#include <opencv2/objdetect.hpp>

#include <opencv2/highgui.hpp>

#include <opencv2/imgproc.hpp>

#include <iostream>

using namespace std;

using namespace cv;

// Function for Face Detection

void detectAndDraw(Mat& img, CascadeClassifier& cascade,

CascadeClassifier& nestedCascade, double scale);

string cascadeName, nestedCascadeName;

int main(int argc, const char\*\* argv)

{

// VideoCapture class for playing video for which faces to be detected

VideoCapture capture;

Mat frame, image;

// PreDefined trained XML classifiers with facial features

CascadeClassifier cascade, nestedCascade;

double scale = 1;

// Load classifiers from "opencv/data/haarcascades" directory

nestedCascade.load("haarcascade\_eye\_tree\_eyeglasses.xml");

// Change path before execution

cascade.load("haarcascade\_frontalcatface.xml");

// Start Video..1) 0 for WebCam 2) "Path to Video" for a Local Video

capture.open(0);

if (capture.isOpened())

{

// Capture frames from video and detect faces

cout << "Face Detection Started...." << endl;

while (1)

{

capture >> frame;

if (frame.empty())

break;

Mat frame1 = frame.clone();

detectAndDraw(frame1, cascade, nestedCascade, scale);

char c = (char)waitKey(10);

// Press q to exit from window

if (c == 27 || c == 'q' || c == 'Q')

break;

}

}

else

cout << "Could not Open Camera";

return 0;

}

void detectAndDraw(Mat& img, CascadeClassifier& cascade,

CascadeClassifier& nestedCascade,

double scale)

{

vector<Rect> faces, faces2;

Mat gray, smallImg;

cvtColor(img, gray, COLOR\_BGR2GRAY); // Convert to Gray Scale

double fx = 1 / scale;

// Resize the Grayscale Image

resize(gray, smallImg, Size(), fx, fx, INTER\_LINEAR);

equalizeHist(smallImg, smallImg);

// Detect faces of different sizes using cascade classifier

cascade.detectMultiScale(smallImg, faces, 1.1,

2, 0 | CASCADE\_SCALE\_IMAGE, Size(30, 30));

// Draw circles around the faces

for (size\_t i = 0; i < faces.size(); i++)

{

Rect r = faces[i];

Mat smallImgROI;

vector<Rect> nestedObjects;

Point center;

Scalar color = Scalar(255, 0, 0); // Color for Drawing tool

int radius;

double aspect\_ratio = (double)r.width / r.height;

if (0.75 < aspect\_ratio && aspect\_ratio < 1.3)

{

center.x = cvRound((r.x + r.width \* 0.5) \* scale);

center.y = cvRound((r.y + r.height \* 0.5) \* scale);

radius = cvRound((r.width + r.height) \* 0.25 \* scale);

circle(img, center, radius, color, 3, 8, 0);

}

else

rectangle(img, cv::Point(cvRound(r.x \* scale), cvRound(r.y \* scale)),

cv::Point(cvRound((r.x + r.width - 1) \* scale),

cvRound((r.y + r.height - 1) \* scale)), color, 3, 8, 0);

if (nestedCascade.empty())

continue;

smallImgROI = smallImg(r);

// Detection of eyes int the input image

nestedCascade.detectMultiScale(smallImgROI, nestedObjects, 1.1, 2,

0 | CASCADE\_SCALE\_IMAGE, Size(30, 30));

// Draw circles around eyes

for (size\_t j = 0; j < nestedObjects.size(); j++)

{

Rect nr = nestedObjects[j];

center.x = cvRound((r.x + nr.x + nr.width \* 0.5) \* scale);

center.y = cvRound((r.y + nr.y + nr.height \* 0.5) \* scale);

radius = cvRound((nr.width + nr.height) \* 0.25 \* scale);

circle(img, center, radius, color, 3, 8, 0);

}

}

// Show Processed Image with detected faces

imshow("Face Detection", img);

}