REPORT

Histogram of oriented gradients

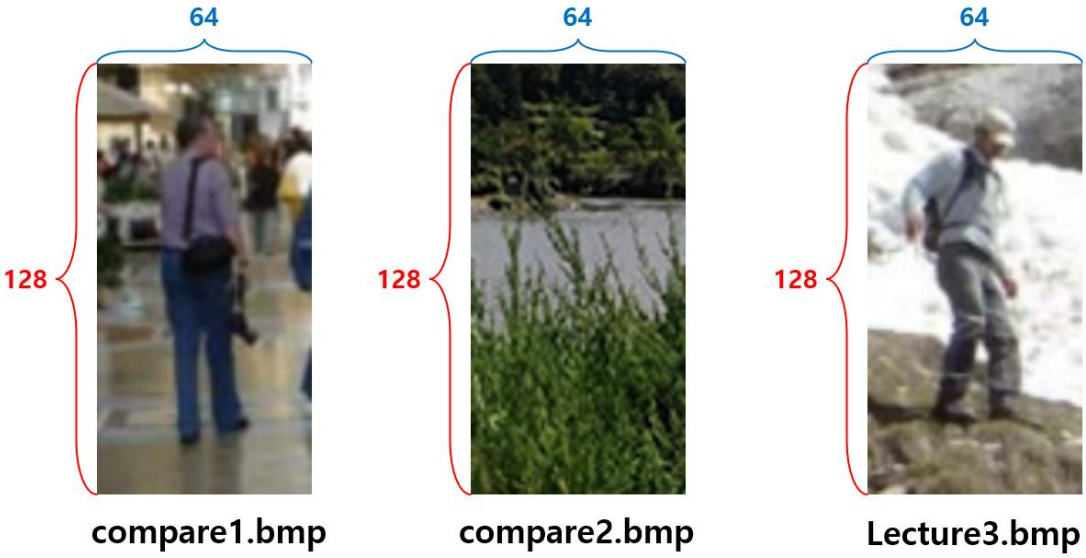
수강과목	전기전자심화설계및소프트웨어실습(2014)
담당교수	김원준
학 과	전기전자공학부
학 번	201810909
이 름	이재현
제출일자	2022. 09. 23(금)

1. Block based Histogram of oriented gradients Algorithm

이미지의 feature 은 대표적으로 conner, edge 등이 있다. image 에서 edge 정보를 extraction 하기 위해 edge detection mask 와 2-D convolution 을 진행한다. edge detection mask 는 x 축, y 축에 대해서 존재하므로 convolution 의 결과는 vector (fx, fy) 로 나타난다. 이때 Magnitude 는 $\sqrt{f_x^2 + f_y^2}$ 로 정의하며, Phase(direction) 은 $\tan^{-1}\frac{fy}{fx}$ 로 구할 수 있다. Edge map 을 얻기 위해 convolution 연산을 수행할 때에는 경계 부분의 indexing 에 주의하여야 한다. image 범위를 벗어나는 픽셀의 경우 ignore 하거나 zero padding, mirroring 등의 방법으로 대처할 수 있다. HOG algorithm 은 histogram 에 direction(degree) 별로 magnitude 를 acculmulation 하여 direction 별 magnitude 분포를 저장하고 이러한 local gradient 를 Image feature 로 활용한다.

