NANYANG TECHNOLOGICAL UNIVERSITY

SINGAPORE

CE4003 Computer Vision

Text Image Segmentation for Optimal Optical Character Recognition

Report by:

Elayne Tan Hui Shan

U1921730C

Table of Contents

1 Introduction

1.1 Objective

In this project, we conduct Optical Character Recognition (OCR) to detect and recognise texts in images through Computer Vision (CV). This involves text image binarization, connected component labelling and character recognition using classifiers.

We first implement the Otsu global thresholding algorithm for text image binarization, where we convert colour and grayscale images into binary images with multiple foreground regions. We then feed the binarized images to the OCR software to analyse the OCR accuracy, with the aim of evaluating the Otsu global thresholding algorithm.

For the second task of the project, we formulate algorithms and explore if they can address the limitations of the Otsu global thresholding algorithm.

Lastly, we also implemented the optional task, whereby we experiment on enhancing recognition algorithms for more robust and accurate character recognition.

Refer to *CE4003_Project.ipynb* for the main code implementation, and *otsu.py* and *wer.py* for the supporting user-defined functions. Outputs are stored in the folder *output*.

1.2 Tesseract

The OCR algorithm that will be used in this project is an open-source OCR software, Tesseract [1]. It is one of the most popular OCR engines with support for more than 100 languages.

1.3 OCR Accuracy Metric

Two common metrics used to evaluate OCR outputs are Character Error Rate (CER) and Word Error Rate (WER). We utilise error rates to measure the extent to which the OCR output text and the ground truth text differ from each other. We consider 3 different types of errors:

1. Substitution error: Misspelled characters/words
2. Deletion error: Lost or missing characters/words
3. Insertion error: Incorrect inclusion of characters/words

CER and WER makes use of Levenshtein distance to measure the extent of errors. It is a distance metric that computes the difference between two string sequences, and the minimum number of single character/word corrections required due to any of the three errors mentioned above. As the OCR output text varies more from the ground truth text, the number of corrections required increases and the Levenshtein distance becomes larger.

In this project, we will be using WER as our OCR accuracy metric as it is more suitable for our text images that are in paragraphs. WER is calculated by the formula $WER = \frac{S+D+I}{N}$, where S, D, I, and N corresponds to the number of substitutions, deletions, insertions, and the number of words in the ground truth text.

We define a Python function *wer(test, answer)* in *wer.py* to calculate the WER based on the formula above. Since we are using uint8, the WER calculation will only work for iterables up to 254 elements. test is the OCR output text string that is to be tested, while answer is the ground truth text string.

2 Basic analysis of images

2.1 sample01.png

Firstly, we read and show the image *sample01.png*.

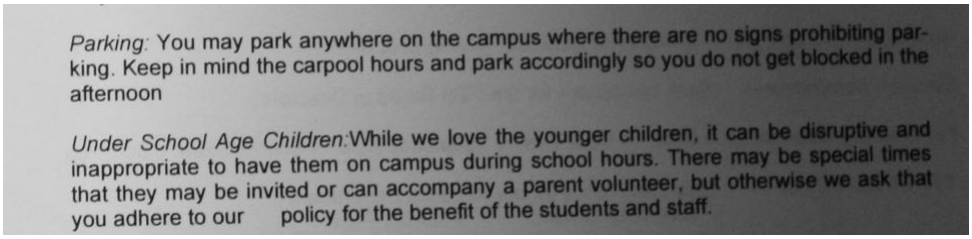

Figure 1: sample01.png

We then obtain the OCR output text (Figure 2) by using *pytesseract.image_to_string()*.

```
Parking: You may park anywhere on the cé
king. Keep in mind the carpool hours and park
afternoon

Under School Age Children:While we love
inappropriate to have them on campus @ J
that they may be invited or can accompany :
you adhere to our _ policy for the benefit of
```

Figure 2: OCR output text of sample01.png

We also evaluate the OCR accuracy using WER. The ground truth text is manually typed out for comparison purposes and is stored in *answer01.txt* and *answer02.txt* respectively for *sample01.png* and *sample02.png*. We obtain an OCR WER accuracy of 0.484.

This is approximately half, and we can also observe this from the OCR output text in Figure 2 where only the left portion of the image is detected. This is likely to be because the right portion of the image is too dark to be detected without going through any processing.

We plot the histogram of the original *sample01.png* (Figure 3) for analysis purposes.

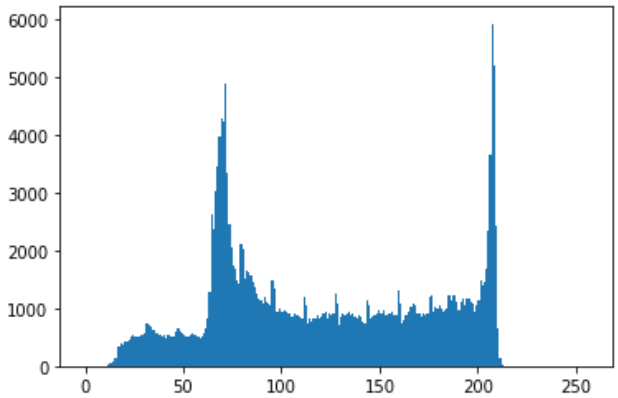

Figure 3: sample1_hist.png

The peaks at the two ends of the histogram support the point that there is a significant amount of difference in colour within the same image.

2.2 sample02.png

We repeat the same steps, from basic analysis to our self-designed algorithms, for *sample02.png*. We will only be showcasing the experiment results for *sample02.png*, and not provide thorough explanations for our steps since they are the same as *sample01.png*'s.

