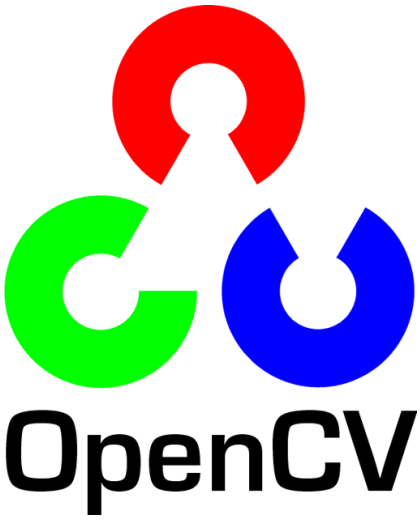
PROJECT

FACE N EYES, CAR AND PEDESTRIAN DETECTION USING openCV

ATUL KUMAR VERMA--19B090004



Introduction

Face Detection, a widely popular subject with a huge range of applications. Modern day Smartphones and Laptops come with in-built face detection softwares, which can authenticate the identity of the user. There are numerous apps that can capture, detect and process a face in real time, can identify the age and the gender of the user, and also can apply some really cool filters. The list is not limited to these mobile apps, as Face Detection also has a wide range of applications in Surveillance, Security and Biometrics as well. But the origin of its Success stories dates back to *2001*, when *Viola and Jones* proposed the first ever Object Detection Framework for Real Time Face Detection in Video Footage.

Computer Vision Plays a vital role in traffic management and surveillance systems and has been an active research area in the past years. In systems like these, the detection of vehicles and also classification of the vehicle plays a major role. The datasets are traffic videos of urban environments taken from various cities around the world which were used to train the classifier hence generating a robust classifier. The proposed approach is computationally less expensive with faster processing speed. The experiments on-road prove it to be a robust and real time algorithm which is highly competitive with the existing architecture.

OpenCV (Open Source Computer Vision Library: http://opencv.org) is an open-source BSD-licensed library that includes several hundreds of computer vision algorithms.

Haar Cascade Classifiers : We will implement our use case using the Haar Cascade classifier. Haar Cascade classifier is an effective object detection approach which was proposed by Paul Viola and Michael Jones in their paper, "**Rapid Object Detection using a Boosted Cascade of Simple Features**" in 2001.

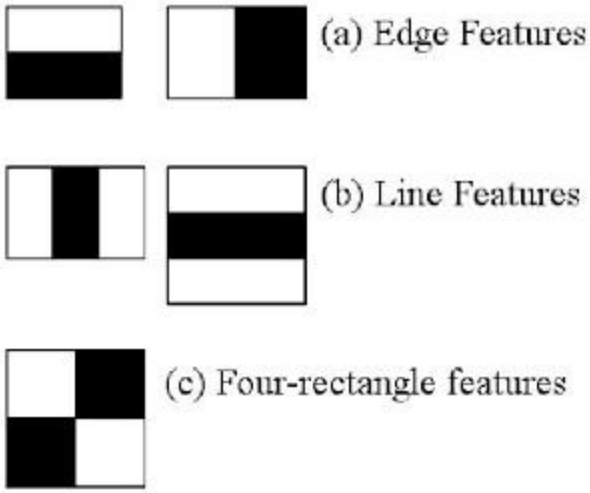
Theory

FACE AND EYE DETECTION

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

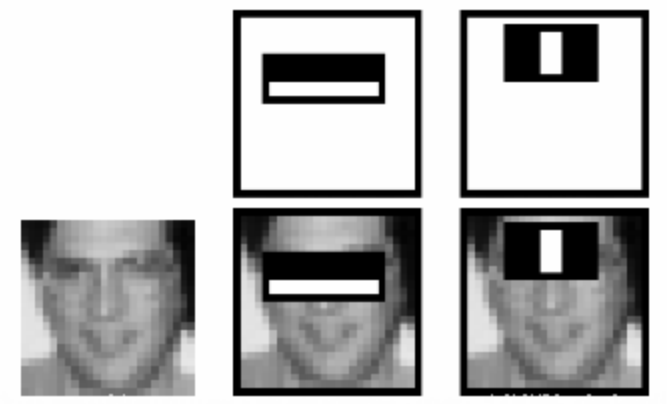
convolutional kernel. Each feature is a single value obtained by subtracting the sum of pixels under the white rectangle from the sum of pixels under the black rectangle.



