# HW9

*Date: 2021. 12. 01*
*Student ID: 21600635*
*Name: Bomoon JUNG*

## 1. Homework 9

1) Develop a program that has the following three functions.

- ✓ Input" 'Faces.mp4'
  - It is up to you whether you read it as grayscale or not. (but colorful video recommended)
- ✓ Output: A video window showing the result according to each function
  - Repeat the video playback.
- ✓ All functions need to be implemented in one single program
- ✓ How you implement this program is totally up to you, which means you can use any methods as long as the results satisfy the function requirements.
- ✓ Record a working video showing all the results according to the functions, upload the video on YouTube, and write down the video the video address in your report.

Develop a program that has following functions.

- ✓ Function 1 (7 points)
  - When user presses 'b' key, subtract the background from the original image(so that the only human regions will remain).
  - When user presses 'b' key again, return to the original video.
  - During background subtraction, count the number of people and write the number on the result window.
    - ✧ Use morphological operation (for this case, do not user face detection)
    - ✧ You may use tracking techniques.

Develop a program that has following functions.

- ✓ Function 2 (5 points)
  - When user presses 'f' key, detect all faces and draw a rectangle box on every face region.
    - ✧ Use green color for the nearest face, blue color for the farthest face, and yellow color for the other face.
  - When user presses 'f' key again, remove all boxes.

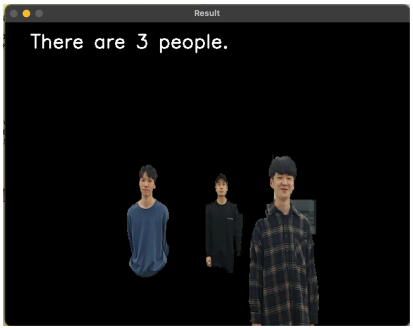
Develop a program that has following functions.

✓ Function 3 (8 points)

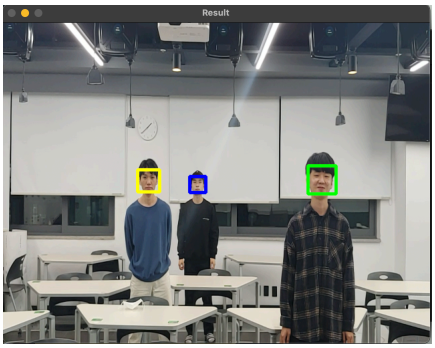
- When user presses ‘g’ key, detect all faces areas (not rectangular area) and fill the rest with a virtual background.
  - ✧ The virtual background image can be arbitrarily.
  - ✧ You may use grabCut() function for segmenting a face area. (refer to the next slide)
- When user presses ‘’ key again, return to the original video.

2) explanation

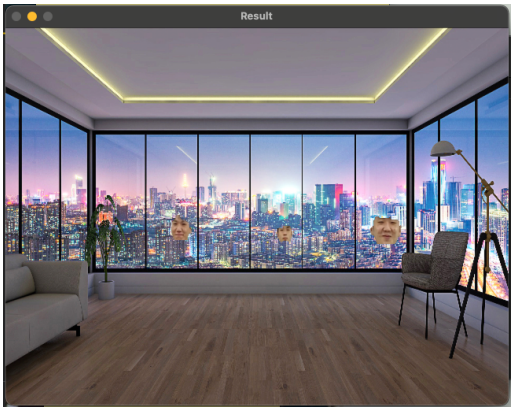
1. main: The 'main' function reads the image, and the output is implemented differently depending on the key pressed by the user. Also, if you press the pressed key again, the original image is output.
2. func1: When the user presses ‘b’, the main function puts one frame of the video as a parameter of ‘func1()’. ‘func1()’ recognizes a person using deep learning technology, and extracts only the human shape through grabCut() of the recognized person’s box. It also prints the number of recognized people.
3. get_foreground: This function is used in ‘func1’, and the foreground can be extracted through the grabCut() function.




4. func2: When the user presses ‘f’, the main function puts one frame of the video as a parameter of ‘func2()’. ‘func2()’ recognizes a face through ‘face detection’ and then changes the color of the box according to the distance. The nearest face is marked with a green box, the farthest face with a blue box, and the others with a yellow box. To implement this, the size of the recognized face area becomes smaller as the distance increases, so the color is changed according to the size.



5. func3: When the user presses 'g', the main function puts one frame of the video as a parameter of 'func3()'. ‘func3()’ detects a face through ‘face detection’, extracts only the face through grabCut(), and puts the extracted face on the virtual background.
6. change_bacground: This function is used for ‘func3()’, and it is a function that puts a face on the virtual background through the grabCut() function.




3) YouTube Link

https://youtu.be/KnGBZNM5B4s

4) Source code

```
// g++ $(pkg-config --cflags --libs opencv) HW9_21600635.cpp -o ./a
#include <opencv2/opencv.hpp>
#include <iostream>
#include <opencv2/dnn.hpp>
#include <fstream>
#include <string>

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


Mat
get_foreground ( Mat src, Mat foreground, Rect rectangle) {

    Mat result, bgModel, fgModel, result2 ;

    grabCut(src, result, rectangle, bgModel, fgModel, 4, GC_INIT_WITH_RECT) ;

    compare(result, GC_PR_FGD, result, CMP_EQ) ;

    // image 에 result(사람만 흰색)인 것을 덮는다. 이때 foreground 는 복사 안 된다.
    src.copyTo(foreground, result) ; // (output, mask)
```

```
    return foreground ;
}


Mat
func1 ( Mat src ) {
    cout << "im func1" << endl ;

    String modelConfiguration = "deep_2/yolov2-tiny.cfg" ;
    String modelBinary = "deep_2/yolov2-tiny.weights" ;
    Net net = readNetFromDarknet(modelConfiguration, modelBinary) ;
    vector<String> classNamesVec ;
    ifstream classNamesFile("deep_2/coco.names") ;

    if ( classNamesFile.is_open() ) {
        string className = "" ;
        while (std::getline(classNamesFile, className))
            classNamesVec.push_back(className) ;
    }


    Mat result = src.clone() ;
    if ( result.channels() == 4 )
        cvtColor(result, result, COLOR_BGRA2BGR) ;

    // Convert Mat to batch of images
    Mat inputBlob = blobFromImage( result, 1 / 255.F, Size( 416, 416 ), Scalar(),
true, false ) ;

    // set the network inputMat detection
    net.setInput( inputBlob, "data" ) ;

    // compute output
    Mat detectionMat = net.forward( "detection_out" ) ;

    // by default
    float confidenceThreshold = 0.24 ;
    int count_person = 0 ;
    Mat foreground = Mat(result.size(), CV_8UC3, Scalar(0, 0, 0)) ;
    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 ;
        // prediction probability of each class
        float confidence = detectionMat.at<float>( i, (int)objectClass +
probability_index ) ;
```

```
        // for drawing labels with name and confidence
        if ( confidence > confidenceThreshold ) {
            float x_center = detectionMat.at<float>( i, 0 ) * result.cols ;
            float y_center = detectionMat.at<float>( i, 1 ) * result.rows ;
            float width = detectionMat.at<float>( i, 2 ) * result.cols ;
            float height = detectionMat.at<float>( i, 3 ) * result.rows ;
            Point p1( cvRound( x_center - width / 2 ), cvRound( y_center - height /
2 ) ) ;
            Point p2( cvRound( x_center + width / 2 ), cvRound( y_center + height /
2 ) ) ;

            // object 에 box rect
            Rect object( p1, p2 ) ;
            foreground = get_foreground( src, foreground, object ) ;

            // // classNamesVec[ objectClass ] == 이름
            String className = objectClass < classNamesVec.size() ? classNamesVec[
objectClass ] : cv::format( "unknown(%d)", objectClass ) ;


            if ( className.compare( "person?" ) ) {
                count_person++ ;
            }
        }
    }
    putText( foreground, format( "There are %d people.", count_person), Point( 40,
40 ), FONT_HERSHEY_SIMPLEX, 1, Scalar( 255, 255, 255 ), 2 ) ;

    return foreground ;
}



Mat
func2 ( Mat src ) {
    Mat result = src.clone() ;
    cout << "im func2" << endl ;

    CascadeClassifier face_classifier ;
    Mat grayframe ;
    vector<Rect> faces ;

    //face detection configuration
    face_classifier.load( "./haarcascade_frontalface_alt.xml" ) ;


    // green
    cvtColor( result, grayframe, COLOR_BGR2GRAY ) ;
    face_classifier.detectMultiScale( grayframe, faces, 1.1, // increase search
scale by 10% each pass
```

```
                                                    3, // merge groups of three
detections
                                                    0, // not used for a new
cascade
                                                    Size( 20, 20 ), // in size
제일 작은 사람 20
                                                    Size( 50, 50 ) ) ;

    // draw the results
    for ( int i = 0 ; i < faces.size() ; i++ ) {
        Point lb( faces[i].x + faces[i].width, faces[i].y + faces[i].height ) ;
        Point tr( faces[i].x, faces[i].y ) ;
        if ( faces[i].area() < 630 ) {// bgr
            rectangle( result, lb, tr, Scalar(255, 0, 0), 3, 4, 0 ) ; // farthest
blue
        }
        else if ( faces[i].area() > 1285 && faces[i].area() < 2000 ) { // nearest
green
            rectangle( result, lb, tr, Scalar(0, 255, 0), 3, 4, 0 ) ;
        }
        else { // other
            rectangle( result, lb, tr, Scalar(0, 255, 255), 3, 4, 0 ) ;
        }
    }

    return result ;
}


Mat
change_background ( Mat src, Mat foreground, Rect rectangle) {

    Mat result, bgModel, fgModel, result2 ;

    grabCut(src, result, rectangle, bgModel, fgModel, 4, GC_INIT_WITH_RECT) ;

    compare(result, GC_PR_FGD, result, CMP_EQ) ;

    // image 에 result(사람만 흰색)인 것을 덮는다. 이때 foreground 는 복사 안 된다.
    src.copyTo(foreground, result) ; // (output, mask)

    return foreground ;
}

Mat
func3 ( Mat src, Mat background ) {
    Mat result = background.clone() ;
    cout << "im func3" << endl ;

    CascadeClassifier face_classifier ;
```

```
    Mat grayframe ;
    vector<Rect> faces ;

    //face detection configuration
    face_classifier.load( "./haarcascade_frontalface_alt.xml" ) ;


    // green
    cvtColor( src, grayframe, COLOR_BGR2GRAY ) ;
    face_classifier.detectMultiScale( grayframe, faces, 1.1, // increase search
scale by 10% each pass
                                                          3, // merge groups of three
detections
                                                          0, // not used for a new
cascade
                                                          Size( 20, 20 ), // in size
제일 작은 사람 20
                                                          Size( 50, 50 ) ) ;

    // draw the results
    for ( int i = 0 ; i < faces.size() ; i++ ) {
        Point lb( faces[i].x + faces[i].width, faces[i].y + faces[i].height ) ;
        Point tr( faces[i].x, faces[i].y ) ;

        Rect object( lb, tr ) ;
        result = change_background( src, result, object ) ;
    }

    return result ;
}




int
main () {
    VideoCapture cap ;

    if ( cap.open( "./Faces.mp4") == 0 ) {
        cout << "no such file" << endl ;
        return 0 ;
    }

    Mat result ;
    Mat background = imread( "./background.jpg" ) ;
    if ( background.empty() ) {
        cout << "no such file" << endl ;
        return 0 ;
    }
    resize(background, background, Size(640, 480));
```

```
    int key ;
    bool b = false, f = false, g = false ;
    while ( true ) {
        if ( cap.grab() == 0 )
            break ;

        // option
        if ( key == 98 ) // 'b'
            b = !b ;
        else if ( key == 102 )
            f = !f ;
        else if ( key == 103 )
            g = !g ;

        // operation
        cap.retrieve( result ) ;
        resize(result, result, Size(640, 480));
        if ( b ) {
            // func1
            result = func1( result ) ;
        }
        else if ( f ) {
            // func2
            result = func2( result ) ;
        }
        else if ( g ) {
            // func3
            result = func3( result, background ) ;
        }

        imshow( "Result", result ) ;

        key = waitKey( 33 ) ;
    }

    cout << "end" << endl ;

    return 0 ;
}
```