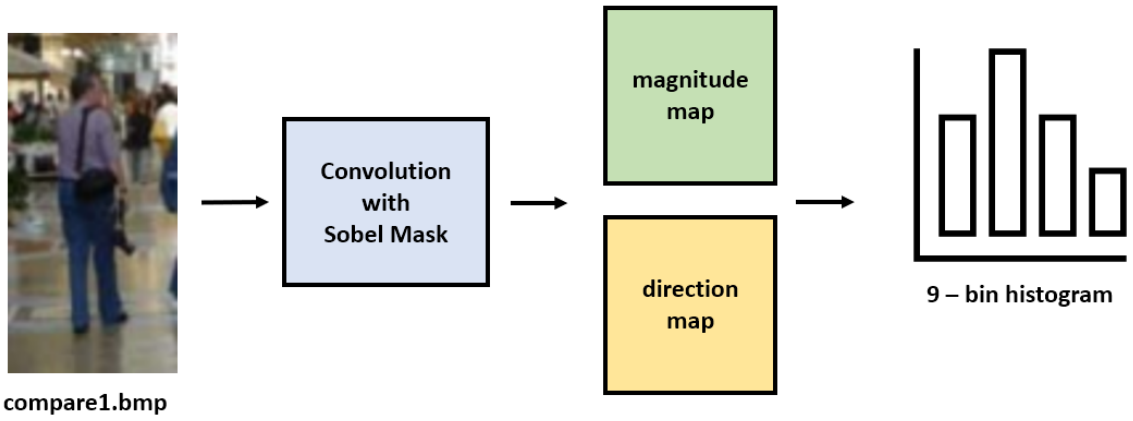
다만 gradient 값은 밝기 변화에 민감하기 때문에 normalization 을 해주는 것이 좋다. normalization 방법론은 다양하게 존재하지만 가장 많이 사용하는 L-2 normalization 방법을 사용하여 구현하도록 한다. 본 실습에서는 Block - based edge oridentation histograms Algorithm 을 구현하여 (Block size = 16 , slicing interval 8, 9 - bin histogram) 서로 다른 두 이미지의 feature 을 extraction 해서 유사도(subtraction)를 비교해본다. histogram 은 하나의 Probability Density Function 으로 볼 수 있기 때문에 두 Histogram 간 유사도를 측정하기 위해 단순 subtraction 보다 KL divergence 를 사용하는 것이 옳은 방법이지만, 본 실습에서는 Block based HOG Algorithm 의 구현이 main topic 이므로 구현의 용이성을 위해 단순히 Euclidean distance 으로 유사도를 계산하기로 한다.

아래의 compare1.bmp, compare2.bmp, Lecture3.bmp 에 대해서 Block - based HOG Alogirthm 을 수행하여 얻은 histogram 을 이용하여 compare1.bmp 와 Lecture3.bmp 의 유사도 그리고 compare2.bmp 와 Lecture3.bmp 와의 유사도를 Euclidean distance 를 이용하여 측정한다. 동일한 보행자 이미지인 compare1.bmp 와 Lecture3.bmp 가 더 유사도가 높을 것임을 예상할 수 있다.

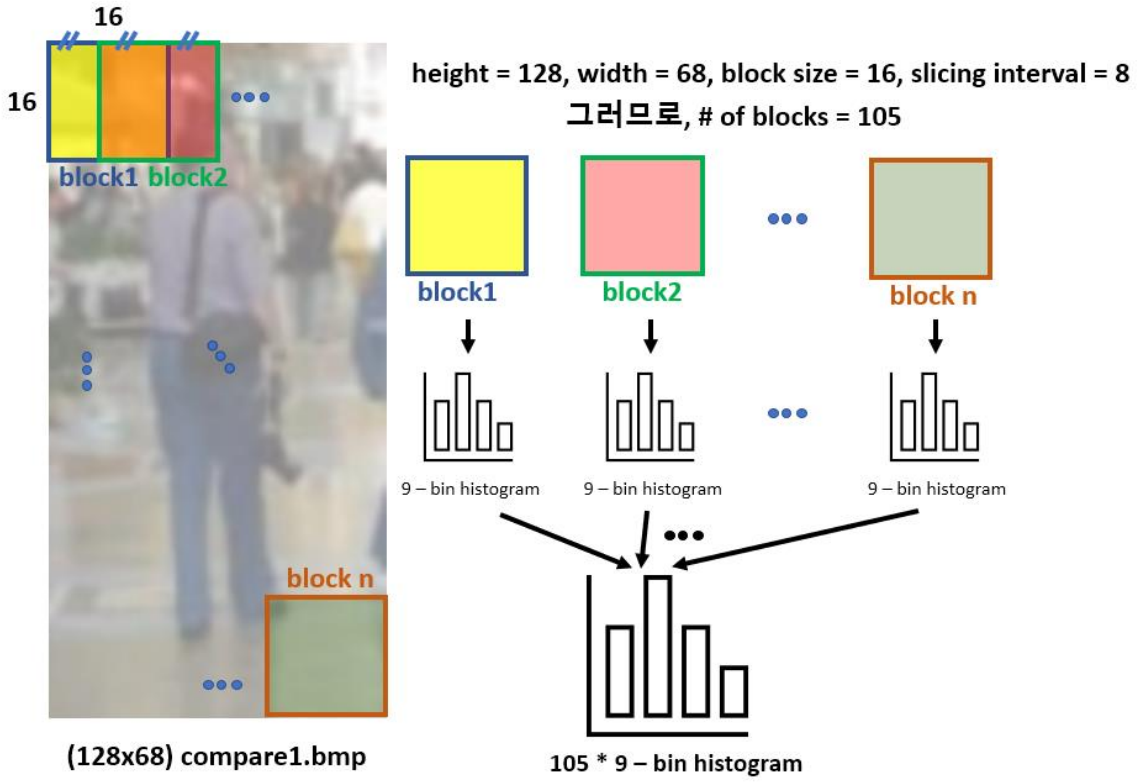


2. Implementation

하나의 block 에서 수행하는 작업의 대략적인 흐름은 아래와 같다.



