Digital Image Processing

LabWork #CV2

Professor : Jin-Woo Jung

* Using the given CPixel class, design a program with the following stops :

1.

* 1. Make histograms of each input image after converting to gray scale images

(Input:A=lenna.jpg, B=stuff\_color\_1.jpg file in given materials)

|  |  |  |  |
| --- | --- | --- | --- |
| **Input** | | **Output** | |
|  |  |  |  |
| code | | | |
| 1. lenna.png   #include <opencv2/opencv.hpp>  #include "Pixel.h"  using namespace cv;  int main()  {  CPixel processing; //pixel processing을 위한 클래스를 생성한다.  Mat convert; //input image를 8bit gray-scale로 변환하고 저장하기 위한 변수를 선언한다.  Mat image = imread("lenna.png", IMREAD\_COLOR); // lenna.png 를 불러온다. 이때 IMREAD\_COLOR 파라미터는 컬러 이미지를 불러온다는 것을 알려주기 위한 플래그 역할을 한다.  cvtColor(image, convert, CV\_BGR2GRAY); //image 변수에 저장된 이미지를 convert 변수에 8bit gray-scale로 변환하여 저장한다. 이때 3번째 파라미터에는 HSV로 변환하는 COLOR\_BGR2HSV, YCrCb로 변환하는 COLOR\_BGR2YCrCb, Luv로 변환하는 COLOR\_BGR2Luv 등등이 올 수 있다.  Mat histogram = processing.GS\_imhist(convert, 256); //gray-scale로 변환된 이미지을 이용하며 명암 히스토그램을 생성하고, 저장한다. 이 때 첫번째 파라미터는 히스토그램으로 표현할 이미지, 두번째 파라미터로는 픽셀의 빈도수 최댓값으로 기본적으로 256이라는 값이 들어간다.  //이 연산을 수행하면 히스토그램을 생성될 뿐만 아니라 빈도수 최댓값을 찾고, 그것을 이용하여 그래프 개형을 다시 그리게 된다.  imshow("image", histogram); // 히스토그램을 출력한다. 이때 첫번째 파라미터는 이미지를 출력하기 위해 띄우는 창의 제목표시줄에 출력할 제목이고, 두 번째 파라미터는 출력하기 위한 이미지를 가리키는 변수이다.  waitKey(); //키 입력이 있을 때까지 창을 유지하기 위해 waitkey() 함수를 이용한다.  return 0;  } | | | |
| 1. stuff\_color\_1.png   #include <opencv2/opencv.hpp>  #include "Pixel.h"  using namespace cv;  int main()  {  CPixel processing; //pixel processing을 위한 클래스를 생성한다.  Mat convert; //input image를 8bit gray-scale로 변환하고 저장하기 위한 변수를 선언한다.  Mat image = imread("stuff\_color\_1.png", IMREAD\_COLOR); // stuff\_color\_1.png 를 불러온다. 이때 IMREAD\_COLOR 파라미터는 컬러 이미지를 불러온다는 것을 알려주기 위한 플래그 역할을 한다.  cvtColor(image, convert, CV\_BGR2GRAY); //image 변수에 저장된 이미지를 convert 변수에 8bit gray-scale로 변환하여 저장한다. 이때 3번째 파라미터에는 HSV로 변환하는 COLOR\_BGR2HSV, YCrCb로 변환하는 COLOR\_BGR2YCrCb, Luv로 변환하는 COLOR\_BGR2Luv 등등이 올 수 있다.  Mat histogram = processing.GS\_imhist(convert, 256); //gray-scale로 변환된 이미지을 이용하며 명암 히스토그램을 생성하고, 저장한다. 이 때 첫번째 파라미터는 히스토그램으로 표현할 이미지, 두번째 파라미터로는 픽셀의 빈도수 최댓값으로 기본적으로 256이라는 값이 들어간다.  //이 연산을 수행하면 히스토그램을 생성될 뿐만 아니라 빈도수 최댓값을 찾고, 그것을 이용하여 그래프 개형을 다시 그리게 된다.  imshow("image", histogram); // 히스토그램을 출력한다. 이때 첫번째 파라미터는 이미지를 출력하기 위해 띄우는 창의 제목표시줄에 출력할 제목이고, 두 번째 파라미터는 출력하기 위한 이미지를 가리키는 변수이다.  waitKey(); //키 입력이 있을 때까지 창을 유지하기 위해 waitkey() 함수를 이용한다.  return 0;  } | | | |

