전기전자심화설계 및 소프트웨어실습

〈Assignment 4〉

담당교수 : 김원준

이름 : 201810528 고려욱

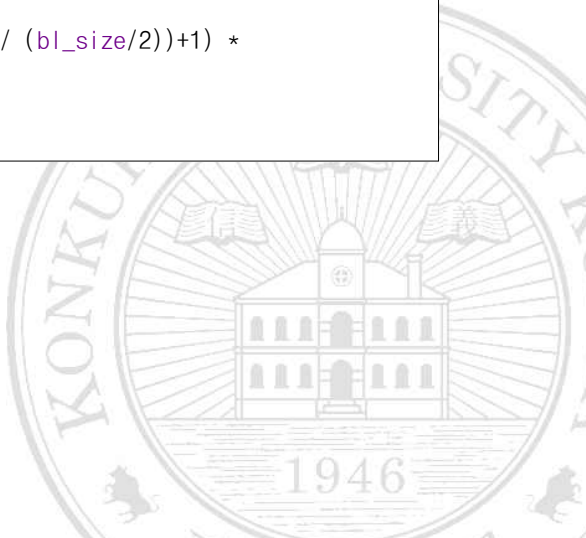
1. Compute Similarity Face Image

1) Source Code

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

#include <stdio.h>
#include <stdlib.h>
#define PI 3.141592
#define bl_size 12
#define quan 9

using namespace cv;

void ref_HOG(Mat input, float* output)
{
	int x, y, xx, yy;
	int height = input.rows;
	int width = input.cols;
	float conv_x, conv_y;
	float dir;
	int deg;
	float* mag = (float*)calloc(height * width, sizeof(float));
	float* o_dir = (float*)calloc(height * width, sizeof(float));
	int mask_y[9] = { -1,-1,-1,0,0,0,1,1,1 };
	int mask_x[9] = { -1,0,1,-1,0,1,-1,0,1 };

	//compute fx and fy
	for (y = 0; y < height; y++) {
	for (x = 0; x < width; x++) {
	conv_x = 0;
	conv_y = 0;

	//inner product
	for (yy = y - 1; yy <= y + 1; yy++) {
	for (xx = x - 1; xx <= x + 1; xx++) {
	if (yy >= 0 && yy < height && xx >= 0 && xx < width) {
	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;

	mag[y * width + x] = sqrt(conv_x * conv_x + conv_y * conv_y);
	dir = atan2(conv_y, conv_x);
	dir = dir * 180.0 / PI;
	if (dir < 0) dir += 180;
	o_dir[y * width + x] = dir;
	}
	}

	int idx = 0;

	float normal_pow;
	for (y = 0; y <= height - bl_size; y += bl_size / 2) {
	for (x = 0; x <= width - bl_size; x += bl_size / 2) {
	for (yy = y; yy < y + bl_size; yy++) {
	for (xx = x; xx < x + bl_size; xx++) {
	deg = o_dir[yy * width + xx] / 20.0;
	output[idx*quan+deg] += mag[yy * width + xx];
	}
	}
	idx++;
```

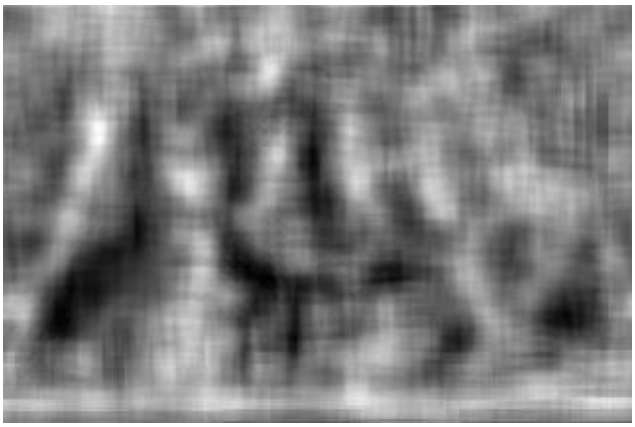
```
		}
		}
		for (x = 0; x < idx; x++) {
		normal_pow = 0.0;
		for (y = 0; y < quan; y++) {
		normal_pow += (output[x*quan+y] * output[x * quan + y]);
		}
		for (y = 0; y < quan; y++) {
		output[x * quan + y] = output[x * quan + y] / (sqrt(normal_pow));
		if (normal_pow == 0.0)
		output[x * quan + y] = 0.0;
		}
		}
}