각각의 이미지에 대해서 Sobel Mask 와의 Convolution 을 통해서 magnitude 와 direction 을 계산한다. 이때 direction 을 degree 단위로 변환해주는 것에 유의한다. 그리고 direction 에 따라서 9 – bin histogram 배열에 magnitude 값을 누적한다.



주어진 이미지의 크기는 128x68 이고 block size 를 16, slicing interval 을 8 으로 설정하였으므로 block 의 개수는 총 105 개가 될 것이다.각각의 block 당 9 – bin histogram 을 하나 씩 계산할 수 있고 이를 appending 하여 105*9 – bin histogram 을 구할 수 있다. 또한 gradient 는 조명과 같은 빛 변화에 민감하므로 L2 – Normalization 을 후처리로 진행한다.

```
float* mag = (float*)calloc(height * width, sizeof(float));
float* dir_arr = (float*)calloc(height * width, sizeof(float));
float* histogram = (float*)calloc(9*(height/(BLK/2) - 1) * (width/(BLK/2) - 1), sizeof(float));
int mask_x[9] = { -1, -2, -1, 0, 0, 0, 1, 2, 1 }; // sobel mask
int mask_y[9] = { -1,0,1,-2,0,2,-1,0,1 };


float min = 1000000, max = -1;

Mat result(height, width, CV_8UC1);

for (y = 0; y < height; y++) { // calculate magnitude and direction
    for (x = 0; x < width; x++) {
        // cur
        conv_x = 0;
        conv_y = 0;

        for (yy = y - 1; yy <= y + 1; yy++) {
            for (xx = x - 1; xx <= x + 1; xx++) { // calc conv_x, conv_y
                if (yy >= 0 && yy < height && xx >= 0 && xx < width) {
                    // indexing 에 주의!
                    conv_x += input.at<uchar>(yy, xx) * mask_x[(yy - (y - 1)) * 3 + (xx - (x - 1))];
                    conv_y += input.at<uchar>(yy, xx) * mask_y[(yy - (y - 1)) * 3 + (xx - (x - 1))];
                }
            }
        }
        conv_x /= 9.0;
        conv_y /= 9.0; // scaling
        mag[y * width + x] = sqrt(conv_x * conv_x + conv_y * conv_y); // calc magninute
        dir = atan2(conv_y, conv_x) * 180.0 / PI; // calc direction ( radian to degree )
        if (dir < 0) dir += 180.0;
        dir_arr[y * width + x] = dir;

        //histogram[(int)(dir / 20)] += mag[y * width + x];
        if (max < mag[y * width + x]) max = mag[y * width + x];
        if (min > mag[y * width + x]) min = mag[y * width + x];
    }
}
```

위와 같이 magnitude 값과 direction 을 계산하여 dir_arr 와 mag 배열에 저장한다. 위에서 언급했듯이 atan2 함수는 radian 값을 반환하므로 이를 degree 로 변환하는 것을 유의한다.

```
int idx = 0;
// block-based hog algorithm
float* temp_histogram = (float*)calloc(9, sizeof(float));
for (int i = 0; i <= input.cols - BLK; i += BLK/2) { // block size = 16 , and slicing interval = block_size/2
    for (int j = 0; j <= input.rows - BLK; j += BLK/2) {
        fill(temp_histogram, temp_histogram+9,  0); // set all zero
        double dir_val = 0.0;

        for (int m = 0; m < BLK; m++) { // calculate histogram at each block
            for (int n = 0; n < BLK; n++) {
                dir_val = dir_arr[(i + m) * width + (j + n)];
                temp_histogram[(int)dir_val / 20] += mag[(i + m) * width + (j + n)];
            }
        }
        for (int i = 0; i < 9; i++) {
            histogram[idx++] = temp_histogram[i];
        }
    }
}
```

