Facial Recognition using OpenCV

Article · March 2012

CITATIONS
51

READS
46,760

2 authors, including:

Shervin Emami
The University of Queensland
7 PUBLICATIONS **106** CITATIONS

SEE PROFILE

Facial Recognition using OpenCV

Shervin EMAMI[1], Valentin Petruț SUCIU[2]
[1]*Senior Systems Software Engineer*
Mobile Computer Vision Team, NVIDIA
AUSTRALIA
[2]*IT&C Security Master*
Department of Economic Informatics and Cybernetics
Bucharest University of Economic Studies
ROMANIA
shervin.emami@gmail.com, petrut.suciu@gmail.com

Abstract: The growing interest in computer vision of the past decade. Fueled by the steady doubling rate of computing power every 13 months, face detection and recognition has transcended from an esoteric to a popular area of research in computer vision and one of the better and successful applications of image analysis and algorithm based understanding. Because of the intrinsic nature of the problem, computer vision is not only a computer science area of research, but also the object of neuro-scientific and psychological studies, mainly because of the general opinion that advances in computer image processing and understanding research will provide insights into how our brain work and vice versa.
Because of general curiosity and interest in the matter, the author has proposed to create an application that would allow user access to a particular machine based on an in-depth analysis of a person's facial features. This application will be developed using Intel's open source computer vision project, OpenCV and Microsoft's .NET framework.

Key-Words: *face, detection, recognition, system, OpenCV, Eigenface*

1. Introduction

The goal of this article is to provide an easier human-machine interaction routine when user authentication is needed through face detection and recognition.

With the aid of a regular web camera, a machine is able to detect and recognize a person's face; a custom login screen with the ability to filter user access based on the users' facial features will be developed.

The objectives of this thesis are to provide a set of detection algorithms that can be later packaged in an easily-portable framework amongst the different processor architectures we see in machines (computers) today. These algorithms must provide at least a 95% successful recognition rate, out of which less than 3% of the detected faces are false positives.

2. Problem Definition

Over the past decade face detection and recognition have transcended from esoteric to popular areas of research in computer vision and one of the better and successful applications of image analysis and algorithm based understanding. Because of the intrinsic nature of the problem, computer vision is not only a computer science area of research, but also the object of neuro-scientific and psychological studies also, mainly because of the general opinion that advances in computer image processing and understanding research will provide insights into how our brain work and vice versa.
A general statement of the face recognition problem (in computer vision) can be formulated as follows: given still or video images of a scene, identify or verify one or more persons in the scene using a stored database of faces.
Facial recognition generally involves two stages:
Face Detection where a photo is searched to find a face, then the image

ISSN 2067 – 4074

is processed to crop and extract the person's face for easier recognition.
Face Recognition where that detected and processed face is compared to a database of known faces, to decide who that person is.
Since 2002, face detection can be performed fairly easily and reliably with Intel's open source framework called OpenCV [1]. This framework has an in-built Face Detector that works in roughly 90-95% of clear photos of a person looking forward at the camera. However, detecting a person's face when that person is viewed from an angle is usually harder, sometimes requiring 3D Head Pose Estimation. Also, lack of proper brightness of an image can greatly increase the difficulty of detecting a face, or increased contrast in shadows on the face, or maybe the picture is blurry, or the person is wearing glasses, etc.

Face recognition however is much less reliable than face detection, with an accuracy of 30-70% in general. Face recognition has been a strong field of research since the 1990s, but is still a far way away from a reliable method of user authentication. More and more techniques are being developed each year. The Eigenface technique is considered the simplest method of accurate face recognition, but many other (much more complicated) methods or combinations of multiple methods are slightly more accurate.