* 1. Make equalized images of each input images (hint: use GS\_histeq function)

|  |  |  |  |
| --- | --- | --- | --- |
| **Input** | | **Output** | |
|  |  |  |  |
| code | | | |
| 1. lenna.png   #include <opencv2/opencv.hpp>  #include "Pixel.h"  using namespace cv;  int main()  {  CPixel processing; //pixel processing을 위한 클래스를 생성한다.  Mat convert; //input image를 8bit gray-scale로 변환하고 저장하기 위한 변수를 선언한다.  Mat image = imread("lenna.png", IMREAD\_COLOR); // lenna.png 를 불러온다. 이때 IMREAD\_COLOR 파라미터는 컬러 이미지를 불러온다는 것을 알려주기 위한 플래그 역할을 한다.  cvtColor(image, convert, CV\_BGR2GRAY); //image 변수에 저장된 이미지를 convert 변수에 8bit gray-scale로 변환하여 저장한다. 이때 3번째 파라미터에는 HSV로 변환하는 COLOR\_BGR2HSV, YCrCb로 변환하는 COLOR\_BGR2YCrCb, Luv로 변환하는 COLOR\_BGR2Luv 등등이 올 수 있다.  Mat equlization = processing.GS\_histeq(convert); // gray-scale로 변환된 이미지를 평활화 작업한다.  imshow("image", equlization); // 평활화 된 이미지를 출력한다. 이때 첫번째 파라미터는 이미지를 출력하기 위해 띄우는 창의 제목표시줄에 출력할 제목이고, 두 번째 파라미터는 출력하기 위한 이미지를 가리키는 변수이다.  waitKey(); //키 입력이 있을 때까지 창을 유지하기 위해 waitkey() 함수를 이용한다.  return 0;  }   1. stuff\_color\_1.png   #include <opencv2/opencv.hpp>  #include "Pixel.h"  using namespace cv;  int main()  {  CPixel processing; //pixel processing을 위한 클래스를 생성한다.  Mat convert; //input image를 8bit gray-scale로 변환하고 저장하기 위한 변수를 선언한다.  Mat image = imread("stuff\_color\_1.png", IMREAD\_COLOR); // stuff\_color\_1.png 를 불러온다. 이때 IMREAD\_COLOR 파라미터는 컬러 이미지를 불러온다는 것을 알려주기 위한 플래그 역할을 한다.  cvtColor(image, convert, CV\_BGR2GRAY); //image 변수에 저장된 이미지를 convert 변수에 8bit gray-scale로 변환하여 저장한다. 이때 3번째 파라미터에는 HSV로 변환하는 COLOR\_BGR2HSV, YCrCb로 변환하는 COLOR\_BGR2YCrCb, Luv로 변환하는 COLOR\_BGR2Luv 등등이 올 수 있다.  Mat equlization = processing.GS\_histeq(convert); // gray-scale로 변환된 이미지를 평활화 작업한다.  imshow("image", equlization); // 평활화 된 이미지를 출력한다. 이때 첫번째 파라미터는 이미지를 출력하기 위해 띄우는 창의 제목표시줄에 출력할 제목이고, 두 번째 파라미터는 출력하기 위한 이미지를 가리키는 변수이다.  waitKey(); //키 입력이 있을 때까지 창을 유지하기 위해 waitkey() 함수를 이용한다.  return 0;  } | | | |

* 1. Make equalized histograms of each equalized image (A\_equlizedHist, B\_equlizedHist)

