HW9 Report

Computer Vision
Section 02
21800181 Kim Jisu

HW 9

- Purpose

The user inputs the key. When the key is "b", background subtraction is performed, and when "g", grabcut is performed. When "f" is input, face detection is performed.

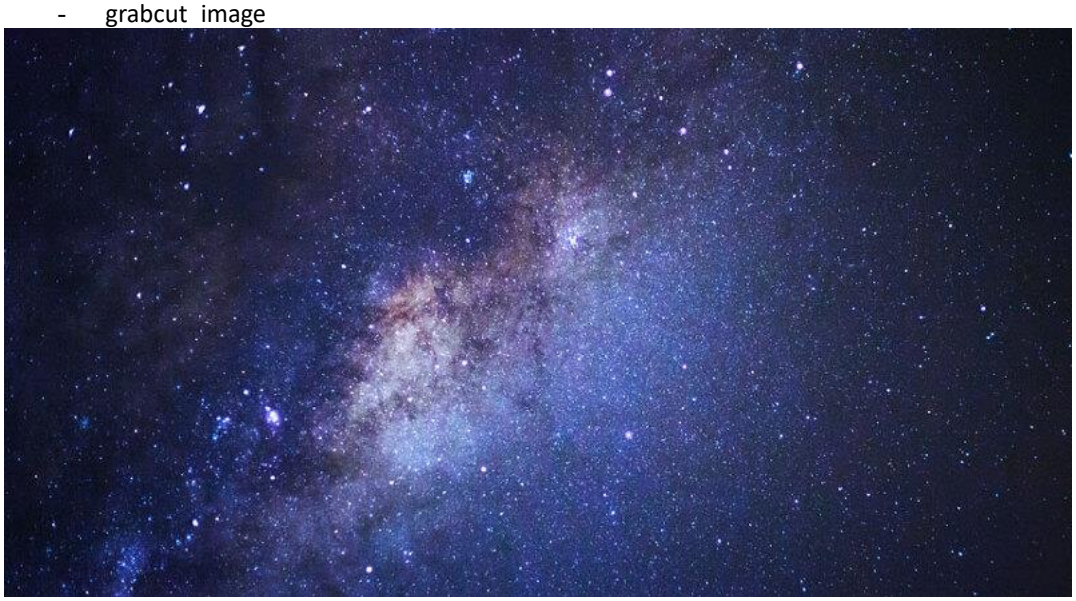
- Principle

void cv::grabCut (InputArray img, InputOutputArray mask, Rect rect, InputOutputArray bgdModel, InputOutputArray fgdModel, int iterCount, int mode = GC_EVAL)

Mat cv::dnn::blobFromImage(InputArray image, double scalefactor = 1.0, const Size & size = Size(), const Scalar & mean = Scalar(), bool swapRB = false, bool crop = false, int ddepth = CV_32F)

Creates 4-dimensional blob from image. Optionally resizes and crops image from center, subtract mean values, scales values by scalefactor, swap Blue and Red channels.

- grabcut image



space.jpg

- Result

https://youtu.be/2cqKrIUdEic

- Code

```
#include "opencv2/opencv.hpp"
#include "opencv2/core.hpp"
#include <iostream>
```

```
#include "opencv2/objdetect.hpp"
#include "opencv2/highgui.hpp"
#include "opencv2/imgproc.hpp"
#include "opencv2/videoio.hpp"
#include "cv.hpp"
#include <opencv2/dnn.hpp>
#include <fstream>
#include <string.h>

using namespace std;
using namespace cv;
using namespace dnn;

Mat faces_detect(Mat image);
Mat universe(Mat image);
Mat grabcut(Mat image,Rect rectangle);
Mat background(Mat image,Mat background_, Mat left, Mat right);
int yolo(Mat frame);

int main(){
	bool face = false;
	bool g_cut = false;
	bool mog = false;
	Mat frame;
	VideoCapture capture;
	if(capture.open("Faces.mp4")==0){
		cout<<"no wuch file!"<<endl;
		return 0;
	}
	Mat original;
	capture >> original;

	Mat background_ = original.clone();
	Mat left;
	Mat right;
	int c = 0;
	while (1) {
		capture >> original;
		c++;
```

```
            if (original.empty()) {
                break;
            }
            if(c == 203) left = original.clone();
            if(c == 319) right = original.clone();
        }
        Mat left_roi_2 = left(Rect(540,300,130,420));
        Mat right_roi_2 = right(Rect(650,273,100,420));

        while(1){
            Mat frame;
            VideoCapture cap;
            if(cap.open("Faces.mp4")==0){
                cout<<"no wuch file!"<<endl;
                return 0;
            }
            while (1) {
                cap >> frame;
                if (frame.empty()) {
                    break;
                }
                if (g_cut){
                    frame = universe(frame);
                }
                if(mog){
                    int count = yolo(frame);
                    frame = background(frame,background_,left_roi_2,right_roi_2);
                    putText(frame, format("human: %d",count ), Point(50, 80), FONT_HERSHEY_SIMPLEX, 1,
Scalar(255, 255, 0), 4);
                }
                if (face){
                    frame = faces_detect(frame);
                }
                imshow("video", frame);
                int key;

                key = waitKey(30);
                if (key == 102){
                    if(face==true) face = false;
                    else face = true;
```

```
                }
                if (key == 98){
                        if(mog==true) mog = false;
                        else mog = true;
                }
                if (key == 103){
                        if(g_cut==true) g_cut = false;
                        else g_cut = true;
                }
            }
        }
}

Mat background(Mat image,Mat back, Mat left, Mat right){
        Mat background = image.clone();
        Mat roi_1 = background(Rect(300,300,950,420));
        Mat roi_2 = back(Rect(300,300,950,420));
        roi_2.copyTo(roi_1);
        Mat left_roi_2 = background(Rect(540,300,130,420));
        Mat right_roi_2 = background(Rect(670,300,100,420));
        left.copyTo(left_roi_2);
        right.copyTo(right_roi_2);
        Mat element = getStructuringElement(MORPH_ELLIPSE, Size(10, 10));
        Mat gray, result, foregroundMask, foregroundImg;
        cvtColor(background, background, CV_BGR2GRAY);
        cvtColor(image, gray, CV_BGR2GRAY);
        absdiff(background, gray, foregroundMask);
        threshold(foregroundMask, foregroundMask, 50, 255, CV_THRESH_BINARY);
        erode(foregroundMask, foregroundMask, element);
        dilate(foregroundMask, foregroundMask, element);
        foregroundMask.copyTo(foregroundImg);
        image.copyTo(foregroundImg, foregroundMask);

        return foregroundImg;
}

int yolo(Mat frame){
        if (frame.channels() == 4) cvtColor(frame, frame, COLOR_BGRA2BGR);

        String modelConfiguration = "deep/yolov2-tiny.cfg";
```

```
        String modelBinary = "deep/yolov2-tiny.weights";

        Net net = readNetFromDarknet(modelConfiguration, modelBinary);

        vector<String> classNamesVec;
        ifstream classNamesFile("deep/coco.names");
        if (classNamesFile.is_open()) {
                string className = "";
                while (std::getline(classNamesFile, className)) classNamesVec.push_back(className);

        }
        Mat inputBlob = blobFromImage(frame, 1 / 255.F, Size(416, 416), Scalar(), true, false);
        net.setInput(inputBlob, "data");                                    //set the network input
        Mat detectionMat = net.forward("detection_out");      //compute output
        int count = 0;
        float confidenceThreshold = 0.24; //by default
        for (int i = 0; i < detectionMat.rows; i++)       {
                const int probability_index = 5;
                const int probability_size = detectionMat.cols - probability_index;
                float *prob_array_ptr = &detectionMat.at<float>(i, probability_index);
                size_t objectClass = max_element(prob_array_ptr, prob_array_ptr + probability_size) - prob_array_ptr;
                float confidence = detectionMat.at<float>(i, (int)objectClass + probability_index);
                if (confidence > confidenceThreshold) {
                        String className = objectClass < classNamesVec.size() ? classNamesVec[objectClass] :
cv::format("unknown(%d)", objectClass);
                        if(strncmp(className.c_str(),"person",5)==0){
                                count++;
                        }
                }
        }
                        return count;
}



Mat faces_detect(Mat image){
        Mat gray_img;
        CascadeClassifier face_classifier;
        face_classifier.load("haarcascade_frontalface_alt.xml");
        cvtColor(image, gray_img, COLOR_BGR2GRAY);
```

```
	vector<Rect> small_faces;
	vector<Rect> medium_faces;
	vector<Rect> big_faces;

	face_classifier.detectMultiScale(
			gray_img,
			small_faces,
			1.1, // increase search scale by 10% each pass
			3,	// merge groups of three detections
			0,	// not used for a new cascade
			Size(30, 30), //min size
			Size(40, 40)
		);
	face_classifier.detectMultiScale(
			gray_img,
			medium_faces,
			1.1, // increase search scale by 10% each pass
			3,	// merge groups of three detections
			0,	// not used for a new cascade
			Size(55, 55), //min size
			Size(58, 58)
		);
	face_classifier.detectMultiScale(
			gray_img,
			big_faces,
			1.1, // increase search scale by 10% each pass
			3,	// merge groups of three detections
			0,	// not used for a new cascade
			Size(83, 83), //min size
			Size(100, 100)
		);
	for (int i = 0; i < medium_faces.size(); i++) {
			Point lb(medium_faces[i].x + medium_faces[i].width, medium_faces[i].y +
medium_faces[i].height);
			Point tr(medium_faces[i].x, medium_faces[i].y);
			rectangle(image, lb, tr, Scalar(255, 0, 255), 3, 4, 0);
		}

	for (int i = 0; i < small_faces.size(); i++) {
			Point lb(small_faces[i].x + small_faces[i].width, small_faces[i].y + small_faces[i].height);
```

```
                Point tr(small_faces[i].x, small_faces[i].y);
                rectangle(image, lb, tr, Scalar(255, 0, 0), 3, 4, 0);
            }
        for (int i = 0; i < big_faces.size(); i++) {
                Point lb(big_faces[i].x + big_faces[i].width, big_faces[i].y + big_faces[i].height);
                Point tr(big_faces[i].x, big_faces[i].y);
                rectangle(image, lb, tr, Scalar(0, 255, 0), 3, 4, 0);
            }
        return image;
}




Mat universe(Mat image){
        Mat Gray;
        Gray = image.clone();
        Gray.setTo(Vec3b(255,255,255));
        Mat space = imread("space.jpg");
        resize(space, space, Size(Gray.cols,Gray.rows));
        Mat gray_img;
        CascadeClassifier face_classifier;
        face_classifier.load("haarcascade_frontalface_alt.xml");
        cvtColor(image, gray_img, COLOR_BGR2GRAY);
        vector<Rect> faces;
        face_classifier.detectMultiScale(
                gray_img,
                faces,
                1.1, // increase search scale by 10% each pass
                3,      // merge groups of three detections
                0,      // not used for a new cascade
                Size(30, 30), //min size
                Size(100, 100)
            );
        Mat temp[3];
        Mat notemp[3];
        for (int i = 0; i < faces.size(); i++) {
                Mat img_1,space_1;
                Point lb(faces[i].x + faces[i].width, faces[i].y + faces[i].height);
                Point tr(faces[i].x, faces[i].y);
```

```
            img_1 = image(Rect(tr.x,tr.y,abs(tr.x-lb.x)+20,abs(tr.y-lb.y)));
            space_1 = space(Rect(tr.x,tr.y,abs(tr.x-lb.x),abs(tr.y-lb.y)));
            img_1.copyTo(space_1);
            temp[i] = grabcut(image,Rect(tr.x,tr.y,abs(tr.x-lb.x),abs(tr.y-lb.y)));
            notemp[i] = ~temp[i];
            }
        Mat temp_re = temp[0].clone();
        for (int i = 1; i < faces.size(); i++) {
            add(temp_re,temp[i],temp_re);
        }
        Mat t = ~temp[0];
        Mat notemp_re = t.clone();
        for (int i = 1; i < faces.size(); i++) {
            Mat t2 = ~temp[i];
            bitwise_and(notemp_re,t2,notemp_re);
        }
        Mat foreground = Mat(image.size(), CV_8UC3, Scalar(255, 255, 255));
        image.copyTo(foreground, temp_re);
        threshold(foreground, foreground, 254, 255, THRESH_TOZERO_INV );
        Mat masked;
        resize(space, space, image.size());
        cvtColor(notemp_re, notemp_re, CV_GRAY2BGR);
        bitwise_and(notemp_re, space,masked);
        add(foreground,masked,masked);
        return masked;
}

Mat grabcut(Mat image,Rect rectangle){
        Mat fn_result[2];
        Mat result, bgModel, fgModel, foreground;
        grabCut(image, result, rectangle, bgModel, fgModel, 10, GC_INIT_WITH_RECT);
        compare(result, GC_PR_FGD, result, CMP_EQ);
        foreground = Mat(image.size(), CV_8UC3, Scalar(255, 255, 255));
        image.copyTo(foreground, result);
        return result;
}
```

l References

https://docs.opencv.org/master/