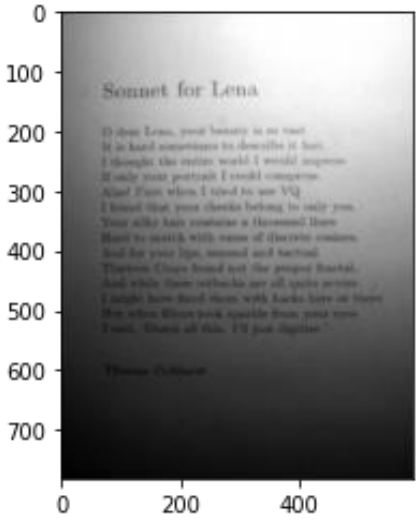

Figure 4: sample02.png

```
Sonnet for Lena
```

Figure 5: OCR output text for sample02.png

We obtain an OCR WER accuracy of 0.0259. This is much lower than the accuracy obtained for *sample01.png*, because *sample02.png* has even more drastic changes in colours, and some of the words at the bottom of the page are tedious to read even with our human eye.

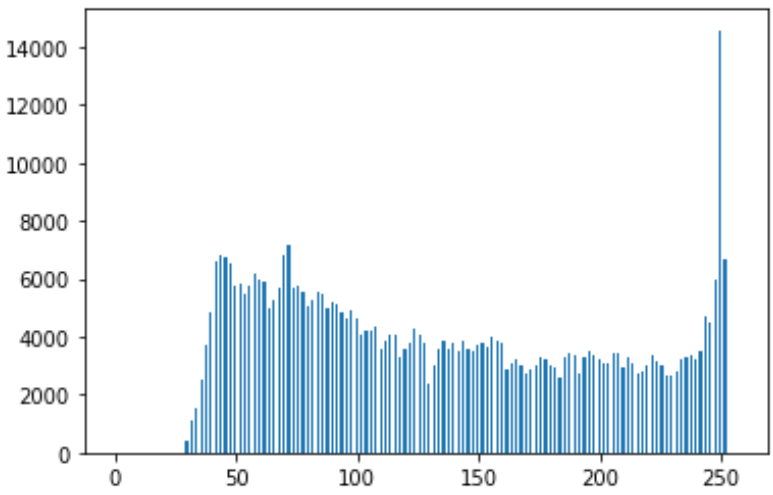

Figure 6: sample2_hist.png

The point mentioned above is supported by the histogram. The histogram for *sample02.png* occupies a wider range of bins as compared to *sample01.png*, and even has a distinct peak of up to 14,000, whereas the histogram for *sample01.png* only has values of up to 6,000.

3 Otsu global thresholding algorithm

In Otsu thresholding, the threshold value is automatically determined from the image histogram, instead of us having to decide on the threshold value. We define a Python file *otsu.py* that implements 3 different Otsu thresholding algorithms, starting off with the standard Otsu global thresholding algorithm *global_threshold(image)*.

It performs the following steps:

1. Generates the image histogram
2. Iterates over all bins of the histogram to obtain the respective probabilities
3. Calculates mean and variance of the classes and the corresponding interclass variance
4. Finds and returns the best threshold by comparing the interclass variances

After performing *global_threshold(image)*, we binarize the image and convert it to np.uint8 for ease of use with Tesseract.

3.1 sample01.png

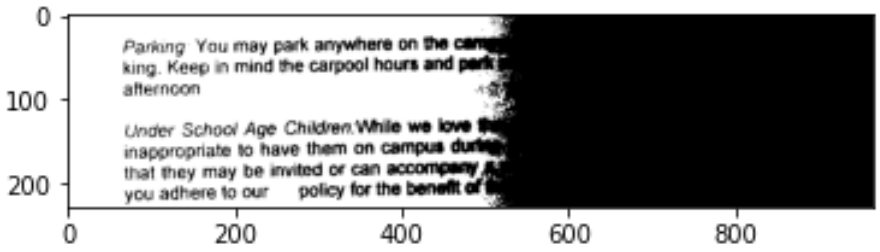

Figure 7: sample1_binarized.png

```
Parking You may park anywhere on the ct
king. Keep in mind the carpool hours and peri
afternoon

Under School Age Children:While we love
inappropriate to have them on campus @ .
that they may be invited or can accompany J
you adhere to our _policy for the benefit of
```

Figure 8: OCR output text of sample1_binarized.png

After implementing the Otsu global thresholding algorithm, we obtain an OCR WER accuracy of 0.473, which is similar to the accuracy obtained without the global thresholding. With reference to Figure 4 and 5, we can observe that the difference in colour between the left and right portion of the original *sample01.png* is too drastic. This led to our results from the global thresholding being similar to, or even slightly lower than, the one without thresholding. This is because likewise, only the left portion of the image can be detected.

3.2 sample02.png

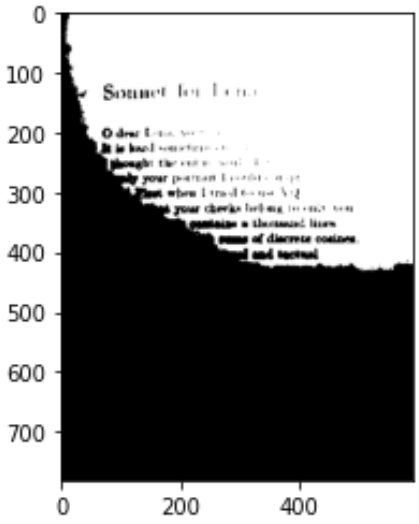

Figure 9: sample2_binarized.png

```
Sonnet for lens
```

Figure 10: OCR output text for sample2_binarized.png

We obtain an OCR WER accuracy of 0.0172. Like *sample01.png*, we obtain an accuracy that is slightly lower than the one without thresholding, due to the extreme difference in colour. The difference in accuracy for *sample02.png* and *sample2_binarized.png* is even wider as the colour differs more in *sample02.png*, as observed in our basic analysis.

From this experiment, we can deduce that the Otsu global thresholding algorithm is limited when the text images have drastically varying colours and applying the same threshold to the whole text image would not produce effective results.