|  |  |  |  |
| --- | --- | --- | --- |
| **Input** | | **Output** | |
|  |  |  |  |
| code | | | |
| 1. lenna.png   #include <opencv2/opencv.hpp>  #include "Pixel.h"  using namespace cv;  int main()  {  CPixel processing; //pixel processing을 위한 클래스를 생성한다.  Mat convert; //input image를 8bit gray-scale로 변환하고 저장하기 위한 변수를 선언한다.  Mat image = imread("lenna.png", IMREAD\_COLOR); // lenna.png 를 불러온다. 이때 IMREAD\_COLOR 파라미터는 컬러 이미지를 불러온다는 것을 알려주기 위한 플래그 역할을 한다.  cvtColor(image, convert, CV\_BGR2GRAY); //image 변수에 저장된 이미지를 convert 변수에 8bit gray-scale로 변환하여 저장한다. 이때 3번째 파라미터에는 HSV로 변환하는 COLOR\_BGR2HSV, YCrCb로 변환하는 COLOR\_BGR2YCrCb, Luv로 변환하는 COLOR\_BGR2Luv 등등이 올 수 있다.  Mat equlization = processing.GS\_histeq(convert); // gray-scale로 변환된 이미지를 평활화 작업한다.  Mat histogram = processing.GS\_imhist(equlization, 256);//평활화 된 이미지로 히스토그램을 생성하고, 저장한다. 이 때 첫번째 파라미터는 히스토그램으로 표현할 이미지, 두번째 파라미터로는 픽셀의 빈도수 최댓값으로 기본적으로 256이라는 값이 들어간다.  //이 연산을 수행하면 히스토그램을 생성될 뿐만 아니라 빈도수 최댓값을 찾고, 그것을 이용하여 그래프 개형을 다시 그리게 된다.  imshow("image", histogram); // 히스토그램을 출력한다. 이때 첫번째 파라미터는 이미지를 출력하기 위해 띄우는 창의 제목표시줄에 출력할 제목이고, 두 번째 파라미터는 출력하기 위한 이미지를 가리키는 변수이다.  waitKey(); //키 입력이 있을 때까지 창을 유지하기 위해 waitkey() 함수를 이용한다.  return 0;  }   1. stuff\_color\_1.png   #include <opencv2/opencv.hpp>  #include "Pixel.h"  using namespace cv;  int main()  {  CPixel processing; //pixel processing을 위한 클래스를 생성한다.  Mat convert; //input image를 8bit gray-scale로 변환하고 저장하기 위한 변수를 선언한다.  Mat image = imread("stuff\_color\_1.png", IMREAD\_COLOR); // stuff\_color\_1.png 를 불러온다. 이때 IMREAD\_COLOR 파라미터는 컬러 이미지를 불러온다는 것을 알려주기 위한 플래그 역할을 한다.  cvtColor(image, convert, CV\_BGR2GRAY); //image 변수에 저장된 이미지를 convert 변수에 8bit gray-scale로 변환하여 저장한다. 이때 3번째 파라미터에는 HSV로 변환하는 COLOR\_BGR2HSV, YCrCb로 변환하는 COLOR\_BGR2YCrCb, Luv로 변환하는 COLOR\_BGR2Luv 등등이 올 수 있다.  Mat equlization = processing.GS\_histeq(convert); // gray-scale로 변환된 이미지를 평활화 작업한다.  Mat histogram = processing.GS\_imhist(equlization, 256);//평활화 된 이미지로 히스토그램을 생성하고, 저장한다. 이 때 첫번째 파라미터는 히스토그램으로 표현할 이미지, 두번째 파라미터로는 픽셀의 빈도수 최댓값으로 기본적으로 256이라는 값이 들어간다.  //이 연산을 수행하면 히스토그램을 생성될 뿐만 아니라 빈도수 최댓값을 찾고, 그것을 이용하여 그래프 개형을 다시 그리게 된다.  imshow("image", histogram); // 히스토그램을 출력한다. 이때 첫번째 파라미터는 이미지를 출력하기 위해 띄우는 창의 제목표시줄에 출력할 제목이고, 두 번째 파라미터는 출력하기 위한 이미지를 가리키는 변수이다.  waitKey(); //키 입력이 있을 때까지 창을 유지하기 위해 waitkey() 함수를 이용한다.  return 0;  } | | | |

* 1. Open the project folder to check output images

Compare original(grayscale) image and equalized image

(A\_originalImg vs A\_equlizedImg, B\_originalImg vs B\_equlizedImg)

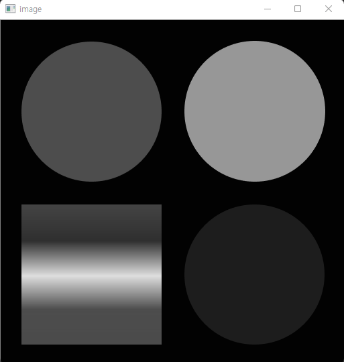
텍스트, 사람, 모자, 하얀색이(가) 표시된 사진

자동 생성된 설명 텍스트, 사람, 여자이(가) 표시된 사진

자동 생성된 설명

(A\_originalImg) (A\_ equlizedImg)

* A\_ equlizedImg가 A­\_ originalImg에 비해 채도가 높아진 것 같은 느낌이 들었습니다. 밝은 부분은 더 밝게 처리가 되었고, 어두운 부분은 더 어둡게 처리가 되어 영상이 보다 더 선명해졌습니다.

