Open Software Project

Assignment 05 Technical Report

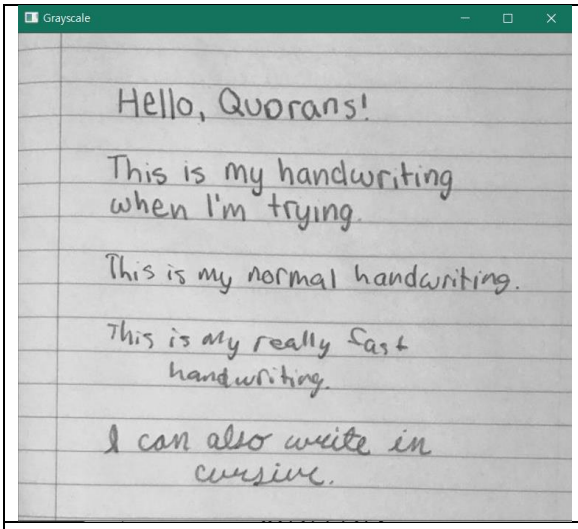
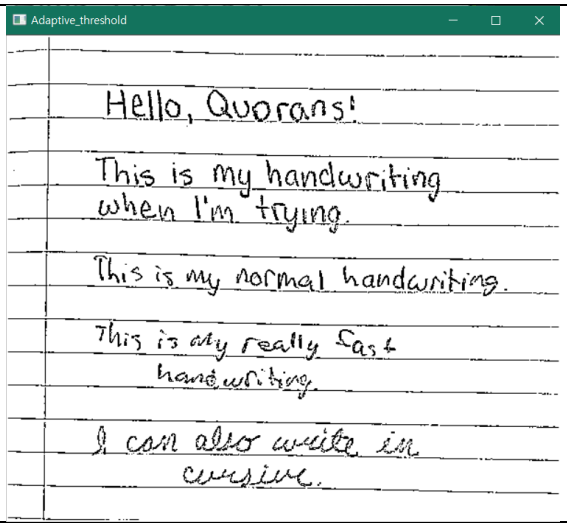
1976235 오진솔

1. AdaptiveThreshold.cpp

```
Mat kernel;

int row = input.rows;
int col = input.cols;
int kernel_size = (2 * n + 1);

// Initialiazing Kernel Matrix
// To simplify, as the filter is uniform. All elements of the kernel value are same.
kernel = Mat::ones(kernel_size, kernel_size, CV_32F) / (float)(kernel_size * kernel_size);
float kernelvalue = kernel.at<float>(0, 0);

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;
 // Fill code that finds the mean intensity using uniform mean filtering with zero paddle border process.
        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)) {
                    sum1 += kernelvalue * (float)(input.at<G>(i + a, j + b));
                }
            }
        }
        float temp = bnumber*(G)sum1;

// Fill code that makes output image's pixel intensity to 255 if the intensity of the input image is bigger
// than the temp value else 0.
        if (input.at<G>(i, j) > temp)     output.at<G>(i, j) = 255;
        else                              output.at<G>(i, j) = 0;
    }
}
return output;
```

uniform mean filtering을 적용한 후 weight b를 곱해서 temp에 저장한다.
픽셀의 intensity 값이 temp보다 크면 255, 작으면 0으로 thresholding한다.

즉 해당 픽셀의 값이 주변 픽셀의 평균보다 밝으면 흰색이 되고 어두우면 검정이 된다.

	
original	adaptive threshold

2. kmeans

```
#define color_type "RGB" // RGB or Grayscale
#define feature_space "intensity_and_position"   //intensity_only or intensity_and_position
```