4 Self-designed algorithms

After identifying the limitations of the Otsu global thresholding algorithm, we design more algorithms with the aim of addressing these limitations. We perform adaptive thresholding by implementing 2 additional variations of the Otsu global thresholding algorithm – segment thresholding and sliding window thresholding.

4.1 Segment thresholding

Segment thresholding is implemented by *segment_threshold(image, vert_seg, hori_seg)* in *otsu.*py, where *vert*_seg represents the number of vertical segments and *hori*_seg represents the number of horizontal segments. It performs the following steps:

1. Calculates segment sizes based on *vert_seg* and *hori_seg*
2. Runs for loops to iterate through all segments individually
3. Applies the Otsu global thresholding algorithm *(global_threshold(image))* to individual segments of the image

4.1.1 sample01.png

From our human eye observation of the original image *sample01.png*, we make a guess to implement 2 vertical segments and 3 horizontal segments.

Likewise, after we perform thresholding, we binarize the image and convert it for ease of use with Tesseract.

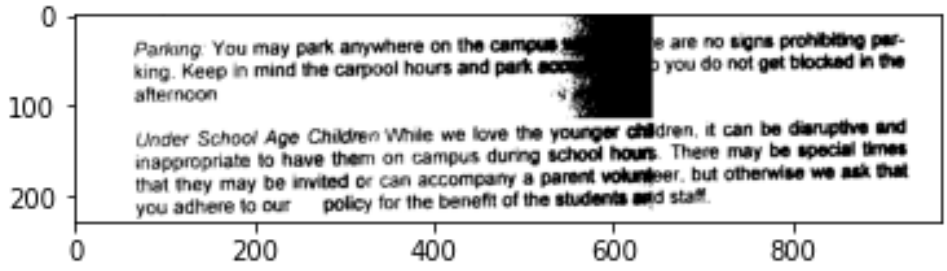

Figure 11: sample1_segment1.png

We observe that there is a black vertical line in the middle of the binarized image, which hints that we might need more horizontal segments. We then adjust the number of segments to be 2 vertical segments and 8 horizontal segments to see if it yields better results.

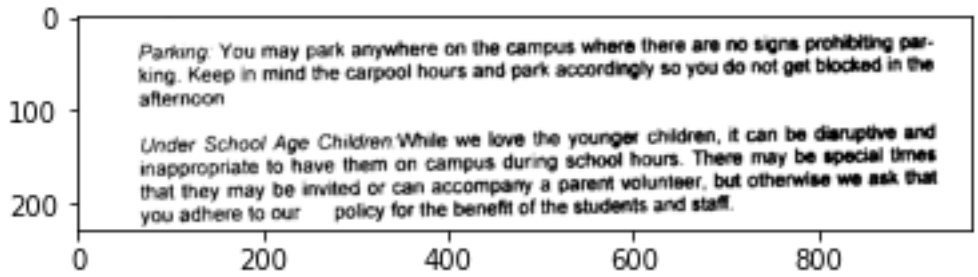

Figure 12: sample1_segment2.png

```
Parking: You may park anywhere on the campus where there are no
signs prohibiting par-
king. Keep in mind the carpool hours and park accordingly so you
do not get blocked in the
afternoon

Under School Age Children:While we love the younger children, it
can be disruptive and
inappropriate to have them on campus during school hours. There
may be special times
that they may be invited or can accompany a parent volunteer, bu
t otherwise we ask that
you adhere to our —_ policy for the benefit of the students and
staff.
```

Figure 13: OCR output text of sample1_segment2.png

We obtain an OCR WER accuracy of 0.978 but based on our manual comparison with the original image, the actual WER accuracy is 1.0, except for the '—_policy' portion, which may be caused by the borders of the image or other unknown external reasons. That '—_policy' portion might have caused the OCR WER accuracy to not be 1.0 and 0.989 instead.

We also investigate if splitting the image into more segments will lead to better accuracies by running the threshold with 2 vertical segments and 12 horizontal segments instead.

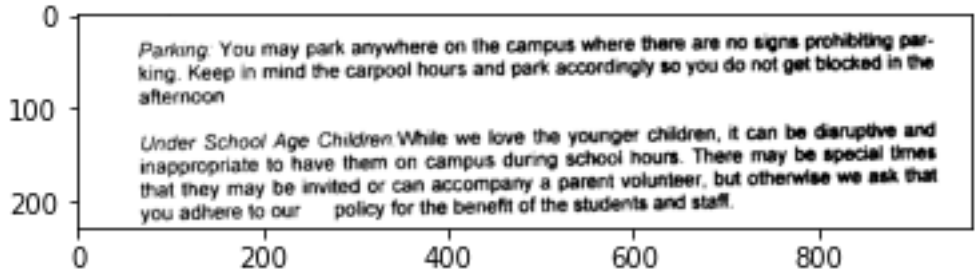

Figure 14: sample1_segment2.png

```
Parking: You may park anywhere on the campus where there are no
signs prohibiting par-
king. Keep in mind the carpool hours and park accordingly so you
do not get blocked in the
afternoon

Under School Age Children:While we love the younger children, it
can be disruptive and
inappropriate to have them on campus during school hours. There
may be special times
that they may be invited or can accompany a parent volunteer, bu
t otherwise we ask that
you adhere to our —_ policy for the benefit of the students and
staff.
```

Figure 15: OCR output text of sample1_segment2.png

The OCR WER accuracy obtained is the same – 0.978. This suggests that there will be no more increase in accuracy even if we continue to adjust the number of segments. Hence, we conclude the optimal number to be 2 vertical segments and 8 horizontal segments for *sample01.png*.

4.1.2 sample02.png

From surface observation of the original image *sample02.png*, we make a guess to implement 10 vertical segments and 8 horizontal segments.

