Viola Jones Object Detection

**ECE579 Intelligent Robotics II**

Colin Moffitt

moffitt@pdx.edu

2/22/2014

Introduction

The purpose of this project is to explore Viola Jones object detection algorithms. Harr Cascade detection is used for this purpose and leverages pre-existing OpenCV algorithms. Facial recognition is used as the first study and later nested classifiers are used for eyes and glasses. Several facial types are considered under different conditions and the advantages and disadvantages to Viola Jones are discussed.

Framework

The framework used for this project was OpenCV 3.0 compiled in Visual Studio 2013 and running on windows. Frame processing times were well under 15ms, even when many faces appeared in the frame at once, and the code ran smoothly on a modern laptop with an attached webcam.

Training

The Cascade Classifier Training used by the Harr object detection functions were pre-built using the OpenCV opencv\_traincascade function. Three classifiers were used, one for faces, one for eyes, and one for glasses. The eyes and glasses classifiers are used in a nested-cascade setup where the search space is inside the already detected face.

Implementation

Once loading the desired classifier xml files, the OpenCV code enters a loop to pull frames from the attached camera and runs CascadeClassifier.detectMultiScale() on each frame. The results (faces in this case) are then passed to the nested classifier if one is chosen. For each feature found, a circle is drawn around that feature.

Results

Following are screenshots of the results. Clearly having a large beard made detecting my own face more difficult, but surprisingly the Harr classifier did well. Testing on other faces worked almost always, with a few undetected faces and a few false positives.

 Figure 1 and 2: Single face detection, no nested classifiers

 

Figures 3, 4, 5, 6: Multiple face detection

  

As can be seen in the above figures 3-6, several faces were not detected at oblique angles or because of obstructions. Also figure 3 has two false positives in the lower left, and the baby face was missed in fig. 6.

Figures 7, 8, 9: Nested eyes classifier detections

  

In the nested classifier for eyes the results worked well for front facing subjects. Figure 7 shows the maximum angle that was achieved while still correctly classifying the face with both eyes.

Figures 10, 11: Nested glasses classifier

 

The nested glasses classifier was less of a success. Although it worked at times the contrast of the glasses frames needed to be very pronounced and the subject very front-facing and vertical. I was unable to get any nested detection using my own face with this classifier (fig. 11), but in some rare cases was able to get correct detection (fig. 10).

Conclusion

The OpenCV implementation of the Viola Jones algorithm using Harr Object Classification and Detection proved to be a powerful tool in the right circumstances. Some subjects were not recognized and some false positives were generated though. Since the algorithm simplifies the search by only looking for forward facing upright faces, subjects that did not fit this pose or that were partially obscured were not recognized. False positives may be reduced by increasing or changing the classification training to better suit specific scenarios.

Appendix: Source Code

#include "opencv2/objdetect.hpp"

#include "opencv2/imgcodecs.hpp"

#include "opencv2/videoio.hpp"

#include "opencv2/highgui.hpp"

#include "opencv2/imgproc.hpp"

#include "opencv2/core/utility.hpp"

#include "opencv2/videoio/videoio\_c.h"

#include "opencv2/highgui/highgui\_c.h"

#include <cctype>

#include <iostream>

#include <iterator>

#include <stdio.h>

using namespace std;

using namespace cv;

static void help()

{

 cout << "\nThis program demonstrates the cascade recognizer.\n"

 "This classifier can recognize many kinds of rigid objects, once the appropriate classifier is trained.\n"

 "Usage:\n"

 "./facedetect [--cascade=<cascade\_path> this is the primary trained classifier such as frontal face]\n"

 " [--nested-cascade[=nested\_cascade\_path this an optional secondary classifier such as eyes]]\n"

 " [--scale=<image scale greater or equal to 1, try 1.3 for example>]\n"

 " [--try-flip]\n"

 " [filename|camera\_index]\n\n"

 "see facedetect.cmd for one call:\n"

 "./facedetect --cascade=\"../../data/haarcascades/haarcascade\_frontalface\_alt.xml\" --nested-cascade=\"../../data/haarcascades/haarcascade\_eye.xml\" --scale=1.3\n\n"

 "During execution:\n\tHit any key to quit.\n"

 "\tUsing OpenCV version " << CV\_VERSION << "\n" << endl;