이어서 각각의 block 에 대해서 temp_histogram(9 – bin) 에 magnitude 값을 저장한다. 9 – bin 이므로 temp_histogram[(int)dir_val/20] 과 같이 해서 direction 값에 따라 0~8 index 로 magnitude 값들이 누적될 수 있도록 한다. 하나의 block 에 대해서 9 – bin

histogram 을 얻어낸 뒤에는 histogram 에 appending 하는 것을 반복하여 105*9 – bin histogram 을 최종적으로 얻어낸다.

```
// L-2 normalization
float normalization_sum = 0.0;
for (int i = 0; i < 9 * (height / (BLK / 2) - 1) * (width / (BLK / 2) - 1); i++) { normalization_sum += histogram[i] * histogram[i]; }
normalization_sum = sqrt(normalization_sum);
for (int i = 0; i < 9 * (height / (BLK / 2) - 1) * (width / (BLK / 2) - 1); i++) histogram[i] /= normalization_sum;

// for visualization
for (y = 0; y < height; y++) {
    for (x = 0; x < width; x++) {
        result.at<uchar>(y, x) = 255 - 255 * (mag[y * width + x] - min) / (max - min); // scaling and negative
    }
}
return histogram; // return histogram
```

이어서 빛 변화 등에 민감한 gradient 값을 사용했기 때문에, 보다 robust 할 수 있도록 L-2 normalization 을 후처리로 진행한 다음 해당 histogram 을 반환할 수 있도록 하였다.

```
int main(int ac, char** av) {
    int height, width;
    float* histo_comp1 = nullptr;
    float* histo_comp2 = nullptr;
    float* histo_lec3 = nullptr;
    float score_comp1_and_lec3 = 0.0;
    float score_comp2_and_lec3 = 0.0;
    int comp1_histo_length, comp2_histo_length, lec3_histo_length;

    Mat comp1 = imread("images/Lecture3/compare1.bmp", CV_LOAD_IMAGE_GRAYSCALE);
    Mat comp2 = imread("images/Lecture3/compare2.bmp", CV_LOAD_IMAGE_GRAYSCALE);
    Mat lecture3 = imread("images/Lecture3/lecture3.bmp", CV_LOAD_IMAGE_GRAYSCALE);

    histo_comp1 = get_hog_histogram(comp1);
    histo_comp2 = get_hog_histogram(comp2);
    histo_lec3 = get_hog_histogram(lecture3);

    comp1_histo_length = 9 * (comp1.rows / (BLK / 2) - 1) * (comp1.cols / (BLK / 2) - 1);
    comp2_histo_length = 9 * (comp2.rows / (BLK / 2) - 1) * (comp2.cols / (BLK / 2) - 1);
    lec3_histo_length = 9 * (lecture3.rows / (BLK / 2) - 1) * (lecture3.cols / (BLK / 2) - 1);

    score_comp1_and_lec3 = get_similarity(histo_comp1, histo_lec3, comp1_histo_length);
    score_comp2_and_lec3 = get_similarity(histo_comp2, histo_lec3, comp2_histo_length);

    cout << "Euclidean distance compare1 between lecture3 : " << score_comp1_and_lec3 << '\n';
    cout << "Euclidean distance compare2 between lecture3 : " << score_comp2_and_lec3 << '\n';

    waitKey(0);
    free(histo_comp1);
    free(histo_comp2);
    free(histo_lec3);
    return 0;
}
```

main 함수에서는 위에서 구현한 get_hog_histogram 함수를 이용해서 comp1, comp2, lecture3 에 대한 hog histogram 을 얻는다. 그리고 get_similarity 함수를 이용하여 compare1.bmp 와 lecture3.bmp 간의 Euclidean distance 그리고 compare2.bmp 와 lecture3.bmp 간의 Eculidean distance 를 얻어 화면에 출력하도록 하였다.

```
float get_similarity(float* obj1, float* obj2, int size) {
    float score = 0.0;

    for (int i = 0; i < size; i++) {
        score += (abs(obj1[i] - obj2[i])) * (abs(obj1[i] - obj2[i]));
    }

    score = sqrt(score);
    // use Euclidean distance
    // 더 작을수록 유사도가 높은 것
    return score;
}
```

```
Microsoft Visual Studio 디버그 콘솔
Euclidean distance compare1 between lecture3 : 0.732983
Euclidean distance compare2 between lecture3 : 0.785572
```

출력값을 확인해 보았을 때, lecture3.bmp 와의 Euclidean Distacne 는 compare2.bmp 가 compare1.bmp 보다 크기 때문에 유사도가 적음을 확인할 수 있었다. 실제로 compare1.bmp 와 lecture3.bmp 는 둘다 동일한 보행자 이미지이고 compare2.bmp 는 보행자 이미지가 아니기 때문에 위와 같이 compare1.bmp 와 lecture3.bmp 간의 유사도가 compare2.bmp 보다 높은 결과가 나타난 것으로 해석하였다.

