CZ4003 Project

Report

Hans Tananda U1720251K

Introduction

This project aims to explore and develop image binarization algorithms that optimizes the accuracy of Optical Character Recognition(OCR). OCR itself is a process that converts texts on an image into machine-encoded text(based on Wikipedia). This project uses Google's Tesseract-OCR Engine as the OCR algorithm used to recognize the texts in the images.

The project is built using Python as the programming language. It uses pytesseract as a wrapper for Google's Tesseract-OCR Engine. It also uses some of OpenCV filtering algorithms mainly for verification and comparison purposes.

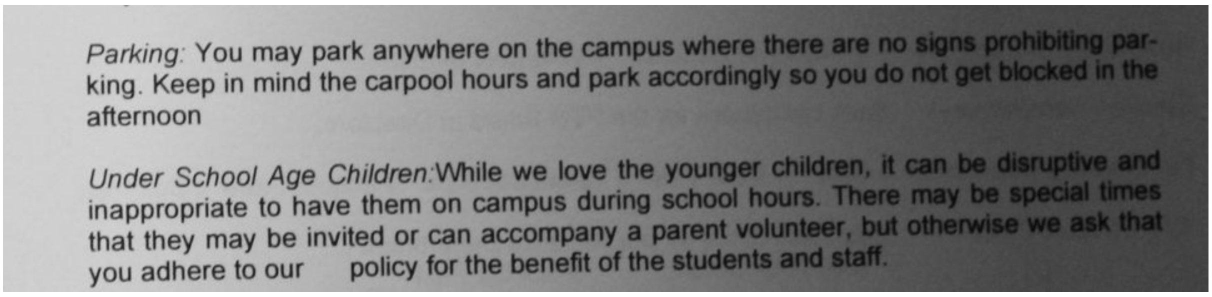
Another library used in this project is Pillow to load, preprocess, and output the resulting image.

We also used Numpy and Scipy to handle mathematical computations for the image processing and filtering used in this project.

Dataset Information

The two given images for this assignment, ‘sample01.png’ and ‘sample02.png’ will be used as the basis for OCR experimentation in this project. These images supposedly do not perform well when it is put directly to the Tesseract OCR engine. Below are the original images used in this project:

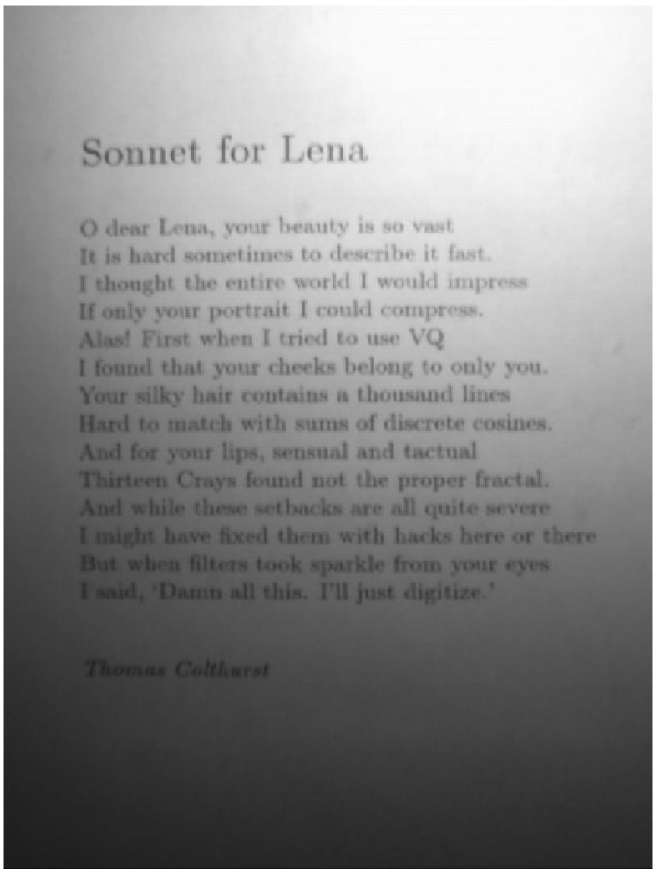
Sample01:



Parking: You may park anywhere on the campus where there are no signs prohibiting parking. 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, but otherwise we ask that you adhere to our policy for the benefit of the students and staff.

Sample02:



Sonnet for Lena

O dear Lena, your beauty is so vast
It is hard sometimes to describe it fast.
I thought the entire world I would impress
If only your portrait I could compress.
Alas! First when I tried to use VQ
I found that your cheeks belong to only you.
Your silky hair contains a thousand lines
Hard to match with sums of discrete cosines.
And for your lips, sensual and tactual
Thirteen Crays found not the proper fractal.
And while these setbacks are all quite severe
I might have fixed them with hacks here or there
But when filters took sparkle from your eyes
I said, ‘Damn all this. I’ll just digitize.’

Thomas Colthurst

The base texts, which will be used to measure the accuracy of the generated OCR texts, are manually typed and saved into ‘sample01.txt’ and ‘sample02.txt’ respectively. These texts can be seen in the appendix.

Experiments

Image OCR

As mentioned in the previous section, we use pytesseract as a wrapper to Tesseract to obtain the texts within the given image.

Below is the source snippet used to obtain the texts inside a given image:

```
result = pytesseract.image_to_string(image)
```

Otsu Thresholding

I decided to implement the Otsu thresholding algorithm based on several online sources, most notable one of which is this wikipedia page. The main idea of the algorithm is to try to find a value that minimizes the intra-class variance, which is defined as a weighted sum of variances of the two classes:

$\sigma_w^2(t) = w_0(t)\sigma_0^2(t) + w_1(t)\sigma_1^2(t)$

where w_0 and w_1 are the probability of the two classes separated by threshold t, defined as:

$w_0(t) = \Sigma_{i=1}^{t} P(i)$

$w_1(t) = \Sigma_{i=t+1}^{255} P(i)$

Moreover, minimizing the intra-class variance is equivalent to maximizing inter-class variance:

$\sigma_b^2(t) = w_0(t)w_1(t)[\mu_0(t) - \mu_1(t)]^2$

Where μ_0 and μ_1 are the class means of w_0 and w_1 respectively.
With the equations shown above, we proceed with calculating the threshold shown below:

```
def otsu_thresholding_in(image, max_value=255):
      # Image must be in grayscale
      image_np = np.array(image)
      # Set total number of bins in the histogram
      number_of_bins = 256  # Since our image is 8 bits, we used 256 for
now
      # Get the image histogram
      histogram, bin_edges = np.histogram(image_np, bins=number_of_bins)

      # Calculate centers of bins
      bin_center = (bin_edges[:-1] + bin_edges[1:]) / 2.
      # Iterate over all thresholds (indices) and get the probabilities
\w_0(t), \w_1(t)
      w_0 = np.cumsum(histogram)
      w_1 = np.cumsum(histogram[::-1])[::-1]

      # Get the class means \mu0(t)
      m_0 = np.cumsum(histogram * bin_center) / w_0
      # Get the class means \mu1(t)
      m_1 = (np.cumsum((histogram * bin_center)[::-1]) /
w_1[::-1])[::-1]

      # Calculate the inter-class variance
      inter_var = w_0[:-1] * w_1[1:] * (m_0[:-1] - m_1[1:]) ** 2

      # Minimize intra-class variance, which is equal to maximize the
inter_class_variance function val
      max_val_index = np.argmax(inter_var)

      # Get the threshold value
      thresh = bin_center[:-1][max_val_index]
      # Get the image by performing the thresholding
      image_result = threshold_image(image_np, thresh)

      return image_result, thresh
```

After we have obtained the threshold value, we do a simple thresholding by using the following function that I have implemented in Python to get the binary image:

```
def threshold_image(image_np, threshold=0, op = '<'):
      # Set pixels with value less than threshold to 0, otherwise set is
as 255
      if op == '<':
      image_result_np = np.where(image_np < threshold, 0, 1)
      else:
      image_result_np = np.where(image_np > threshold, 0, 1)
      # Convert numpy array back to PIL image object
      image_result = Image.fromarray((image_result_np *
255).astype(np.uint8))
      return image_result
```

Accuracy Calculation

To calculate the accuracy of the OCR, I have developed a simple evaluation mechanism using built-in Python difflib so that we can obtain a reliable and easily reproducible evaluation algorithm. To be more specific, we will base our accuracy using the `ratio()` function in the difflib library, which returns a value between 0 and 1, indicating the similarity measure of two texts.

