OpenCV Object Detection in Video

By Brenden Bishop

OpenCV is a library for computer visions designed for analyze, process, and understand the objects from images with the aim to produce information. This software can be used to detect a multitude of objects in an image or a video. This process is used for facial recognition, object detection, people detection, object tracking, etc. This guide is to help those who are looking to learn some **OpenCV** for the purpose of object detection in videos of any and or several different types of objects.

First Step: Getting the OpenCV software and setup

Visit http://opencv.willowgarage.com/wiki/InstallGuide for the OpenCV setup guide. Once installed and setup with all the necessary libraries continue to step too.

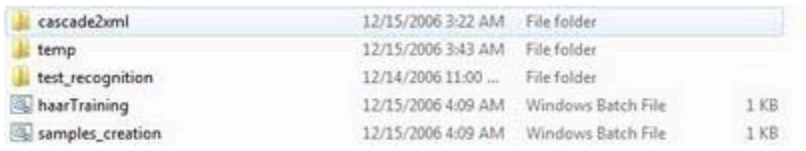
Second Step: Prepping for the Object Detection training

Now that you have installed OpenCV, you are ready.

To start, download the "tools.exe" from the link below:
http://www.4shared.com/file/mpbQkWp4/tools.html

These tools are provided by **Amit Arup Nayak** from http://nayakamitarup.blogspot.com/. For the startup of object detection I will be highlighting parts of his guide which is slightly outdated. For object detection as a whole you can use Nayak's guide for references/further help and this source too:
http://www.intorobotics.com/how-to-detect-and-track-object-with-opencv/

Unzip the tools.exe into a folder. You will find the contents like given below:



cascade2xml	12/15/2006 3:22 AM	File folder	
temp	12/15/2006 3:43 AM	File folder	
test_recognition	12/14/2006 11:00 ...	File folder	
haarTraining	12/15/2006 4:09 AM	Windows Batch File	1 KB
samples_creation	12/15/2006 4:09 AM	Windows Batch File	1 KB

For our training purposes the only tool needed in this zip is the temp folder(zip file provided by http://nayakamitarup.blogspot.com/2011/07/how-to-make-your-own-haar-trained-xml.html). Copy the temp folder to some known directory where the openCV libraries were contained to keep everything organized. For our training we will use the **opencv_createsamples.exe** and **opencv_traincascade.exe** files provided by openCV. To setup for the training, create a .txt file named haartraining1 for example. Inside this text file(notepad) include something like this;

Bold words are just comments, when entering in text file continues each line with just a single space separation. And all the positive, negative, and data folders are located in the temp folder provided above.

cd C:\WhereYouInstalled \opencv\opencv-2.4.3\win32\bin

opencv_createsamples.exe -info C:\someLocation\Project1\temp\positive\info.dat

-This will be where your opencv libraries are contained and the subfolder where stored your positives from the object marker in the info.txt file. More about this later.

-vec C:\someLocation\Project1\temp\data\samples.vec

-This is the same location as the main folder of before, but different subfolder called data. In this folder to start should only have one subfolder called "cascade". This line in command prompt will create the .vec file from the createsamples.exe and save it to the data folder.

-bg C:\someLocation\Project1\temp\negative\negatives.dat -num 400

-This is the location of the bg(background) images, aka the negative samples. –num 400 = the number of positive samples you have.

opencv_traincascade.exe -data C:\someLocation\Project1\temp\data\cascade

-This is the command to create the .xml cascade files and they're to be stored in the cascade folder inside the data folder.

-vec C:\someLocation\utilities\Project1\temp\data\samples.vec

-This is where the vec file called "samples" will be created and stored in the data folder.

-bg C:\someLocation\Project1\temp\negative\negatives.dat -numPos 400 -numNeg 1477 -numStages 14 -minHitRate 0.999

-This is the final command. This does the training of the samples created. -numPos is the number of positive samples it must match number put in createsamples.exe portion. The -numNeg is the number of negative samples. -numStages is the number of stages you want to run. More stages takes much longer time but creates more accurate results, however it is possible to over-train so more stages is not always better. -maxFalseAlarmRate is the rate of which you declare a match. So the lower the falseAlarmRate is the fewer amounts of false positives will be detected, but a too low rate and many positive matches will be discarded as well. Generally I found a 0.2 rate at 6-10 stages to meet my needs best. -minHitRate you always want to keep 0.999 for best results. And keep in mind, ALL of these must match the folders and filenames of the createsamples.exe portion.

pause...

-This just keeps the command prompt open for you to view the process when it's complete.