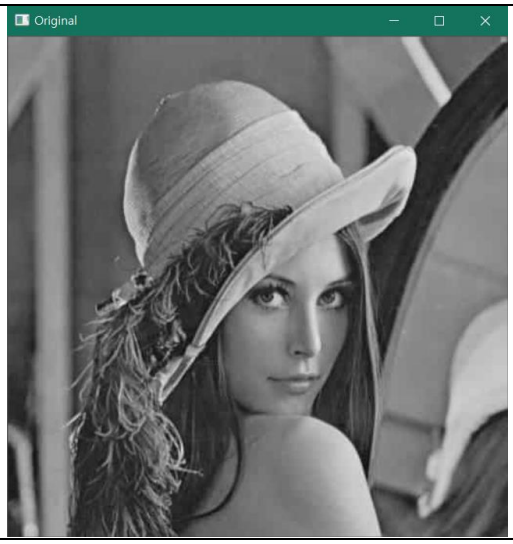
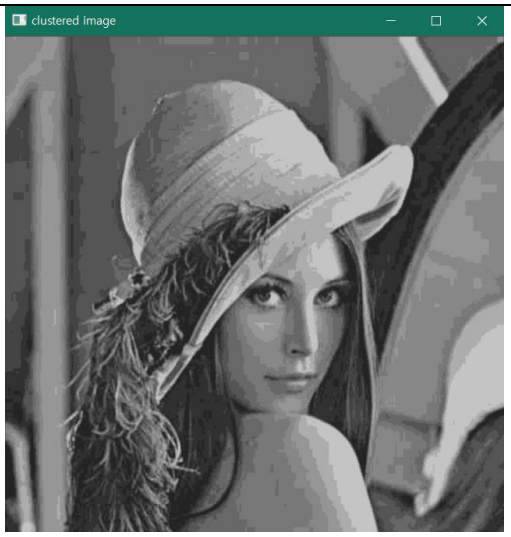
color_type과 feature_space 값을 변경하여 4가지 경우의 k-means clustering을 수행할 수 있습니다.

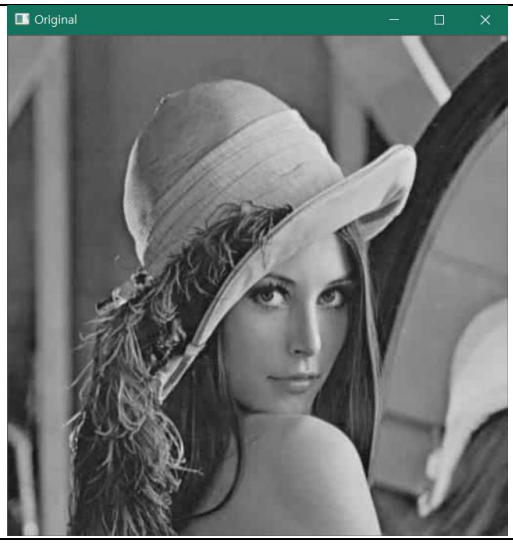
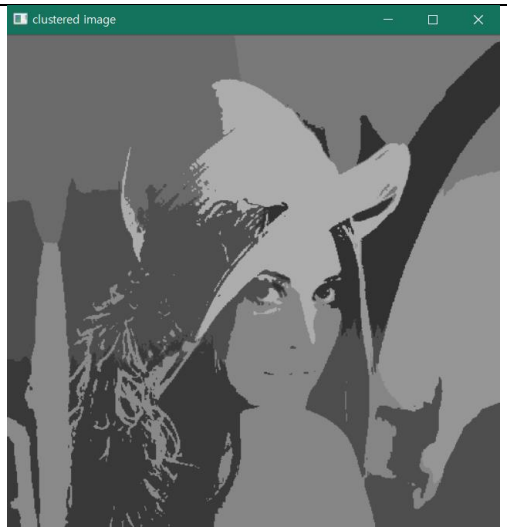
- Grayscale, intensity_only
- Grayscale, intensity_and_position
- RGB, intensity_only
- RGB, intensity_and_position

```
<Grayscale>
if (!strcmp(color_type, "Grayscale")) {
    cvtColor(input, input, CV_RGB2GRAY);
    if (!strcmp(feature_space, "intensity_only")) {
        samples = Mat(input.rows * input.cols, 1, CV_32F);
        for (int y = 0; y < input.rows; y++
            for (int x = 0; x < input.cols; x++)
                samples.at<float>(y + x * input.rows, 0) = input.at<uchar>(y, x);
    }
    else if (!strcmp(feature_space, "intensity_and_position")) {
        samples = Mat(input.rows * input.cols, 3, CV_32F);
        for (int y = 0; y < input.rows; y++) {
            for (int x = 0; x < input.cols; x++) {
                samples.at<float>(y + x * input.rows, 0) = (float)(input.at<uchar>(y, x)) / 255;
                samples.at<float>(y + x * input.rows, 1) = (float)y / input.rows;
                samples.at<float>(y + x * input.rows, 2) = (float)x / input.cols;
}}}}

(중략)

kmeans(samples, clusterCount, labels, TermCriteria(CV_TERMCRIT_ITER | CV_TERMCRIT_EPS, 10000,
0.0001), attempts, KMEANS_PP_CENTERS, centers);

(중략)

if (!strcmp(color_type, "Grayscale")) {
    if (!strcmp(feature_space, "intensity_only"))
        new_image.at<uchar>(y, x) = centers.at<float>(cluster_idx, 0);
    else if (!strcmp(feature_space, "intensity_and_position"))
        new_image.at<uchar>(y, x) = centers.at<float>(cluster_idx, 0) * 255;
```

intensity_only 는 intensity 값만 사용해서 clustering 한다.
Kmeans 를 수행한 뒤 labels 에는 해당 픽셀의 cluster 번호가, centers 에는 cluster center 의 좌표와 intensity 값이 들어있다. 따라서 두 결과를 조합해서 output 이미지를 만든다.