Below is the source snippet used to evaluate the two texts given:

```
def evaluate(actual, expected, print_score=True):
      s = difflib.SequenceMatcher(None, actual, expected)
      if print_score:
      print("{:.5f}".format(s.ratio()))
      # print(s.get_matching_blocks())
      return s.ratio()
```

Base Text Accuracy Testing

We first obtain the accuracy of the base image, which we have yet to perform any manipulation to the image. This will be referred to as our base accuracy.

```
for idx, image_name in enumerate(images):
      image = Image.open(os.path.join(image_folder, image_name))
      print(image.format, image.mode)
      image = image.convert("RGB")
      result = pytesseract.image_to_string(image)

      with open(os.path.join(text_folder, texts[idx]), 'r') as f:
            base_text = f.readlines()
            base_text = "".join(base_text)
            # base_text = [line.strip() for line in base_text]

      print(result)
      evaluate(result, base_text)
```

Result & Discussion

Here we obtain an accuracy of 0.42293 for the first image and 0.05207 for the second image. To see what is going on, I tried to print the texts recognized by the OCR algorithm, which are shown below:

Sample01:

```
Parking: You may park anywhere on the ce
```

```
king. Keep in mind the carpool hours and park
afternoon

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

Sample02:

```
Sonnet for Lena
```

It can be observed that the first part can recognize the text in the bright area very well. Meanwhile, it is almost unable to recognize anything in the dark region.
Unfortunately, the OCR algorithm can only detect the title of the Poem in the second image.

This project aims to improve this so that it can correctly recognize as many characters in the image as possible.

Otsu Thresholding Accuracy Testing

We then perform an otsu thresholding algorithm to the image and check the accuracy of the resulting image.

```python
for idx, image_name in enumerate(images):
      image = Image.open(os.path.join(image_folder, image_name))
      # print(image.format, image.mode)
      image = image.convert("L")

      with open(os.path.join(text_folder, texts[idx]), 'r') as f:
            base_text = f.readlines()
            base_text = "".join(base_text)
            # base_text = [line.strip() for line in base_text]

      image_th, thresh = otsu_thresholding_in(image)
      print(f"Threshold pixel value={thresh}")
      image_th.show()
      result_th = pytesseract.image_to_string(image_th)


      evaluate(result_th, base_text)
```

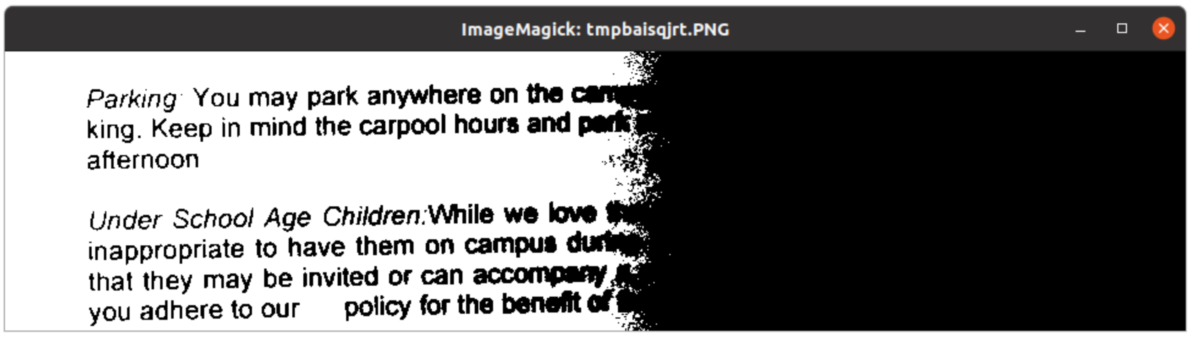
Also, to verify that our current implementation of the Otsu algorithm is correct, I did a comparison with the Otsu thresholding algorithm implemented in OpenCV, shown below:

```
for idx, image_name in enumerate(images):
      image = Image.open(os.path.join(image_folder, image_name))
      # print(image.format, image.mode)
      image = image.convert("L")

      with open(os.path.join(text_folder, texts[idx]), 'r') as f:
            base_text = f.readlines()
            base_text = "".join(base_text)
            # base_text = [line.strip() for line in base_text]
      img_cv = numpy.array(image)
      ret, image_th_cv = cv.threshold(img_cv, 0, 255, cv.THRESH_BINARY +
cv.THRESH_OTSU)
      image_th = Image.fromarray(image_th_cv)
      result_th = pytesseract.image_to_string(image_th)
      image_th.show()

      evaluate(result_th, base_text)
```

Result & Discussion

Surprisingly, we obtain a lower accuracy than the result of our base image, which are 0.41019 and 0.03374 respectively. The result from the OpenCV Otsu algorithm also produces the same accuracy result.
The following are the result of the binarized images:

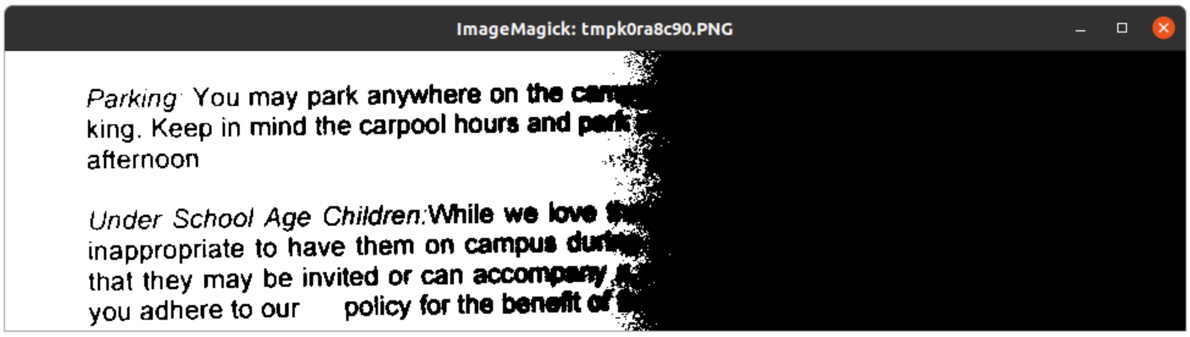
Sample01 (Our implementation):


ImageMagick: tmpbaisqjrt.PNG

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

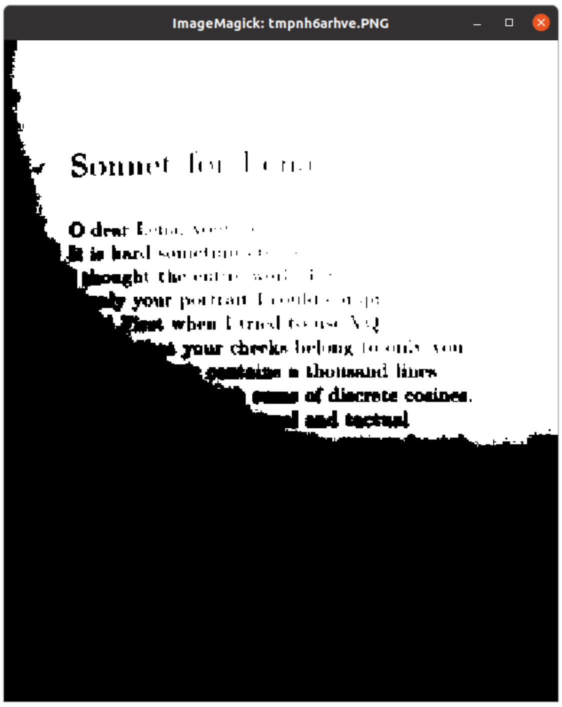
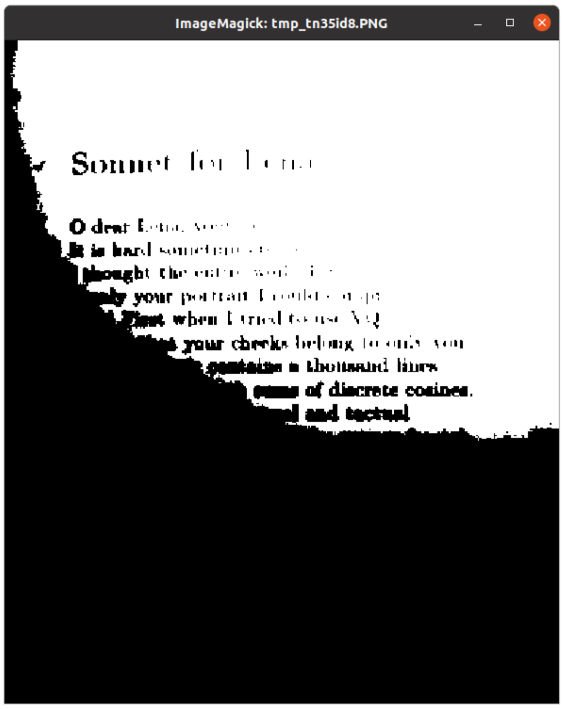
*Under School Age Children:*While we love th
inappropriate to have them on campus duri
that they may be invited or can accompany
you adhere to our policy for the benefit of

Sample 01 (OpenCV implementation):


ImageMagick: tmpk0ra8c90.PNG

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

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

Sample02 (Our implementation):	Sample02 (OpenCV implementation):
	

It can be seen that the resulting implementation of our own Otsu thresholding and OpenCV implementation looks the same. It can also be observed that the darker regions of the image are now totally black and thus ineligible. Moreover, certain parts in the edge between the dark and the bright region of the images that were previously still pretty much visible are now getting blurred as well. This explains why the accuracy drops after the otsu thresholding algorithm is performed.

Local Adaptive Thresholding

It is known that the Otsu thresholding algorithm has limitations, especially if the image has different lighting conditions in different areas. Unfortunately, this is the case for both of our sample images, which are shown in the previous section. To address the problem of the Otsu global thresholding algorithm, several techniques are tested and one of the promising techniques is local adaptive thresholding.

In the local adaptive thresholding algorithm, it calculates the threshold for a small region in the image. Therefore, we will obtain different thresholds for different regions of the same image and thus it can give us better results for images with varying brightness.

To be more specific, in the local adaptive thresholding algorithm, we examine each pixel and its neighborhood and calculate the statistics (mean, median, gaussian weighted mean) to be used in determining whether it is a background or foreground pixel.

It must be noted that the size of the neighborhood to be considered must be large enough to cover enough background and foreground pixels or otherwise the algorithm will perform poorly. On the other hand, if the chosen neighborhood size is too big, we will lose the uniformity in the illumination of the selected pixels.

Adaptive Gaussian Thresholding

I implemented the Adaptive Gaussian Thresholding which is mainly based on an open source Matlab implementation by Guanglei Xiong at Tsinghua University, with several explanations from the resource university website.

Below is the source code for the implementation in python:

```
def adaptive_gaussian_thresholding_in(image, max_value=255,
block_size=7, C=0, std=1):
      # Image must be in grayscale
      image_np = np.array(image)

      kernel = gaussian_kernel(block_size, std=std)
      # print(f"kernel={kernel}")

      image_convolved_np = scipy.signal.convolve2d(image_np, kernel,
mode='same', boundary='symm')
      image_result_np = image_convolved_np - image_np - C
      # print(image_result_np)

      image_result = threshold_image(image_result_np)

      return image_result
```

With the following function to create the gaussian kernel(based on this StackOverflow question):

```
def gaussian_kernel(kernel_size=7, std=1, normalize=True):
      gaussian_kernel_1d = scipy.signal.gaussian(kernel_size,
std=std).reshape(kernel_size, 1)
```

```
      gaussian_kernel_2d = np.outer(gaussian_kernel_1d,
gaussian_kernel_1d)
      if normalize:
      return gaussian_kernel_2d / gaussian_kernel_2d.sum()
      else:
      return gaussian_kernel_2d
```

Adaptive Gaussian Thresholding Accuracy Testing

I proceed with testing the accuracy after Adaptive Gaussian Thresholding has been performed to the images. I decided to try to initially try to use the kernel size of 7x7, which is the one used in the explanation link above.

```
for idx, image_name in enumerate(images):
      image = Image.open(os.path.join(image_folder, image_name))
      # print(image.format, image.mode)
      image = image.convert("L")

      with open(os.path.join(text_folder, texts[idx]), 'r') as f:
            base_text = f.readlines()
            base_text = "".join(base_text)
            # base_text = [line.strip() for line in base_text]

      image_th = adaptive_gaussian_thresholding_in(image, block_size=7,
std=2, C=0)
      image_th.show()
      result_th = pytesseract.image_to_string(image_th)


      evaluate(result_th, base_text)
```

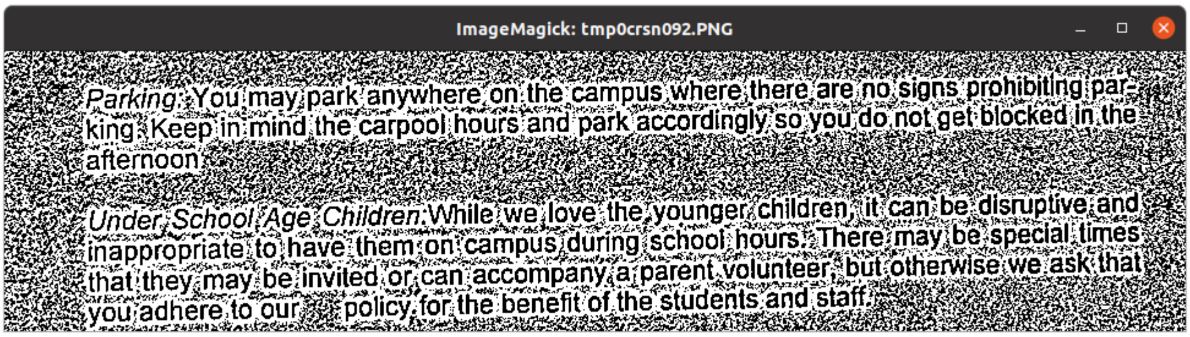
To verify the correctness of my implementation, I also compared my implementation of the Adaptive Gaussian Thresholding with the one implemented in OpenCV.

```
for idx, image_name in enumerate(images):
      image = Image.open(os.path.join(image_folder, image_name))
      # print(image.format, image.mode)
      image = image.convert("L")

      with open(os.path.join(text_folder, texts[idx]), 'r') as f:
            base_text = f.readlines()
            base_text = "".join(base_text)
            # base_text = [line.strip() for line in base_text]
      img_cv = numpy.array(image)
      img_th_cv = cv.adaptiveThreshold(img_cv, 255,
cv.ADAPTIVE_THRESH_GAUSSIAN_C, \
                              cv.THRESH_BINARY, 7, 0)

      image_adaptive_gaussian = Image.fromarray(img_th_cv)
      # image_adaptive_gaussian.show()
      result_adaptive_gaussian =
pytesseract.image_to_string(image_adaptive_gaussian)
      # print(result_adaptive_gaussian)
      print("Adaptive gaussian:")
      evaluate(result_adaptive_gaussian, base_text)
```

Result & Discussion

The resulting accuracy for both images in both implementations is either 0 or nearly 0. It seems that the OCR cannot identify any texts within the image. To figure out why, I decided to take a look at the resulting images, shown below:

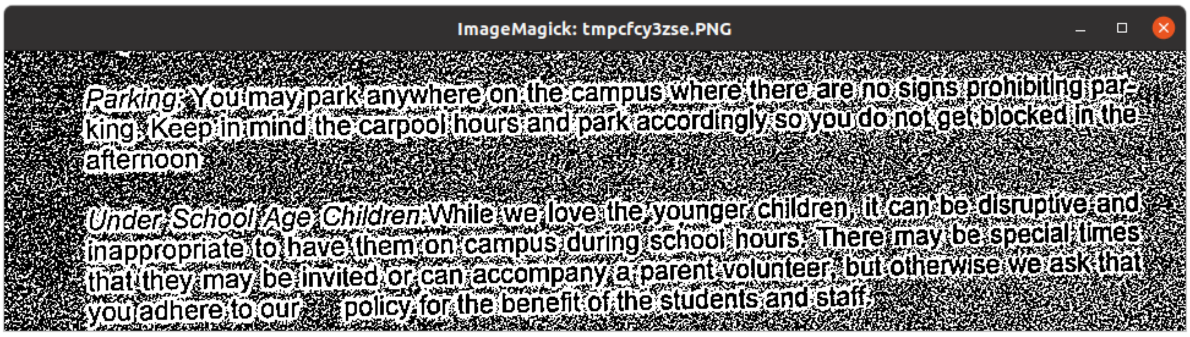
Sample01 (Our implementation):



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

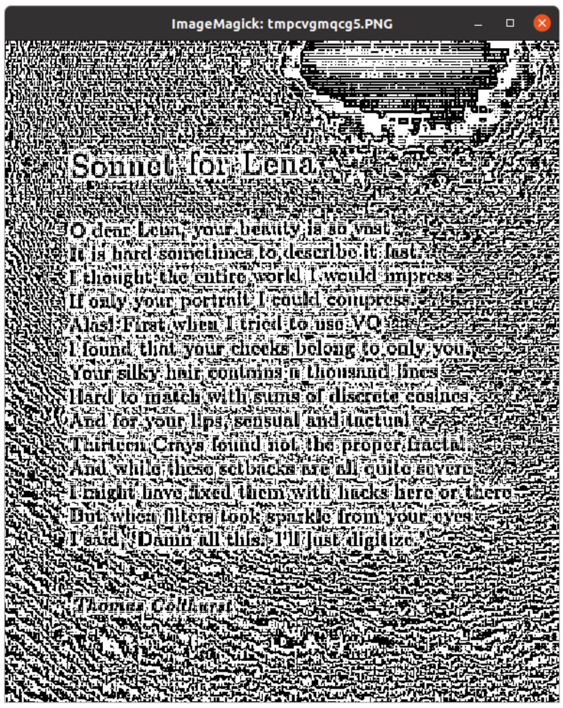
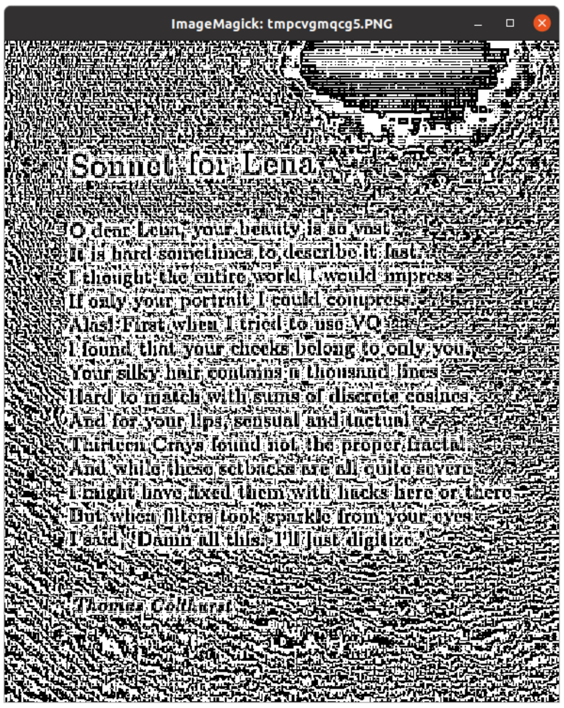
Under School Age Children: While we love the younger children, it can be disruptive and inappropriate to have them on campus during school hours. There may be special times that they may be invited or can accompany a parent volunteer, but otherwise we ask that you adhere to our policy for the benefit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

Sample01 (OpenCV implementation):



Parking: You may park anywhere on the campus where there are no signs prohibiting parking. 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, but otherwise we ask that you adhere to our policy for the benefit of the students and staff.

Sample02 (Our implementation):	Sample02 (OpenCV implementation):
	

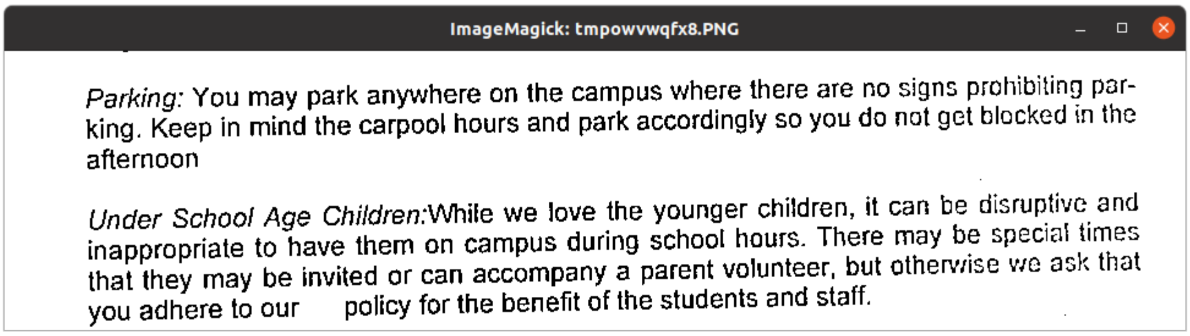
From the images above, it can be confirmed that our implementation produces similar results with the one implemented in OpenCV.
It can be seen that both images now have similar brightness. Moreover, it can be seen that the algorithm is particularly successful in the area near the text. However, it seems that there is a lot of noise picked up by the algorithm everywhere else in the image. This was most likely be caused by the fact that the intensity values between the local neighborhood are very similar, and thus the mean is very close to the value in the center pixel. As a result, the algorithm has a difficulty in binarizing those values. Therefore, I tried to experiment with increasing the constant C so that values with neighboring values nearing the mean can still be classified as background.

Testing C-values for Adaptive Gaussian Thresholding

Since the image looks very noisy, I tried to start off with a C value that is quite large, which is 10.

Result & Discussion

Sample01 (Our implementation):

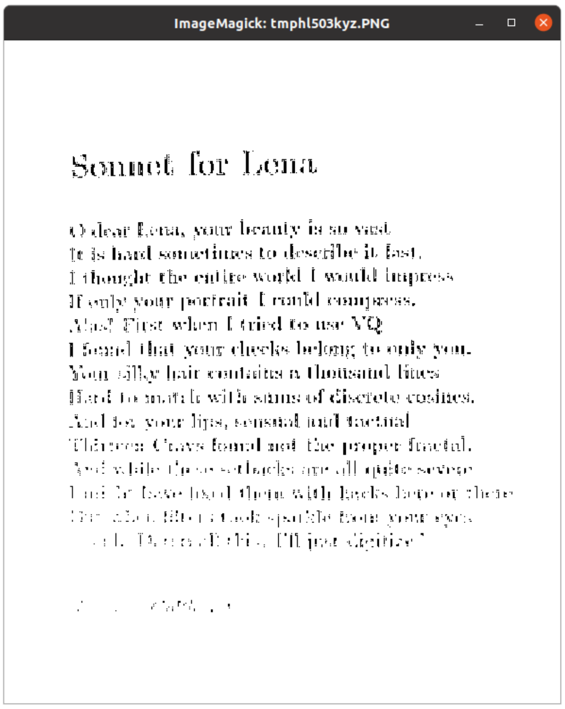


Parking: You may park anywhere on the campus where there are no signs prohibiting parking. Keep in mind the carpool hours and park accordingly so you do not get blocked in the afternoon

*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, but otherwise we ask that you adhere to our policy for the benefit of the students and staff.

Accuracy: 0.98837

Sample02(Our Implementation):



Sonnet for Lena

O dear Lena, your beauty is so vast
It is hard sometimes to describe it fast.
I thought the entire world I would impress
If only your portrait I could compress.
Alas! First when I tried to use VQ
I found that your cheeks belong to only you.
Your silky hair contains a thousand lines
Hard to match with sums of discrete cosines.
And for your lips, sensual and tactual
Thirteen Crays found not the proper fractal.
And while these setbacks are all quite severe
I might have fixed them with hacks here or there
[illegible] took sparkle from your eyes
[illegible] I'll just digitize.'

[illegible]

Accuracy: 0.04952

It can be seen that the result is now much better than previously. The accuracy for the first image is also near perfect. Yet, we are still unable to see any improvements of OCR accuracy for the second image.

Adaptive Gaussian Thresholding Parameters Fine-tuning

I decided to continue testing systematically the parameters for each image, by changing each parameters within a specific range and testing the accuracy result for each of them. By doing so, we can know what could be the best parameters for each image and what is the highest accuracy we could get by only relying on this method.

Below is the code used for the testing:

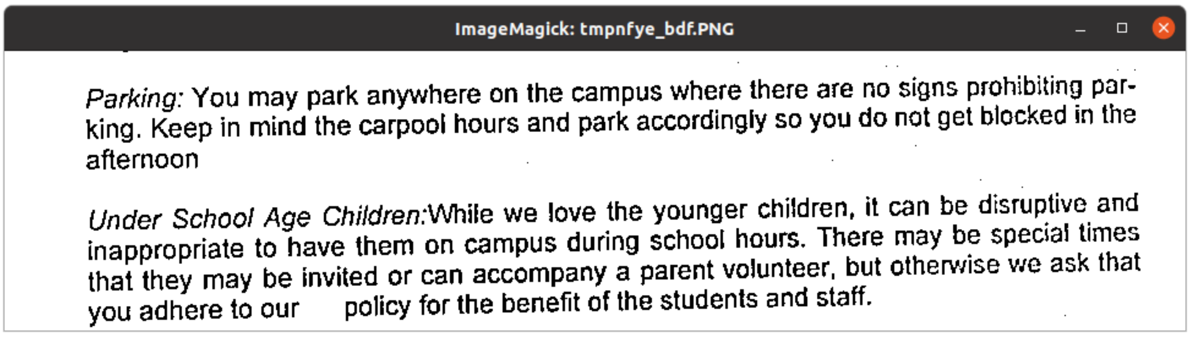
```
# Parameters fine-tuning
accuracy = [0,0]
block_size_optimum = [0,0]
std_optimum = [0,0]
```

```
C_optimum = [0,0]
for idx, image_name in enumerate(images):
      image = Image.open(os.path.join(image_folder, image_name))
      # print(image.format, image.mode)
      image = image.convert("L")

      with open(os.path.join(text_folder, texts[idx]), 'r') as f:
            base_text = f.readlines()
            base_text = "".join(base_text)
            # base_text = [line.strip() for line in base_text]
      for C in range(0,10):
      for block_size in range(3,15,2):
            for std in range(1,3):
                  image_th = adaptive_gaussian_thresholding_in(image,
block_size=block_size,std=std,C=C)
                  # image_th.show()
                  result_th = pytesseract.image_to_string(image_th)
                  score = evaluate(result_th, base_text,False)
                  if accuracy[idx] < score:
                  print(f"Found better accuracy of {score} for image
{image_name} with parameters {block_size} {std} {C}")
                  accuracy[idx] = score
                  block_size_optimum[idx] = block_size
                  std_optimum[idx] = std
                  C_optimum[idx] = C
                  # print(f"{block_size} | {std} | {C} | {score:.5f}")
print(accuracy)
print(block_size_optimum)
print(std_optimum)
print(C_optimum)
```

Result & Discussion

The following are the resulting image with its best parameters and its corresponding accuracy:

Sample01 (Our implementation):



ImageMagick: tmpnfye_bdf.PNG

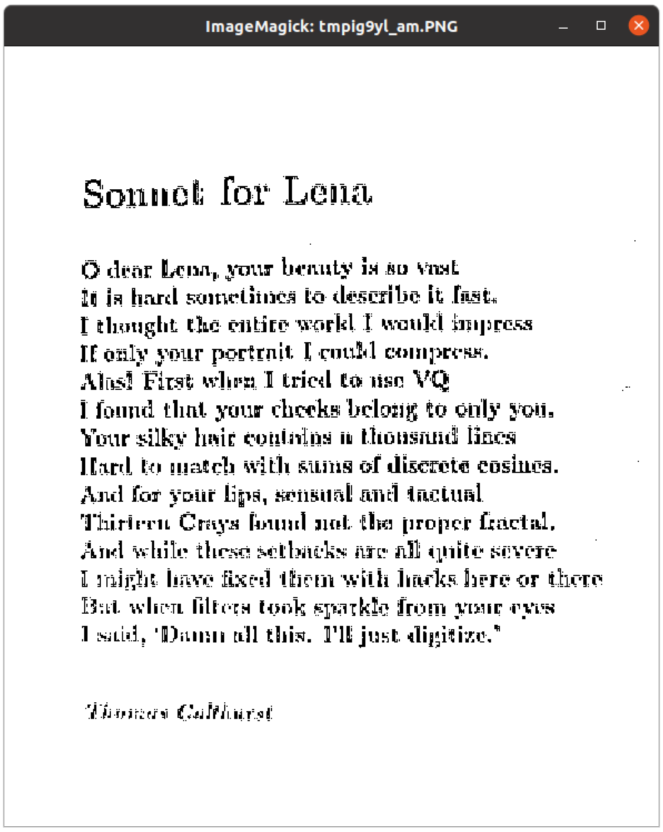
Parking: You may park anywhere on the campus where there are no signs prohibiting parking. Keep in mind the carpool hours and park accordingly so you do not get blocked in the afternoon

*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, but otherwise we ask that you adhere to our   policy for the benefit of the students and staff.

Accuracy: 0.99516

Parameters: Kernel size=11, std=2, C=8

Sample02 (Our implementation):



Sonnet for Lena

O dear Lena, your beauty is so vast
It is hard sometimes to describe it fast.
I thought the entire world I would impress
If only your portrait I could compress.
Alas! First when I tried to use VQ
I found that your cheeks belong to only you.
Your silky hair contains a thousand lines
Hard to match with sums of discrete cosines.
And for your lips, sensual and tactual
Thirteen Crays found not the proper fractal.
And while these setbacks are all quite severe
I might have fixed them with hacks here or there
But when filters took sparkle from your eyes
I said, 'Damn all this. I'll just digitize.'

Thomas Colthurst

Accuracy: 0.29385
Parameters: Kernel size=9, std=2, C=4

It can be seen here that the accuracy for the first image is near perfect. This can be observed from the fact that the image obtained after the thresholding mechanism is performed is already pretty clear. There is only a little bit of noise in the image.
Meanwhile, we can see that there is also a significant improvement in accuracy for the second image, although the accuracy is still quite low. I suspect this is caused by the fact that the characters in the image are not clear, which might be caused by noise in the image.

Gaussian Blur + Adaptive Gaussian Thresholding

For the next step, I tried to implement some filtering with the hope that it can reduce the noise in the image and make the Adaptive Gaussian Thresholding perform better.

```python
def gaussian_blur_in(image, kernel_size=7, std=1):
      image_np = np.array(image)
      kernel = gaussian_kernel(kernel_size=kernel_size, std=std)
      image_convolved_np = scipy.signal.convolve2d(image_np, kernel, 
mode='same', boundary='symm')
      return Image.fromarray(image_convolved_np)
```

Gaussian Blur + Adaptive Gaussian Thresholding Accuracy Testing

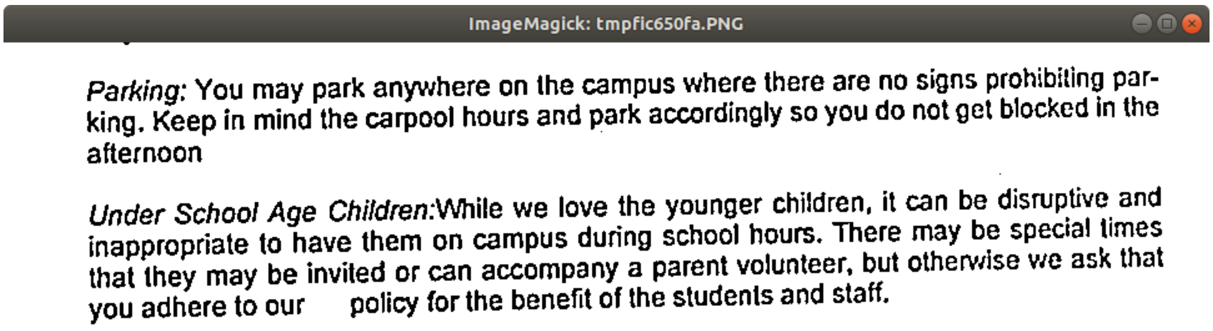
I tested to apply the Gaussian blurring with varying kernel size and sigma to see which one works best.

