Google Custom Search



# Cascade Classifier

## Goal

In this tutorial,

- We will learn how the Haar cascade object detection works.
- We will see the basics of face detection and eye detection using the Haar Feature-based Cascade Classifiers
- We will use the **cv::CascadeClassifier** class to detect objects in a video stream. Particularly, we will use the functions:
  - **cv::CascadeClassifier::load** to load a .xml classifier file. It can be either a Haar or a LBP classifier
  - **cv::CascadeClassifier::detectMultiScale** to perform the detection.

## Theory

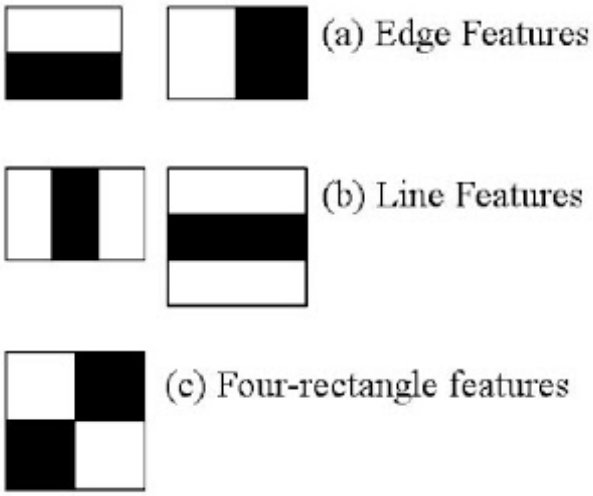
Object Detection using Haar feature-based cascade classifiers is an effective object detection method proposed by Paul Viola and Michael Jones in their paper, "Rapid Object Detection using a Boosted Cascade of Simple Features" in 2001. It is a machine learning based approach where a cascade function is trained from a lot of positive and negative images. It is then used to detect objects in other images.

Here we will work with face detection. Initially, the algorithm needs a lot of positive images (images of faces) and negative images (images without faces) to train the classifier. Then we need to extract features from it. For this, Haar features shown in the below image are used. They are just like our convolutional

Loading [MathJax]/extensions/MathMenu.js

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

kernel. Each feature is a single value obtained by subtracting sum of pixels under the white rectangle from sum of pixels under the black rectangle.



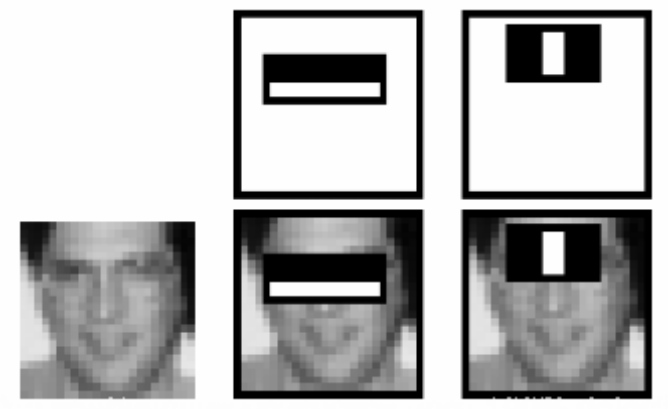

**image**

Now, all possible sizes and locations of each kernel are used to calculate lots of features. (Just imagine how much computation it needs? Even a 24x24 window results over 160000 features). For each feature calculation, we need to find the sum of the pixels under white and black rectangles. To solve this, they introduced the integral image. However large your image, it reduces the calculations for a given pixel to an operation involving just four pixels. Nice, isn't it? It makes things super-fast.

But among all these features we calculated, most of them are irrelevant. For example, consider the image below. The top row shows two good features. The first feature selected seems to focus on the property that the region of the eyes is often darker than the region of the nose and cheeks. The second feature selected relies on the property that the eyes are darker than the bridge of the nose. But the same windows applied to cheeks or any other place is irrelevant. So how do we select the best features out of 160000+ features? It is achieved by **Adaboost**.

Loading [MathJax]/extensions/MathMenu.js

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD



**image**

For this, we apply each and every feature on all the training images. For each feature, it finds the best threshold which will classify the faces to positive and negative. Obviously, there will be errors or misclassifications. We select the features with minimum error rate, which means they are the features that most accurately classify the face and non-face images. (The process is not as simple as this. Each image is given an equal weight in the beginning. After each classification, weights of misclassified images are increased. Then the same process is done. New error rates are calculated. Also new weights. The process is continued until the required accuracy or error rate is achieved or the required number of features are found).

The final classifier is a weighted sum of these weak classifiers. It is called weak because it alone can't classify the image, but together with others forms a strong classifier. The paper says even 200 features provide detection with 95% accuracy. Their final setup had around 6000 features. (Imagine a reduction from 160000+ features to 6000 features. That is a big gain).