}

void detectAndDraw(Mat& img, CascadeClassifier& cascade,

 CascadeClassifier& nestedCascade,

 double scale, bool tryflip);

string cascadeName = "../../data/haarcascades/haarcascade\_frontalface\_alt.xml";

string nestedCascadeName = "../../data/haarcascades/haarcascade\_eye\_tree\_eyeglasses.xml";

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

{

 CvCapture\* capture = 0;

 Mat frame, frameCopy, image;

 const string scaleOpt = "--scale=";

 size\_t scaleOptLen = scaleOpt.length();

 const string cascadeOpt = "--cascade=";

 size\_t cascadeOptLen = cascadeOpt.length();

 const string nestedCascadeOpt = "--nested-cascade";

 size\_t nestedCascadeOptLen = nestedCascadeOpt.length();

 const string tryFlipOpt = "--try-flip";

 size\_t tryFlipOptLen = tryFlipOpt.length();

 string inputName;

 bool tryflip = false;

 help();

 CascadeClassifier cascade, nestedCascade;

 double scale = 2;

 for (int i = 1; i < argc; i++)

 {

 cout << "Processing " << i << " " << argv[i] << endl;

 if (cascadeOpt.compare(0, cascadeOptLen, argv[i], cascadeOptLen) == 0)

 {

 cascadeName.assign(argv[i] + cascadeOptLen);

 cout << " from which we have cascadeName= " << cascadeName << endl;

 }

 else if (nestedCascadeOpt.compare(0, nestedCascadeOptLen, argv[i], nestedCascadeOptLen) == 0)

 {

 if (argv[i][nestedCascadeOpt.length()] == '=')

 nestedCascadeName.assign(argv[i] + nestedCascadeOpt.length() + 1);

 if (!nestedCascade.load(nestedCascadeName))

 cerr << "WARNING: Could not load classifier cascade for nested objects" << endl;

 }

 else if (scaleOpt.compare(0, scaleOptLen, argv[i], scaleOptLen) == 0)

 {

 if (!sscanf(argv[i] + scaleOpt.length(), "%lf", &scale) || scale < 1)

 scale = 1;

 cout << " from which we read scale = " << scale << endl;

 }

 else if (tryFlipOpt.compare(0, tryFlipOptLen, argv[i], tryFlipOptLen) == 0)

 {

 tryflip = true;

 cout << " will try to flip image horizontally to detect assymetric objects\n";

 }

 else if (argv[i][0] == '-')

 {

 cerr << "WARNING: Unknown option %s" << argv[i] << endl;

 }

 else

 inputName.assign(argv[i]);

 }

 if (!cascade.load(cascadeName))

 {

 cerr << "ERROR: Could not load classifier cascade" << endl;

 help();

 return -1;

 }

 if (inputName.empty() || (isdigit(inputName.c\_str()[0]) && inputName.c\_str()[1] == '\0'))

 {

 capture = cvCaptureFromCAM(inputName.empty() ? 0 : inputName.c\_str()[0] - '0');

 int c = inputName.empty() ? 0 : inputName.c\_str()[0] - '0';

 if (!capture) cout << "Capture from CAM " << c << " didn't work" << endl;

 }

 else if (inputName.size())

 {

 image = imread(inputName, 1);

 if (image.empty())

 {

 capture = cvCaptureFromAVI(inputName.c\_str());

 if (!capture) cout << "Capture from AVI didn't work" << endl;

 }

 }

 else

 {

 image = imread("../data/lena.jpg", 1);

 if (image.empty()) cout << "Couldn't read ../data/lena.jpg" << endl;

 }

 cvNamedWindow("ViolaJones-Harr-FaceDetection", 1);

 if (capture)

 {

 cout << "In capture ..." << endl;

 for (;;)

 {

 IplImage\* iplImg = cvQueryFrame(capture);

 frame = cv::cvarrToMat(iplImg);

 if (frame.empty())

 break;

 if (iplImg->origin == IPL\_ORIGIN\_TL)

 frame.copyTo(frameCopy);

 else

 flip(frame, frameCopy, 0);

 detectAndDraw(frameCopy, cascade, nestedCascade, scale, tryflip);

 if (waitKey(10) >= 0)

 goto \_cleanup\_;

 }

 waitKey(0);