 텍스트, 전자기기이(가) 표시된 사진

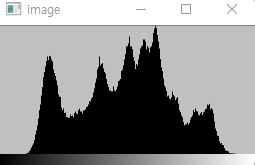
자동 생성된 설명

(B\_originalImg) (B\_ equlizedImg)

B\_equlizedImg 또한 B\_originalImg보다 채도가 높아졌다는 느낌이 들었고, 전체적으로 색상이 밝아졌다는 기분이 들었습니다. 또한 왼쪽 아래에 있는 , 그라데이션으로 처리된 사각형을 보면 색의 구분감이 원래 이미지보다 높아졌음을 볼 수 있었습니다.

Compare original histogram and equalized histogram

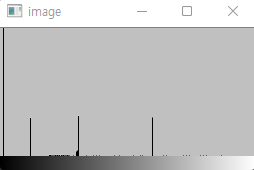
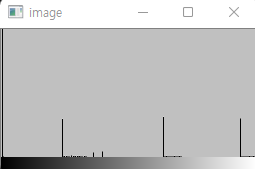
(A\_originalHist vs A\_equlizedHist, B\_originalHist vs B\_equlizedHist)

 음악이(가) 표시된 사진

자동 생성된 설명

(A\_originalHist) (A\_equlizedHist)

A의 원 히스토그램의 구간이 각 양쪽 끝까지 늘어난 것을 볼 수 있으며, 빈도수 또한 골고루 분포되도록 처리가 된 것을 볼 수 있습니다. A의 원 히스토그램을 보면 구간 사이사이의 간격이 동일했지만, 평활화를 진행하면서 구간 사이의 간격이 제각각으로 변한 것을 볼 수 있었습니다.

(B\_originalHist) (B\_equlizedHist)

B의 원 히스토그램을 살펴보면 B의 원 히스토그램 또한 왼쪽으로 치우친 것을 볼 수 있었는데, 평활화 작업을 수행하면서 왼쪽으로 치우친 그래프의 개형에서 골고루 퍼진 그래프의 개형으로 변화된 것을 볼 수 있었습니다.

2.

* 1. Convert an image to gray scale images (Input : A = lenna.jpg)

|  |  |
| --- | --- |
| **Input** | **Output** |
|  |  |
| code | |
| #include <opencv2/opencv.hpp>  #include "Pixel.h"  using namespace cv;  int main()  {  CPixel processing; //pixel processing을 위한 클래스를 생성한다.  Mat convert; //input image를 8bit gray-scale로 변환하고 저장하기 위한 변수를 선언한다.  Mat image = imread("lenna.png", IMREAD\_COLOR); // lenna.png 를 불러온다. 이때 IMREAD\_COLOR 파라미터는 컬러 이미지를 불러온다는 것을 알려주기 위한 플래그 역할을 한다.  cvtColor(image, convert, CV\_BGR2GRAY); //image 변수에 저장된 이미지를 convert 변수에 8bit gray-scale로 변환하여 저장한다. 이때 3번째 파라미터에는 HSV로 변환하는 COLOR\_BGR2HSV, YCrCb로 변환하는 COLOR\_BGR2YCrCb, Luv로 변환하는 COLOR\_BGR2Luv 등등이 올 수 있다.  imshow("image", convert); // gray-sclae로 변환된 이미지를 출력한다. 이때 첫번째 파라미터는 이미지를 출력하기 위해 띄우는 창의 제목표시줄에 출력할 제목이고, 두 번째 파라미터는 출력하기 위한 이미지를 가리키는 변수이다.  waitKey(); //키 입력이 있을 때까지 창을 유지하기 위해 waitkey() 함수를 이용한다.  return 0;  } | |