So now you take an image. Take each 24x24 window. Apply 6000 features to it. Check if it is face or not. Wow.. Isn't it a little inefficient and time consuming? Yes, it is. The authors have a good solution for that.

In an image, most of the image is non-face region. So it is a better idea to have a simple method to check if a window is not a face region. If it is not, discard it in a single shot, and don't process it again. Instead, focus on regions where there can be a face. This way, we spend more time checking possible face regions.

For this they introduced the concept of **Cascade of Classifiers**. Instead of applying all 6000 features on a window, the features are grouped into different stages of classifiers and applied one-by-one. (Normally the first few stages will contain very many fewer features). If a window fails the first stage, discard it. We don't consider the remaining features on it. If it passes, apply the second stage of features and continue the process. The window which passes all stages !

Loading [MathJax]/extensions/MathMenu.js

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

The authors' detector had 6000+ features with 38 stages with 1, 10, 25, 25 and 50 features in the first five stages. (The two features in the above image are actually obtained as the best two features from Adaboost). According to the authors, on average 10 features out of 6000+ are evaluated per sub-window.

So this is a simple intuitive explanation of how Viola-Jones face detection works. Read the paper for more details or check out the references in the Additional Resources section.

## Haar-cascade Detection in OpenCV

C++ Java Python

OpenCV provides a training method (see **Cascade Classifier Training**) or pretrained models, that can be read using the **cv::CascadeClassifier::load** method. The pretrained models are located in the data folder in the OpenCV installation or can be found here.

The following code example will use pretrained Haar cascade models to detect faces and eyes in an image. First, a **cv::CascadeClassifier** is created and the necessary XML file is loaded using the **cv::CascadeClassifier::load** method. Afterwards, the detection is done using the **cv::CascadeClassifier::detectMultiScale** method, which returns boundary rectangles for the detected faces or eyes.

This tutorial code's is shown lines below. You can also download it from here

```
#include "opencv2/objdetect.hpp"
#include "opencv2/highgui.hpp"
#include "opencv2/imgproc.hpp"
#include "opencv2/videoio.hpp"
#include <iostream>

using namespace std;
using namespace cv;

void detectAndDisplay( Mat frame );

CascadeClassifier face_cascade;
CascadeClassifier eyes_cascade;

int main( int argc, const char** argv )
{
    CommandLineParser parser(argc, argv,
                             "{help h||}"
                             "{face_cascade|data/haarcascades/haarcascade_frontalface_alt.xml|Path to face cascade.}"
                             "{eyes_cascade|data/haarcascades/haarcascade_eye_tree_eyeglasses.xml|Path to eyes cascade.}"
                             "{camera|0|Camera device number.}");
```

Loading [MathJax]/extensions/MathMenu.js

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
    parser.about( "\nThis program demonstrates using the cv::CascadeClassifier class to detect objects (Face + eyes) in a
       video stream.\n"
                  "You can use Haar or LBP features.\n\n" );
    parser.printMessage();

    String face_cascade_name = samples::findFile( parser.get<String>("face_cascade") );
    String eyes_cascade_name = samples::findFile( parser.get<String>("eyes_cascade") );

    //-- 1. Load the cascades
    if( !face_cascade.load( face_cascade_name ) )
    {
        cout << "--(!)Error loading face cascade\n";
        return -1;
    };
    if( !eyes_cascade.load( eyes_cascade_name ) )
    {
        cout << "--(!)Error loading eyes cascade\n";
        return -1;
    };

    int camera_device = parser.get<int>("camera");
    VideoCapture capture;
    //-- 2. Read the video stream
    capture.open( camera_device );
    if ( ! capture.isOpened() )
    {
        cout << "--(!)Error opening video capture\n";
        return -1;
    }

    Mat frame;
    while ( capture.read(frame) )
    {
        if( frame.empty() )
        {
            cout << "--(!) No captured frame -- Break!\n";
            break;
        }

        //-- 3. Apply the classifier to the frame
        detectAndDisplay( frame );

        if( waitKey(10) == 27 )
        {
            break; // escape
        }
    }
    return 0;
}