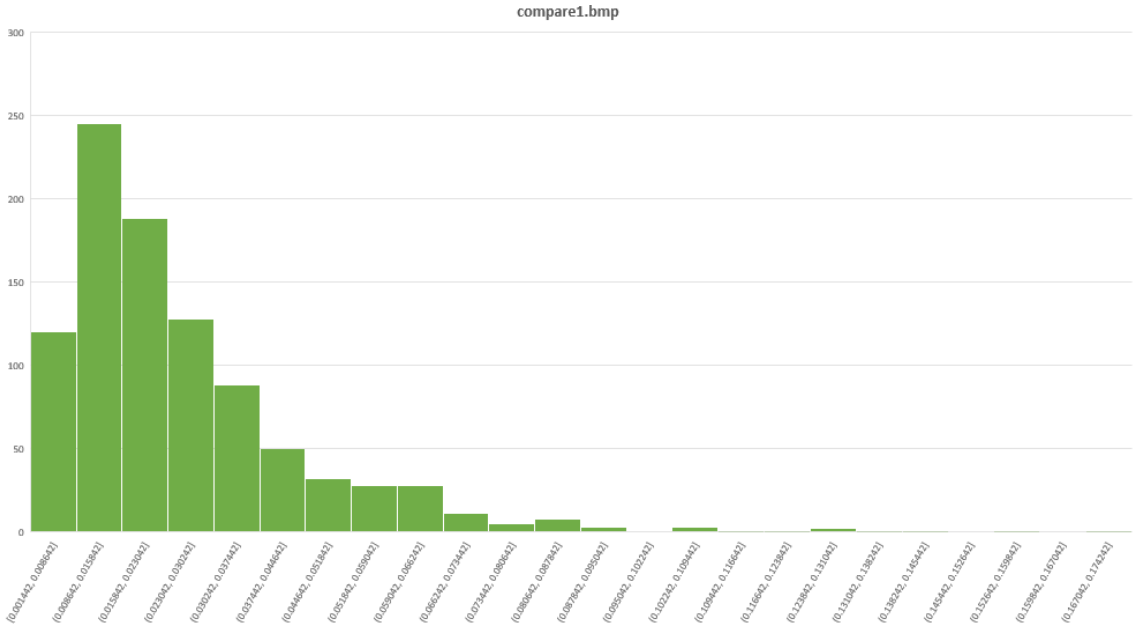
추가적으로 histogram 의 분포를 graphical 하게 확인하기 위해 file writing 관련 코드를 추가하여 compare1.bmp, compare2.bmp, lecture3.bmp 의 histogram 값을 csv 파일로 저장하여, excel 에서 histogram 을 확인하였다.