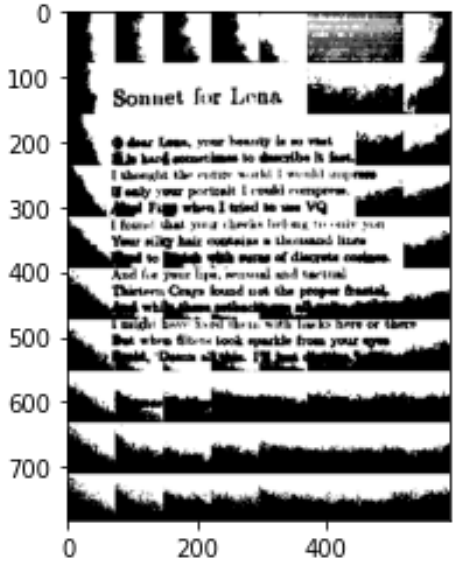

Figure 16: sample2_segment1.png

```
EER.

Sonnet for Lena



deer [eus, your heanty ls po vast
hard point imas to dmacribe It iaat,|
I thonght the entice work! F would aipinee
Wealy our portrat [could ramprew.
Aiegl Figg when 1 tried bo use YQ



Dfound that ving checks belong to oniy yon
Yeour silityhair couteinn @ thotssod lines Ps
40 [fied ith warns of diacrete conimes.
. And fur your Lipa, sensual and tectuel
i hitter; Crays found wot the proper frastal,
aly

sade 2 Then Sth Tees Aer' oF
gp Bet when §item look sparkle from your ayes ,
7" ae
```

Figure 17: OCR output text of sample2_segment1.png

We obtain an OCR WER accuracy of 0.233. We observe that there are black vertical and horizontal lines in the binarized image, which hints that we might need more segments. Hence, we adjust the number of segments to see if it yields better results.

We now run segment thresholding with 40 vertical segments and 30 horizontal segments.

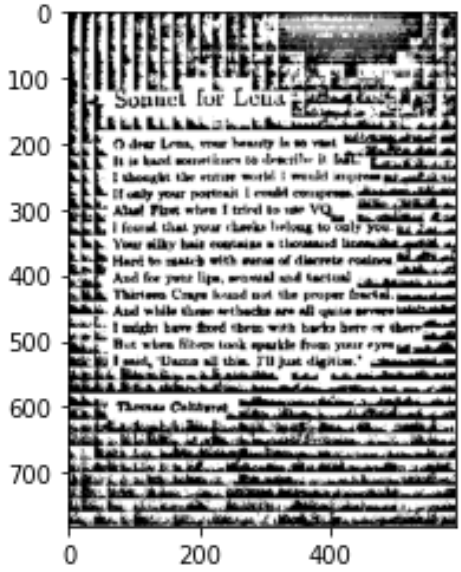

Figure 18: sample2_segment2.png

```
TLE 2:

Sonne fur Leua ae
```

```
" dear Lena, your beauty Inn vent Ml
* Tt ia hard soinetimes to clescribe it fat
- Pebought Uke entire world 1 would impromeh abate

sahil

Tf galy your portrait | exuld compress. «
Ale Firet when f tried to uae V4

1 found that your cheeks Lelong to only ‘you. te
Your silky bait copteina @ thotussud
Hard to match with wurde of discrete coaines.
And for yur lips, seoal and tectual
f Thirteen Crays found not the proper ape
- And while thre acthacks are all vyclle pevers uate
eek Cf might have fhved them with hacks bere or there ier
3. But when Filter took sparkle from your ee
i saicl, ‘Damo all thie. I'l) give.” nites aia
```

Figure 19: OCR output text of sample2_segment2.png

We obtain an OCR WER accuracy of 0.379.

We also investigate if splitting the image into more segments will lead to better accuracies by running the threshold with 45 vertical segments and 35 horizontal segments instead.

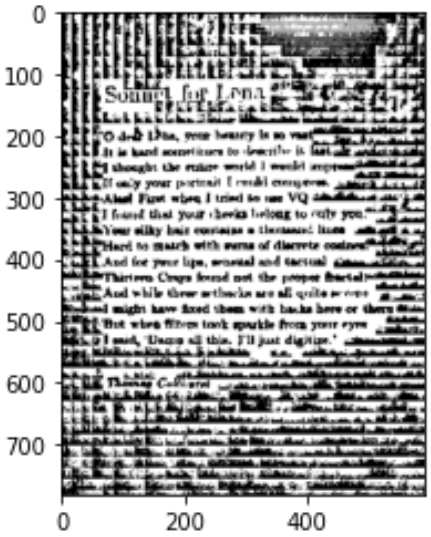

Figure 20: sample2_segment3.png

```
rs] was thee penn or bent Into

J ia ard ainelinuca to cdeacribe it 7

1 thought the entire world | erould impreni aman

HW guly your portrait [ rouki compress, PP were

pA bead Firet when J tried ta we ¥Q tld thy

‘gl found that your cheeks belong to Oty you:

i ‘Your silky hair contaias a truissn lines i

TeeHand to match sins ee
vAnd for your lips, seomal and tectual

SThirters Crays found not the preper rectal meiiadiilideiehs
And. while three eethacke ert all quite eecveny eee

d might have fixed tbem with backs here or therg Mitzi
```

Figure 21: OCR output text of sample2_segment3.png

We obtain an OCR WER accuracy of 0.25 A further increase in number of segments does not increase the accuracy. Therefore, we deem 40 vertical segments and 30 horizontal segments as optimal for *sample02.png*.

