Automatic Noise Detection and Removal Pipeline

Veer Singh

July 31, 2021

1 Introduction

This pipeline takes in an image in grayscale. Then the watermark is removed, followed by checking for the type of noise. Based on if the noise is Gaussian or Impulse (salt and pepper) or none a flag is set up, in the next step the corresponding filter is applied and finally Otsu thresholding is applied.

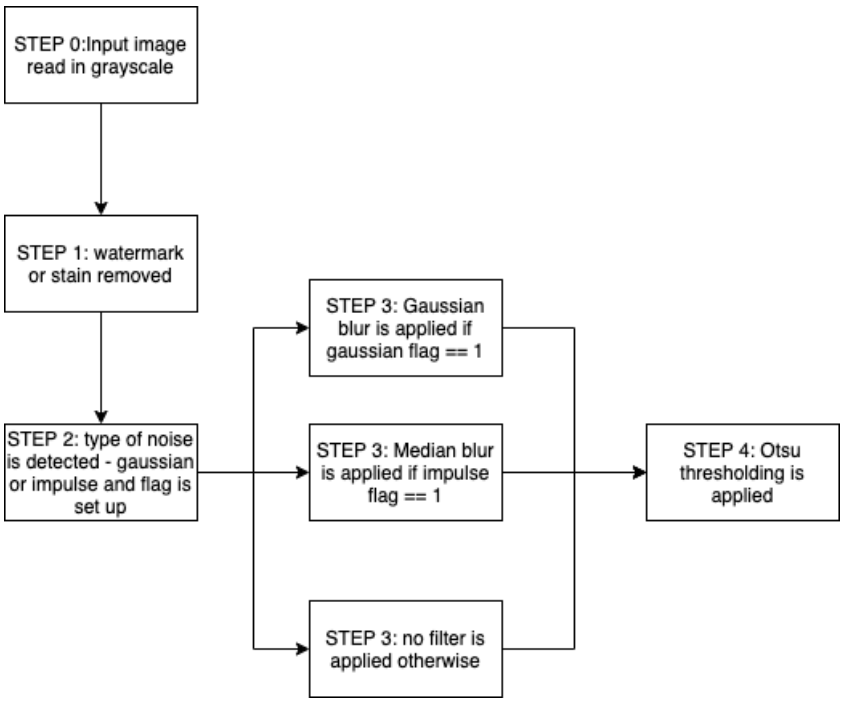

Figure 1: Pipeline Flowchart

2 Watermark Removal

2.1 Approach

I used a median blur with a larger kernel size, when this is applied to the image, it will output an image with only the watermark or stain, this is then subtracted from the original image, this gives us an image without the stain or watermark with flipped pixel values. The pixels are then flipped back with a bitwise not operation.

2.2 Class $\Rightarrow$ *watermark_removal*

Input: Any image file that can be read by cv2.imread()
Output: Image file without watermark/stain

```
import cv2
from preprocessing.watermark_removal import watermark_removal

input_image = 'image.jpg'
output_image = watermark_removal(input_image=input_image).output()

cv2.imwrite('Output.jpg', output_image)
```

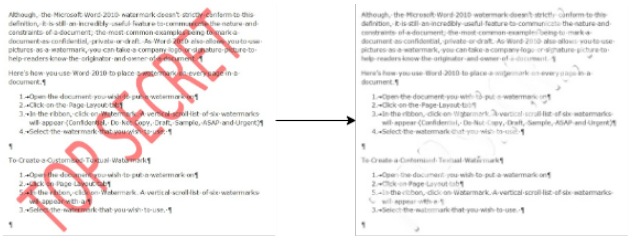

Figure 2: Watermark Removal Example

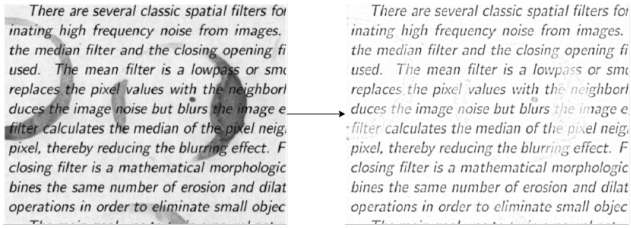

Figure 3: Stain Removal Example

3 Noise Type Detection

3.1 Approach

I implemented the Spike Detection Algorithm described in [1].

A lower bound and upper bound values are calculated for Gaussian type noise and Impulse type noise. A new image is then run through the SDT algorithm, if it falls inside any of these ranges then the Gaussian Flag or the Impulse Flag is set, otherwise no flag is set.

