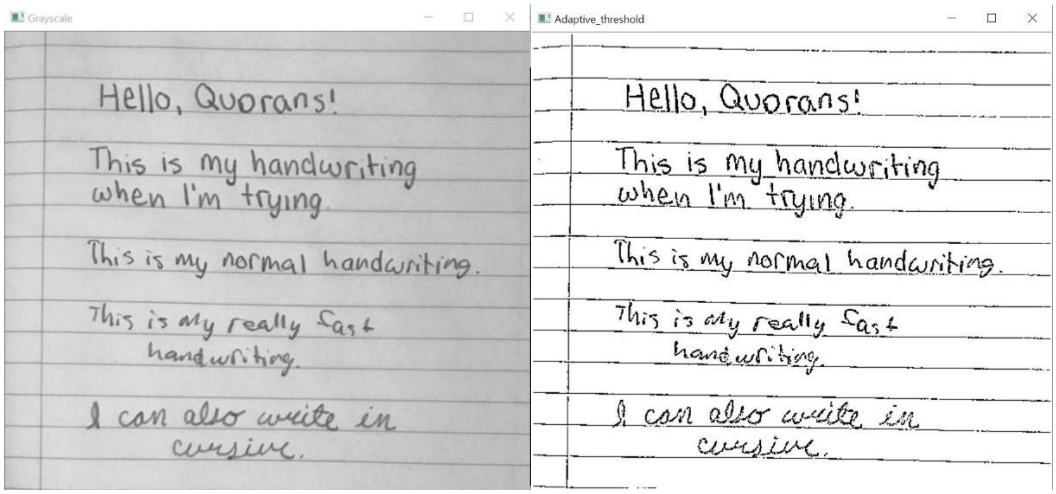
Technical Report

1829008 김민영

Practice : Adaptive Tresholding Using Moving Averages

1. result image



2. Explanation of code (**adaptiveThreshold.cpp**)

Main 함수: 사진을 입력받아와 Adaptice_thres 함수를 통해 결과 image를 출력한다.

Adaptive_thres 함수 : Moving Avearage 를 이용하여 Adaptive Thresholding 을 구현하고 결과를 반환한다.

$$g(i,j) = \begin{cases} 1 & if\ I(i,j) > T(i,j) \\ 0 & otherwise \end{cases}$$

$$T(i,j) = b \times m(i,j)$$

Using the mean intensity $m(i,j)$

$$m(i,j) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s,t) I(i+s, j+t)$$

```
// Initialiazing Kernel Matrix
        kernel = Mat::ones(kernel_size, kernel_size, CV_32F) / (float)(kernel_size *
kernel_size);
        float kernelvalue = kernel.at<float>(0, 0);  // To simplify, as the filter is
uniform. All elements of the kernel value are same.
        Mat output = Mat::zeros(row, col, input.type());
        for (int i = 0; i < row; i++) { //for each pixel in the output
                for (int j = 0; j < col; j++) {
                        float sum1 = 0.0;
                        for (int a = -n; a <= n; a++) { // for each kernel window
                                for (int b = -n; b <= n; b++) {
                                        if ((i + a <= row - 1) && (i + a >= 0) &&
(j + b <= col - 1) && (j + b >= 0)) { //if the pixel is not a border pixel
                                                sum1 += kernelvalue *
(float)(input.at<G>(i + a, j + b));
                                        }
```