void tar_HOG(Mat input, float* output, int pos_y, int pos_x, int w_height, int w_width)
{
		int x, y, xx, yy;
		float conv_x, conv_y;
		float dir;
		int deg;
		float* mag = (float*)calloc(w_height * w_width, sizeof(float));
		float* o_dir = (float*)calloc(w_height * w_width, sizeof(float));
		int mask_y[9] = { -1,-1,-1,0,0,0,1,1,1 };
		int mask_x[9] = { -1,0,1,-1,0,1,-1,0,1 };

		//compute fx and fy
		for (y = pos_y; y < pos_y + w_height; y++) {
		for (x = pos_x; x < pos_x + w_width; x++) {
		conv_x = 0;
		conv_y = 0;

		//inner product
		for (yy = y - 1; yy <= y + 1; yy++) {
		for (xx = x - 1; xx <= x + 1; xx++) {
		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;

		mag[(y-pos_y) * w_width + (x-pos_x)] = sqrt(conv_x * conv_x + conv_y * conv_y);
		dir = atan2(conv_y, conv_x);
		dir = dir * 180.0 / PI;
		if (dir < 0) dir += 180;
		o_dir[(y - pos_y) * w_width + (x - pos_x)] = dir;
		}
		}
		int idx = 0;
		float normal_pow;

		for (y = 0; y <= w_height - bl_size; y += bl_size / 2) {
		for (x = 0; x <= w_width - bl_size; x += bl_size / 2) {
		for (yy = y; yy < y + bl_size; yy++) {
		for (xx = x; xx < x + bl_size; xx++) {
		deg = o_dir[yy * w_width + xx] / 20.0;
		output[idx * quan + deg] += mag[yy * w_width + xx];
		}
		}
		idx++;
		}
		}
```

```
        for (x = 0; x < idx; x++) {
        normal_pow = 0.0;
        for (y = 0; y < quan; y++) {
        normal_pow += (output[x * quan + y] * output[x * quan + y]);
        }
        for (y = 0; y < quan; y++) {
        output[x * quan + y] = output[x * quan + y] / (sqrt(normal_pow));
        if (normal_pow == 0.0)
        output[x * quan + y] = 0.0;
        }
        }
}
float cos_sim(float* ref, float* tar)
{
        float dot = 0.0, size_r = 0.0, size_t = 0.0;
        for (int x = 0; x < 25*quan; x++) {
        dot += ref[x] * tar[x];
        size_r += ref[x] * ref[x];
        size_t += tar[x] * tar[x];
        }
        return dot / (sqrt(size_r) * sqrt(size_t));
}

void compute_sim(Mat input, float * refHOG, int w_height, int w_width)
{
        int height = input.rows;
        int width = input.cols;
        int x, y, xx, yy;
        Mat result(height, width, CV_8UC1);
        float * sim = (float *)calloc(height*width, sizeof(float));
        float max = -1.0;
        float min = 10000000.0;
        for (y = 0; y < height; y++) {
        for (x = 0; x < width; x++) {
        float* tarHOG = (float*)calloc(quan * ((((w_width - bl_size) / (bl_size / 2)) + 1) *
        (((w_height - bl_size) / (bl_size / 2)) + 1)), sizeof(float));
        tar_HOG(input, tarHOG, y, x , w_height, w_width);
        sim[y*width+x] = cos_sim(refHOG, tarHOG);
        if (sim[y*width+x] > max) max = sim[y*width+x];
        if (sim[y*width+x] < min) min = sim[y*width+x];
        free(tarHOG);
        }
        }
        for (y = 0; y < height; y++) {
        for (x = 0; x < width; x++) {
        result.at<uchar>(y, x) = 255 * (sim[y * width + x] - min) / (max - min);
        }
        }
        imwrite("sim.bmp", result);
}