* 1. Threshold the image with threshold value 150 (Threshold type : THRESH\_BINARY)

|  |  |
| --- | --- |
| **Input** | **Output** |
|  |  |
| code | |
| #include <opencv2/opencv.hpp>  #include "Pixel.h"  using namespace cv;  int main()  {  CPixel processing; //pixel processing을 위한 클래스를 생성한다.  Mat convert; //input image를 8bit gray-scale로 변환하고 저장하기 위한 변수를 선언한다.  Mat image = imread("lenna.png", IMREAD\_COLOR); // lenna.png 를 불러온다. 이때 IMREAD\_COLOR 파라미터는 컬러 이미지를 불러온다는 것을 알려주기 위한 플래그 역할을 한다.  cvtColor(image, convert, CV\_BGR2GRAY); //image 변수에 저장된 이미지를 convert 변수에 8bit gray-scale로 변환하여 저장한다. 이때 3번째 파라미터에는 HSV로 변환하는 COLOR\_BGR2HSV, YCrCb로 변환하는 COLOR\_BGR2YCrCb, Luv로 변환하는 COLOR\_BGR2Luv 등등이 올 수 있다.  Mat binarization = processing.GS\_threshold(convert, 150, THRESH\_BINARY); //threshold 값을 이용하여 이진화 작업을 수행한다. 이떄 첫번째 파라미터는 이진화를 수행할 이미지, 두 번째 파라미터는 Threshold 값, 세번째 파라미터는 Threshold 값의 type이 들어간다.  //threshold의 type 종류에는 THRESH\_BINARY(픽셀 값이 treshold 값보다 큰 경우만 픽셀 남김), THRESH\_BINARY\_INV(픽셀 값이 threshold값보다 작은 경우만 픽셀 남김)  //THRESH\_TRUNC(픽셀 값이 threshold값보다 크면 픽셀 값을 threshold 값으로, threshold값보다 작으면 픽셀 값 그대로 유지)  //THRESH\_TOZERO(픽셀 값이 threshold값보다 크면 픽셀 값 그대로, 작으면 0으로 할당), THRESH\_TOZERO\_INV(픽셀 값이 threshold값보다 크면 0, 작으면 픽셀 값 그대로 할당)  //THRESH\_OTSU(otsu의 이진화 알고리즘을 이용하기 위해 사용), TRHESH\_MASK(흑색 이미지로 변환), THRESH\_TRIANGLE(Triangle 알고리즘 사용) 등이 있다.  imshow("image", binarization); // 이진화된 이미지를 출력한다. 이때 첫번째 파라미터는 이미지를 출력하기 위해 띄우는 창의 제목표시줄에 출력할 제목이고, 두 번째 파라미터는 출력하기 위한 이미지를 가리키는 변수이다.  waitKey(); //키 입력이 있을 때까지 창을 유지하기 위해 waitkey() 함수를 이용한다.  return 0;  } | |

* 1. Threshold the image with threshold value 0

(Threshold type : THRESH\_BINARY | THRESH\_OTSU)

