Creating Your Own Haar Cascades for Object Detection

Mudit Goswami[1] & Amolak Ratan Kalra[2]

Abstract— **In this paper, we propose a novel technique of creating a cascade of classifiers based on the Haar-like features for object detection. For the sake of simplicity the implementation of the haar cascades has been limited to the application of Face, Eyes and Watch Detection. This technique could be easily extended to various range of objects. A step-by-step process has been demonstrated for preparing the Haar-Cascade Classifier in order to perform robust object detection.**

Keywords - Face Detection, Eye Detection, Watch Detection, Haar Features, Haar-Wavelet, Image Processing, Computer Vision, Classification, Weak Classifiers, Haar-Training, XML file, OpenCV.

I. INTRODUCTION

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 Featuresïn 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.

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 are used. They are just like our convolutional kernel. Each feature is a single value obtained by subtracting sum of pixels under white rectangle from sum of pixels under black rectangle.

Basically, the steps involved in training the haar-like classifier are as follows

1. Collection of positive and negative training images.
2. Marking the positive images.
3. Creating a .vec file for positive marked images.
4. Training the classifier
5. Running the classifier.

The main contribution of this paper is to develop a step-by-step process for training a Haar-like classifier for performing object detection for a wide range of objects.

II. OVERVIEW OF FACE DETECTION USING THE HAAR CASCADES

In the previous section, we define the haar features as convolutional kernel, where each feature is a single value obtained by subtracting sum of pixels under white rectangle from sum of pixels under black rectangle. The haar features are shown in figure 1.

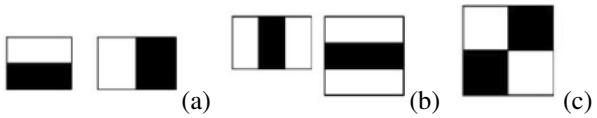

Fig. 1: (a). Edge Features, (b). Line Features, (c). Four-rectangle features

Now all possible sizes and locations of each kernel is used to calculate plenty 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 sum of pixels under white and black rectangles. To solve this,the integral images is used. It simplifies calculation of sum of pixels, how large may be the number of pixels, to an operation involving just four pixels.It makes operations super-fast.

But among all these features we calculated, most of them are irrelevant. For example, consider the Figure 2. 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 applying on cheeks or any other place is irrelevant. So how do we select the best features out of 160000+ features? It is achieved by Adaboost.

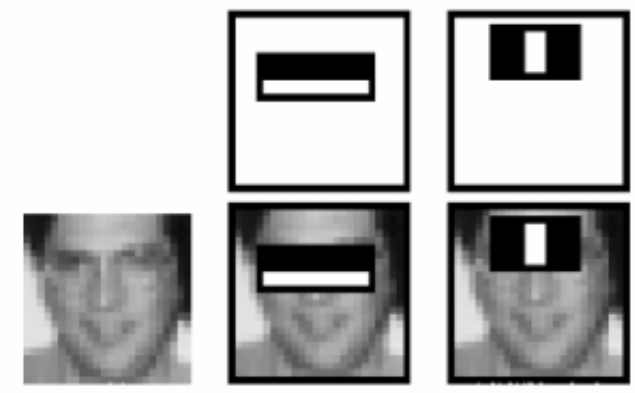

Fig. 2: Good Features

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. But obviously, there will be errors or misclassifications. We select the features with minimum error rate, which means they are

This work was done as an additional assignment for the course on Digital Image Processing taught by Prof. C. Patwardhan

[1]Mudit Goswami and [2]AR. Kalra are with Faculty of Engineering, Electrical Engineering with spcl. in Computer Science, Datalbagh Educational Institute, Agra, U.P, India muditgos29@gmail.com

the features that best classifies 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 again same process is done. New error rates are calculated. Also new weights. The process is continued until required accuracy or error rate is achieved or required number of features are found).

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 in [1] 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. In an image, most of the image region 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. Don't process it again. Instead focus on region where there can be a face. This way, we can find more time to check a possible face region.

For this purpose the concept of Cascade of Classifiers was used. Instead of applying all the 6000 features on a window, group the features into different stages of classifiers and apply one-by-one. (Normally first few stages will contain very less number of features). If a window fails the first stage, discard it. We don't consider 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. 6000+ features with 38 stages with 1, 10, 25, 25 and 50 features in first five stages. (Two features in the above image is actually obtained as the best two features from Adaboost). According to paper [1], on an average, 10 features out of 6000+ are evaluated per sub-window.