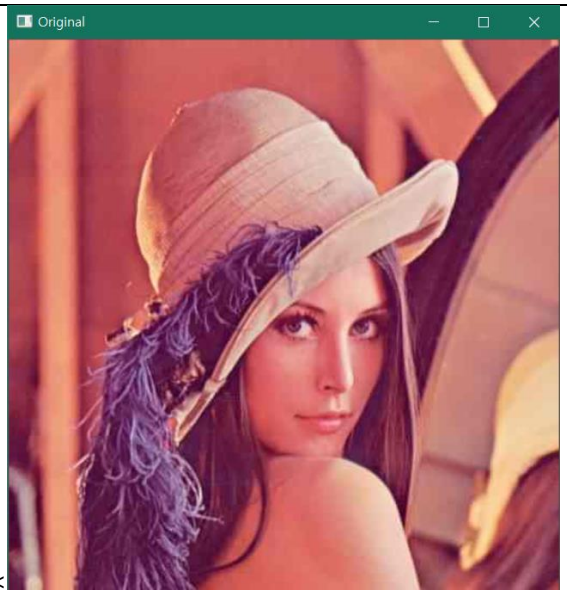
intensity_and_position 은 intensity 값과 위치를 이용해서 clustering 한다. 즉 intensity 가 같은 픽셀이라도 위치가 멀리 떨어져 있다면 서로 다른 cluster 에 배정될 가능성이 있다.
intensity 와 position 의 영향을 균등하게 하기 위해 두 값을 모두 normalize 하고 위와 같은 작업을 수행한다.

	
original	clustered (intensity only)

	
original	clustered (intensity and position)

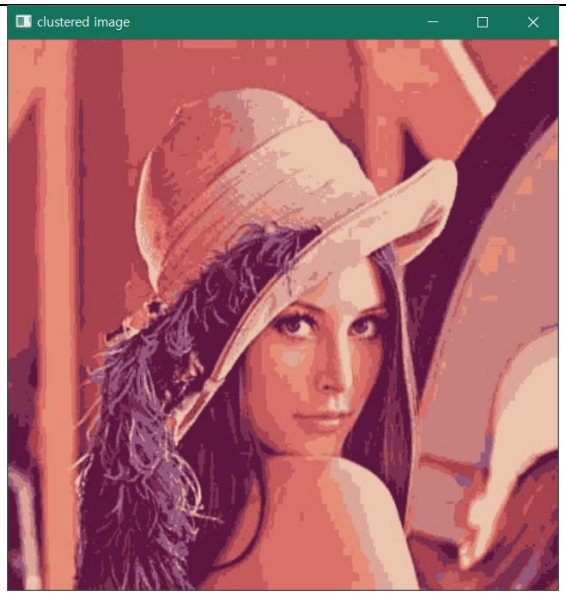
```
<RGB>
if (!strcmp(color_type, "RGB")) {
    if (!strcmp(feature_space, "intensity_only")) {
        samples = Mat(input.rows * input.cols, 3, CV_32F);
        for (int y = 0; y < input.rows; y++)
            for (int x = 0; x < input.cols; x++)
                for (int z = 0; z < 3; z++)
                    samples.at<float>(y + x * input.rows, z) = input.at<Vec3b>(y, x)[z]; }
    else if (!strcmp(feature_space, "intensity_and_position")) {
        samples = Mat(input.rows * input.cols, 5, CV_32F);
        for (int y = 0; y < input.rows; y++) {
           for (int x = 0; x < input.cols; x++){
               for (int z = 0; z < 3; z++){
                   samples.at<float>(y + x * input.rows, z) = (float)(input.at<Vec3b>(y, x)[z]) / 255;
               }
               samples.at<float>(y + x * input.rows, 3) = (float)y / input.rows;
               samples.at<float>(y + x * input.rows, 4) = (float)x / input.cols;
           }
        }
    }
}

(중략)

kmeans(samples, clusterCount, labels, TermCriteria(CV_TERMCRIT_ITER | CV_TERMCRIT_EPS, 10000,
0.0001), attempts, KMEANS_PP_CENTERS, centers);

(중략)

else if (!strcmp(color_type, "RGB")) {
    if (!strcmp(feature_space, "intensity_only")) {
        new_image.at<Vec3b>(y, x)[0] = centers.at<float>(cluster_idx, 0);
        new_image.at<Vec3b>(y, x)[1] = centers.at<float>(cluster_idx, 1);
        new_image.at<Vec3b>(y, x)[2] = centers.at<float>(cluster_idx, 2);
    }
    else if (!strcmp(feature_space, "intensity_and_position")) {
        new_image.at<Vec3b>(y, x)[0] = centers.at<float>(cluster_idx, 0) * 255;
        new_image.at<Vec3b>(y, x)[1] = centers.at<float>(cluster_idx, 1) * 255;
        new_image.at<Vec3b>(y, x)[2] = centers.at<float>(cluster_idx, 2) * 255;
    }
}
```