|  |  |
| --- | --- |
| **Input** | **Output** |
|  |  |
| code | |
| #include <opencv2/opencv.hpp>  #include "Pixel.h"  using namespace cv;  int main()  {  CPixel processing; //pixel processing을 위한 클래스를 생성한다.  Mat convert; //input image를 8bit gray-scale로 변환하고 저장하기 위한 변수를 선언한다.  Mat image = imread("lenna.png", IMREAD\_COLOR); // lenna.png 를 불러온다. 이때 IMREAD\_COLOR 파라미터는 컬러 이미지를 불러온다는 것을 알려주기 위한 플래그 역할을 한다.  cvtColor(image, convert, CV\_BGR2GRAY); //image 변수에 저장된 이미지를 convert 변수에 8bit gray-scale로 변환하여 저장한다. 이때 3번째 파라미터에는 HSV로 변환하는 COLOR\_BGR2HSV, YCrCb로 변환하는 COLOR\_BGR2YCrCb, Luv로 변환하는 COLOR\_BGR2Luv 등등이 올 수 있다.  Mat binarization1 = processing.GS\_threshold(convert, 0, THRESH\_BINARY); //threshold 값을 이용하여 이진화 작업을 수행한다. 이떄 첫번째 파라미터는 이진화를 수행할 이미지, 두 번째 파라미터는 Threshold 값, 세번째 파라미터는 Threshold 값의 type이 들어간다.  //threshold의 type 종류에는 THRESH\_BINARY(픽셀 값이 treshold 값보다 큰 경우만 픽셀 남김), THRESH\_BINARY\_INV(픽셀 값이 threshold값보다 작은 경우만 픽셀 남김)  //THRESH\_TRUNC(픽셀 값이 threshold값보다 크면 픽셀 값을 threshold 값으로, threshold값보다 작으면 픽셀 값 그대로 유지)  //THRESH\_TOZERO(픽셀 값이 threshold값보다 크면 픽셀 값 그대로, 작으면 0으로 할당), THRESH\_TOZERO\_INV(픽셀 값이 threshold값보다 크면 0, 작으면 픽셀 값 그대로 할당)  //THRESH\_OTSU(otsu의 이진화 알고리즘을 이용하기 위해 사용), TRHESH\_MASK(흑색 이미지로 변환), THRESH\_TRIANGLE(Triangle 알고리즘 사용) 등이 있다.  Mat binarization2 = processing.GS\_threshold(convert, 0, THRESH\_OTSU); //threshold 값을 이용하여 이진화 작업을 수행한다. 이떄 첫번째 파라미터는 이진화를 수행할 이미지, 두 번째 파라미터는 Threshold 값, 세번째 파라미터는 Threshold 값의 type이 들어간다.    Mat OR = processing.GS\_gray\_logic(binarization1, binarization2, 2); // 두 이미지를 이용하여 bitwise을 수행한다. 이때 세번째 파라미터에 따라 AND(0), NAND(1), OR(2), NOR(3), XOR(4), DIFFERENCE(5) 연산을 수행할 수 있다.  imshow("image", OR); // 이진화된 이미지를 출력한다. 이때 첫번째 파라미터는 이미지를 출력하기 위해 띄우는 창의 제목표시줄에 출력할 제목이고, 두 번째 파라미터는 출력하기 위한 이미지를 가리키는 변수이다.  waitKey(); //키 입력이 있을 때까지 창을 유지하기 위해 waitkey() 함수를 이용한다.  return 0;  } | |

* 1. Adaptive threshold the image with maxvalue 150

(Adaptive method: ADAPTIVE\_THRESH\_MEAN\_C, Threshold type : THRESH\_BINARY)

|  |  |
| --- | --- |
| **Input** | **Output** |
|  |  |
| code | |
| #include <opencv2/opencv.hpp>  #include "Pixel.h"  using namespace cv;  int main()  {  CPixel processing; //pixel processing을 위한 클래스를 생성한다.  Mat convert; //input image를 8bit gray-scale로 변환하고 저장하기 위한 변수를 선언한다.  Mat image = imread("lenna.png", IMREAD\_COLOR); // lenna.png 를 불러온다. 이때 IMREAD\_COLOR 파라미터는 컬러 이미지를 불러온다는 것을 알려주기 위한 플래그 역할을 한다.  cvtColor(image, convert, CV\_BGR2GRAY); //image 변수에 저장된 이미지를 convert 변수에 8bit gray-scale로 변환하여 저장한다. 이때 3번째 파라미터에는 HSV로 변환하는 COLOR\_BGR2HSV, YCrCb로 변환하는 COLOR\_BGR2YCrCb, Luv로 변환하는 COLOR\_BGR2Luv 등등이 올 수 있다.  Mat adapt\_thresh = processing.GS\_adaptive\_threshold(convert, 150, ADAPTIVE\_THRESH\_MEAN\_C, THRESH\_BINARY); //적응형 이진화 작업을 수행한다. 첫번째 파라미터는 이진화를 수행할 이미지, 두번째 파라미터는 threshold의 값, 세번째 파라미터는 adaptive method, 네번째 파라미터는 Threshold 값의 type이 들어간다.  //adaptive method의 종류에는 ADAPTIVE\_THRESH\_MEAN\_C(주변영역의 평균값으로 결정), ADAPTIVE\_THRESH\_GAUSSIAN\_C(가중치들의 합을 활용)가 있다.  ////threshold의 type 종류에는 THRESH\_BINARY(픽셀 값이 treshold 값보다 큰 경우만 픽셀 남김), THRESH\_BINARY\_INV(픽셀 값이 threshold값보다 작은 경우만 픽셀 남김)  //THRESH\_TRUNC(픽셀 값이 threshold값보다 크면 픽셀 값을 threshold 값으로, threshold값보다 작으면 픽셀 값 그대로 유지)  //THRESH\_TOZERO(픽셀 값이 threshold값보다 크면 픽셀 값 그대로, 작으면 0으로 할당), THRESH\_TOZERO\_INV(픽셀 값이 threshold값보다 크면 0, 작으면 픽셀 값 그대로 할당)  //THRESH\_OTSU(otsu의 이진화 알고리즘을 이용하기 위해 사용), TRHESH\_MASK(흑색 이미지로 변환), THRESH\_TRIANGLE(Triangle 알고리즘 사용) 등이 있다.  imshow("image", adapt\_thresh); // 이진화된 이미지를 출력한다. 이때 첫번째 파라미터는 이미지를 출력하기 위해 띄우는 창의 제목표시줄에 출력할 제목이고, 두 번째 파라미터는 출력하기 위한 이미지를 가리키는 변수이다.  waitKey(); //키 입력이 있을 때까지 창을 유지하기 위해 waitkey() 함수를 이용한다.  return 0;  } | |