When all said and done your txt file, inside should look like this:

```
cd C:\WhereYouInstalled \opencv\opencv-2.4.3\win32\bin

opencv_createsamples.exe -info C:\someLocation\Project1\temp\positive\info.dat -vec
C:\someLocation\Project1\temp\data\samples.vec -bg
C:\someLocation\Project1\temp\negative \negatives.dat -num 400

opencv_traincascade.exe -data C:\someLocation\Project1\temp\data\cascade -vec
C:\someLocation \utilities\Project1\temp\data\samples.vec -bg
C:\someLocation\Project1\temp\negative\negatives.dat -numPos 400 -numNeg 1477 -
numStages 14  -minHitRate 0.999

pause...
```

Now save it to the same directory where your temp folder is located. Once the txt file is saved, click on it and rename it and change the .txt to .bat it will prompt you that changing the extension will do so and so, hit okay.

Step Three: Creating Positive and Negative Samples

Now that you have setup all the folder locations and the command prompt .bat file. You now need to create all your positive and negative samples. In my circumstance I was detecting multiple objects in a video file. If this is the case the positive samples can be cropped out from each frame of the video where the object you want is located. To do this you can use a program called VirtualDub.

Download here: http://sourceforge.net/projects/virtualdub/files/virtualdub-win/1.9.11.32842/VirtualDub-1.9.11.zip/download

Simply load in the video file and then hit File-Export- then choose your settings to export as. **Note that your positive image samples must be in the .bmp format to use the ObjectMarker tool.**

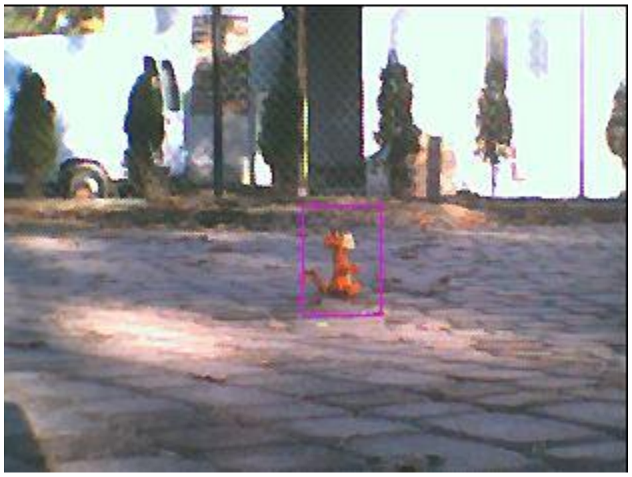
Samples have best accuracy of a ratio of at least 5:1 negative to positive images. Positive samples are best with a confined amount of DETAILED images rather than an abundance of instances.

Negative images should be the entire background of the frame WITHOUT the object being looked for in it.

Smaller sample objects (i.e. < 80x80), best results are taken from cropping out that object in an even square. This way you can get around using the **ObjectMarker** tool which is very time consuming. For example the object itself could be 30x42 in a 1920x1200 photo, it's best to crop out a 50x50 area around that small object. A universal size for all the cropped positive images works best so that the size will include ALL the object samples. Because at different angles in the video the object may appear bigger than other instances. You need to find the biggest instance of it and crop around that and then apply this size +/- 10 pixels to all other instances.

If the object is larger than 100x100, (now this threshold is not definite, it just happened to work in my case, only testing will tell for your object detection), if it's larger than that 100x100 I found it best to simple save the whole frame, and then manually crop out the object using the object marker tool.

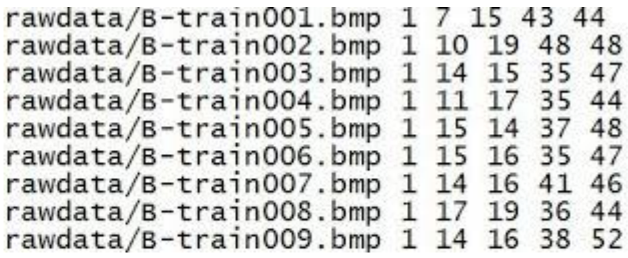
Place all the positive images (images with object) inside the folder **"\temp\positive\rawdata**".
Now that you have placed all the positives, execute the "**objectmarker.exe**". It will be located in the positive folder. Upon executing it, it will show the image, and you need to mark the area containing the object and you get something like this:



(Picture by: Amit Arup Nayak)

Then press "Space Bar". It will feed the coordinates of the bounding rectangle into a text file "info.txt". Then Press "Enter" to go to next image, and continue similarly till u are done with all the images. Note for some images you draw on, it may be off center where you start to draw may draw the box in another area separate from where you clicked. Don't worry just move your mouse to line it up, even if where you click and draw is far away from the object, as long as the actual color drawn box is around the object you're good.