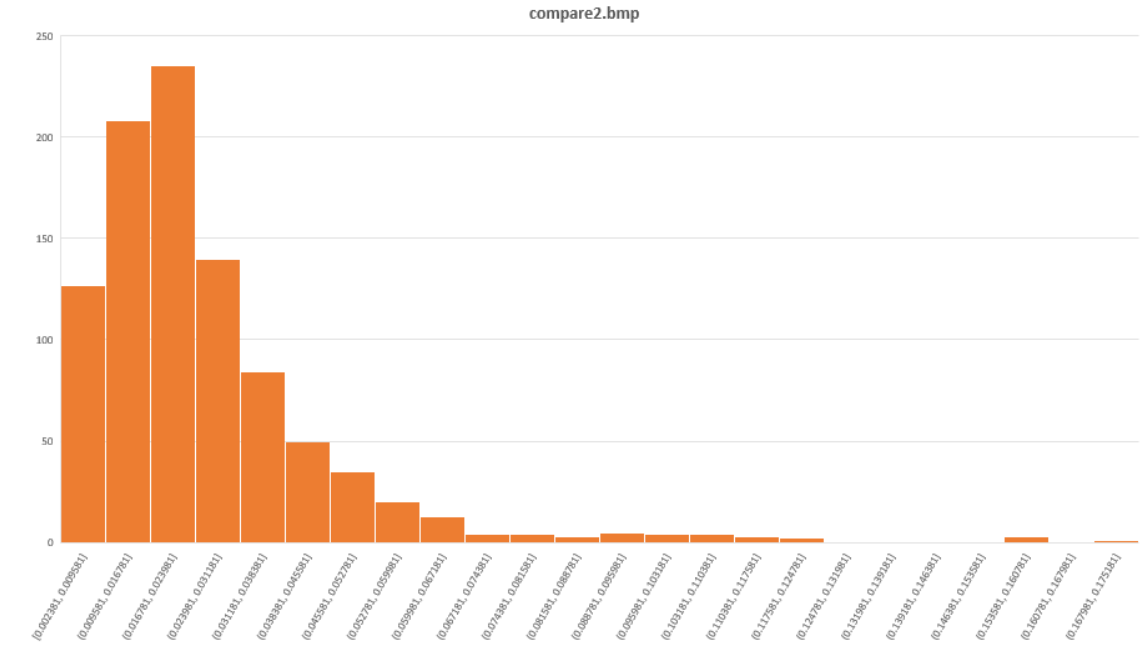
```
float* get_hog_histogram(Mat input, const char* filename) {
    int x, y, xx, yy;
    int height = input.rows;
    int width = input.cols;
    float conv_x, conv_y, dir;
    ⋮
    fp = fopen(filename, "wt");
    for (int i = 0; i < 9 * (height / (BLK / 2) - 1) * (width / (BLK / 2) - 1); i++) {
        fprintf(fp, "%f\n", histogram[i]);
    }
    fclose(fp);
    ⋮
}
```

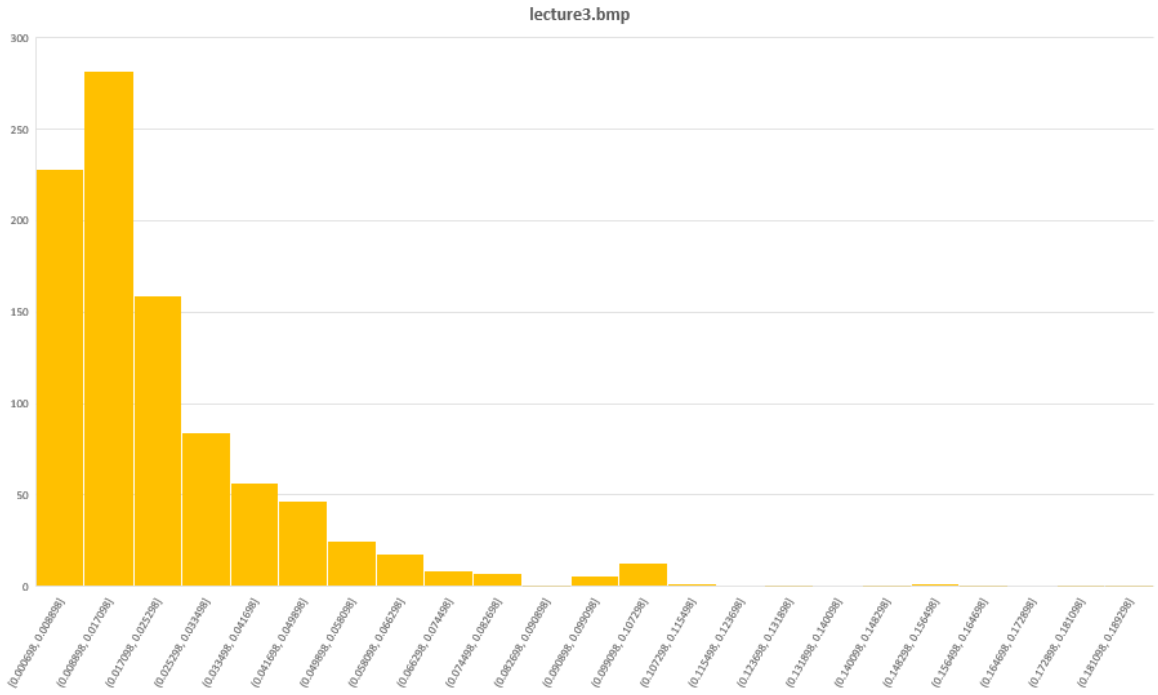
```
int main() {
    ⋮
    histo_comp1 = get_hog_histogram(comp1, "Assign2_hog/compare1.csv");
    histo_comp2 = get_hog_histogram(comp2, "Assign2_hog/compare2.csv");
    histo_lec3 = get_hog_histogram(lecture3, "Assign2_hog/histo_lec3.csv");
    ⋮
}
```

compare1.csv	2022-09-23 오후 4:44	Microsoft Excel ...
compare2.csv	2022-09-23 오후 4:44	Microsoft Excel ...
histo_lec3.csv	2022-09-23 오후 4:44	Microsoft Excel ...