t frame )
```

Loading [MathJax]/extensions/MathMenu.js

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

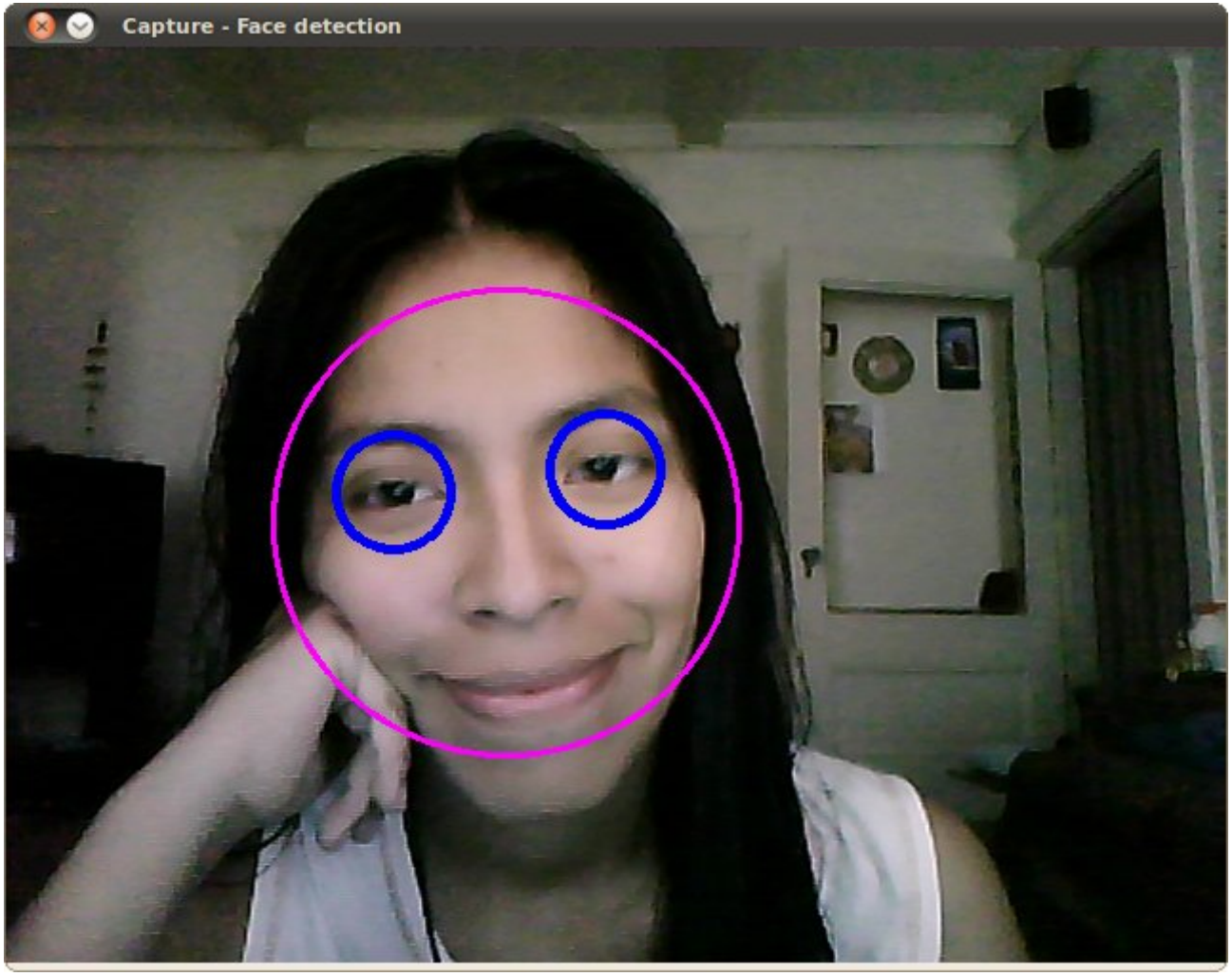
```
    Mat frame_gray;
    cvtColor( frame, frame_gray, COLOR_BGR2GRAY );
    equalizeHist( frame_gray, frame_gray );

    //-- Detect faces
    std::vector<Rect> faces;
    face_cascade.detectMultiScale( frame_gray, faces );

    for ( size_t i = 0; i < faces.size(); i++ )
    {
        Point center( faces[i].x + faces[i].width/2, faces[i].y + faces[i].height/2 );
        ellipse( frame, center, Size( faces[i].width/2, faces[i].height/2 ), 0, 0, 360, Scalar( 255, 0, 255 ), 4 );

        Mat faceROI = frame_gray( faces[i] );

        //-- In each face, detect eyes
        std::vector<Rect> eyes;
        eyes_cascade.detectMultiScale( faceROI, eyes );

        for ( size_t j = 0; j < eyes.size(); j++ )
        {
            Point eye_center( faces[i].x + eyes[j].x + eyes[j].width/2, faces[i].y + eyes[j].y + eyes[j].height/2 );
            int radius = cvRound( (eyes[j].width + eyes[j].height)*0.25 );
            circle( frame, eye_center, radius, Scalar( 255, 0, 0 ), 4 );
        }
    }