III. Creating positive and negative training images

For training a boosted cascade of weak classifiers we need a set of positive samples (containing actual objects you want to detect) and a set of negative images (containing everything you do not want to detect). The set of negative samples must be prepared manually, whereas set of positive samples is created using the opencv_createsamples application, which is a functionality offered by OpenCV Library.

III-A. Negative Samples

Negative samples are taken from arbitrary images, not containing objects you want to detect. These negative images, from which the samples are generated, should be listed in a special negative image file containing one image path per line (can be absolute or relative). Note that negative samples and sample images are also called background samples or background images.

Fig. 3: Negative Sample Images

Described images may be of different sizes. However, each image should be equal or larger than the desired training window size (which corresponds to the model dimensions, most of the times being the average size of your object), because these images are used to subsample a given negative image into several image samples having this training window size.

III-B. Positive Samples

Positive samples are created by the opencv_createsamples application. They are used by the boosting process to define what the model should actually look for when trying to find your objects of interest. The application supports two ways of generating a positive sample dataset.

1. You can generate a bunch of positives from a single positive object image.
2. You can supply all the positives yourself and only use the tool to cut them out, resize them and put them in the opencv needed binary format.

Fig. 4: Positive Sample Image

While the first approach works decently for fixed objects, like very rigid logo's, it tends to fail rather soon for less rigid objects. In that case we must use the second approach. The first approach takes a single object image with for example a company logo and creates a large set of positive samples from the given object image by randomly rotating the object, changing the image intensity as well as placing the image on arbitrary backgrounds. The amount and range of randomness can be controlled by command line arguments of the opencv_createsamples application.

IV. Marking the positive images

When running opencv_createsamples in this way, the following procedure is used to create a sample object instance: The given source image is rotated randomly around all three axes. The chosen angle is limited by -maxxangle, -maxyangle and -maxzangle. Then pixels having the intensity from the [bg_color-bg_color_threshold; bg_color+bg_color_threshold] range are interpreted as transparent. White noise is added to the intensities of the foreground. If the -inv key is specified then foreground pixel

intensities are inverted. If -randinv key is specified then algorithm randomly selects whether inversion should be applied to this sample. Finally, the obtained image is placed onto an arbitrary background from the background description file, resized to the desired size specified by -w and -h and stored to the vec-file, specified by the -vec command line option.

Fig. 5: Positive Image Marked on Negative Sample Image

Positive samples also may be obtained from a collection of previously marked up images, which is the desired way when building robust object models. This collection is described by a text file similar to the background description file. Each line of this file corresponds to an image. The first element of the line is the filename, followed by the number of object annotations, followed by numbers describing the coordinates of the objects bounding rectangles (x, y, width, height).
In our root folder we must have the following files and folders, **data**, for storing the Cascade classifier at each stage of training, **info**, for storing all the positive marked images generated by the opencv_createsamples command, **neg**, contains the negative set of images, **bg.txt**, contains the location of all negative images. The command for creating the positive samples from a single image of the object is given as,
opencv_createsamples -img watch5050.jpg -bg bg.txt -info info/info.lst -pngoutput info -maxxangle 0.5 -maxyangle 0.5 -maxzangle 0.5 -num 1950

V. Creating a .vec (vector) file for the positive marked images

Now we create a positive vector file which contains information about the location of the info file, how many images we want to contain in the file, what dimension should the image be in the vector file and finally where to output the results. These files can be made larger by using images larger than 20 x 20, which will result in exponential growth of training time.The command for making a .vec file is as follows:
opencv_createsamples -info info/info.lst -num 1950 -w 20 -h 20 -vec positives.vec

VI. Training the classifier

Now at this point we have completed all our requirements for training the Haar-Cascade classifier. We'll start the training by specifying the parameters like the no. of stages, number of positive samples, number of negative samples, location of positive and negative images. The command for running the haar-cascade training is as follows:
opencv_traincascade -data data -vec positives.vec -bg bg.txt -numPos 1800 -numNeg 900 -numStages 10 -w 20 -h 20