R,G,B 채널을 나누어서 같은 작업을 수행한다.

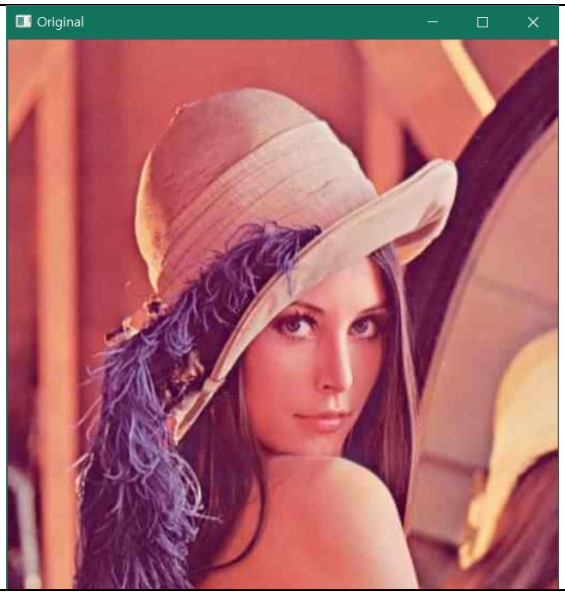


<

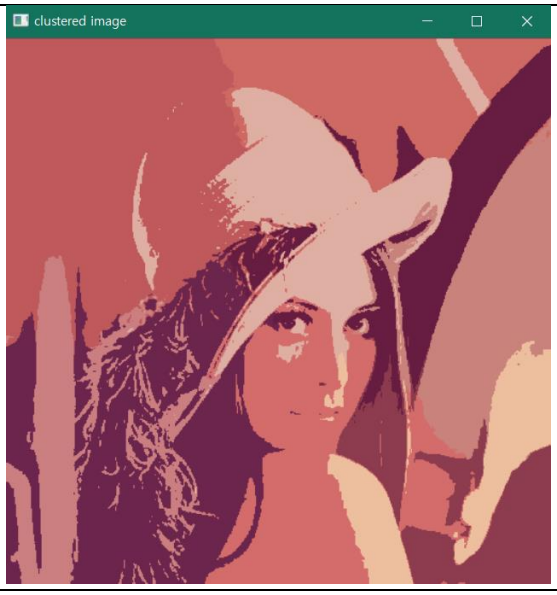
original



clustered (intensity only)