105*9 bin - Histogram 값을 excel 파일에 저장







3. Conclusion

Assignment 02 실습에서는 Image feature extraction 과 관련하여 gradient 성분을 토대로 feature 을 뽑아내는 block based – HOG algorithm 을 구현하여 compare1.bmp, compare2.bmp 와 lecture3.bmp 파일의 Euclidean distance 를 계산하는 프로그램을 구현하였다. 각각의 이미지의 block 마다, gradient 의 magnitude 를 direction 에 따라 9 – bin histogram 에 축적한 뒤에 이를 이어 붙인 다음 빛 변화에 robust 하도록 L2 – Normalization 을 수행한 105*9 bin – histogram 을 최종적으로 계산하여 csv 파일로 저장하였다. 그리고 이 histogram 을 이용해서 compare1.bmp, compare2.bmp 와 lecture3.bmp 파일의 Euclidean distance 를 계산하였고 실제로 동일한 보행자 이미지인

compare1.bmp 와 lecture3.bmp 의 Euclidean distance 가 적게(유사도가 높게) 나타나는 것을 확인하였다. 이와 같이 image 의 gradient(혹은 edge map)을 image 의 feature 으로 활용하여 유사도를 측정하거나 분류 등의 작업을 수행할 수 있음을 확인하였다.