4.2 Sliding window thresholding

Sliding window thresholding is implemented by *sliding_window_threshold(image, wind_height, wind_width, vert_step, hori_step)* in *otsu.*py, where *wind_*height and wind_*width* represent the height and width of the sliding window, and *vert_step* and *hori_step* represents the number of vertical and horizontal steps to take. It performs the following steps:

1. Runs for loops to iterate through all sliding windows individually
2. Applies the Otsu global thresholding algorithm *(global_threshold(image))* to individual windows of the image
3. Keeps count of the number of times the Otsu global thresholding algorithm is applied on each portion of the image
4. Generates and returns the threshold by getting average

There is a need to keep count of the number of times the algorithm is repeated on various portions as the sliding windows overlap, and hence we utilise the count to generate the average at the end.

4.2.1 sample01.png

We use the information from segment thresholding to determine the optimal height and width of our sliding window. Based on our calculations, the optimal window height and width is 115 and 121 respectively. We start off with 8 as the number of vertical and horizontal steps.

We perform sliding window thresholding and once again, binarize and convert the image.

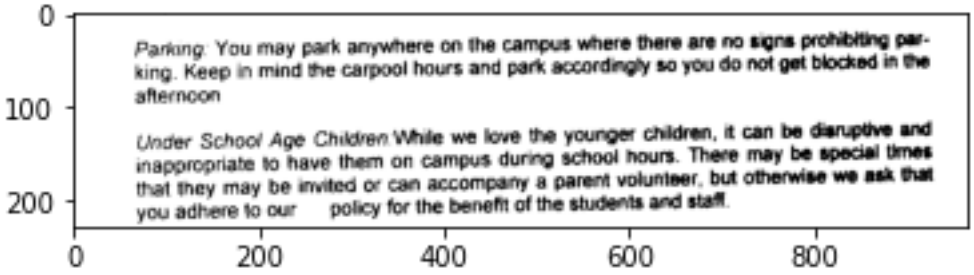

Figure 22: sample1_window1.png

```
Parking: You may park anywhere on the campus where there are no
signs prohibiting par-
king. Keep in mind the carpool hours and park accordingly so you
do not get blocked in the
afternoon

Under School Age Children:While we love the younger children, it
can be disruptive and
inappropriate to have them on campus during school hours. There
may be special times
that they may be invited or can accompany a parent volunteer, bu
t otherwise we ask that
you adhere to our —_—policy for the benefit of the students and
staff.
```

Figure 23: OCR output text of sample1_window1.png

We obtain an OCR WER accuracy of 0.989.

We further explore on how the number of vertical and horizontal steps will affect the OCR accuracy.

With 24 as the number of vertical and horizontal steps, we obtain the following results:

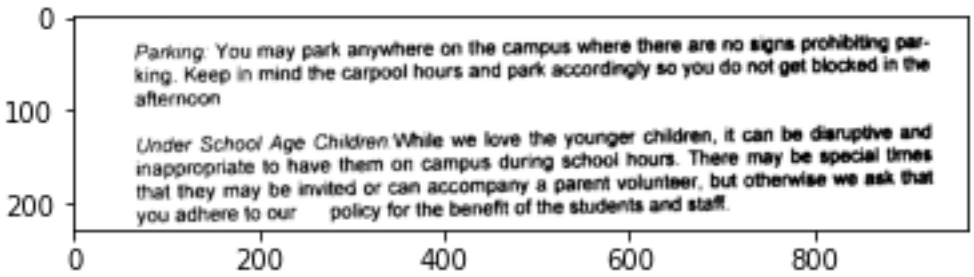

Figure 24: sample1_window2.png

```
Parking: You may park anywhere on the campus where there are no
signs prohibiting par-
king. Keep in mind the carpool hours and park accordingly so you
do not get blocked in the
afternoon

Under School Age Children:While we love the younger children, it
can be disruptive and
inappropriate to have them on campus during school hours. There
may be special times
that they may be invited or can accompany a parent volunteer, bu
t otherwise we ask that
you adhere to our _policy for the benefit of the students and st
aff.
```

Figure 25: OCR output text of sample1_window2.png

We obtain the same OCR WER accuracy of 0.989. This suggests that further increasing the number of vertical and horizontal steps will not yield better accuracies. Hence, we deem 115, 121, 8, and 8 to be the optimal parameters corresponding to *sample01.*png's *wind_height, wind_width, vert_step,* and *hori_step* respectively.

4.2.2 sample02.png

Previously, we deemed 40 vertical segments and 30 horizontal segments as optimal. Likewise, we use this information from segment thresholding to determine the optimal height and width of our sliding window. Based on our calculations, the optimal window height and width are 20. We start off with 4 as the number of vertical and horizontal steps.

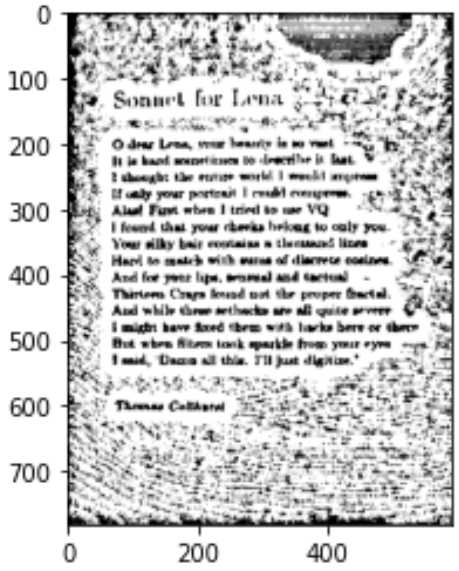

Figure 26: sample2_window1.png

```
ge SSE
your heanty js oo vert
Ht ia bard animetioes to cleecribe it fast.
Pitbought Ube eritire worl | would impress”
Tf only your portrait | could compress,
| Alaa! Fizet when J tried to use VQ
[ found that, your chesis belong te only you.
‘Your silky bait contadinn « Unvisnod lines
Hard to match with sutaa of discrete cosines.

And for your lips, sensual and tactual ~
Thirteen Craya found not the proper fractal.

! Ad while then setbacks ane all quite severe Fig
t might bave fixed them with hacke bere or there
But when Alter: tonk sparkle from your cyra
Yasid, ‘Damo oll this, I'll Just digitize.” 2%

ee
```

Figure 27: OCR output text of sample2_window1.png

We obtain an OCR WER accuracy of 0.466. We further explore on how the number of vertical and horizontal steps will affect the OCR accuracy.