OpenCV was started at Intel in 1999 by Gary Bradski for the purposes of accelerating research in and commercial applications of computer vision in the world and, for Intel, creating a demand for ever more powerful computers by such applications. Vadim Pisarevsky joined Gary to manage Intel's Russian software OpenCV team. Over time the OpenCV team moved on to other companies and other Research. Several of the original team eventually ended up working in robotics and found their way to Willow Garage. In 2008, Willow Garage saw the need to rapidly advance robotic perception capabilities in an open way that leverages the entire research and commercial community and began actively supporting OpenCV, with Gary and Vadim once again leading the effort [2].
Intel's open-source computer-vision library can greatly simplify computer-vision programming. It includes advanced capabilities - face detection, face tracking, face recognition, Kalman filtering, and a variety of artificial-intelligence (AI) methods - in ready-to-use form. In addition, it provides many basic computer-vision algorithms via its lower-level APIs.
OpenCV has the advantage of being a multi-platform framework; it supports both Windows and Linux, and more recently, Mac OS X.
OpenCV has so many capabilities it can seem overwhelming at first. A good understanding of how these methods work is the key to getting good results when using OpenCV. Fortunately, only a select few need to be known beforehand to get started.
OpenCV's functionality that will be used for facial recognition is contained within several modules. Following is a short description of the key namespaces:
CXCORE namespace contains basic data type definitions, linear algebra and statistics methods, the persistence functions and the error handlers. Somewhat oddly, the graphics functions for drawing on images are located here as well.
CV namespace contains image processing and camera calibration methods. The computational geometry functions are also located here.
CVAUX namespace is described in OpenCV's documentation as containing obsolete and experimental code. However, the simplest interfaces for face recognition are in this module. The code behind them is specialized for face recognition, and they're widely used for that purpose.
ML namespace contains machine-learning interfaces.
HighGUI namespace contains the basic I/O interfaces and multi-platform windowing capabilities.
CVCAM namespace contains interfaces for video access through DirectX on 32-bit Windows platforms.

Eigenfaces is considered the simplest method of accurate face recognition, but many other (much more complicated) methods or combinations of multiple methods are slightly more accurate.
Most resources on face recognition are for basic Neural Networks, which usually don't work as well as Eigenfaces does. And unfortunately there are only some basic explanations for better type of face recognition than Eigenfaces, such as recognition from video and other techniques at the Face Recognition Homepage [4] or 3D Face Recognition Wikipedia page [5] and Active Appearance Models page [6]. But for other techniques, you should read some recent computer vision research papers from CVPR and other computer vision conferences. Most computer vision or machine vision conferences include new advances in face detection and face recognition that give slightly better accuracy. So for example you can look for the CVPR10 and CVPR09 conferences [7].

3. Proposed Solution

When image quality is taken into consideration, there is a plethora of factors that influence the system's accuracy.
It is extremely important to apply various image pre-processing techniques to standardize the images that you supply to a face recognition system. Most face recognition algorithms are extremely sensitive to lighting conditions, so that if it was trained to recognize a person when they are in a dark room, it probably won't recognize them in a bright room, etc. This problem is referred to as "lumination dependent", and there are also many other issues, such as the face should also be in a very consistent position within the images (such as the eyes being in the same pixel coordinates), consistent size, rotation angle, hair and makeup, emotion (smiling, angry, etc), position of lights (to the left or above, etc). This is why it is so important to use a good image preprocessing filters before applying face recognition. You should also do things like removing the pixels around the face that aren't used, such as with an elliptical mask to only show the inner face region, not the hair and image background, since they change more than the face does.