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**.



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).

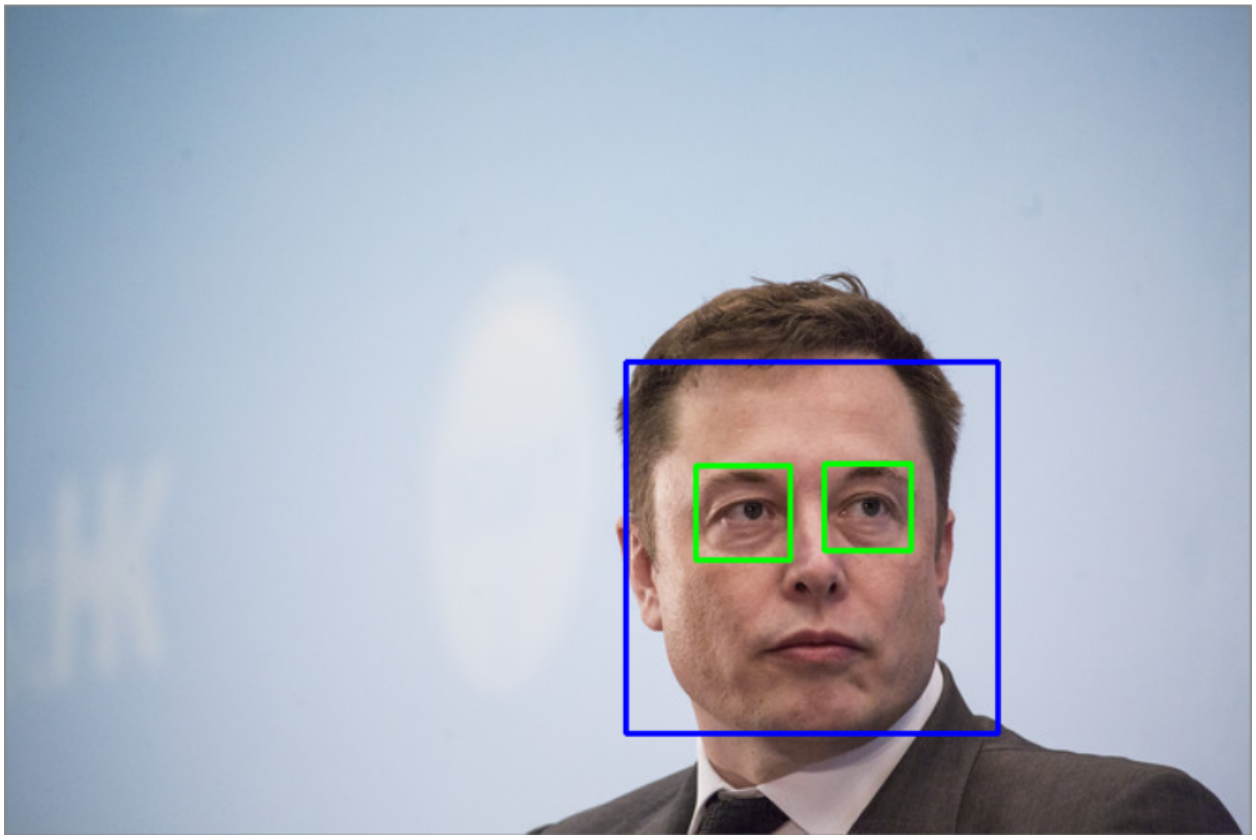
So now you take an image. Take each 24x24 window. Apply 6000 features to it. Check if it is face or not. Wow.. Isn't it a little inefficient and time consuming? Yes, it is. The authors have a good solution for that.

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 few(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 is a face region. How is that plan!

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.



Vehicle detection from streaming video

We implement one more use case from the haar cascade classifier. In this use-case we will be detecting the vehicles from a streaming video.I have implemented these use-cases to show how it works. There are whole bunch of other xmls also for this classifier which we could use to implement to implement few other computer vision cases as well. The implementation here is same as the one we did for face detection.

Step 1

In order to detect the features of a vehicle we need to import the **haarcascade_car.xml.**

Use the **VideoCapture** of cv2 and store the value in **cap**

Reading (`cap.read()`) from a `VideoCapture` returns a tuple `(ret, frame)`. With the first item you check whether the reading was successful, and if it was then you proceed to use the returned `frame`.

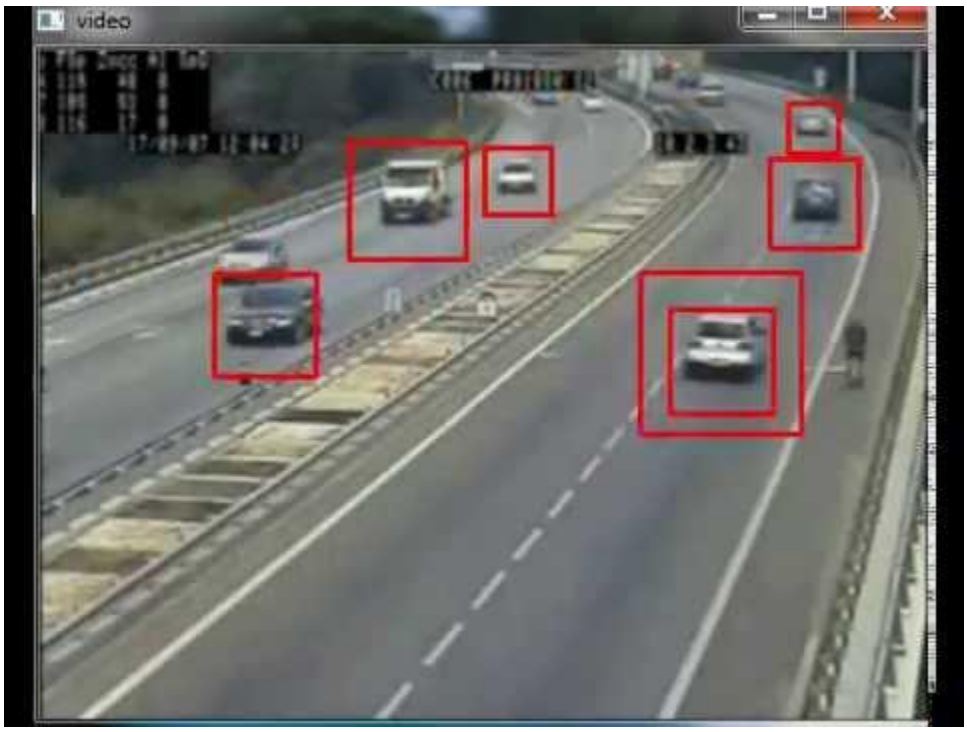
Step 2

Now that we have the tuple of `(ret, frame)` , we will convert the BGR channel image to gray channel. Reasons being the same, we are converting the

image to gray scale and using the classifier function **detectMultiScale** to extract the x-coordinate, y-coordinate, width (w) and height(h), and gray scale is used for better performance throughput.

Step 3

Based on the extracted features/dimensions of the cars, we will loop through them and draw a rectangle around each frame of the image.



Pedestrian detection from a streaming video

The implementation is completely the same as the vehicle detection. The only difference here will be, we will be using the **haarcascade_fullbody.xml** to identify the features of the pedestrian's body.