 \_cleanup\_:

 cvReleaseCapture(&capture);

 }

 else

 {

 cout << "In image read" << endl;

 if (!image.empty())

 {

 detectAndDraw(image, cascade, nestedCascade, scale, tryflip);

 waitKey(0);

 }

 else if (!inputName.empty())

 {

 /\* assume it is a text file containing the

 list of the image filenames to be processed - one per line \*/

 FILE\* f = fopen(inputName.c\_str(), "rt");

 if (f)

 {

 char buf[1000 + 1];

 while (fgets(buf, 1000, f))

 {

 int len = (int)strlen(buf), c;

 while (len > 0 && isspace(buf[len - 1]))

 len--;

 buf[len] = '\0';

 cout << "file " << buf << endl;

 image = imread(buf, 1);

 if (!image.empty())

 {

 detectAndDraw(image, cascade, nestedCascade, scale, tryflip);

 c = waitKey(0);

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

 break;

 }

 else

 {

 cerr << "Aw snap, couldn't read image " << buf << endl;

 }

 }

 fclose(f);

 }

 }

 }

 cvDestroyWindow("ViolaJones-Harr-FaceDetection");

 return 0;

}

void detectAndDraw(Mat& img, CascadeClassifier& cascade,

 CascadeClassifier& nestedCascade,

 double scale, bool tryflip)

{

 int i = 0;

 double t = 0;

 vector<Rect> faces, faces2;

 const static Scalar colors[] = { CV\_RGB(0, 0, 255),

 CV\_RGB(0, 128, 255),

 CV\_RGB(0, 255, 255),

 CV\_RGB(0, 255, 0),

 CV\_RGB(255, 128, 0),

 CV\_RGB(255, 255, 0),

 CV\_RGB(255, 0, 0),

 CV\_RGB(255, 0, 255) };

 Mat gray, smallImg(cvRound(img.rows / scale), cvRound(img.cols / scale), CV\_8UC1);

 cvtColor(img, gray, COLOR\_BGR2GRAY);

 resize(gray, smallImg, smallImg.size(), 0, 0, INTER\_LINEAR);

 equalizeHist(smallImg, smallImg);

 t = (double)cvGetTickCount();

 cascade.detectMultiScale(smallImg, faces,

 1.1, 2, 0

 //|CASCADE\_FIND\_BIGGEST\_OBJECT

 |CASCADE\_DO\_ROUGH\_SEARCH

 //| CASCADE\_SCALE\_IMAGE

 ,

 Size(30, 30));

 if (tryflip)

 {

 flip(smallImg, smallImg, 1);

 cascade.detectMultiScale(smallImg, faces2,

 1.1, 2, 0

 //|CASCADE\_FIND\_BIGGEST\_OBJECT

 |CASCADE\_DO\_ROUGH\_SEARCH

 //| CASCADE\_SCALE\_IMAGE

 ,

 Size(30, 30));

 for (vector<Rect>::const\_iterator r = faces2.begin(); r != faces2.end(); r++)

 {

 faces.push\_back(Rect(smallImg.cols - r->x - r->width, r->y, r->width, r->height));

 }

 }

 t = (double)cvGetTickCount() - t;

 printf("detection time = %g ms\n", t / ((double)cvGetTickFrequency()\*1000.));

 for (vector<Rect>::const\_iterator r = faces.begin(); r != faces.end(); r++, i++)

 {

 Mat smallImgROI;

 vector<Rect> nestedObjects;

 Point center;

 Scalar color = colors[i % 8];

 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, cvPoint(cvRound(r->x\*scale), cvRound(r->y\*scale)),

 cvPoint(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);

 nestedCascade.detectMultiScale(smallImgROI, nestedObjects,

 1.1, 2, 0

 //|CASCADE\_FIND\_BIGGEST\_OBJECT

 |CASCADE\_DO\_ROUGH\_SEARCH

 //|CASCADE\_DO\_CANNY\_PRUNING

 //|CASCADE\_SCALE\_IMAGE

 ,

 Size(30, 30));

 for (vector<Rect>::const\_iterator nr = nestedObjects.begin(); nr != nestedObjects.end(); nr++)

 {

 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);

 }

 }

 cv::imshow("ViolaJones-`-FaceDetection", img);

}