Now once it's done you can see the "info .txt" now in your positive folder. Please note that if you do **objectMarker** multiple times each time it will overwrite the info.txt. This is why before I told you to create the separate info.dat file and once the i**nfo.txt** is created, open it copy its contents, and paste them into the **info.dat** file. This way if you do **objectMarker** again or accidently do it, it will overwrite the **info.txt** but your **.dat** version will be untouched and **THIS** version is what is being trained, **NOT** the **.txt** and inside it will look something like this:



```
rawdata/B-train001.bmp 1 7 15 43 44
rawdata/B-train002.bmp 1 10 19 48 48
rawdata/B-train003.bmp 1 14 15 35 47
rawdata/B-train004.bmp 1 11 17 35 44
rawdata/B-train005.bmp 1 15 14 37 48
rawdata/B-train006.bmp 1 15 16 35 47
rawdata/B-train007.bmp 1 14 16 41 46
rawdata/B-train008.bmp 1 17 19 36 44
rawdata/B-train009.bmp 1 14 16 38 52
```

(Picture by: Amit Arup Nayak)

Now for the smaller positives cropped out you will need to manually create the info.dat file of positives. The easiest way to do this is just run the ObjectMarker on these images but do not waste your time drawing around the small borders, just hit enter repeatedly through till the end. Then it will create the info.txt file with as above, except it will look like this:

```
rawdata/B-train001.bmp 0 0 0 0 0
rawdata/B-train002.bmp 0 0 0 0 0
rawdata/B-train003.bmp 0 0 0 0 0
rawdata/B-train004.bmp 0 0 0 0 0
```

Now all you need to do is change the first 0 to a 1, then the next 2 zeros are the coordinates, those you leave at (0,0) meaning the drawing will start from the top left corner of the image. And the last 2 digits are the universal size chosen. For example, if your cropped image of the object is 40x50, then your last two digits will be 40 and 50 respectively.

```
rawdata/B-train001.bmp 1 0 0 40 50
rawdata/B-train002.bmp 1 0 0 40 50
rawdata/B-train003.bmp 1 0 0 40 50
rawdata/B-train004.bmp 1 0 0 40 50
….etc
```

Once you have done this then proceed to copy its content and paste it into the info.dat file.

Now for the negatives; your negative folder at the moment should only a few items. The **create_list.bat** file, 2 images which you can delete, the **infofile.txt**, and the **sample_list.txt** file. The only file you need is the create_list.bat, delete everything else. Now after it's just that file, create a new txt file called "**negatives.txt**" keep the txt file blank, just change the extension to **.dat** instead of **.txt** and then open up the **create_list.bat** file with notepad and change the contents to something like this:

```
dir /b *.jpg >negatives.dat
```

Save the file and close. This takes all the negative images(.jpg)s in the negatives folder and puts them ALL into the negatives.dat file when you run the create_list.bat file. For the negatives I used .jpg format to conserve space. The positives use .bmp to work with ObjectMarker but .bmp files are significantly larger file size than .jpg, if you use both for positive images and negative images, your computer might not have enough memory to run the training. Use jpgs for your negatives to prevent this from happening.

Now that you have the negatives folder setup simply export ALL your negative images (where the object isn't located) into the negatives folder. To get better results you may also want to edit the frames where the object is found and simply “cover up” the object and blend it into the background. This way you have the exact frame where your object IS located, for your negative samples but WITHOUT the object. This prevents the program from picking up other objects similar to yours in the same frame where your positive object is located.

****If you have more than one objects to be detected you can simply do this by adding the exact same format of above.****

The only difference is that in the temp folder, you will then have Data2 folder, Negative2 folder, and Positive2 folders. All the folders should have the same subfolders as the first instance, the only difference inside the folders is in the data folder. The samples.vec has to be changed to samples2.vec, samples3.vec and create_list2.bat, negatives2.dat and so on. Remember to change the content in create_list2.bat to something like:

```
dir /b *.jpg >negatives2.dat
```

And for the command prompt in the text file in step 2 simply copy and paste its contents into a new .bat file called Haartraining2, for example, and then edit the folders and files accordingly. So your second training .bat file might look something like this:

```
cd C:\WhereYouInstalled \opencv\opencv-2.4.3\win32\bin

opencv_createsamples.exe -info C:\someLocation\Project1\temp\positive2\info.dat -vec
C:\someLocation\Project1\temp\data\samples2.vec -bg
C:\someLocation\Project1\temp\negative2 \negatives2.dat -num 400

opencv_traincascade.exe -data C:\someLocation\Project1\temp\data2\cascade -vec
C:\someLocation \utilities\Project1\temp\data2\samples2.vec -bg
C:\someLocation\Project1\temp\negative2\ negatives2.dat -numPos 400 -numNeg 1477 -
numStages 14  -minHitRate 0.999

pause...
```

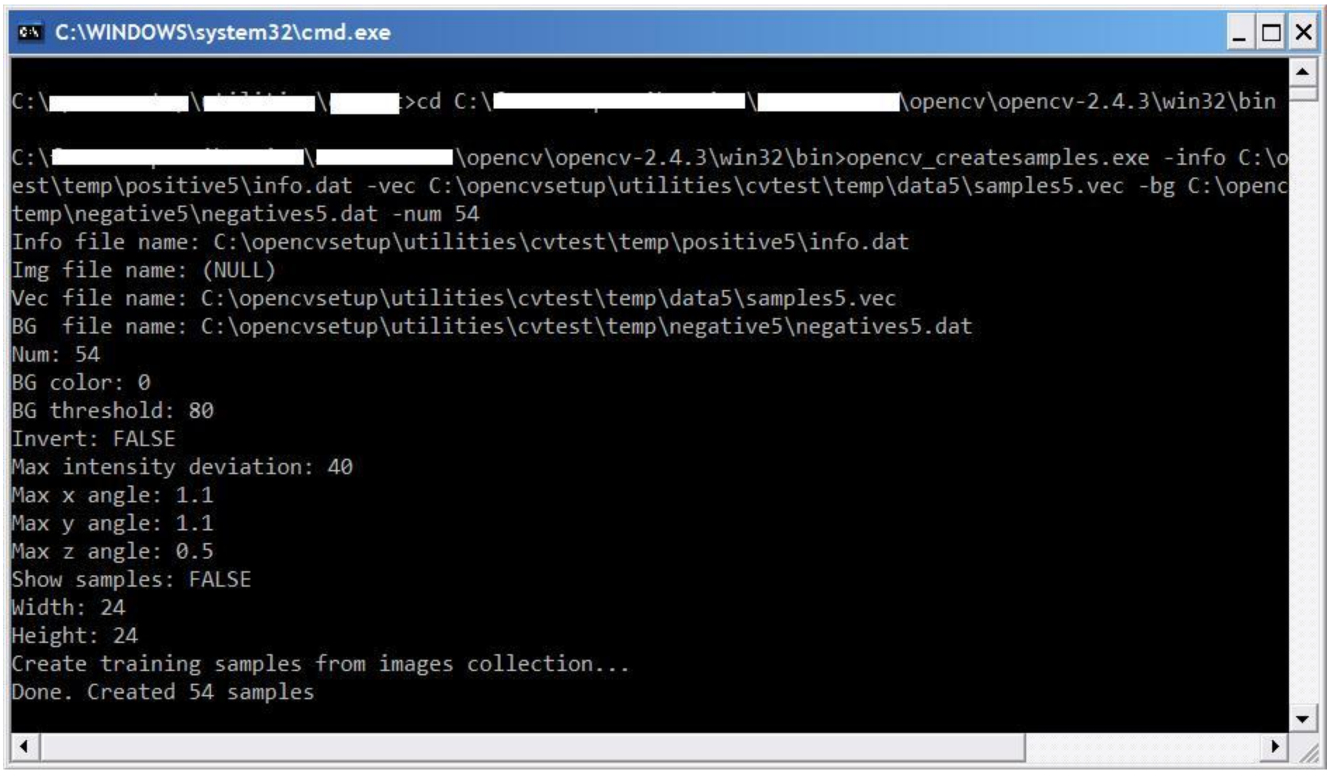
Step 4: The Sampling and Training

Now that steps 1-3 are completed it is time to actually run the training. For this open the directory where your haartraining.bat files, that we created in step 2, are located. Upon running the bat file the command prompt window will open and start to run the commands.

There are two common types of errors when this is being run. First one happens when the numbers in the haartraining.bat files entered to not match. For example for the **opencv_createsamples.exe** portion you put **-num 400** and then in the **opencv_traincascade.exe** portion you put in **-numPos 410**, you will get a mismatch error, make sure they are the same value.

The next error occurs for both positives and negatives when the value you input into the .bat file does not match the actual number of images listed in the info.dat (for positives) or negatives.dat(for negatives). If these lists have 54 lines, aka 54 images, and you enter 55 into the haartraining.bat file for –num, -numPos, or –numNeg, an error will occur.

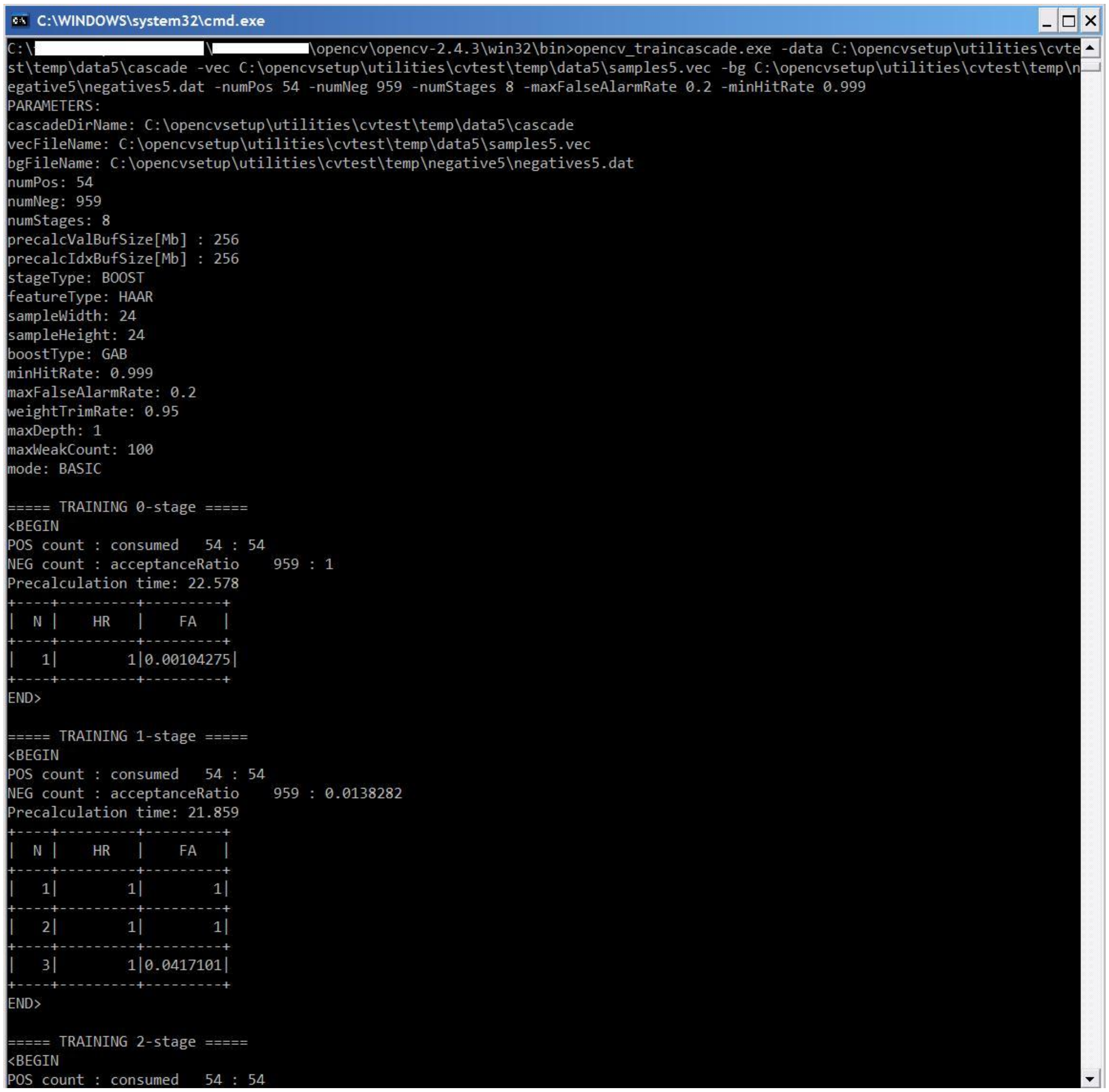
Here is what the createsamples portion will look like:



C:\WINDOWS\system32\cmd.exe

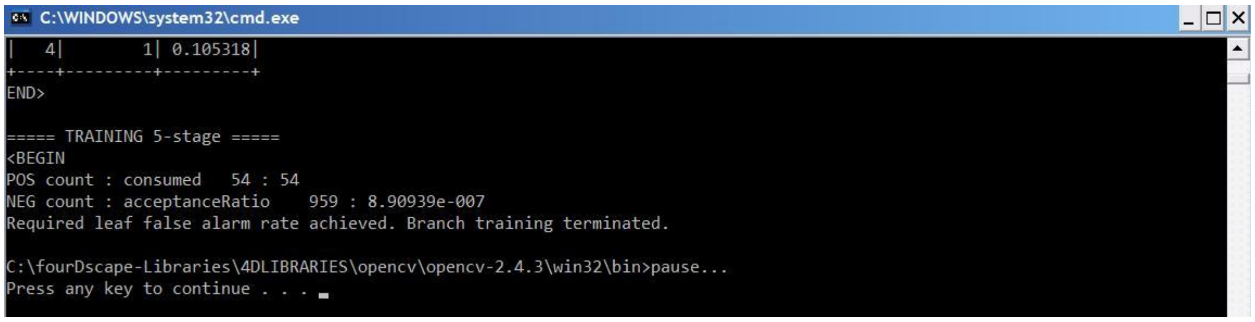
```
C:\[redacted]\[redacted]\[redacted]>cd C:\[redacted]\[redacted]\opencv\opencv-2.4.3\win32\bin

C:\[redacted]\[redacted]\opencv\opencv-2.4.3\win32\bin>opencv_createsamples.exe -info C:\o
est\temp\positive5\info.dat -vec C:\opencvsetup\utilities\cvtest\temp\data5\samples5.vec -bg C:\openc
temp\negative5\negatives5.dat -num 54
Info file name: C:\opencvsetup\utilities\cvtest\temp\positive5\info.dat
Img file name: (NULL)
Vec file name: C:\opencvsetup\utilities\cvtest\temp\data5\samples5.vec
BG  file name: C:\opencvsetup\utilities\cvtest\temp\negative5\negatives5.dat
Num: 54
BG color: 0
BG threshold: 80
Invert: FALSE
Max intensity deviation: 40
Max x angle: 1.1
Max y angle: 1.1
Max z angle: 0.5
Show samples: FALSE
Width: 24
Height: 24
Create training samples from images collection...
Done. Created 54 samples
```

Once that's complete it will continue to run into the traincascade portion which looks like this:



```
C:\[redacted]\[redacted]\opencv\opencv-2.4.3\win32\bin>opencv_traincascade.exe -data C:\opencvsetup\utilities\cvte
st\temp\data5\cascade -vec C:\opencvsetup\utilities\cvtest\temp\data5\samples5.vec -bg C:\opencvsetup\utilities\cvtest\temp\n
egative5\negatives5.dat -numPos 54 -numNeg 959 -numStages 8 -maxFalseAlarmRate 0.2 -minHitRate 0.999
PARAMETERS:
cascadeDirName: C:\opencvsetup\utilities\cvtest\temp\data5\cascade
vecFileName: C:\opencvsetup\utilities\cvtest\temp\data5\samples5.vec
bgFileName: C:\opencvsetup\utilities\cvtest\temp\negative5\negatives5.dat
numPos: 54
numNeg: 959
numStages: 8
precalcValBufSize[Mb] : 256
precalcIdxBufSize[Mb] : 256
stageType: BOOST
featureType: HAAR
sampleWidth: 24
sampleHeight: 24
boostType: GAB
minHitRate: 0.999
maxFalseAlarmRate: 0.2
weightTrimRate: 0.95
maxDepth: 1
maxWeakCount: 100
mode: BASIC

===== TRAINING 0-stage =====
<BEGIN
POS count : consumed   54 : 54
NEG count : acceptanceRatio    959 : 1
Precalculation time: 22.578
+----+---------+---------+
|  N |    HR   |    FA   |
+----+---------+---------+
|   1|        1|0.00104275|
+----+---------+---------+
END>

===== TRAINING 1-stage =====
<BEGIN
POS count : consumed   54 : 54
NEG count : acceptanceRatio    959 : 0.0138282
Precalculation time: 21.859
+----+---------+---------+
|  N |    HR   |    FA   |
+----+---------+---------+
|   1|        1|        1|
+----+---------+---------+
|   2|        1|        1|
+----+---------+---------+
|   3|        1|0.0417101|
+----+---------+---------+
END>

===== TRAINING 2-stage =====
<BEGIN
POS count : consumed   54 : 54
```