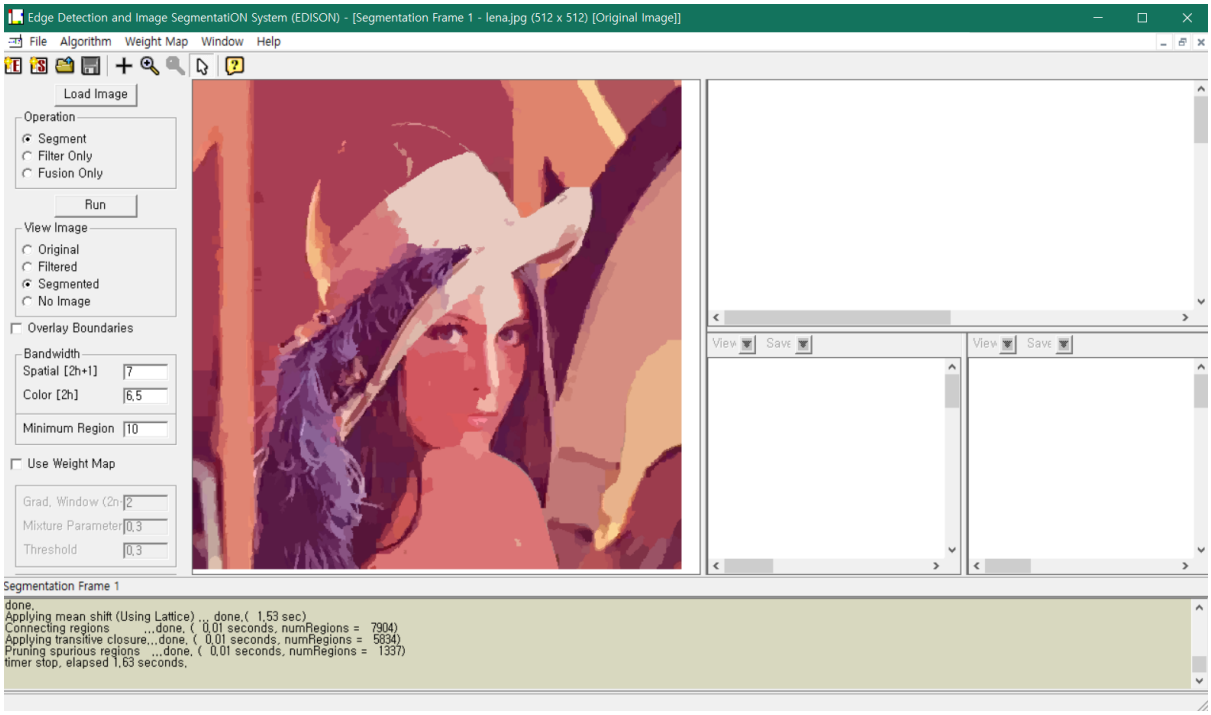


original

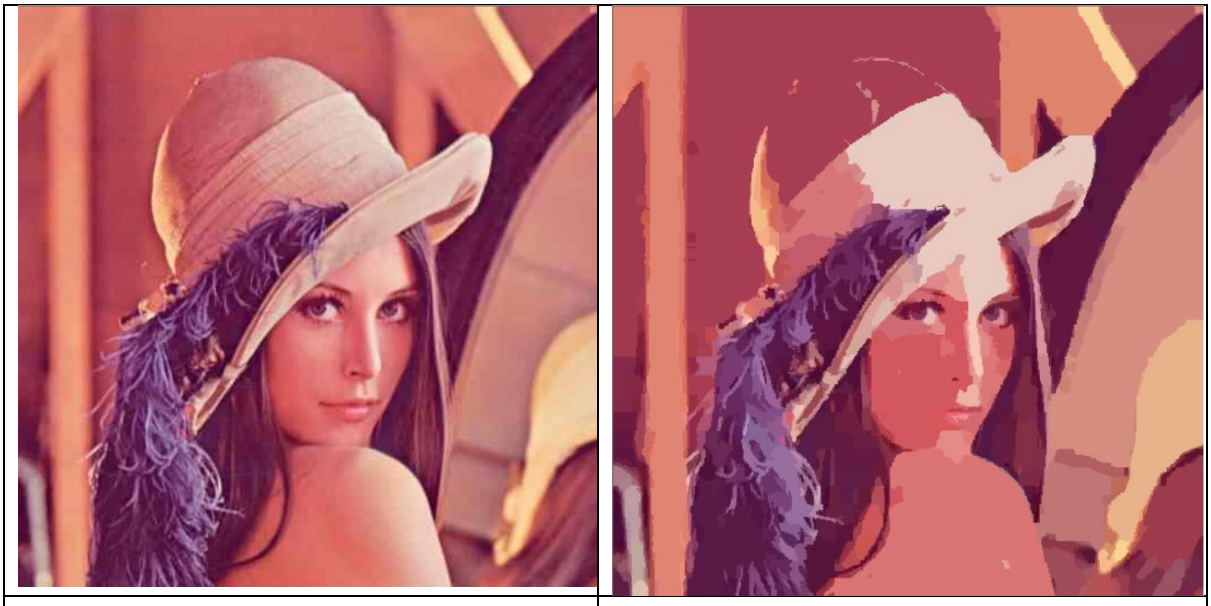


clustered (intensity and position)

3. Mean shift segmentation



Edge Detection and Image SegmentatiON (EDISON) System v1.1 를 사용함.



original	segmented