3.2 Spike Detection Algorithm

- This histogram of the grayscale image is calculated. We get 256 values, frequency of each intensity value from 0 - 255

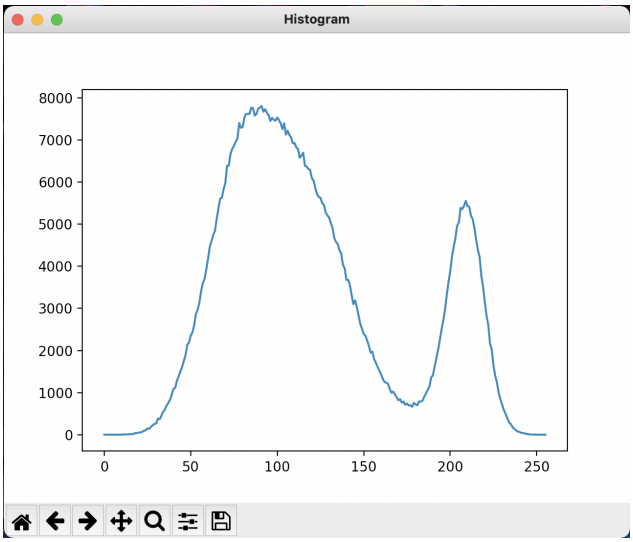

Figure 4: Histogram of Grayscale Image

- A vector D is calculated which is the difference of neighbouring intensity values.
  $D_i = H_{(i+1)} + H_i$ for all i=0...255

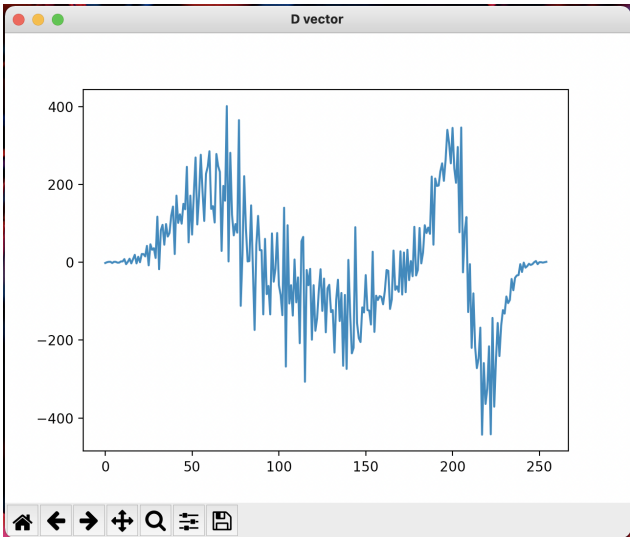

Figure 5: Vector D

- Minima NL1 and Maxima NL2 are calculated from vector D.
- Distance is calculated.
  $distance = NL2 - NL1$
- Finally the distance is normalized by diving it with the image size and multiplying with an arbitrary number, i chose 100000

3.3 Getting Range of Values for Gaussian and Impulse Noise using SDT Algorithm

I used the following dataset - `https://www.kaggle.com/veersingh230799/grayscale-images-with-gaussian-or-impulse-noise`

This dataset contains 1200 images with Gaussian noise in training set, 1200 images with Impulse noise in training and 200 images with Gaussian noise in test set and 200 images with Impulse noise.

The distance value range for each type of noise was calculated in the following way:

- SDT algorithm was applied to all 1200 images for each type of noise.
- This gave 1200 distance values. The mean and standard deviation were calculated.
- lower bound = mean - 2 * standard deviation
- upper bound = mean + 2 * standard deviation

3.4 Testing the Accuracy

I used the 200 images each in the test set to test the accuracy. The test images would run the SDT algorithm and their distance would be calculated. If this distance value was inside the range calculated previously then it was flagged as having that type of noise. Since all 200 images of a particular set had the noise, if 200 images were flagge then the accuracy would be 100%.

Output:

```
-----Gaussian Noise-----
lower bound = 96.04917198684099
upper bound = 326.5743861507359
Total test images = 200
Total flagged images = 144
Accuracy = 72.0%

-----Impulse Noise-----
lower bound = 4039.43981828374
upper bound = 8989.931753143906
Total test images = 200
Total flagged images = 191
```

```
Accuracy = 95.5%

Process finished with exit code 0
```

3.5 Class ⇒ *noise_type_detector*

Input: cv2.imread(img, cv2.IMREAD_GRAYSCALE)
Output: gaussian_flag, impulse_flag values set to either 0 or 1

```
import cv2
from preprocessing.noise_type_detector import noise_type_detector

# This image was taken from the impulse noise dataset
input_image = cv2.imread('image.jpg', cv2.IMREAD_GRAYSCALE)
gaussian_flag, impulse_flag = output_image = noise_type_detector(input_image).flag()

print(f'Gaussian Flag = {gaussian_flag}\n'
      f'Impulse Flag = {impulse_flag}')


########################################
Gaussian Flag = 0
Impulse Flag = 1

Process finished with exit code 0
########################################

import cv2
from preprocessing.noise_type_detector import noise_type_detector

# This image was taken from the gaussian noise dataset
input_image = cv2.imread('image.jpg', cv2.IMREAD_GRAYSCALE)
gaussian_flag, impulse_flag = output_image = noise_type_detector(input_image).flag()

print(f'Gaussian Flag = {gaussian_flag}\n'
      f'Impulse Flag = {impulse_flag}')


########################################
Gaussian Flag = 1
Impulse Flag = 0

Process finished with exit code 0
########################################
```

