

**NANYANG**  
**TECHNOLOGICAL**  
**UNIVERSITY**

**CZ4003**

**Computer Vision**

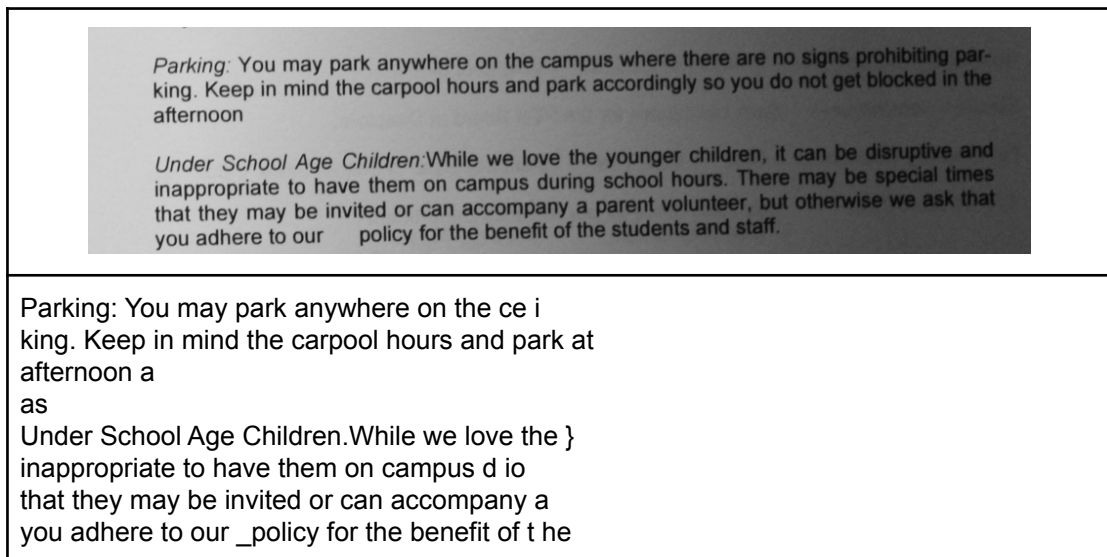
**Project Report**

## **Contents**

<b>Introduction</b>	<b>2</b>
<b>Otsu thresholding</b>	<b>2</b>
<b>Improvements</b>	<b>4</b>
Improvement 1: Using Edge Detection	4
Sobel mask	4
Prewitt Mask	4
Canny	5
Improvement 2: Pixel Division	6
Improvement 3: Adaptive Thresholding	7
<b>Possible improvements to recognition algorithm</b>	<b>8</b>
<b>Conclusion</b>	<b>8</b>
<b>Appendix</b>	<b>9</b>

## 1. Introduction

Optical Character Recognition (OCR) is becoming more widely employed by businesses on various platforms as a means to provide smart services to cater to their users' needs. It is one of the most successfully and widely implemented Computer Vision techniques in the commercial world. Typically, a neural network can be trained on an in-the-wild image dataset with labelled data. One of the most popular off-the-shelf OCR out there is Tesseract OCR. It is an open-source OCR engine that is actively being maintained. The engine has implementations in various languages such as C++, python and javascript, thus making it widely accessible to many engineers. However, a key issue with tesseract or any other OCR engine is the existence of 'noisy images' that may hinder the OCR engine's ability to extract words. We can see from a simple demonstration that the OCR is unable to perform well if we simply feed it an in-the-wild text image.



## 2. Otsu thresholding

A simple solution is to add a pre-processing step before feeding the image to the OCR engine. We can introduce binarization to present the image as black and white with the text being essentially segmented from the background. Otsu global thresholding algorithm is an algorithm for binarizing the input images and feeding the binarized image into the OCR engine. It is considered an image segmentation technique as we sought to find and segment pixels within a threshold range. But to do that we must first determine the threshold value. The objective of otsu global thresholding can be defined as:

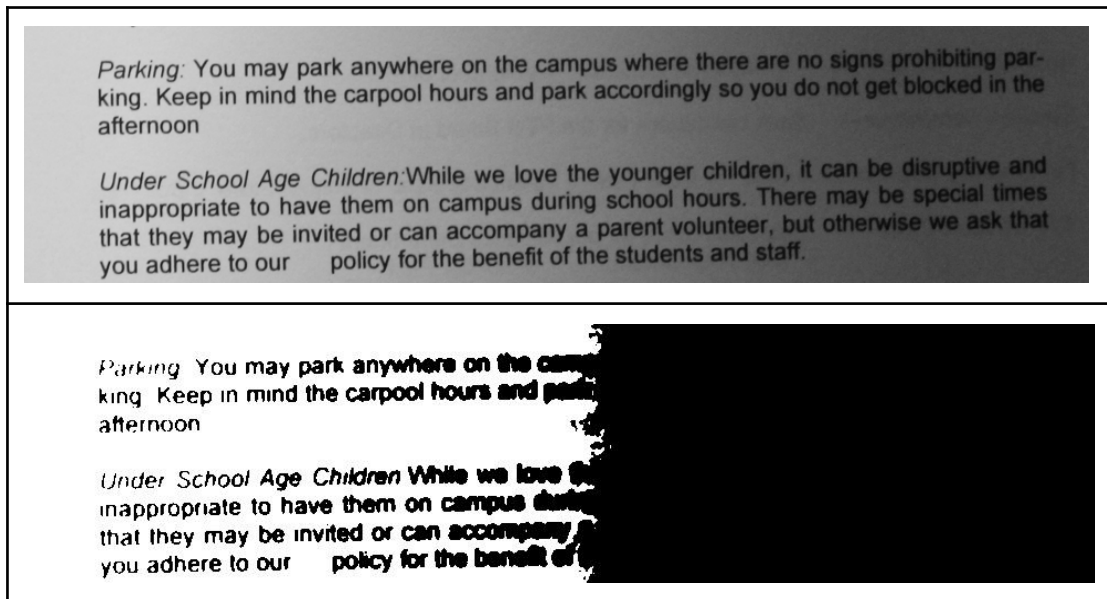
Global thresholding means a single threshold will be used for the entirety of the input image. We can break down the algorithm into 3 simple steps:

1. Process image and obtain its histogram
2. Compute threshold value  $T$
3. Replace image pixels in region whose saturation is greater than  $T$  to white and black for opposite case

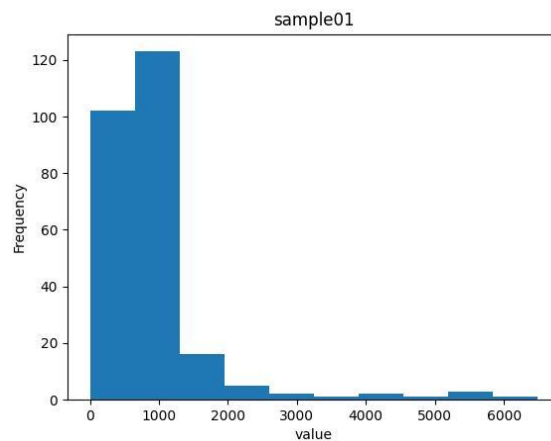
The threshold value  $T$  would vary from images to images. We can organize the process into simple steps as well:

1. Perform normalization on histogram
2. Get centers of bins
3. Iterate over all possible thresholds and get probabilities weight 1 and weight 2
4. Get class means  $\mu_0(t)$
5. Get class means  $\mu_1(t)$
6. Get inter-class variance
7. Obtain threshold value

After obtaining the threshold value, we will apply the thresholding to the image to obtain the binarized image:



As we can observe, the result of the binarization is extremely poor. The output image will in fact cause the OCR engine to perform worse as a significant portion of the text has been blacked out. A reason for the poor binarization is that we are using global thresholding which does not take into account relationship between pixels and assumes that the image has a bimodal histogram (another way we can put it: even illumination). We can visually inspect the image and see that it is not the case. We can plot the histogram for visualization:



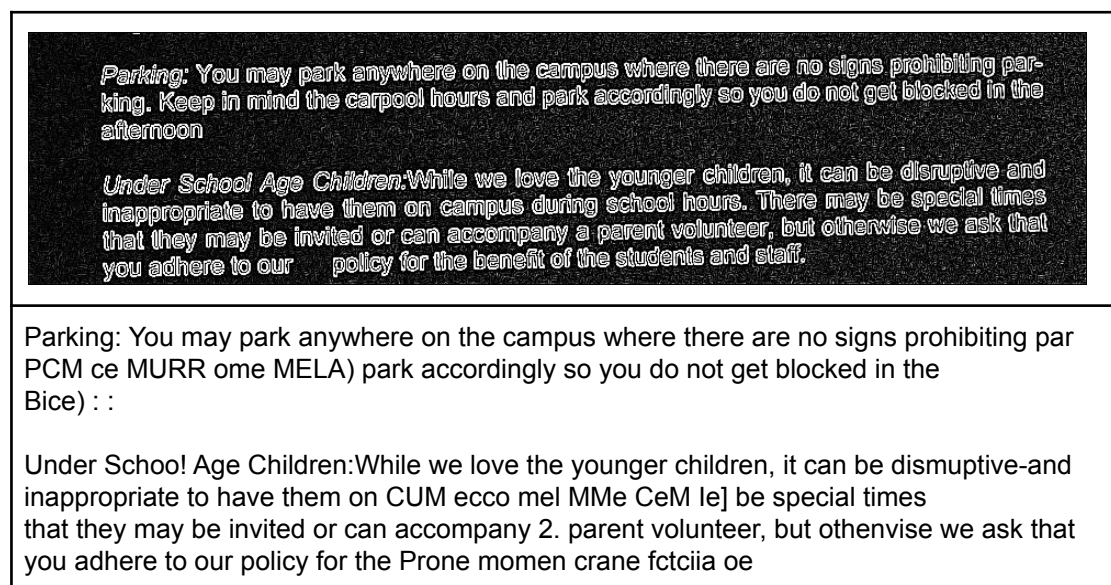
We can clearly see that the histogram is not bimodal. The background transitions from light pixels to dark which will cause an issue when the algorithm tries to calculate an optimal threshold. This is the limitation of global otsu thresholding.

### 3. Improvements

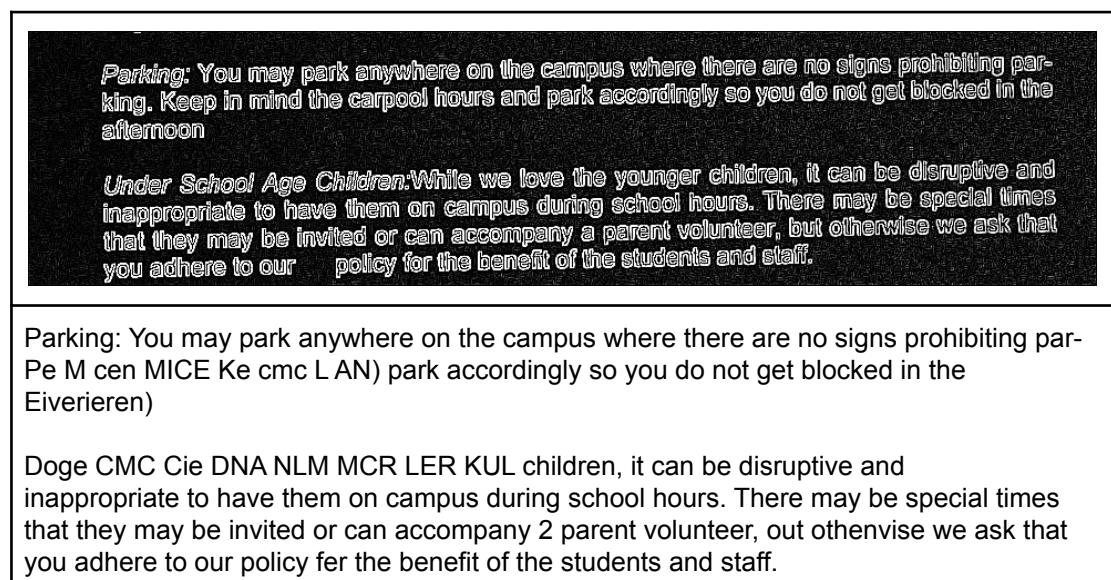
#### Improvement 1: Using Edge Detection

A simple way to improve OCR accuracy is to perform edge detection on the image and such that only the outline of the text can be seen on the image. This way we can circumvent uneven illumination within the image. We perform various different edge detection algorithms to observe the accuracy.

##### a. Sobel mask



##### b. Prewitt Mask



### c. Canny

*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.