```python
accuracy = [0,0]
kernel_size_optimum = [0,0]
std_optimum = [0,0]
for idx, image_name in enumerate(images):
      for kernel_size in range(3,17,2):
      for std in [0.5,1,2]:
            image = Image.open(os.path.join(image_folder, image_name))
            # print(image.format, image.mode)
            image = image.convert("L")

            with open(os.path.join(text_folder, texts[idx]), 'r') as f:
                  base_text = f.readlines()
                  base_text = "".join(base_text)
                  # base_text = [line.strip() for line in base_text]
            image = gaussian_blur_in(image, kernel_size=kernel_size, 
std=std)
            image_th = adaptive_gaussian_thresholding_in(image, 
block_size=9, std=2, C=4)
            # image_th.show()
            result_th = pytesseract.image_to_string(image_th)
            score = evaluate(result_th, base_text, print_score=False)
            if accuracy[idx] < score:
                  print(f"Found better accuracy of {score} for image 
{image_name} with parameters {kernel_size} {std}")
                  accuracy[idx] = score
                  kernel_size_optimum[idx] = kernel_size
                  std_optimum[idx] = std
            # print(f"Gaussian blur ({kernel_size},{kernel_size})
std={std} + Adaptive gaussian for {image_name} score: {score:.5f}")
print(accuracy)
print(kernel_size_optimum)
print(std_optimum)
```

Result & Discussion

The following are the resulting image with its best parameters and its corresponding accuracy:

Sample01 (Our Implementation):



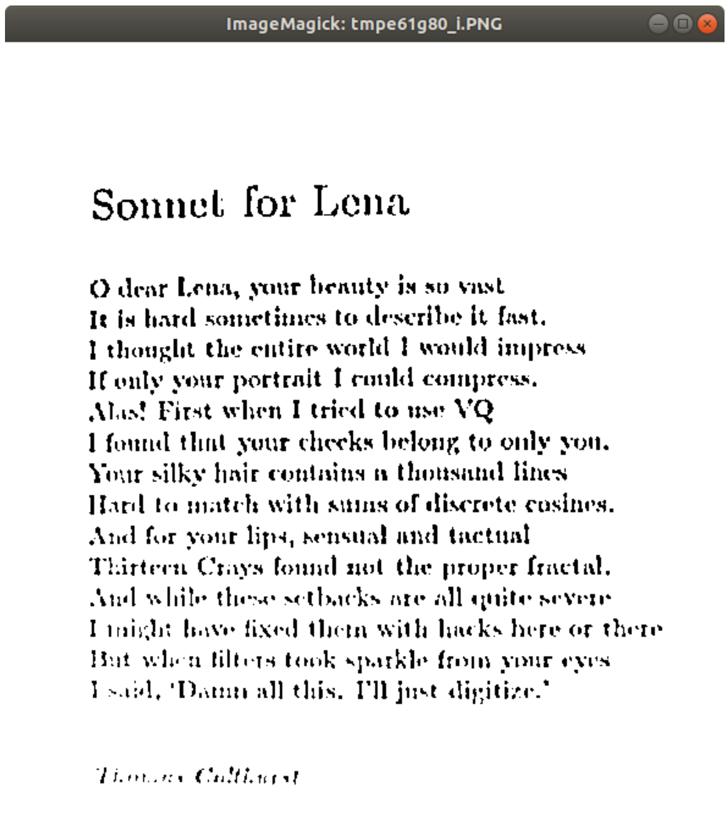
Parking: You may park anywhere on the campus where there are no signs prohibiting parking. Keep in mind the carpool hours and park accordingly so you do not get blocked in the afternoon

*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, but otherwise we ask that you adhere to our   policy for the benefit of the students and staff.

Accuracy: 0.99516

Gaussian parameter: Kernel size =3, std=0.5

Sample02 (Our Implementation):



Sonnet for Lena

O dear Lena, your beauty is so vast
It is hard sometimes to describe it fast.
I thought the entire world I would impress
If only your portrait I could compress.
Alas! First when I tried to use VQ
I found that your cheeks belong to only you.
Your silky hair contains a thousand lines
Hard to match with sums of discrete cosines.
And for your lips, sensual and tactual
Thirteen Crays found not the proper fractal.
And while these setbacks are all quite severe
I might have fixed them with hacks here or there
But when filters took sparkle from your eyes
I said, 'Damn all this. I'll just digitize.'

Thomas Colthurst

Accuracy: 0.58457
Gaussian parameter: Kernel size =3, std=1

Now, it can be seen that our accuracy in the second image increased by around as twice as much. We can also see that the characters in the image are now more robust.

Let’s take a closer look at the obtained text for both image:
Sample01:

```
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 limes
that they may be invited or can accompany a parent volunteer, but
```

```
otherwise we ask that
you adhere toour policy for the benefit of the students and staff.
```

Sample02:

```
Sonnet for Lena

OQ dear Lena, your beauty ix su vast

It is bard sometimes to deseribe it fast.
Tthonglit the entire world | would impress
If only your portrait [ could compress,

Alas! First when [tried to use VQ

1 found that your cheeks belong to only you.
Your silky hair contains n thousand lines
Hard ta mates with sums of discrete cosines.
And for your lips, sensual and tactual
Hirteen Crays fonnd not the proper fractal,
ryernie





And while these setbacks are all quite
Tinisgit bave fixed thet with hacks here or there
Bat when filters took sparkle from yon eyes
Teaidl, Datne all this, PM juse digitize."
```

We can see that the resulting text from the OCR of *Sample01* is almost perfect. The only difference is that in the last word of line 5, it is detected as “limes” instead of “times”. Also, we can see that there is a missing space between the word “to” and “our” in the last line.
Meanwhile, in the resulting text of the second image *Sample02*, we can see that the algorithm mostly failed to recognize texts in the bottom left region of the image, which is the dark region of the image. This can be attributed to the fact that in the resulting pre-processed image, that area is somewhat ‘corrupted’, quite a significant part of each character in that region is missing.

Additional Possible Enhancement

We can see that after all the processing that we have done, the resulting characters are still unclear, especially in the bottom-left part of the image, which region is darker in the original image. If we look closely at the original image, we can see that the original image itself is already blurry and rather noisy. Moreover, it is likely that the camera captures more noise in the darker region of the image. Therefore, what we might be able to do is to have a more sophisticated filter that can remedy this issue better.

However, another approach that we might be able to do instead is we try to sharpen the image itself. By doing so, we can get a clearer picture and thus the OCR library will be able to perform better. There are several approaches that can be utilized to achieve this, the simplest of which is by Laplacian Pyramid decomposition where we try to supersample the image. A much better approach would be to apply more sophisticated super-resolution methods. Recently, there are quite a lot of machine learning approaches to perform this task. As far as I know of, TecoGAN is one of the latest examples with the best result using this machine-learning approach.

Conclusion

In general, to improve the recognition algorithms for more robust and accurate character recognition, we can try to apply a blurring method to remove the noise in the image followed by an adaptive gaussian filter to make the image have a similar brightness, as what we have shown in the experiments on this project. The only problem now is that we need to manually try different parameters for our noise filters and check which one produces the best image.
Moreover, in the real-life scenario, we will most likely not have the original plaintext of the image beforehand. Therefore, we cannot simply compute the accuracy of the recognized text.

There are several methods that can be utilized to handle this issue. The easiest one is by systematically applying different parameters like what we have done in this experiment, and then taking the result which produces the most detected characters (greedy approach).
By doing so, we assume that when the recognition algorithm can detect the most characters, it means that it is when the image is in the best condition. A better version of this would be to train a neural network that can check the validity and correctness of the resulting text. We can then try to select the image that makes the most sense as a phrase to the ML and while maintaining to get as many recognized words as possible.

Another approach that we can use to improve the accuracy for the recognition algorithm in general is to actually train the OCR itself with high quantities of poor quality images. By doing this, the OCR algorithm itself can be more robust and able to recognize texts in degraded images. Moreover, we can add another Neural Network that can predict what is the most probable word given the current recognized text, which is apparently already starting to be implemented in the latest version of Tesseract. However, we can still see that this might not be that robust yet, since we can see that there are still a lot of mistakes seen on our last text results that should have been fixed by this. Therefore, what we can do is either to train it further or use a more complicated neural network like a transformer-based language model.

References

- Anthony Kay. 2007. Tesseract: an open-source optical character recognition engine. *Linux J.* 2007, 159 (July 2007), 2.
- Bradski, G. (2000). The OpenCV Library. *Dr. Dobb's Journal of Software Tools*.
- Choi, Myungsub, et al. "Channel Attention Is All You Need for Video Frame Interpolation." *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 07, Apr. 2020, pp. 10663–71. *DOI.org (Crossref)*, doi:10.1609/aaai.v34i07.6693.
- Clark, A. (2015). *Pillow (PIL Fork) Documentation*. readthedocs. Retrieved from https://buildmedia.readthedocs.org/media/pdf/pillow/latest/pillow.pdf
- Local Adaptive Thresholding. https://www.mathworks.com/matlabcentral/fileexchange/8647-local-adaptive-thresholding. Accessed 27 Nov. 2020.
- Oliphant, T. E. (2006). *A guide to NumPy* (Vol. 1). Trelgol Publishing USA.
- "Optical Character Recognition." Wikipedia, 18 Nov. 2020. Wikipedia, https://en.wikipedia.org/w/index.php?title=Optical_character_recognition&oldid=989353224.
- *Otsu's Thresholding with OpenCV*. https://www.learnopencv.com/otsu-thresholding-with-opencv/. Accessed 27 Nov. 2020.
- *Point Operations - Adaptive Thresholding*. https://homepages.inf.ed.ac.uk/rbf/HIPR2/adpthrsh.htm. Accessed 27 Nov. 2020.
- "Pyramid (Image Processing)." *Wikipedia*, 8 Sept. 2020. *Wikipedia*, https://en.wikipedia.org/w/index.php?title=Pyramid_(image_processing)&oldid=977329364.
- "Python - How to Calculate a Gaussian Kernel Matrix Efficiently in Numpy?" *Stack Overflow*, https://stackoverflow.com/questions/29731726/how-to-calculate-a-gaussian-kernel-matrix-efficiently-in-numpy. Accessed 27 Nov. 2020.
- "Super-Resolution Imaging." *Wikipedia*, 25 Nov. 2020. *Wikipedia*, https://en.wikipedia.org/w/index.php?title=Super-resolution_imaging&oldid=990692045.
- Virtanen, P., Gommers, R., Oliphant, Travis E., Haberland, M., Reddy, T., Cournapeau, D., … Contributors, SciPy 1. 0. (2020). SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*.

Appendix

The source code for this project, as well as the image and base texts used in this project will also be accessible in https://github.com/hanstananda/CZ4003-Project.
The source code used in this project is shown below:

```
import cv2 as cv
import numpy
from PIL import Image
import scipy.ndimage
import scipy.signal
import pytesseract
import difflib
import os
import numpy as np


# In[3]:


image_folder = "./images"
text_folder = "./source"
images = ["sample01.png", "sample02.png"]
texts = ["sample01.txt", "sample02.txt"]


# In[4]:


def evaluate(actual, expected, print_score=True):
      s = difflib.SequenceMatcher(None, actual, expected)
      if print_score:
      print("{:.5f}".format(s.ratio()))
      # print(s.get_matching_blocks())
      return s.ratio()


# # Base Image with OCR

# In[5]:


for idx, image_name in enumerate(images):
      image = Image.open(os.path.join(image_folder, image_name))
      print(image.format, image.mode)
      image = image.convert("RGB")
      result = pytesseract.image_to_string(image)

      with open(os.path.join(text_folder, texts[idx]), 'r') as f:
            base_text = f.readlines()
            base_text = "".join(base_text)
            # base_text = [line.strip() for line in base_text]


      print(result)
      evaluate(result, base_text)


# # Otsu thresholding

# In[5]:


def threshold_image(image_np, threshold=0, op = '<'):
      # Set pixels with value less than threshold to 0, otherwise set is
as 255
      if op == '<':
      image_result_np = np.where(image_np < threshold, 0, 1)
      else:
      image_result_np = np.where(image_np > threshold, 0, 1)
      # Convert numpy array back to PIL image object
      image_result = Image.fromarray((image_result_np *
255).astype(np.uint8))
      return image_result


# In[6]:


def otsu_thresholding_in(image, max_value=255):
      # Image must be in grayscale
      image_np = np.array(image)
      # Set total number of bins in the histogram
      number_of_bins = 256  # Since our image is 8 bits, we used 256 for
now
      # Get the image histogram
      histogram, bin_edges = np.histogram(image_np, bins=number_of_bins)

      # Calculate centers of bins
      bin_center = (bin_edges[:-1] + bin_edges[1:]) / 2.
      # Iterate over all thresholds (indices) and get the probabilities
\w_0(t), \w_1(t)
```

```
      w_0 = np.cumsum(histogram)
      w_1 = np.cumsum(histogram[::-1])[::-1]

      # Get the class means \mu0(t)
      m_0 = np.cumsum(histogram * bin_center) / w_0
      # Get the class means \mu1(t)
      m_1 = (np.cumsum((histogram * bin_center)[::-1]) /
w_1[::-1])[::-1]

      # Calculate the inter-class variance
      inter_var = w_0[:-1] * w_1[1:] * (m_0[:-1] - m_1[1:]) ** 2

      # Minimize intra-class variance, which is equal to maximize the
inter_class_variance function val
      max_val_index = np.argmax(inter_var)

      # Get the threshold value
      thresh = bin_center[:-1][max_val_index]
      # Get the image by performing the thresholding
      image_result = threshold_image(image_np, thresh)

      return image_result, thresh


# In[89]:


for idx, image_name in enumerate(images):
      image = Image.open(os.path.join(image_folder, image_name))
      # print(image.format, image.mode)
      image = image.convert("L")

      with open(os.path.join(text_folder, texts[idx]), 'r') as f:
            base_text = f.readlines()
            base_text = "".join(base_text)
            # base_text = [line.strip() for line in base_text]
      img_cv = numpy.array(image)
      ret, image_th_cv = cv.threshold(img_cv, 0, 255, cv.THRESH_BINARY +
cv.THRESH_OTSU)
      image_th = Image.fromarray(image_th_cv)
      result_th = pytesseract.image_to_string(image_th)
      image_th.show()

      evaluate(result_th, base_text)
```

```
# ### Self implementation of Otsu thresholding

# In[58]:


for idx, image_name in enumerate(images):
      image = Image.open(os.path.join(image_folder, image_name))
      # print(image.format, image.mode)
      image = image.convert("L")

      with open(os.path.join(text_folder, texts[idx]), 'r') as f:
            base_text = f.readlines()
            base_text = "".join(base_text)
            # base_text = [line.strip() for line in base_text]

      image_th, thresh = otsu_thresholding_in(image)
      print(f"Threshold pixel value={thresh}")
      image_th.show()
      result_th = pytesseract.image_to_string(image_th)


      evaluate(result_th, base_text)


# # Adaptive Gaussian

# In[7]:


#
https://stackoverflow.com/questions/29731726/how-to-calculate-a-gaussian
-kernel-matrix-efficiently-in-numpy
def gaussian_kernel(kernel_size=7, std=1, normalize=True):
      gaussian_kernel_1d = scipy.signal.gaussian(kernel_size,
std=std).reshape(kernel_size, 1)
      gaussian_kernel_2d = np.outer(gaussian_kernel_1d,
gaussian_kernel_1d)
      if normalize:
      return gaussian_kernel_2d / gaussian_kernel_2d.sum()
      else:
      return gaussian_kernel_2d


# In[8]:
```

```
#
https://www.mathworks.com/matlabcentral/fileexchange/8647-local-adaptive
-thresholding
# https://homepages.inf.ed.ac.uk/rbf/HIPR2/adpthrsh.htm
def adaptive_gaussian_thresholding_in(image, max_value=255,
block_size=7, C=0, std=1):
      # Image must be in grayscale
      image_np = np.array(image)

      kernel = gaussian_kernel(block_size, std=std)
      # print(f"kernel={kernel}")

      image_convolved_np = scipy.signal.convolve2d(image_np, kernel,
mode='same', boundary='symm')
      image_result_np = image_convolved_np - image_np - C
      # print(image_result_np)

      image_result = threshold_image(image_result_np, op='>')

      return image_result


#
https://www.mathworks.com/matlabcentral/fileexchange/8647-local-adaptive
-thresholding
def adaptive_mean_thresholding_in(image, max_value=255, block_size=7,
C=0):
      # Image must be in grayscale
      image_np = np.array(image)

      kernel = np.ones((block_size, block_size)) / (block_size ** 2)
      image_convolved_np = scipy.signal.convolve2d(image_np, kernel,
mode='same', boundary='symm')
      image_result_np = image_convolved_np - image_np - C
      image_result = threshold_image(image_result_np, op='>')

      return image_result


# In[10]:


print(gaussian_kernel(3,1))