After this is done it will finish and pause and it will say something like this…

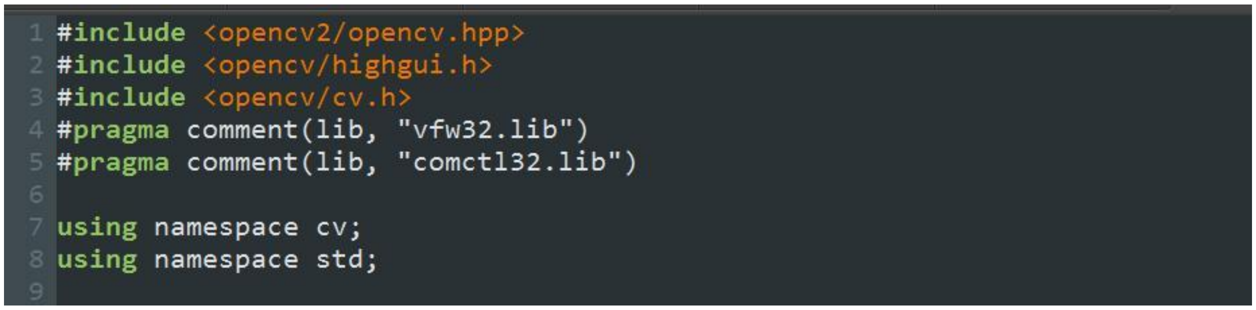


```
|   4|        1| 0.105318|
+----+---------+---------+
END>

===== TRAINING 5-stage =====
<BEGIN
POS count : consumed   54 : 54
NEG count : acceptanceRatio    959 : 8.90939e-007
Required leaf false alarm rate achieved. Branch training terminated.

C:\fourDscape-Libraries\4DLIBRARIES\opencv\opencv-2.4.3\win32\bin>pause...
Press any key to continue . . .
```

Step 5: Locating the XML file and creating your program

Once the training is complete a series of files will be exported in to your **temp/data/cascade** folder. In there you will see a **cascade.xml** file. This is the file that will be called into your openCV detect object program and the data in that xml file will map out the matches to be drawn. I will show you the basics to setup the code and the algorithm but you will have to write your code on your own.

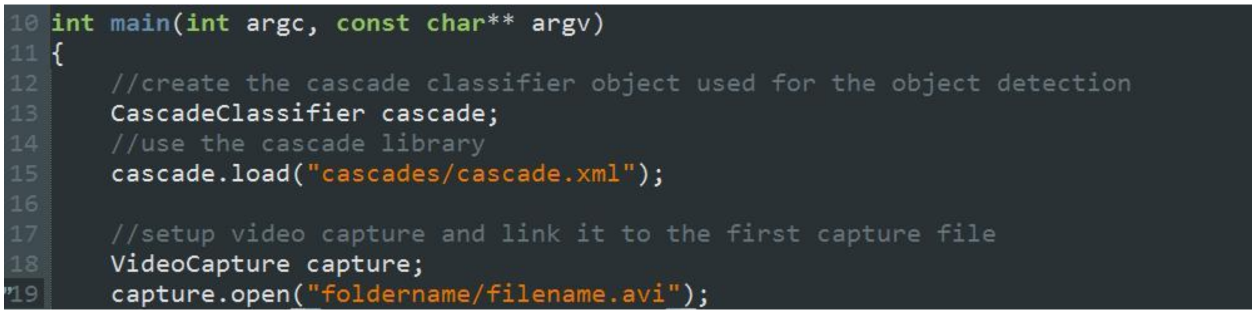
For starting out with the code, C++, for example, you first have to setup the libraries and includes in order to make sure you can use certain commands of the cascade and cvCapture classes.



```
#include <opencv2/opencv.hpp>
#include <opencv/highgui.h>
#include <opencv/cv.h>
#pragma comment(lib, "vfw32.lib")
#pragma comment(lib, "comctl32.lib")

using namespace cv;
using namespace std;

```

It was tricky setting up because my compiler kept giving me include mismatch errors which is why you see the non-conventional addition of the #pragma comment(lib, "_ .lib") there. You may not need this in your code, but for me, in order to get the VideoCapture class to work and compile I needed those extra includes there.

