, we can see that the best method is Gaussian Adaptive Filter with 1 time of (5,5) neighbourhood dilate and Median Filter. And also, we can see through the form is that, firstly, for Median Filter and Gaussian Fiter, Median Filter is more effecive and more efficient than the latter one; And despite there would have some different k parametre and dilate iterations that suits images proceed with different adaptive threshold method, 1 time of (5,5) neighbourhood dilate would always be more efficient in the step of text\_into\_strings.

## Reference

https://www.cnblogs.com/juzicode/p/15520806.html

https://blog.csdn.net/m0\_55320151/article/details/127192801

https://blog.csdn.net/u014003644/article/details/70239528

https://github.com/tesseract-ocr/tesseract