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

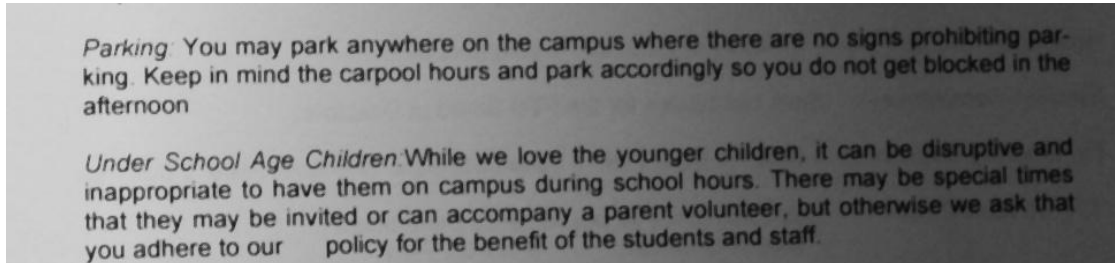
Under School Age Children: While we love the younger children, it can be disruptive and inappropriate to have them on campus during school hours. There may be special times that they may be invited or can accompany a parent volunteer, but otherwise we ask that you adhere to our policy for the benefit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

We can observe that edge detection algorithms do in fact improve the accuracy of the OCR engine as opposed to simply using otsu thresholding. By presenting an image with only the outline of the characters, the OCR engine will have a much easier time detecting the characters. However we can see that the result is far from perfect. Additionally, if the image has other edges detected, it may cause the OCR accuracy to suffer even further. This is still not the optimal solution we hope for.

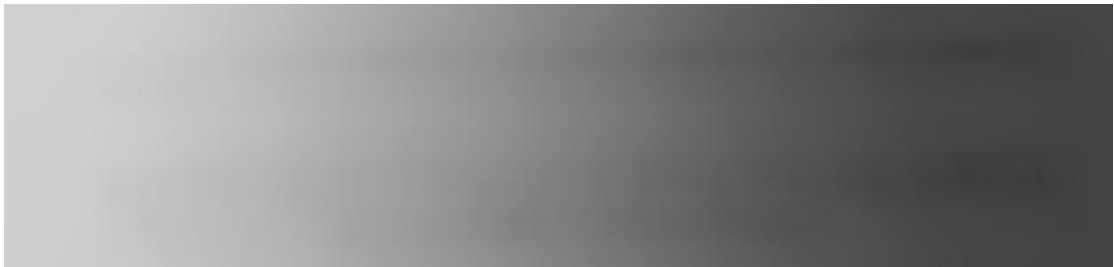
## Improvement 2: Pixel Division

The problem with the sample image is the uneven illumination that can be seen in the image. To counter this problem, we can first apply median blur to obtain the following two images:

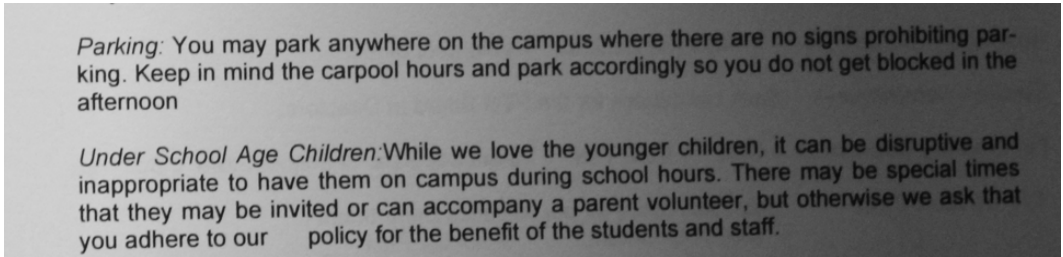
Blurred image 1



Blurred image 2



We can see that the second blurred image obtained is essentially a background mask for the image. Using this mask, we can “remove” the background of the first image to obtain a resulting image with just the foreground text. To do so, we can perform a division operation on the two images. The division between two images (basically two numpy arrays in our code) gives us a fractional change between the corresponding pixels. This is also known as ratioing. Given the fractional change we can scale it back to the original range of pixel intensity and the resulting image is in fact the foreground text. We can visually deduce that the only difference between the first and second image is the foreground text. In practice, we perform a numpy masked array division between the two blurred images before scaling back the pixel intensity to get the foreground text (second image in the table below).


<p><i>Parking:</i> 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</p> <p><i>Under School Age Children:</i> 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.</p>

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.

As we can see, the result is almost perfect with the exception of the “\_”. With only the foreground text left in the image, the OCR engine will have an easier time identifying the characters. This method appears to be highly effective. We can explore further to see what other techniques we can apply.

### Improvement 3: Adaptive Thresholding

We apply adaptive thresholding to address the shortcomings of global thresholding. In adaptive thresholding we calculate the threshold value for different smaller regions of an image. Within these regions, the illumination should be even. We apply the different thresholds within each region accordingly to obtain the final binarized image.

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.

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.

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.

From the result we can observe that the translation is perfect with the exception of an extra “\_” in place of a space. This outperforms all previous improvements by a huge margin. The potential issue with adaptive thresholding would be deciding on the size of the regions. If the regions are too large the illumination may not be uniform enough causing some degradation to the binarized image.



#### **4. Possible improvements to recognition algorithm**

Tesseract is a Long Short Term Memory (LSTM) based engine. It is a type of Recurrent Neural Network (RNN) which is a type of network which specializes in Natural Language Processing. LSTMs are designed to address exploding and vanishing gradient problems which may appear in RNN training. The key concept that LSTM employs is a “state” which is a persistent module representing a past history allowing the LSTM to “remember” context. This in turn allows gradients to flow for a long duration and allows weights to learn based on context.

One possible improvement to tackle the issue of image degradation is to use image upscaling and character reconstruction. Sometimes images that are fed to the OCR are in low resolution which affects character recognizability. We can address this by upsampling the image to achieve a higher resolution image. We can use General Adversarial Networks (GANs). During training, a high resolution image is downsampled to low resolution and the generator of the GANs up samples it again. The discriminator compares the new image with the original and backpropagates the loss. Through increasing image resolution, we can increase clarity of the text characters.

Sometimes the characters may be deformed leading to reduced accuracy. We can perform reconstruction on the characters using a CNN or in some cases, simply perform edge linking with hough transform. Theoretically, the properly reconstructed characters would be easily recognizable by the OCR engine.

Another potential improvement to the OCR engine is to use a Transformer. Much recent research has been done on this topic and results are very promising. LSTM was created to address the issue of disappearing or vanishing gradients due to long-term dependencies in a normal RNN. However, as it needs to “remember” past history it has many parameters that will cause the memory size to go up. Transformers on the other hand are completely attention-based and no recursion is required. It relies on parallel computation resulting in lowered training time and computing resources. Transformers are being used on image captioning, speech recognition, text summarization and many other tasks. It is quite likely that it will be able to outperform LSTMs such as Tesseract.

#### **5. Conclusion**

In conclusion, we can observe that preprocessing is important leading to a high OCR accuracy. Otsu thresholding is one such preprocessing but suffers when images do not have a bimodal histogram. Thus to solve this issue, we can use adaptive thresholding on the basis that individual regions within the image have a bimodal histogram. An important factor in this method is the size of the region. Outside of thresholding, edge detection algorithms can create an image with only the characters’ outline and circumvent the uneven illumination. However in the event where other edges appear in the image, it most likely will cause the OCR accuracy to suffer. Another method we explored is the use of a background mask to remove the background of the image leaving only the foreground text. We create the mask using median blurring. The resulting accuracy of the OCR was promising but alas not as good as adaptive thresholding. From the experiments we can conclude that adaptive thresholding is the best technique we can use to improve the OCR accuracy.

## 6. Appendix

```
import cv2

import argparse

import numpy as np

import pytesseract

import matplotlib.pyplot as pl

import math


debug_dir = './debug'

im_dir = './images'


# otsu global thresholding implementation

def otsu_t (img):

    print('Calculating Otsu threshold')

    ## de-noising

    # add gaussian blur to reduce image noise

    img = cv2.GaussianBlur(img, (5,5),0)


    cv2.imwrite(f'{debug_dir}/gblur_{args.imname}.jpg', img) # checkpoint 1 gaussian blur


    # get image histogram

    bins = 256

    hist, bin_edges = np.histogram(img, bins=bins)


    # save histogram plot for analysis later

    pl.hist(hist)

    pl.title(f'{args.imname}')

    pl.xlabel("value")

    pl.ylabel("Frequency")

    pl.savefig(f'{debug_dir}/histplot_{args.imname}.jpg')


    # perform normalization
```

```

hist = np.divide(hist.ravel(), hist.max())

# get centers of bins
bin_mids = (bin_edges[:-1] + bin_edges[1:]) / 2

# iterate over all possible thresholds and get probabilities w1 and w2
w1 = np.cumsum(hist)
w2 = np.cumsum(hist[::-1])[::-1]

# get mean of class u0(t) and u1(t)
mu0 = np.cumsum(hist * bin_mids) / w1
mu1 = (np.cumsum((hist * bin_mids)[::-1]) / w2[::-1])[::-1]

# get inter class variance
inter_class_variance = w1[::-1] * w2[1:] * (mu0[::-1] - mu1[1:]) ** 2

# maximise inter class variance
i = np.argmax(inter_class_variance)

# get threshold value
th = bin_mids[::-1][i]

print(f'Threshold calculated: {th}')

_, output = cv2.threshold(img, th, 255, cv2.THRESH_BINARY)
print('Otsu thresholding Applied')

return output

```

### # Using edge detectors

# takes in an additional flag parameter to

```
def improvement1 (img, flag=1):
```

```
    if flag == 1:
```

```
        # create sobel mask
```

```
        Gx = np.array([[1.0, 0.0, -1.0], [2.0, 0.0, -2.0], [1.0, 0.0, -1.0]])
```

```
        Gy = np.array([[1.0, 2.0, 1.0], [0.0, 0.0, 0.0], [-1.0, -2.0, -1.0]])
```

```
        [rows, columns] = np.shape(img) # we need to know the shape of the input grayscale image
```

```
        output = np.zeros(shape=(rows, columns)) # initialization of the output image array (all elements are 0)
```

```
        # Compute output in x and y directions
```

```
        for i in range(rows - 2):
```

```
            for j in range(columns - 2):
```

```
                gx = np.sum(np.multiply(Gx, img[i:i + 3, j:j + 3]))
```

```
                gy = np.sum(np.multiply(Gy, img[i:i + 3, j:j + 3]))
```

```
                output[i + 1, j + 1] = np.sqrt(gx ** 2 + gy ** 2)
```

```
        kernel = np.array([[-1,-1,-1], [-1,9,-1], [-1,-1,-1]])
```

```
        output = cv2.filter2D(output, -1, kernel)
```

```
    elif flag == 2:
```

```
        # create prewitt mask
```

```
        Gx = np.array([[1.0, 0.0, -1.0], [1.0, 0.0, -1.0], [1.0, 0.0, -1.0]])
```

```
        Gy = np.array([[1.0, 1.0, 1.0], [0.0, 0.0, 0.0], [-1.0, -1.0, -1.0]])
```

```
        [rows, columns] = np.shape(img)
```

```
        output = np.zeros(shape=(rows, columns))
```

```
        # Compute output in x and y directions
```

```
        for i in range(rows - 2):
```

```
            for j in range(columns - 2):
```

```
                gx = np.sum(np.multiply(Gx, img[i:i + 3, j:j + 3]))
```

```
                gy = np.sum(np.multiply(Gy, img[i:i + 3, j:j + 3]))
```

```
                output[i + 1, j + 1] = np.sqrt(gx ** 2 + gy ** 2)
```

```

    kernel = np.array([[ -1,-1,-1], [-1,9,-1], [-1,-1,-1]])
    output = cv2.filter2D(output, -1, kernel)
    output =otsu_t(output)
elif flag == 3:
    # use canny edge detector
    output = cv2.Canny(img, 40, 70)
return output

```

### **# Pixel Division**

```

def improvement2(img):
    # image
    output1 = cv2.medianBlur(img,3)
    # background mask
    output2 = cv2.medianBlur(img,51)

    # divide the 2 blur images to remove the background. Afterwards, normalize pixel intensity.
    output = np.ma.divide(output1, output2).data
    output = np.uint8(255 * output / output.max())

    # use simple 2D filter to sharpen the image
    kernel = np.array([[ -1,-1,-1], [-1,9,-1], [-1,-1,-1]])
    output = cv2.filter2D(output, -1, kernel)
return output