With 12 as the number of vertical and horizontal steps, we obtain the following results:

Figure 28: sample2_window2.png

```
a O dear Lens, your heauty ja eo ae; Se

4 ft is bart conn hines to cleacribe it fast.

a Vr ehonght uhe entire world 1 weuld impr
if Lf only your portrait | could cinpres,

4 Alas! First when [tried to use YQ 2:

1 found that, your cheeks belong te only you.

Your silky bait coutaion a Uhoisand lines “4



Thirteen Crays fownd not the proper fractal,
S And while thew setbacks are all quite aevere 9°
a 1 might bave fixed them with backe here or therr
But when Alter took sparkle from your eyes
sais) Sasid, ‘Dasa all :
```

Figure 29: OCR output text of sample2_window2.png

We obtain a lower OCR WER accuracy of 0.371.

Since *sample02.*png originally has a much longer height than width, we then experiment if setting the number of vertical and horizontal steps to be 12 and 8 respectively will affect the results. The results are as follows:

Figure 30: sample2_window3.png

```
fi ia hard sometimes to ceeczibe it fast,
Pebought Ube entire world | would impream |
If only your portrait Traub! compress, ree
Alaa! First when 1 tried to use VQ > 2273
[found that your cheeks belong to only y ‘you.
Your silky bait coutadon a those! lines“
© Hard to match with mums of discrete cosines.

And for your lips, sensual and tactuad - ---
Thirteen Crays fownd not the proper fractal,
And while these setbacks are all quite aercre “9°
‘oc. | might bave fixed them with hacks here or there
But when Aitem took mparkle from your even
. . 1 just digi i
```

Figure 31: OCR output text of sample2_window3.png

This set of parameters produced the highest OCR WER accuracy of 0.491.

As a result, we deem that 20, 20, 12, and 8 to be the optimal parameters for *sample02.png*'s *wind_height, wind_width, vert_step,* and *hori_step* respectively.

5 Enhancing recognition algorithms

5.1 Objective

Text images are often affected by varying kinds of image degradation. In this section, we will experiment on enhancing recognition algorithms for more resilient and precise character recognition.

Tesseract performs image processing operations internally, using the Leptonica library), before performing the actual OCR [2]. Conventionally, Tesseract does this well but in certain cases a lack of image processing will lead to a notable decrease in accuracy.

Hence, in this section we conduct various image processing techniques with the aim of achieving higher OCR accuracy. Most of the techniques are processed on *sample02.png* as *sample01.png* achieved a high accuracy of 0.989 after going through segment thresholding.

5.2 Noise Removal

Noise is the random variation of brightness or colour in an image that increases the difficulty of reading the image text. Certain kinds of noise are not removable by Tesseract; hence we reduce the noise in the text images before inputting it to the Otsu threshold algorithms.

5.2.1 Median filter

We run a median filter on *sample02.png* by using *scipy.ndimage.median_filter()*.

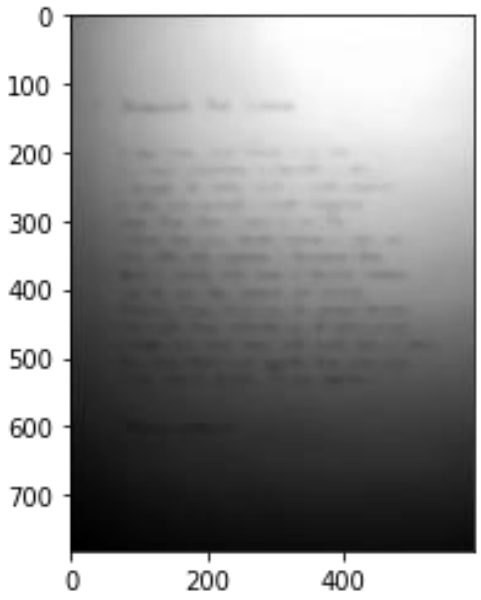

Figure 32: sample2_median.png

We can observe that the details are smoothened out and are not detectable anymore. Hence, we obtain a blank OCR output text and attain an OCR WER accuracy of 0. Median filters are not suitable for such text images as they reduce the visibility of the details, which is what we require to detect texts.

5.2.2 Gaussian filter

We run a Gaussian filter on *sample02.png* by using *scipy.ndimage.gaussian_filter()*.

We first use a sigma of 0.5.

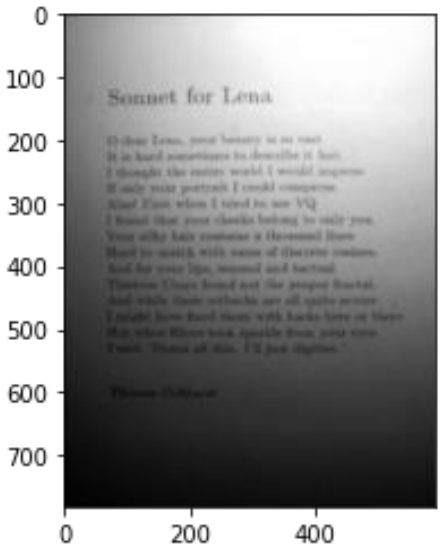

Figure 33: sample2_gaussian.png

```
Sonnet for Le:

O dear Lena

VQ
% to only you
```

Figure 34: OCR output text of sample2_gaussian.png

We obtain an OCR WER accuracy of 0.0603 with a sigma of 0.5. We then test out various sigma values to observe how it will affect our images and accuracies.

After testing various sigma values, we eventually decide on a Gaussian filter with a sigma value of 0.05. We utilise sliding window thresholding with parameters 20, 20, 12, and 8 as window height, window width, vertical step, and horizontal step respectively as this set of parameters produced the highest accuracy previously.

Figure 35: sample2_gaussian_window.png

```
fi ia hard sometimes to ceeczibe it fast,
Pebought Ube entire world | would impream |
If only your portrait Traub! compress, ree
Alaa! First when 1 tried to use VQ > 2273
[found that your cheeks belong to only y ‘you.
Your silky bait coutadon a those! lines“
© Hard to match with mums of discrete cosines.

And for your lips, sensual and tactuad - ---
Thirteen Crays fownd not the proper fractal,
And while these setbacks are all quite aercre “9°
‘oc. | might bave fixed them with hacks here or there
But when Aitem took mparkle from your even
. . 1 just digi i
```

Figure 36: OCR output text of sample2_gaussian_window.png

We obtained an OCR WER accuracy of 0.491, which is the same as previously.

This suggests that median and Gaussian filters are not suitable for such images.

5.3 Dilation and Erosion

Bold or thin characters affect the recognition of details and sometimes lead to a drop in recognition accuracy. In this case, we allow edges of the characters to dilate and grow. We perform dilation using OpenCV. We first define our kernel by calling *np.ones((2,2), np.uint8)*.

We conduct dilation on the original *sample02.png* by using *cv2.dilate()* with our defined kernel.

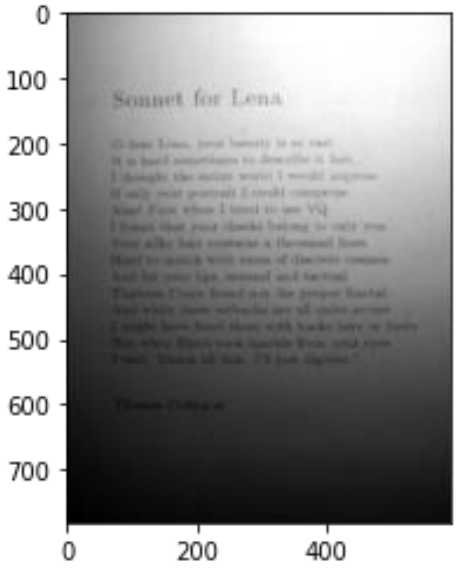

Figure 37: sample2_dilation.png

```
Sonnet

1 dear Lena

Wi is bard sone
0 the ws
fy your portrait [cor
when | tried to use
```

Figure 38: OCR output text of sample2_dilation.png

We obtain an OCR WER accuracy of 0.0862, which is higher than the original OCR WER accuracy of 0.0259 for *sample02.png*.

5.4 Rotation

OCR accuracy of text images might also be affected by how skewed the image is. When the text image is too skewed, it significantly decreases the quality of Tesseract's line segmentation. For this portion, we will be performing rotation on sample01.png instead of sample02.png as sample02.png is not skewed whereas we can tell that sample01.png is slightly skewed. We do so by using OpenCV, calling *imutils.rotate()* with an angle of -0.8, which is determined by trial and error.

We then perform segment thresholding on the rotated image with the previously determined optimal parameters of 2 vertical segments and 8 horizontal segments.

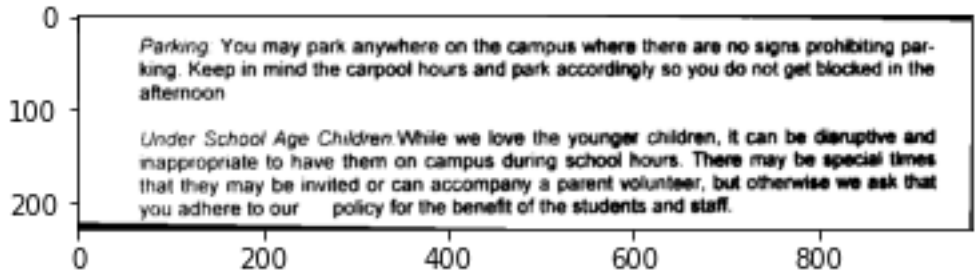

Figure 39: sample1_rotated.png

```
Parking: You may park anywhere on the campus where there are no
signs prohibiting par-
king. Keep in mind the carpool hours and park accordingly so you
do not get blocked in the
afternoon

Under School Age Children:While we love the younger children, it
can be disruptive and
inappropriate to have them on campus during school hours. There
may be special times
that they may be invited or can accompany a parent volunteer, bu
t otherwise we ask that
you adhere to our _ policy for the benefit of the students and s
taff.
```

Figure 40: OCR output text of sample1_rotated.png

We can observe from Figure 40 that the space between 'our' and 'policy' in the second last line, which was previously identified as a problem, is now resolved in the rotated image. The OCR output text now accurately identifies the space as '_' and achieves an OCR WER accuracy of 1.0.

6 Conclusion

In conclusion, there are various approaches to improve the OCR accuracy. Pre-algorithm approaches include working on image enhancement and processing our images before feeding them into the algorithms. We can also implement enhancements to our algorithms, such as using adaptive thresholding. Different practices render varying outcomes, and it boils down to finding the most suitable approach for our given image.

7 References

[1] Tesseract, “Tesseract Open Source OCR Engine,” November 2021. [Online]. Available: https://github.com/tesseract-ocr/tesseract.

[2] Tesseract, “Improving the quality of the output,” November 2021. [Online]. Available: https://tesseract-ocr.github.io/tessdoc/ImproveQuality.html.

[3] OpenCV, “Image Thresholding,” 19 November 2021. [Online]. Available: https://docs.opencv.org/3.4/d7/d4d/tutorial_py_thresholding.html.

[4] K. Leung, “Evaluate OCR Output Quality with Character Error Rate (CER) and Word Error Rate (WER),” 24 June 2021. [Online]. Available: https://towardsdatascience.com/evaluating-ocr-output-quality-with-character-error-rate-cer-and-word-error-rate-wer-853175297510.