For simplicity, the face recognition system presented in this paper is Eigenfaces using grayscale images. The paper demonstrates how easily is to convert color images to grayscale (also called 'grayscale'), and then to apply Histogram Equalization [8] as a very simple method of automatically standardizing the brightness and contrast of your facial images. For better results, you could use color face recognition (ideally with color histogram fitting in HSV or another color space instead of RGB), or apply more processing stages such as edge enhancement, contour detection, motion detection, etc. Also, this code is resizing images to a standard size, but this might change the aspect ratio of the face. In [9] is described a method on how to resize an image while keeping its aspect ratio the same.
OpenCV uses a type of face detector called a Haar Cascade classifier.
Given an image, which can come from a file or from live video, the face detector examines each image location and classifies it as "Face" or "Not Face." Classification assumes a fixed scale for the face, say 50x50 pixels. Since faces in an image might be smaller or larger than this, the classifier runs over the image several times, to search for faces across a range of scales. This may seem an enormous amount of processing, but thanks to algorithmic tricks, explained in the sidebar, classification is very fast, even when it's applied at several scales.
The classifier uses data stored in an XML file to decide how to classify each image location. The OpenCV download includes four flavors of XML data for frontal face detection, and one for profile faces. It also includes three non-face XML files - one for full body (pedestrian) detection, one for upper body, and one for lower body.

ISSN 2067 – 4074

You'll need to tell the classifier where to find the data file you want it to use. The one I'll be using is called *haarcascade_frontalface_default.xml*. In OpenCV version 1.0, it's located at *[OPENCV_ROOT]/data/haarcascades/haarcascade_frontalface_default.xml* where *[OPENCV_ROOT]* is the path to your OpenCV installation. For example, if you're on Windows XP, and you selected the default installation location, you'd *use [OPENCV_ROOT] = "C:/Program Files/OpenCV"* (If you're working with an older, 16-bit version of Windows, you'd use '\' as the directory separator, instead of '/'.)
It's a good idea to locate the XML file you want to use, and make sure your path to it is correct, before you code the rest of your face-detection program.
It is very easy to use a webcam stream as input to the face recognition system instead of a file list. Basically you just need to grab frames from a camera instead of from a file, and you run forever until the user wants to quit, instead of just running until the file list has run out.

Grabbing frames from a webcam can be implemented easily using this function:

```
// Grab the next camera frame. Waits until the next frame is ready, and
// provides direct access to it, so do NOT modify or free the returned image!
// Will automatically initialize the camera on the first frame.
IplImage*       getCameraFrame(CvCapture* &camera)
{
IplImage *frame;
int w, h;

// If the camera hasn't been initialized, then open it.
if (!camera) {
printf("Acessing the camera ...\n");
camera = cvCreateCameraCapture( 0 );
if (!camera) {
printf("Couldn't access the camera.\n");
exit(1);
}

// Try to set the camera resolution to 320 x 240.

cvSetCaptureProperty(camera, CV_CAP_PROP_FRAME_WIDTH, 320);

cvSetCaptureProperty(camera, CV_CAP_PROP_FRAME_HEIGHT, 240);

// Get the first frame, to make sure the camera is initialized.
frame = cvQueryFrame( camera );

if (frame) {
       w = frame->width;
       h = frame->height;
       printf("Got the camera at %dx%d resolution.\n", w, h);
}

// Wait a little, so that the camera can auto-adjust its brightness.
Sleep(1000);  // (in milliseconds)
}

// Wait until the next camera frame is ready, then grab it.
frame = cvQueryFrame( camera );

if (!frame) {
printf("Couldn't grab a camera frame.\n");
exit(1);
}
return frame;
}
```

This function can be used like this:

```
CvCapture* camera = 0;
// The camera device.

// Quit on "Escape" key.
while ( cvWaitKey(10) != 27 ) {
IplImage            *frame            = getCameraFrame(camera);

…

}
// Free the camera.
cvReleaseCapture( &camera );
```

4. Conclusions

To improve the recognition performance, there are MANY things that can be improved here, some of them being fairly easy to implement. For example, you could add color processing, edge detection, etc.

You can usually improve the face recognition accuracy by using more input images, at least 50 per person, by taking more photos of each person, particularly from different angles and lighting conditions. If you can't take more photos, there are several simple techniques you could use to obtain more

training images, by generating new images from your existing ones:
You could create mirror copies of your facial images, so that you will have twice as many training images and it won't have a bias towards left or right.
You could translate or resize or rotate your facial images slightly to produce many alternative images for training, so that it will be less sensitive to exact conditions.
You could add image noise to have more training images that improve the tolerance to noise.