void main()
{
        Mat imgref = imread("face_ref.bmp", CV_LOAD_IMAGE_GRAYSCALE);
        Mat imgtar = imread("face_tar.bmp", CV_LOAD_IMAGE_GRAYSCALE);
        int height = imgref.rows;
        int width = imgref.cols;
        float* refHOG = (float*)calloc(quan * (((width - bl_size) / (bl_size/2))+1) *
        (((height - bl_size) / (bl_size/2))+1), sizeof(float));
        ref_HOG(imgref, refHOG);
        compute_sim(imgtar, refHOG, height, width);
}
```



2) 결과 사진

(1) similarity 나타낸 사진



(2) face_tar.bmp

두 사진을 비교해보면 similarity를 보여주는 사진에서 가장 밝은 하얀 점이 face_tar에서 사람들의 얼굴에 위치해 있음을 볼 수 있다. 하지만, 결과에 몇 가지 문제점들을 볼 수 있다.

먼저, face_tar에 대한 HOG를 처리할 때의 문제이다. Image ref와 동일한 크기로 tar의 사진을 잘라 HOG를 분석하기 위해 tar 사진을 1pixel씩 이동하며 동일한 36*36의 크기로 잘라 HOG를 취했다. 코드에서 먼저, 자른 사진에서 gradient를 구하는 과정에서 자른 크기 외의 pixel과 mask가 filtering 되는 것을 막기 위해서 boundary condition을 주고 계산을 진행하였다. 하지만, 그 결과를 보았을 때, 전혀 관계없음을 보여주는 검정색에 해당하는 similarity image의 pixel들이 상당수 적어졌기 때문에, 자른 크기 바깥의 pixel또한 filtering 계산에 들어가게 계산을 하였다. 그 결과 가장자리의 pixel의 계산이 달라졌을 수 있어 가장자리에서 조금은 밝은 결과들이 도출되었다고 분석된다.

두번째로, tar image에서 5사람의 가운데 위치한 원판이 조금은 관련성이 있다고 보인다. 조금은 밝은 하얀색으로 원판의 similarity가 관측되었다. 해당 위치의 similarity를 정확한 수치로 출력해 보지 않아 실제 사람의 얼굴과 원판의 유사도 차이가 얼마나 나는지는 확인하지 못했다. 또한, 아직 결과상에서 해당 위치에 사각형을 그려 해당 pixel 주변이 사람의 얼굴로 판정되게 구현하지는 않았다. 원판이 높은 관련성을 보였기 때문에, 사각형을 그리는 threshold값을 높은 점수로 주어 사각형을 그려야 실제 사람 5명의 얼굴 주위에만 사각형이 찍힐 수 있을 것으로 예상된다.

마지막으로, 수행속도에 있다. 상당히 많은 반복문을 수행하기 때문에 IDE환경에서도 파일 출력까지 완성하는데 많은 시간이 걸렸다. 가장 많은 시간을 소비하는 곳은 tar이미지를 1pixel씩 이동하여 유사도를 측정하기 때문에 한 pixel의 grdient가 반복되서 계산되는 점에 있다. 이는, 코드를 더 발전시켜 gradient는 한번 구해주고 pixel을 이동하며 HOG계산시에 저장된 값만 불러오도록하면 더 줄어들 수 있을 것이라 생각한다. 하지만, 1pixel씩 이동하며 모든 pixel의 유사도를 측정하는 것 외의 알고리즘을 사용하여야 본질적으로 수행속도를 줄일 수 있을 것이라 생각한다.