[5] T. Anh, “Word Error Rate (WER) and Word Recognition Rate (WRR) with Python,” 2019. [Online]. Available: https://holianh.github.io/portfolio/Cach-tinh-WER/.

[6] GeeksforGeeks, “Erosion and Dilation of images using OpenCV in python,” 25 June 2021. [Online]. Available: https://www.geeksforgeeks.org/erosion-dilation-images-using-opencv-python/.

[7] AskPython, “2 ways to rotate an image by an angle in Python,” 2021. [Online]. Available: https://www.askpython.com/python/examples/rotate-an-image-by-an-angle-in-python.

8 Source Code

8.1 otsu.py

```
import cv2
import numpy as np

'''
Standard Otsu global threshold algorithm
cv2.threshold(image, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU,)
'''
def global_threshold(image):
    channels = [0]  # grayscale - [0]
    mask = None
    histSize = [256]
    ranges = [0, 256]
    hist = cv2.calcHist([image], channels, mask, histSize, ranges, )

    # iterate over all and get probabilities
    weights = [num_bin[0] / image.size for num_bin in hist]

    # init
    result = -1
    min_var = float('inf')

    for threshold in range(len(hist)):
        weight1 = sum(weights[:threshold + 1])
        weight2 = sum(weights[threshold + 1:])

        if weight1 != 0 and weight2 != 0:
            # class mean and variance for class1
            mean1 = sum([pixel_val * weight for pixel_val, weight in
                         enumerate(weights[:threshold + 1])])
            mean1 /= weight1
            var1 = sum([((pixel_val - mean1) ** 2) * weight for pixel_val,
                        weight in enumerate(weights[:threshold + 1])])
            var1 /= weight1

            # class mean and variance for class2
            mean2 = sum([pixel_val * weight for pixel_val, weight in
                         enumerate(weights[threshold + 1:])])
            mean2 /= weight2
            var2 = sum([((pixel_val - mean2) ** 2) * weight for pixel_val,
                        weight in enumerate(weights[threshold + 1:])])
            var2 /= weight2

            # find best threshold
            interclass_var = weight1 * var1 + weight2 * var2
            if interclass_var < min_var:
                result = threshold
                min_var = interclass_var

    # return image with same shape but with fill_value as result threshold
    return np.full_like(image, fill_value=result)


'''
Otsu algorithm performed on segmented blocks of the image
vert_seg: number of vertical segments
hori_seg: number of horizontal segments
```

```
'''
def segment_threshold(image, vert_seg, hori_seg):
    height, width = image.shape
    # calculate segment sizes based on number of segments in each direction
    seg_height = height // vert_seg + 1
    seg_width = width // hori_seg + 1

    # init
    threshold = np.zeros_like(image)

    # how much to offset for 1 segment in each direction
    for vert_offset in range(0, height, seg_height):
        for hori_offset in range(0, width, seg_width):
            # retrieve only that segment of the image
            segment = image[vert_offset: seg_height + vert_offset,
                      hori_offset: seg_width + hori_offset]

            # apply global Otsu threshold to that segment
            segment_threshold = global_threshold(segment)
            threshold[vert_offset: seg_height + vert_offset,
            hori_offset: seg_width + hori_offset] = segment_threshold

    return threshold


'''
Otsu algorithm performed on sliding window over image
wind_height, wind_width: height and width of window
vert_step, hori_step: number of steps to take in each direction
'''
def sliding_window_threshold(image, wind_height, wind_width, vert_step,
hori_step):
    height, width = image.shape

    # init
    threshold, count = np.zeros_like(image, dtype=np.float64),
np.zeros_like(image)

    # how much to offset for 1 window in each direction
    for vert_offset in range(0, height - wind_height + vert_step,
vert_step):
        for hori_offset in range(0, width - wind_width + hori_step,
hori_step):
            # retrieve only that window of the image
            window = image[vert_offset: wind_height + vert_offset,
                     hori_offset: wind_width + hori_offset]

            # apply global Otsu threshold to that window
            window_threshold = global_threshold(window)
            threshold[vert_offset: wind_height + vert_offset,
            hori_offset: wind_width + hori_offset] +=
np.mean(window_threshold)

            # how many times it is repeated (due to overlap)
            # so that we can get average later on
            count[vert_offset: wind_height + vert_offset,
            hori_offset: wind_width + hori_offset] += 1

    return threshold / count
```

8.2 wer.py

```
import numpy

'''
Calculate WER with Levenshtein distance
Works only for iterables up to 254 elements (uint8)
test: string to be tested for accuracy
answer: string with correct answer
'''
def wer(test, answer):
    # initialisation
    test = test.split()
    answer = answer.split()
    d = numpy.zeros((len(test)+1)*(len(answer)+1),
                    dtype=numpy.uint8)
    d = d.reshape((len(test)+1, len(answer)+1))

    for i in range(len(test)+1):
        for j in range(len(answer)+1):
            if i == 0:
                d[0][j] = j
            elif j == 0:
                d[i][0] = i

    # computation
    for i in range(1, len(test)+1):
        for j in range(1, len(answer)+1):
            if test[i-1] == answer[j-1]:
                d[i][j] = d[i-1][j-1]
            else:
                substitution = d[i-1][j-1] + 1
                insertion = d[i][j-1] + 1
                deletion = d[i-1][j] + 1
                d[i][j] = min(substitution, insertion, deletion)

    num_errors = d[len(test)][len(answer)]
    total_words = len(answer)
    num_correct = total_words - num_errors
    return num_correct/total_words
```

8.3 CE4003_Project.ipynb

CE4003_Project.ipynb is unable to be appended, as it is a Jupyter notebook, and hence will be zipped in the file along with the other source codes, text files, and output images.