4. Full Source code

```
#define _CRT_SECURE_NO_WARNINGS

// Assignment 02 전기전자심화설계및소프트웨어실습
// 2022. 09. 23. 전기전자공학부 201810909 이재현

#include <opencv2/imgproc.hpp>
#include <opencv2/highgui.hpp>

#include <iostream>
#include <stdio.h>

#define PI 3.141592
#define BLK 16 // Block size

using namespace cv;
using namespace std;

FILE* fp;

float* get_hog_histogram(Mat input, const char* filename) {
	int x, y, xx, yy;
	int height = input.rows;
	int width = input.cols;
	float conv_x, conv_y, dir;

	float* mag = (float*)calloc(height * width, sizeof(float));
	float* dir_arr = (float*)calloc(height * width, sizeof(float));
	float* histogram = (float*)calloc(9*(height/(BLK/2) - 1) * (width/(BLK/2) - 1), sizeof(float));
	int mask_x[9] = { -1, -2, -1, 0, 0, 0, 1, 2, 1 }; // sobel mask
	int mask_y[9] = { -1,0,1,-2,0,2,-1,0,1 };


	float min = 1000000, max = -1;

	Mat result(height, width, CV_8UC1);

	for (y = 0; y < height; y++) { // calculate magnitude and direction
		for (x = 0; x < width; x++) {
			// cur
			conv_x = 0;
			conv_y = 0;

			for (yy = y - 1; yy <= y + 1; yy++) {
				for (xx = x - 1; xx <= x + 1; xx++) { // calc conv_x, conv_y
					if (yy >= 0 && yy < height && xx >= 0 && xx < width) {
						// indexing 에 주의!
						conv_x += input.at<uchar>(yy, xx) * mask_x[(yy - (y - 1))
* 3 + (xx - (x - 1))];
```

```
                                                conv_y += input.at<uchar>(yy, xx) * mask_y[(yy - (y - 1))
* 3 + (xx - (x - 1))];
                                        }
                                }
                        }
                        conv_x /= 9.0;
                        conv_y /= 9.0; // scaling
                        mag[y * width + x] = sqrt(conv_x * conv_x + conv_y * conv_y); // calc magninute
                        dir = atan2(conv_y, conv_x) * 180.0 / PI; // calc direction ( radian to degree )
                        if (dir < 0) dir += 180.0;
                        dir_arr[y * width + x] = dir;

                        //histogram[(int)(dir / 20)] += mag[y * width + x];
                        if (max < mag[y * width + x]) max = mag[y * width + x];
                        if (min > mag[y * width + x]) min = mag[y * width + x];
                }
        }

        int idx = 0;
        // block-based hog algorithm
        float* temp_histogram = (float*)calloc(9, sizeof(float));
        for (int i = 0; i <= input.cols - BLK; i += BLK/2) { // block size = 16 , and slicing interval =
block_size/2
                for (int j = 0; j <= input.rows - BLK; j += BLK/2) {
                        fill(temp_histogram, temp_histogram+9,  0); // set all zero
                        double dir_val = 0.0;

                        for (int m = 0; m < BLK; m++) { // calculate histogram at each block
                                for (int n = 0; n < BLK; n++) {
                                        dir_val = dir_arr[(i + m) * width + (j + n)];
                                        temp_histogram[(int)dir_val / 20] += mag[(i + m) * width + (j +
n)];
                                }
                        }
                        for (int i = 0; i < 9; i++) {
                                histogram[idx++] = temp_histogram[i];
                        }
                }
        }

        // L-2 normalization
        float normalization_sum = 0.0;
        for (int i = 0; i < 9 * (height / (BLK / 2) - 1) * (width / (BLK / 2) - 1); i++)
{ normalization_sum += histogram[i] * histogram[i]; }
        normalization_sum = sqrt(normalization_sum);
        for (int i = 0; i < 9 * (height / (BLK / 2) - 1) * (width / (BLK / 2) - 1); i++) histogram[i] /=
normalization_sum;
        // for visualization
        for (y = 0; y < height; y++) {
                for (x = 0; x < width; x++) {
                        result.at<uchar>(y, x) = 255 - 255 * (mag[y * width + x] - min) / (max - min); //
scaling and negative
                }
        }
```

```
        fp = fopen(filename, "wt");
        for (int i = 0; i < 9 * (height / (BLK / 2) - 1) * (width / (BLK / 2) - 1); i++) {
                fprintf(fp, "%f₩n", histogram[i]);
        }
        fclose(fp);

        return histogram; // return histogram
}

float get_similarity(float* obj1, float* obj2, int size) {
        float score = 0.0;

        for (int i = 0; i < size; i++) {
                score += (abs(obj1[i] - obj2[i])) * (abs(obj1[i] - obj2[i]));
        }

        score = sqrt(score);
        // use Euclidean distance
        // 더 작을수록 유사도가 높은 것
        return score;
}

int main(int ac, char** av) {
        int height, width;
        float* histo_comp1 = nullptr;
        float* histo_comp2 = nullptr;
        float* histo_lec3 = nullptr;
        float score_comp1_and_lec3 = 0.0;
        float score_comp2_and_lec3 = 0.0;
        int comp1_histo_length, comp2_histo_length, lec3_histo_length;

        Mat comp1 = imread("images/Lecture3/compare1.bmp", CV_LOAD_IMAGE_GRAYSCALE);
        Mat comp2 = imread("images/Lecture3/compare2.bmp", CV_LOAD_IMAGE_GRAYSCALE);
        Mat lecture3 = imread("images/Lecture3/lecture3.bmp", CV_LOAD_IMAGE_GRAYSCALE);

        histo_comp1 = get_hog_histogram(comp1, "Assign2_hog/compare1.csv");
        histo_comp2 = get_hog_histogram(comp2, "Assign2_hog/compare2.csv");
        histo_lec3 = get_hog_histogram(lecture3, "Assign2_hog/histo_lec3.csv");

        comp1_histo_length = 9 * (comp1.rows / (BLK / 2) - 1) * (comp1.cols / (BLK / 2) - 1);
        comp2_histo_length = 9 * (comp2.rows / (BLK / 2) - 1) * (comp2.cols / (BLK / 2) - 1);
        lec3_histo_length = 9 * (lecture3.rows / (BLK / 2) - 1) * (lecture3.cols / (BLK / 2) - 1);

        score_comp1_and_lec3 = get_similarity(histo_comp1, histo_lec3, comp1_histo_length);
        score_comp2_and_lec3 = get_similarity(histo_comp2, histo_lec3, comp2_histo_length);

        cout << "Euclidean distance compare1 between lecture3 : " << score_comp1_and_lec3 << '₩n';
        cout << "Euclidean distance compare2 between lecture3 : " << score_comp2_and_lec3 << '₩n';

        waitKey(0);
        free(histo_comp1);
        free(histo_comp2);
        free(histo_lec3);
        return 0;
}
```