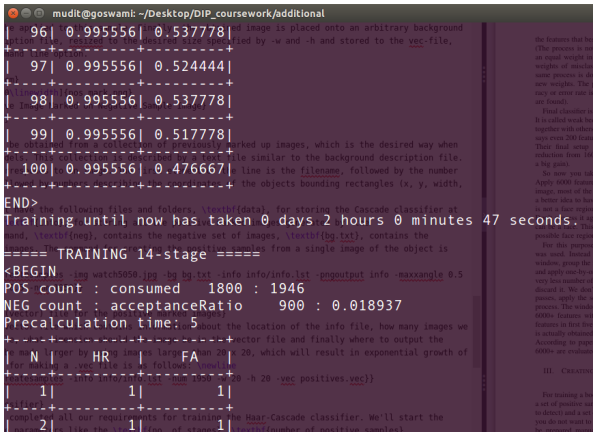

Fig. 6: Training Snapshot

VII. Results

This section includes the experimentation results of various scenarios tested with the trained haar caascade classifier. The haar cascade classifier for Face and Eyes are already available in the OpenCV Libraries, we used those haar-cascades for detecting Face and Eye. The haar-cascade for the watch detection was trained for 14 stages. The training was done using the OpenCV libraries. The object detection includes the detection of face, eyes, watch, multiple faces. the results for the following are as follows:

Fig. 7: Watch Detection

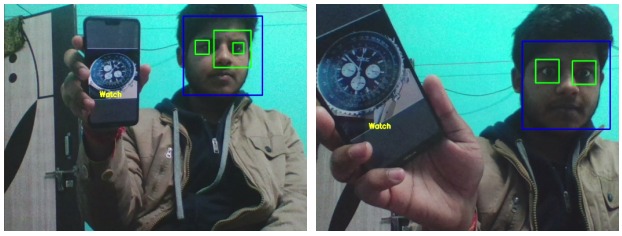

Fig. 8: Face, Eye and Watch Detection

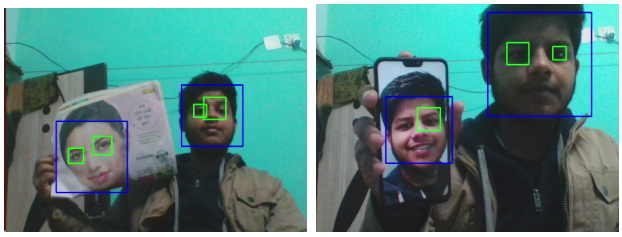

Fig. 9: Multiple Face and Eye Detection

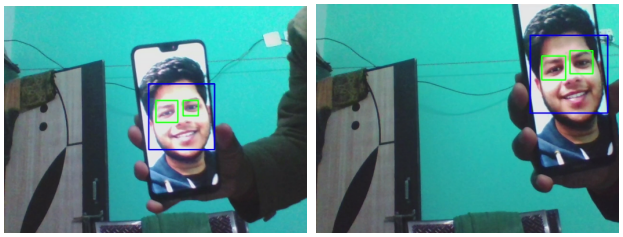

Fig. 10: Face and Eye Detection

VIII. CONCLUSIONS

Thus from the experimentation results we can observe that we can perform object detection on a various range of objects. The only thing required for the recognition is the images of the object to be tracked. The more the number of positive images for training the more robust becomes the classifier, hence the accuracy for the object detection also increases. Here as we say that the training cosumes alot of time, an alternative approach would be the one to use Deep Learning for learning the objects.

APPENDIX

The complete implementation of the project has been done using the OpenCV Libraries. The camera used was a laptop's webcam.

ACKNOWLEDGMENT

I would like to thank Prof. C. Patwardhan for giving this opportunity to work on this project as my additional assignment. It was a great learning experience for us as individuals.

REFERENCES

[1] P.Viola, M. Jones, Rapid Object Detection using a Boosted Cascade of Simple Features, 2001

[2] Mahdi Rezaei, Reinhard Klette, "3D cascade of classifiers for open and closed eye detection in driver distraction monitoring", In Proc. Computer Analysis of Images and Patterns, pp. 171-179, 2011

[3] Mahdi Rezaei, Hossein Ziaei Nafchi, Sandino Morales, "Global Haar-like Features: A New Extension of Classic Haar Features for Efficient Face Detection in Noisy Images", 6th Pacific-Rim Symposium on Image and Video Technology, PSIVT 2013.

[4] OpenCV-3.0.0 Documentation

[5] Mahdi Rezaei, "Artistic Rendering of Human Portraits Paying Attention to Facial Features", Arts and Technology, 101, pp. 90-99, 2012