전기전자심화설계 및 소프트웨어실습 〈Assignment 1〉

담당교수 : 김원준
이름 : 201810528 고려욱

1. Image Resizing

1) Source Code

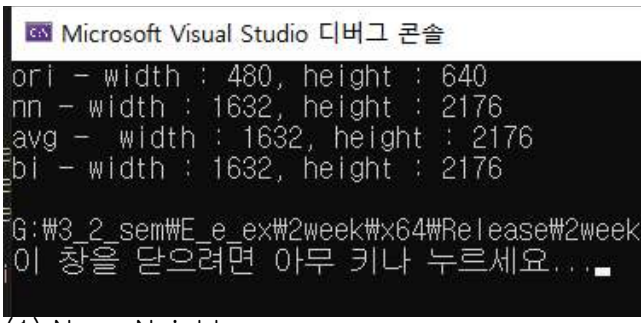
```
void imageResize(Mat input, float scale, int op)
{
    int x, y;
    int height, width;
    int re_height, re_width;
    float pos_x, pos_y;
    int sx, sy;
    float p, q;
    float p1[2], p2[2], p3[2], p4[2];

    height = input.rows;
    width = input.cols;
    re_height = (int)(scale * height);
    re_width = (int)(scale * width);

    Mat result(re_height, re_width, CV_8UC1);

    for (y = 0; y < re_height; y++) {
        for (x = 0; x < re_width; x++) {
            pos_x = (1.0 / scale) * x;
            pos_y = (1.0 / scale) * y;
            //Nearest Neighbor
            if (op == 1) {
                sx = (int)(pos_x + 0.5);
                sy = (int)(pos_y + 0.5);
                result.at<uchar>(y, x) = input.at<uchar>(sy, sx);
            }
            //Average
            else if (op == 2) {
                sx = (int)pos_x;
                sy = (int)pos_y;
                result.at<uchar>(y, x) = 0.25 * (input.at<uchar>(sy, sx) + input.at<uchar>(sy + 1, sx)
                    + input.at<uchar>(sy + 1, sx + 1) + input.at<uchar>(sy, sx + 1));
            }
            //Bi linear
            else if (op == 3) {
                sx = (int)pos_x;
                sy = (int)pos_y;
                p = pos_x - sx;
                q = pos_y - sy;
                result.at<uchar>(y, x) = (1 - p) * (1 - q) * (input.at<uchar>(sy, sx))
                    + p * (1 - q) * (input.at<uchar>(sy, sx + 1))
                    + (1 - p) * q * (input.at<uchar>(sy + 1, sx))
                    + p * q * (input.at<uchar>(sy + 1, sx + 1));
            }
            else {}
        }
    }
    if (op == 1) {
        imwrite("nn.jpg", result);
    }
    else if (op == 2) {
        imwrite("average.jpg", result);
    }
    else {
        imwrite("bi_lin.jpg", result);
    }
```

```
void main()
{
    float scale, degree;
    int option;
    Mat imgColor = imread("test2.jpg", CV_LOAD_IMAGE_GRAYSCALE);

    printf("Input your scale : ");
    scanf("%f", &degree);
    printf("1 : NN, 2 : Average, 3 : Bi-Linear₩n");
    //printf("1 : NN, 2 : Bi-Linear₩n");
    printf("Input your option : ");
    scanf("%d", &option);

    imageResize(imgColor, scale, option);
    //imageRotate(imgColor, degree, option);
}
```

2) 결과 사진

* scale-factor 3.4

* 사진 크기 확대 여부 출력



(1) Near-Neighbor

(2) Bi_linear ver.average

(3) Bi-Linear

(4) 결과분석

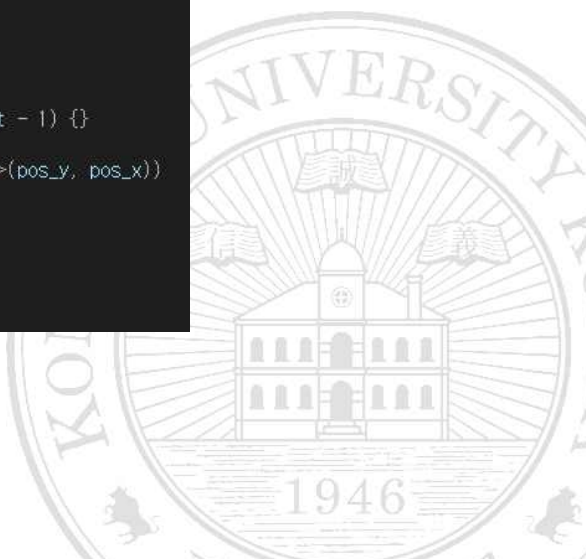
NN과 bi_linear의 사진을 비교 분석해보면 NN의 경우 물체의 테두리의 픽셀이 부드럽게 이어지지 않는다. 이에 반해 bi_linear의 경우 물체를 구별하는 테두리 픽셀이 굉장히 부드럽게 이어지고 있음을 알 수 있다.