# In[208]:
```

```
for idx, image_name in enumerate(images):
      image = Image.open(os.path.join(image_folder, image_name))
      # print(image.format, image.mode)
      image = image.convert("L")

      with open(os.path.join(text_folder, texts[idx]), 'r') as f:
            base_text = f.readlines()
            base_text = "".join(base_text)
            # base_text = [line.strip() for line in base_text]
      img_cv = numpy.array(image)
      img_th_cv = cv.adaptiveThreshold(img_cv, 255,
cv.ADAPTIVE_THRESH_GAUSSIAN_C,
cv.THRESH_BINARY, 11, 8)

      image_adaptive_gaussian = Image.fromarray(img_th_cv)
      # image_adaptive_gaussian.show()
      result_adaptive_gaussian =
pytesseract.image_to_string(image_adaptive_gaussian)
      # print(result_adaptive_gaussian)

      print("Adaptive gaussian:")
      evaluate(result_adaptive_gaussian, base_text)


# ### Self implementation of Adaptive Gaussian thresholding

# In[209]:


for idx, image_name in enumerate(images):
      image = Image.open(os.path.join(image_folder, image_name))
      # print(image.format, image.mode)
      image = image.convert("L")

      with open(os.path.join(text_folder, texts[idx]), 'r') as f:
            base_text = f.readlines()
            base_text = "".join(base_text)
            # base_text = [line.strip() for line in base_text]

      image_th = adaptive_gaussian_thresholding_in(image, block_size=11,
std=2, C=8)
      image_th.show()
      result_th = pytesseract.image_to_string(image_th)
      evaluate(result_th, base_text)


# In[210]:


# Parameters fine-tuning
accuracy = [0,0]
block_size_optimum = [0,0]
std_optimum = [0,0]
C_optimum = [0,0]
for idx, image_name in enumerate(images):
      image = Image.open(os.path.join(image_folder, image_name))
      # print(image.format, image.mode)
      image = image.convert("L")

      with open(os.path.join(text_folder, texts[idx]), 'r') as f:
            base_text = f.readlines()
            base_text = "".join(base_text)
            # base_text = [line.strip() for line in base_text]
      for C in range(0,10):
      for block_size in range(3,13,2):
            for std in range(1,3):
                  image_th = adaptive_gaussian_thresholding_in(image,
block_size=block_size,std=std,C=C)
                  # image_th.show()
                  result_th = pytesseract.image_to_string(image_th)
                  score = evaluate(result_th, base_text,False)
                  if accuracy[idx] < score:
                  print(f"Found better accuracy of {score} for image
{image_name} with parameters {block_size} {std} {C}")
                  accuracy[idx] = score
                  block_size_optimum[idx] = block_size
                  std_optimum[idx] = std
                  C_optimum[idx] = C
                  # print(f"{block_size} | {std} | {C} | {score:.5f}")
print(accuracy)
print(block_size_optimum)
print(std_optimum)
print(C_optimum)


# # Gaussian Blur + Adaptive Gaussian Thresholding

# In[9]:
```

```
def gaussian_blur_in(image, kernel_size=7, std=1):
      image_np = np.array(image)
      kernel = gaussian_kernel(kernel_size=kernel_size, std=std)
      image_convolved_np = scipy.signal.convolve2d(image_np, kernel,
mode='same', boundary='symm')
      return Image.fromarray(image_convolved_np)



# In[26]:


for kernel_size in range(3,17,2):
      image = Image.open(os.path.join(image_folder, image_name))
      # print(image.format, image.mode)
      image = image.convert("L")

      with open(os.path.join(text_folder, texts[idx]), 'r') as f:
                  base_text = f.readlines()
                  base_text = "".join(base_text)
                  # base_text = [line.strip() for line in base_text]
      img_cv = numpy.array(image)
      img_blur = cv.GaussianBlur(img_cv, (kernel_size, kernel_size), 0)
#           img_th_cv = cv.adaptiveThreshold(img_blur, 255,
cv.ADAPTIVE_THRESH_GAUSSIAN_C, \
#                                    cv.THRESH_BINARY, 11, 8)
      image_th = adaptive_gaussian_thresholding_in(img_blur,
block_size=9, std=2, C=4)
      # image_th.show()
      result_th = pytesseract.image_to_string(image_th)
      score = evaluate(result_th, base_text, print_score=False)
      print(f"Gaussian blur ({kernel_size},{kernel_size}) + Adaptive
gaussian for {image_name} score: {score:.5f}")


# In[29]:


accuracy = [0,0]
kernel_size_optimum = [0,0]
std_optimum = [0,0]
for idx, image_name in enumerate(images):
      for kernel_size in range(3,17,2):
      for std in [0.5,1,2]:
            image = Image.open(os.path.join(image_folder, image_name))
            # print(image.format, image.mode)
            image = image.convert("L")

            with open(os.path.join(text_folder, texts[idx]), 'r') as f:
                  base_text = f.readlines()
                  base_text = "".join(base_text)
                  # base_text = [line.strip() for line in base_text]
            image = gaussian_blur_in(image, kernel_size=kernel_size,
std=std)
            image_th = adaptive_gaussian_thresholding_in(image,
block_size=15, std=2, C=4)
            # image_th.show()
            result_th = pytesseract.image_to_string(image_th)
            score = evaluate(result_th, base_text, print_score=False)
            if accuracy[idx] < score:
                  print(f"Found better accuracy of {score} for image
{image_name} with parameters {kernel_size} {std}")
                  accuracy[idx] = score
                  kernel_size_optimum[idx] = kernel_size
                  std_optimum[idx] = std
            # print(f"Gaussian blur ({kernel_size},{kernel_size})
std={std} + Adaptive gaussian for {image_name} score: {score:.5f}")


# In[31]:


print(accuracy)
print(kernel_size_optimum)
print(std_optimum)


# # Additional Testing

# In[13]:


for idx, image_name in enumerate(images):
      image = Image.open(os.path.join(image_folder, image_name))
      # print(image.format, image.mode)
      image = image.convert("L")

      with open(os.path.join(text_folder, texts[idx]), 'r') as f:
            base_text = f.readlines()
            base_text = "".join(base_text)
            # base_text = [line.strip() for line in base_text]
      image = gaussian_blur_in(image, kernel_size=3, std=1)
      image_th = adaptive_gaussian_thresholding_in(image, block_size=9,
std=2, C=4)
      image_th.show()
      result_th = pytesseract.image_to_string(image_th)
      score = evaluate(result_th, base_text, print_score=False)
      print(f"Gaussian blur + Adaptive gaussian for {image_name} score:
{score:.5f}")
      print(result_th)


# In[21]:


for idx, image_name in enumerate(images):
      if idx==0:
      continue
      for kernel_size in range(3,25,2):
      image = Image.open(os.path.join(image_folder, image_name))
      # print(image.format, image.mode)
      image = image.convert("L")

      with open(os.path.join(text_folder, texts[idx]), 'r') as f:
            base_text = f.readlines()
            base_text = "".join(base_text)
            # base_text = [line.strip() for line in base_text]
      image_cv = np.array(image)
      image_cv = cv.pyrUp(image_cv)
      image = Image.fromarray(image_cv)
      image_th = adaptive_gaussian_thresholding_in(image,
block_size=kernel_size, std=2, C=4)
      # image_th.show()
      result_th = pytesseract.image_to_string(image_th)
      score = evaluate(result_th, base_text, print_score=False)
      print(f"Adaptive gaussian {kernel_size} for {image_name} score:
{score:.5f}")
      # print(result_th)
      s = difflib.SequenceMatcher(None, result_th, base_text)


# In[ ]:
```

- “Sample01.txt” (Base text used to calculate the accuracy of the first image sample01.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, but
otherwise we ask that
you adhere to our policy for the benefit of the students and staff.
```

- “Sample02.txt” (Base text used to calculate the accuracy of the second image sample02.png):

```
Sonnet for Lena

O dear Lena, your beauty is so vast
It is hard sometimes to describe it fast.
I thought the entire world I would impress
If only your portrait I could compress.
Alas! First when I tried to use VQ
I found that your cheeks belong to only you.
Your silky hair contains a thousand lines
Hard to match with sums of discrete cosines.
And for your lips, sensual and tactual
Thirteen Crays found not the proper fractal.
And while these setbacks are all quite severe
I might have fixed them with hacks here or there
But when filters took sparkle from your eyes
I said, 'Damn all this.  I'll just digitize.'

Thomas Colthrust
```