4 Applying the Noise Reduction Filters

4.1 Approach

Finally once we have detected the type of noise, we can apply the respective filters. Gaussian Blur is applied if Gaussian flag is set, Median Blur is applied if Impulse flag is set, otherwise no filter is applied. Then Otsu Thresholding is applied and we get the final image.

4.2 Class ⇒ *noise_reduction_apply*

Input: cv2.imread(img, cv2.IMREAD_GRAYSCALE)
Output: Image file with corresponding filter + thresholding applied

```
import cv2
from preprocessing.noise_type_detector import noise_type_detector
from preprocessing.noise_reduction_apply import noise_reduction_apply

image = '/Users/veersingh/Desktop/Internship/data-extraction/assets/impulse.jpg'
img = cv2.imread(image, cv2.IMREAD_GRAYSCALE)
```

```
# Check for gaussian noise or impulse noise
gaussian_flag, impulse_flag = noise_type_detector(img).flag()

# Apply filter depending on the flag
if gaussian_flag == 1:
    third = noise_reduction_apply(img).gaussian_blur()
    print('Gaussian Noise Detected')
elif impulse_flag == 1:
    third = noise_reduction_apply(img).median_blur()
    print('Impulse Noise Detected')
else:
    third = img
    print('None')

# Apply thresholding
output = noise_reduction_apply(third).thresholding()

cv2.imwrite('1.jpg', cv2.imread(image))
cv2.imwrite('2.jpg', third)
cv2.imwrite('3.jpg', output)
```

Figure 6: Testing image with Gaussian Noise

Figure 7: Testing image with Impulse Noise

5 Complete Pipeline

Implementing the complete pipeline.

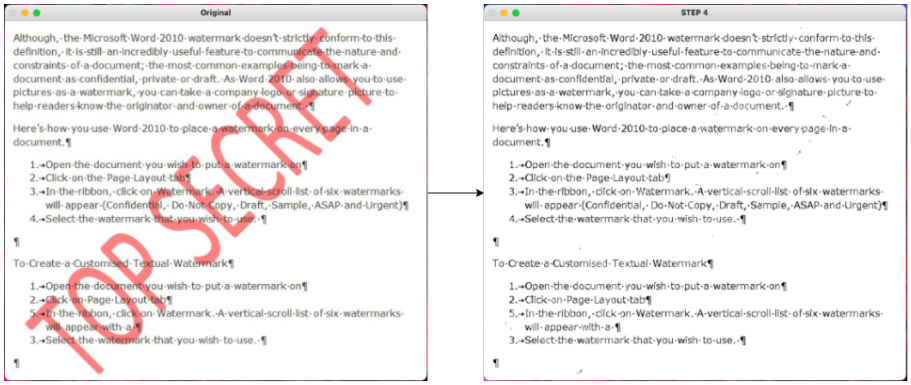

Figure 8: Complete Pipeline Example 1

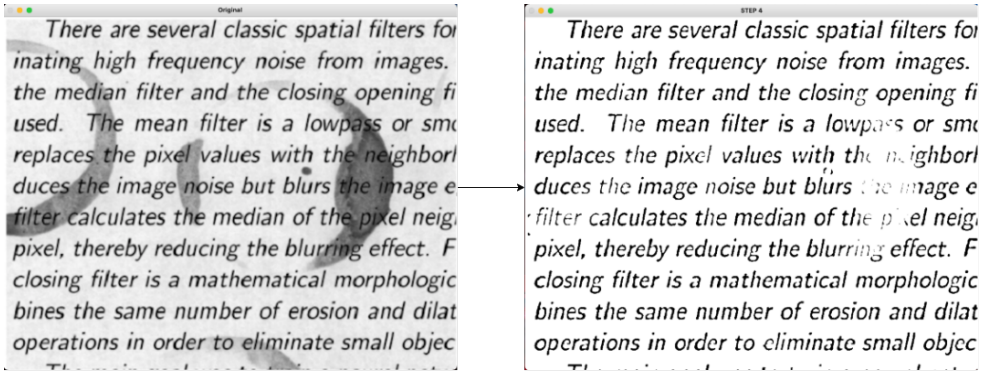

Figure 9: Complete Pipeline Example 2

References

[1] S. Indu and Chaveli Ramesh. A noise fading technique for images highly corrupted with impulse noise. In *2007 International Conference on Computing: Theory and Applications (ICCTA'07)*, pages 627–632, 2007.