NN의 경우 원래 이미지에서 가장 가까운 픽셀을 가져오기 때문에 이미지가 커질 경우 결과 이미지의 픽셀들 중 여러 픽셀이 원래 이미지의 한 픽셀의 값을 그대로 가져오기 때문에 픽셀들이 서로 부드럽게 연결되지 않게 보이게 된다. 반면에 bi_lienar의 경우 scale-factor를 고려한 선형보간법을 3중으로 실행하기 때문에 결과 이미지의 픽셀을 역으로 scale-factor를 고려하여 계산된 가상의 픽셀 주위의 실제 픽셀 4 픽셀의 이미지 값이 모두 고려되기 때문에 NN에 비교하여 훨씬 부드러운 확대/축소 이미지를 얻을 수 있다.

2. Image Rotate

1) Source Code

```
void imageRotate(Mat input, float degree, int op)
{
    int x, y;
    int height, width;
    float rad = degree * PI / 180.0;
    float R[2][2] = { {cos(rad), sin(rad)}, {-sin(rad), cos(rad)} };
    float sx, sy;
    int pos_x, pos_y;
    float p, q;

    height = input.rows;
    width = input.cols;

    Mat result(height, width, CV_8UC1);

    for (y = 0; y < height; y++) {
        for (x = 0; x < width; x++) {
            sx = R[0][0] * (x-width/2) + R[0][1] * (y-height/2);
            sy = R[1][0] * (x-width/2) + R[1][1] * (y-height/2);
            sx = sx + width / 2;
            sy = sy + height / 2;
            if (op == 1) {
                pos_x = (int)(sx + 0.5);
                pos_y = (int)(sy + 0.5);
                if (pos_x < 0 || pos_x > width - 1 || pos_y < 0 || pos_y > height - 1) {}
                else {
                    result.at<uchar>(y, x) = input.at<uchar>(pos_y, pos_x);
                }
            }
            else if (op == 2) {
                pos_x = (int)sx;
                pos_y = (int)sy;
                p = sx-pos_x;
                q = sy-pos_y;
                if (pos_x < 0 || pos_x > width - 1 || pos_y < 0 || pos_y > height - 1) {}
                else {
                    result.at<uchar>(y, x) = (1 - p) * (1 - q) * (input.at<uchar>(pos_y, pos_x))
                        + p * (1 - q) * (input.at<uchar>(pos_y, pos_x + 1))
                        + (1 - p) * q * (input.at<uchar>(pos_y + 1, pos_x))
                        + p * q * (input.at<uchar>(pos_y + 1, pos_x + 1));
                }
            }
```



```
            else {}
        }
    }
    if (op == 1) {
        imwrite("nn-rotate.jpg", result);
    }
    else {
        imwrite("bi-rotate.jpg", result);
    }
}

void main()
{
    float scale, degree;
    int option;
    Mat imgColor = imread("test2.jpg", CV_LOAD_IMAGE_GRAYSCALE);
    printf("Input your degree : ");
    scanf("%f", &degree);
    //printf("1 : NN, 2 : Average, 3 : Bi-Linear₩n");
    printf("1 : NN, 2 : Bi-Linear₩n");
    printf("Input your option : ");
    scanf("%d", &option);

    //imageResize(imgColor, scale, option);
    imageRotate(imgColor, degree, option);
}
```

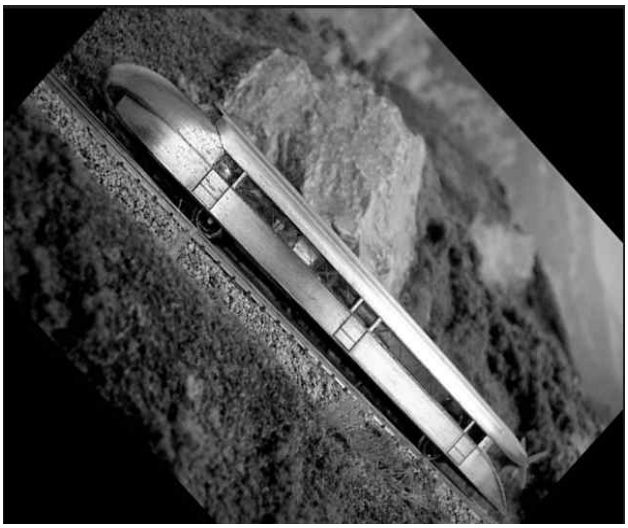
2) 결과 사진

* 회전 각도 : 45%

< 원본 이미지 >

(1) Near-Neighbor rotate

< 전체 이미지>



<픽셀 확대>

(2) Bi-Linear rotate

< 전체 이미지>

<픽셀 확대>

(3) 결과 분석

먼저, 해당 코드는 rotate matrix의 역함수를 표현하는 <float R[2][2] = { {cos(rad), sin(rad)}, {-sin(rad), cos(rad)} };>에 의해 이미지의 중심을 기준으로 오른쪽으로 회전하는 코드이다. 이때, rotate matrix를 지금의 역행렬으로 하고 재차 역행렬을 구해보면 <float R[2][2] = { {cos(rad), -sin(rad)}, {sin(rad), cos(rad)} };>이 된다. 코드에 적용하면 이미지의 중심을 기존으로 왼쪽으로 회전하는 코드가 된다.

NN과 Bi-linear를 분석해보면 Image Resizing과 동일하게 픽셀간의 부드러움에 차이가 있다. 그 이유에 대해서는 Image Resizing에서 분석한 이유와 동일하다.