```

### **# adaptive thresholding**

```
def improvement3(img, threshold = 25):
```

```
    # Original image size
```

```
    orignrows, origncols = img.shape
```

```
    # region size
```

```
    width = int(np.floor(orignrows/16) + 1)
```

```
    height = int(np.floor(origncols/16) + 1)
```

```
    # border padding
```

```
    Mextend = round(width / 2) - 1
```

```
    Nextend = round(height / 2) - 1
```

```
    # image padding
```

```
    aux = cv2.copyMakeBorder(img, top=Mextend, bottom=Mextend,  
left=Nextend, right=Nextend, borderType=cv2.BORDER_REFLECT)
```

```
    region = np.zeros((width,height),np.int32)
```

```
    # calculate image integral
```

```
    imageIntegral = cv2.integral(aux, region, -1)
```

```
    # get integral image size
```

```
    nrows, ncols = imageIntegral.shape
```

```
    result = np.zeros((orignrows, origncols))
```

```
    # calculate cumulative pixels
```

```
    for i in range(nrows - width):
```

```
        for j in range(ncols - height):
```

```
            result[i, j] = imageIntegral[i + width, j + height] - imageIntegral[i, j + height] +  
imageIntegral[i, j] - imageIntegral[i + width, j]
```

```
    binary = np.ones((orignrows, origncols), dtype=np.bool)
```

```

img = (img).astype('float64') * width * height

# image binarization
binary[img <= result * (100.0 - threshold) / 100.0] = False
output = (255 * binary).astype(np.uint8)
return output

## Tesseract function
# takes in an image to be fed into the LSTM-based engine
def translate(img):
    print('Translating image...')
    # Adding custom options
    custom_config = r'--oem 3 --psm 6'
    result = pytesseract.image_to_string(img, config=custom_config)
    return result

# program entry point
if __name__ == '__main__':
    parser = argparse.ArgumentParser(description='Tesseract OCR program parser.')
    parser.add_argument('--imname', default='sample01')
    parser.add_argument('--otsu', default=True)
    parser.add_argument('--improvement1', default=True)
    parser.add_argument('--improvement2', default=True)
    parser.add_argument('--improvement3', default=True)
    args = parser.parse_args()

    print('Reading image...')

    # read image in greyscale mode
    img = cv2.imread(f'{im_dir}/{args.imname}.png', 0)

    ## Base Otsu thresholding
    if args.otsu:

```

```

output = otsu_t(img)
cv2.imwrite(f'./processed_{args.imname}.jpg', output)
ocr_result = translate(output)

with open(f'./result_otsu_{args.imname}.txt', 'w') as tf:
    tf.write(ocr_result)

## Improvement1: Edge detectors
if args.improvement1:
    # sobel mask
    output = improvement1(img, 1)
    cv2.imwrite(f'./processed_improved1_sobel_{args.imname}.jpg', output)
    output = cv2.imread(f'./processed_improved1_sobel_{args.imname}.jpg', 0)
    ocr_result = translate(output)

    with open(f'./result_otsu_improved1_sobel_{args.imname}.txt', 'w') as tf:
        tf.write(ocr_result)

    # prewitt mask
    output = improvement1(img, 2)
    cv2.imwrite(f'./processed_improved1_prewitt_{args.imname}.jpg', output)
    output = cv2.imread(f'./processed_improved1_prewitt_{args.imname}.jpg', 0)
    ocr_result = translate(output)

    with open(f'./result_otsu_improved1_prewitt_{args.imname}.txt', 'w') as tf:
        tf.write(ocr_result)

    # canny
    output = improvement1(img, 3)
    cv2.imwrite(f'./processed_improved1_canny_{args.imname}.jpg', output)
    output = cv2.imread(f'./processed_improved1_canny_{args.imname}.jpg', 0)
    ocr_result = translate(output)

```



```

with open(f'./result_otsu_improved1_canny_{args.imname}.txt', 'w') as tf:
    tf.write(ocr_result)

## Improvement2: Pixel division
if args.improvement2:
    output = improvement2(img)
    cv2.imwrite(f'./processed_improved2_{args.imname}.jpg', output)
    ocr_result = translate(output)

with open(f'./result_otsu_improved2_{args.imname}.txt', 'w') as tf:
    tf.write(ocr_result)

## Improvement3: Adaptive thresholding
if args.improvement3:
    output = improvement3(img)
    cv2.imwrite(f'./processed_improved3_{args.imname}.jpg', output)
    ocr_result = translate(output)

with open(f'./result_improved3_{args.imname}.txt', 'w') as tf:
    tf.write(ocr_result)

```