It is important to have a lot of variation of conditions for each person, so that the classifier will be able to recognize the person in different lighting conditions and positions, instead of looking for specific conditions. But it's also important to make sure that a set of images for a person is not too varied, such as if you rotated some images by 90 degrees. This would make the classifier to be too generic and also give very bad results, so if you think you will have a set of images with too much variance (such as rotation more than 20 degrees), then you could create separate sets of training images for each person. For example, you could train a classifier to recognize "John_Facing_Forward" and another one for "John_Facing_Left" and other ones "Mary_Facing_Forward", "Mary_Facing_Left", etc. Then each classifier can have a lot of variance but not too much, and you simply need to associate the different classifiers for each person with that one person (ie: "John" or "Mary").

That's why you can often get very bad results if you don't use good preprocessing on your images. As I mentioned earlier, Histogram Equalization is a very basic image preprocessing method that can make things worse in some situations, so you will probably have to combine several different methods until you get decent results.

And something important to understand is that at the heart of the algorithm, it is matching images by basically doing the equivalent of subtracting the testing image with a training image to see how similar they are. This would work fairly well if a human performed it, but the computer just thinks in terms of pixels and numbers. So if you imagine that it is looking at one pixel in the test image, and subtracting the gray scale value of that pixel with the value of the pixel in the EXACT same location of each training image, and the lower the difference then the better the match. So if you just move an image by a few pixels across, or use an image that is just a few pixels bigger or has a few more pixels of the forehead showing than the other image, etc, then it will think they are completely different images! This is also true if the background is different, because the code doesn't know the difference between background and foreground (face), which is why its important to crop away as much of the background as you can, such as by only using a small section inside the face that doesn't include any background at all.

Since the images should be almost perfectly aligned, it actually means that in many cases, using small low-res images (such as by shrinking the images to thumbnail size) can give better recognition results than large hi-res images!

Also, even if the images are perfectly aligned, if the testing image is a bit brighter than the training image then it will still think there is not much of a match. Histogram Equalization can help in many cases but it can also make things worse in other cases, so differences in lighting is a difficult & common problem. There are also issues such as if there was a shadow on the left of the nose in the training image and on the right in the test image then it will often cause a bad match, etc.

That's why face recognition is relatively easy to do in real-time if you are training on someone and then instantly

ISSN 2067 – 4074

trying to recognize them after, since it will be the same camera, and background will be the same, their expressions will be almost the same, the lighting will be the same, and the direction you are viewing them from will be the same. So you will often get good recognition results at that moment. But once you try to recognize them from a different direction or from a different room or outside or on a different time of the day, it will often give bad results!
So it's important to do a lot of experimentation if you want better results, and if you still can't get good results after trying many things, perhaps you will need a more complicated face recognition algorithm than PCA (Eigenfaces), such as 3D Face Recognition or Active Appearance Models, mentioned below.

References

[1] Face Detection and Recognition using OpenCV, Article, http://shervinemami.info/faceRecognition.html, Published by Shervin Emami, 2010

[2] Seeing with OpenCV, Article, http://www.cognotics.com/opencv/servo_2007_series/part_1/index.html, Published by Robin Hewitt, 2010

[3] OpenCV Homepage, http://opencv.willowgarage.com

[4] Face Recognition Homepage, http://www.face-rec.org/algorithms/

[5] Wikipedia, Three-dimensional face recognition, http://en.wikipedia.org/wiki/Three-dimensional_face_recognition

[6] Wikipedia, Active appearance model, http://en.wikipedia.org/wiki/Active_appearance_model

[7] Computer Vision Papers, http://www.cvpapers.com/

[8] Wikipedia, Histogram equalization, http://en.wikipedia.org/wiki/Histogram_equalization

[9] Shervin Emami, Rotating or Resizing an Image in OpenCV, http://shervinemami.info/imageTransforms.html