* 1. Open the project folder to check output images

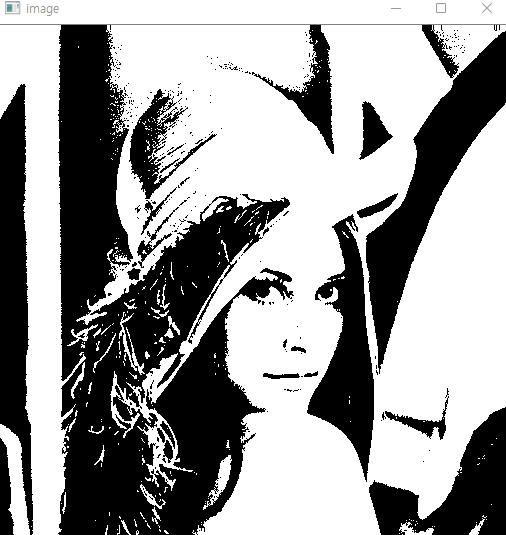
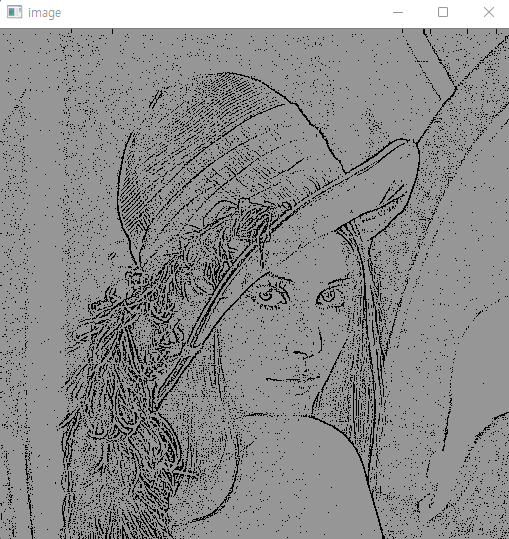
Compare the results of above steps

텍스트, 사람, 모자, 하얀색이(가) 표시된 사진

자동 생성된 설명 텍스트, 책이(가) 표시된 사진

자동 생성된 설명

(original image) (Thresholding image with threshold value 150)

(Thresholding image with otsu’s method) (Adaptive thresholding image)

오른쪽 상단 이미지를 살펴보면 threshold 값이 150으로 설정되어 있어, 150이하 픽셀 값은 모두 Black으로, 150 이상은 white로 처리된 모습입니다. threshold값이 높게 설정되어 있어서 전체적으로 어두운 이미지가 되었고, 배경과 인물이 제대로 분리되지 않은 모습을 볼 수 있습니다.

그러나 왼쪽 하단 이미지를 살펴보면 otsu가 고안한 method를 사용했기 때문에 오른쪽 상단 이미지보다 배경과 인물이 잘 분리 되었음을 볼 수 있고, 보다 선명한 이미지를 띠고 있습니다.

오른쪽 하단 이미지는 adaptive threshold를 활용한 것으로, 이미지의 구역구역마다 threshold를 수행하기 때문에 기존의 threshold 방법보다 배경과 인물이 잘 구분되어 결과로 출력되었음을 볼 수 있었습니다.