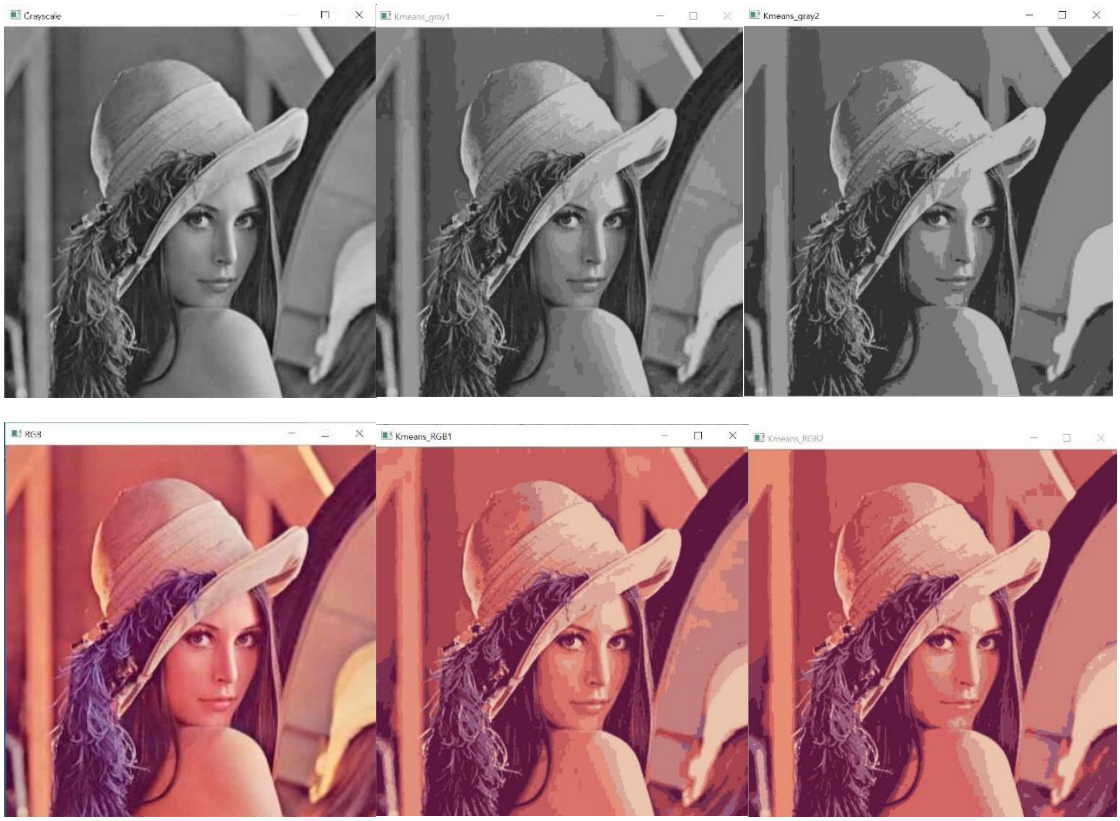
```
                                        }
                                }
                                float temp = bnumber * (G)sum1;
                                if (input.at<G>(i, j) > temp)
                                        output.at<G>(i, j) = 255;
                                else
                                        output.at<G>(i, j) = 0;
                        }
                }
                return output;
```

3. Analysis

Adaptive thresholding bases on moving averages : 물체사이즈가 작으면 잘 작동하며, 문서 처리에 유용한 방법인데 예상한대로 Result image가 잘 나온 것을 확인할 수 있다.

Practice : K-Means Clustering

1. result image



<original> <output1> <output2>

2. Explanation of code **(KMeans.cpp)**

Main 함수 : Kmeans_gray1, Kmeans_gray2, Kmeans_RGB1, Kmeans_RGB2를 호출하고 이미지를 출력한다.

Kmeans_gray1 함수 : intensity I

Kmeans_gray2 함수 : intensity + position(I,x/sigma,y/sigma)

Kmeans_RGB1 함수 : color(r,g,b)

Kmeans_RGB2 함수 : color+position(r,g,b,x/sigma,y/sigma)

로 하여 Kmeans clustering을 하고 각각의 output을 출력한다.

3. Analysis : Intensity + position 으로 pixels들을 그룹핑하니 Intensity 만 고려한 kmeans 보다 더 깔끔하게 clustering 되었다.

+ 추가 : 주어진 프로그램으로 Mean Shift Segmentation 실행하기(실행결과 첨부)