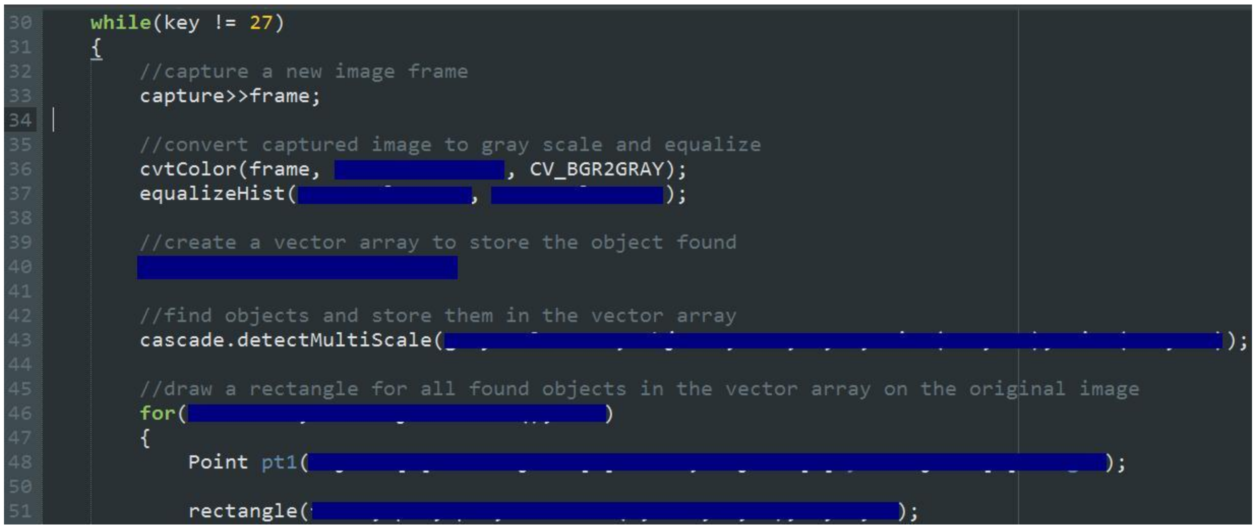
Now for the program's main:



```
int main(int argc, const char** argv)
{
    //create the cascade classifier object used for the object detection
    CascadeClassifier cascade;
    //use the cascade library
    cascade.load("cascades/cascade.xml");

    //setup video capture and link it to the first capture file
    VideoCapture capture;
    capture.open("foldername/filename.avi");
```

The CascadeClassifier is the class you will be using to detect the rectangles from the XML file that then will be drawn in your video thus detecting objects. I also set up a separate cascade folder beside the temp folder to contain all the cascades for all of my objects, it makes it more organized.

Now for the algorithm, you need a while loop to go through each frame of the video and then a for loop to take in each frame and run the cascade detection on that frame, jump out of the while loop, then display the window showing the detection:



```
while(key != 27)
{
    //capture a new image frame
    capture>>frame;

    //convert captured image to gray scale and equalize
    cvtColor(frame, [redacted], CV_BGR2GRAY);
    equalizeHist([redacted], [redacted]);

    //create a vector array to store the object found
    [redacted]

    //find objects and store them in the vector array
    cascade.detectMultiScale([redacted]);

    //draw a rectangle for all found objects in the vector array on the original image
    for([redacted])
    {
        Point pt1([redacted]);

        rectangle([redacted]);
```

For more help on the code, algorithm, and other questions here are some useful sources to visit:

http://opencv.willowgarage.com/documentation/object_detection.html

http://docs.opencv.org/trunk/doc/user_guide/ug_traincascade.html

http://stackoverflow.com/questions/16058080/how-to-train-cascade-properly

Step 6: Testing

This training is VERY temperamental and the program should be updated by openCV to be more efficient. Some flaws consist of long lengths, and the inefficient need to retrain when just adding in another negative sample. Also adding in these negative samples is very unstable. For example one program, one video, it finds all the positive matches of a car, but then finds one false positive of a dumpster. As a result you want to crop out that dumpster image and make sure it is in with the negative samples to help differentiate it from the car. However, doing this can cause the opposite and radical changes. Just one additional negative image can cause the program to change entirely and now it won't pick up all the positives it had before, or now it'll pickup even MORE negatives. A good amount of the time it works as intended but in many random circumstances it can alter everything and negatively.

maxFalseAlarmRate is best 0.5 or lower any higher will cause too many false positives and if too many false positives to start, reduce this rate. Do not go too low or else it will also over-train and not find the positives it should. Same goes for the number of stages the higher num of stages, the more accurate, but it increases the time taken to train and it's possible to over-train which can result in the program not finding the positive matches it's meant to.

Another way to reduce the amount of false positives solely in the code, is to reduce the threshold of the size of rectangles drawn, and/or increase the min number of neighbors parameter in the cascade.detectMultiScale() function. The neighbor parameter can be confusing, basically the higher this value is, the less overall boxes will be drawn(more space between detections), so if there are a lot of false positives, increasing this value can void a good portion of those while leaving the positive matches untouched. Overall this value by increasing it will produce less boxes drawn on your video program, if you have a good amount of positive matches, increasing this number may be enough to reduce the false negatives shown, leaving a good enough amount of positive detections. I advise trying this FIRST before going back to the training and adding in more negatives or editing your positives to be more accurate, because it takes a lot more time.

This training results in large amount of waiting/downtime and much trial and error getting accurate results. The downtime is best used cropping other objects for detection or more negative samples to be added. Here is where your trial and error begins…