    //-- Show what you got
    imshow( "Capture - Face detection", frame );
}
```

## Result

1. Here is the result of running the code above and using as input the video stream of a built-in webcam:

Loading [MathJax]/extensions/MathMenu.js

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

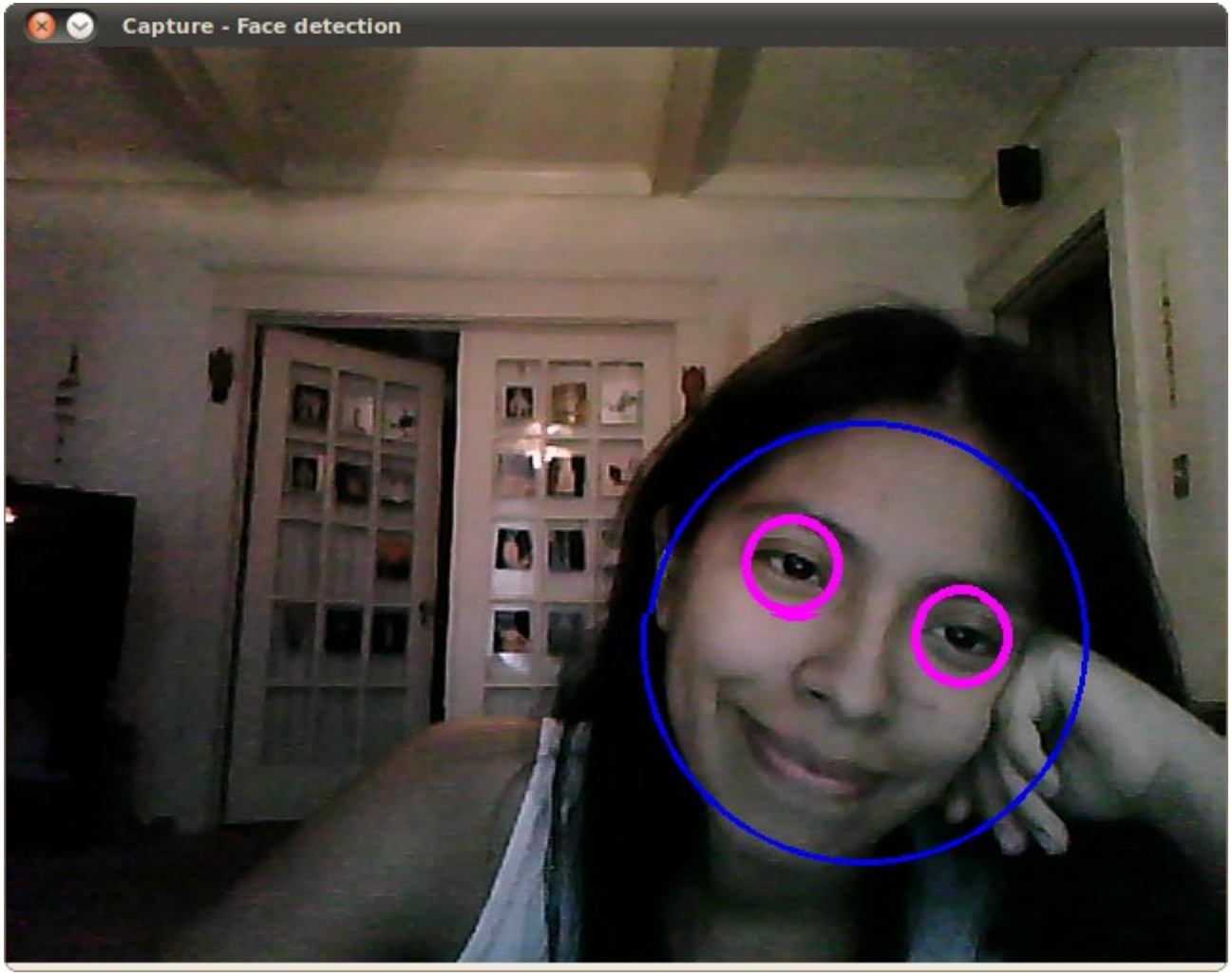



Be sure the program will find the path of files *haarcascade_frontalface_alt.xml* and *haarcascade_eye_tree_eyeglasses.xml*. They are located in *opencv/data/haarcascades*

2. This is the result of using the file *lbpcascade_frontalface.xml* (LBP trained) for the face detection. For the eyes we keep using the file used in the tutorial.

Loading [MathJax]/extensions/MathMenu.js

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD




## Additional Resources

1. Paul Viola and Michael J. Jones. Robust real-time face detection. International Journal of Computer Vision, 57(2):137–154, 2004. **[217]**
2. Rainer Lienhart and Jochen Maydt. An extended set of haar-like features for rapid object detection. In Image Processing. 2002. Proceedings. 2002 International Conference on, volume 1, pages I–900. IEEE, 2002. **[126]**
3. Video Lecture on Face Detection and Tracking
4. ...garding Face Detection by Adam Harvey

Loading [MathJax]/extensions/MathMenu.js

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

5. OpenCV Face Detection: Visualized on Vimeo by Adam Harvey



Loading [MathJax]/extensions/MathMenu.js

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD