How-to build a cascade of boosted classifiers based on Haar-like features

Introduction

This document describes how to train a classifier for object detection. OpenCV comes already with a trained classifier for frontal face detection. In order to show that it also works for a variety of objects, not only for faces, an exemplary training process with bowl is shown. The system used here is a *Win32* machine with *OpenCV beta 3.1* default installation into "`C:\Program Files\OpenCV`" as <*OpenCV-directory*>.

The utilities used here should be in <*OpenCV-directory*>*\bin*. If this is not the case you have to compile them first by using <*OpenCV-directory*>*\apps\HaarTraining\make\full.dsw*. In all compilations select the "release" version as a current configuration in order to avoid creating the debug version (library and program has "d" appended).
In case some files cannot be found, check the paths of the "include files" and "library files" in your development evironment or compile the needed libraries.

In case the steps mentioned above run properly, you should look for any of the missing file(s) in the OpenCV Yahoo!-Group.
When you try to compile one of the most common problems is that "_cvcommon.h" cannot be found because it has simply been forgotten in the distribution package. You can download it from some message attachments of the OpenCV Yahoo!-Group (i.e. http://groups.yahoo.com/group/OpenCV/message/11615 or http://groups.yahoo.com/group/OpenCV/message/12353)

After the orginal code has compiled successfully, you should remove the sample size limitation in the function *icvCreateIntHaarFeatures()* is to be found in "cvhaartraining.cpp". Simply replace s0=36, s1=12, s2=18 and s3=24 with s0=s1=s2=s3=0 and rebuild.

If you are lucky enough to have a multi-processor system or Pentium 4 with "Hyper-Threading" for training, you should use the multi-processor feature in cvhaartraining. Thus the haartraining has to be rebuilt with "OpenMP" enabled. Due to the fact that the system used here doesn't have any of the above mentioned features, we could not experience use of multi-processor features.

Step 1 - Preparation

Tools:
First of all the utilities "`createsamples.exe`", "`haartraining.exe`" and "`performance.exe`" should be available. In our case we work in "`C:\Temp\`".

Negative Samples:
You should have a lot of images (about 5,000 to 10,000) as negative (background) samples. You can get them from well known ressources, i.e. photo-cds (CorelDraw or MS Office) or public databases.
Due to compression artefacts in some compression levels of JPEG images, a BMP image format should be chosen. The negative (background) resolution used here is equivalent to the one the video camera uses for later image acquisition (384x256 pixels). But this is not a necessary condition.
"`C:\Temp\negatives\`" contains the folders of negative images as well as the info files "`train.txt`" and "`test.txt`".
To work with these images in our utilities, a list of the files is needed. Using Win32 as host OS, you can get the list via command promt. Move to the folder where the info file will be stored and type "`dir /b /s > infofile.txt`". Open "infofile.txt" and make the paths relative to the info file (i.e. find "C:\Temp\negatives\" and replace with "").

Positive Samples:
The advantage of object images taken from real world scenes is that you have different reflections, illuminations and backgrounds. Using IPL the "createsamples" tool can simulate such conditions but generally they train not that well.

You can get real object images from several sources such as public data sets (as done for face training) or you can create them by yourself.
The raw images of the bowls were made manually by statues of live video. Several sites with different floor surfaces and illuminations have been taken to achieve a variety of conditions. Additionally at each site the camera was positioned in several angles and distances around the bowls.

You might want to use "objectmarker.exe" to extract the desired objects out of your raw images, i.e. taken with a camera [Fig.1-1]. You browse through the folder named "rawdata" containing all BMP format images and mark your object(s) with a bounding rectangle. You can add each bounding box to the info file "`info.txt`" which you can use for your training and/or testing.

Notes: While marking the rectangle should be close to your objects border. Take care of the info file before you restart this little tool, otherwise it will overwrite it without asking.

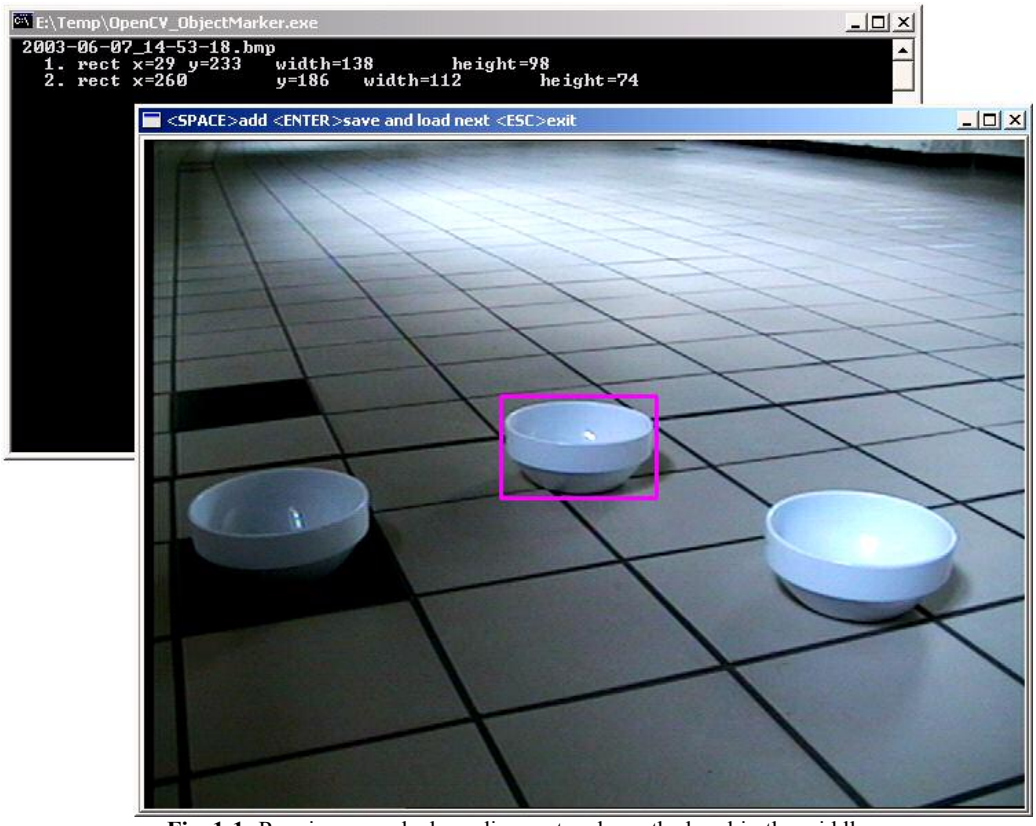

Fig. 1-1: Raw image and a bounding rectangle on the bowl in the middle

If you want to try to train by one example image anyway, you must have "iplPXl.lib" available. Also you have to uncomment "`#define HAVE_IPL`" in file "_cvhaartraining.h" so that *iplWarpPerspectiveQ*() is running correctly for "createsamples.exe". Otherwise when compiled the step of pattern generation from your one sample is not included into the program and only the negatives are put into a vec file.

These positive samples will be stored in a folder "`bowls`" in "`C:\Temp\positives\`" where the info files "`train.txt`" and "`testing.txt`"are located [Fig.1-2].

```
bowls/2003-06-08_17-48-49.bmp 4 209 97 74 43 108 94 71 44 134 172 112 72 249 146 96 68
bowls/2003-06-08_17-48-54.bmp 4 182 87 64 38 259 94 69 41 216 139 90 57 86 153 100 61
bowls/2003-06-08_17-48-59.bmp 3 51 151 106 66 194 146 100 64 342 146 95 55
bowls/2003-06-08_17-49-03.bmp 2 91 164 108 73 245 175 111 79
bowls/2003-06-08_17-49-08.bmp 3 234 163 104 74 140 141 89 58 376 155 98 65
bowls/2003-06-08_17-49-12.bmp 1 322 173 107 65
bowls/2003-06-08_18-07-31.bmp 2 316 169 108 67 98 177 107 70
bowls/2003-06-08_18-07-40.bmp 5 357 126 77 46 250 147 88 56 125 182 104 62 303 218 120 80 422 179 100 67
bowls/2003-06-08_18-07-45.bmp 5 282 110 81 49 151 133 87 55 282 166 108 71 411 141 94 61 115 204 120 83
bowls/2003-06-08_18-07-49.bmp 5 358 125 82 55 430 171 110 77 252 183 115 82 195 133 90 57 49 202 124 81
bowls/2003-06-08_18-07-54.bmp 5 75 164 103 62 237 170 107 74 411 178 115 74 427 128 91 55 273 123 84 52
bowls/2003-06-08_18-08-00.bmp 6 400 135 87 56 306 175 106 74 270 120 75 48 155 110 73 43 171 155 95 65 60 140 88 56
bowls/2003-06-08_18-08-06.bmp 4 469 138 87 65 372 114 75 49 293 95 67 43 305 167 104 68
bowls/2003-06-08_18-08-11.bmp 2 395 148 91 58 187 162 103 59
bowls/2003-06-08_18-08-16.bmp 2 368 174 111 73 130 170 106 70
bowls/2003-06-08_18-08-20.bmp 5 238 127 81 50 144 160 100 58 20 199 115 75 234 229 140 102 346 177 111 70
bowls/2003-06-08_18-08-25.bmp 6 360 121 84 50 489 150 97 64 356 175 107 71 234 135 90 55 91 154 97 55 158 205 123 86
bowls/2003-06-08_18-08-30.bmp 3 478 144 97 66 114 154 107 65 305 151 105 67
bowls/2003-06-08_18-08-34.bmp 4 135 107 77 47 251 112 84 52 38 143 93 62 196 157 102 67
bowls/2003-06-08_18-08-39.bmp 6 227 98 71 44 315 113 79 57 429 131 89 61 93 124 84 56 192 148 99 66 337 177 111 79
bowls/2003-06-08_18-08-44.bmp 2 422 149 95 61 253 178 115 82
bowls/2003-06-08_17-07-46.bmp 3 246 137 98 60 47 206 131 85 315 243 150 101
bowls/2003-06-08_17-07-56.bmp 3 242 113 87 54 81 157 109 78 299 186 126 79
bowls/2003-06-08_17-08-00.bmp 2 196 133 89 57 187 233 137 95
```

Fig. 1-2: Example content of file train.txt (from left to right): BMP file location, number of rectangles, each rectangles x/y coordinate of the upper left corner and width/height from this point x/y.

Step 2 - Sample/Test Creation

Assuming that a sample size of 20x20 is a good choice for most objects, samples are reduced to this size.

Basically there should be four sets of images that you work on:
- a positive sample set representing your object that you want to train
- another positive sample set for testing purpose
- a negative sample set (or so-called backgrounds) for training
- and another negative sample set for testing, too

Note: The testing sets should not contain any images that were used for training.

Of course, defining the number of images in each set depends on how many images you have in total. We use 5500 negatives and split them into 5350 samples for training and 150 samples for testing. As positive samples we have 1350 images from our bowl [Fig.2-1] where 50 are taken for testing.

Fig. 2-1: Examples of how different a simple bowl can appear in a real video image

According to this amount of samples in each set you must specify the number paramters for the training utilities, too.
Once you have all your sets arranged the object images have to be "packed" into a vec-file in the folder "`data`". This can only be done by the createsamples tool, even if you already have a set of object images and don't want to generate artificial object images. The call in our case would be:

```
createsamples.exe -info positives/train.txt -vec data/positives.vec -num 1300 -w 20 –h 20
```

It should be checked if the vec file really contains the desired images. For example when you took the non-IPL version of createsamples to create artificial object images, you will see now that it contains parts of your negative set with no object on it. In our case call following and press <Enter> to scroll through the images in this "highGUI" window:

```
createsamples.exe -vec data/positives.vec -w 20 –h 20
```

Step 3 - Training

Assuming the default values of hitrate (0.995), maxfalsealarm (0.5), weighttrimming (0.95) and boosting type (GAB, "Gentle Ada Boost") are good in most cases, only some parameters will be changed. The extended feature set should be used and the number of stages should be at least 20. If these are too many stages you can abort training at any time. If these are too less stages you can restart the training tool and stages will be added to an existing cascade (starting point is the last completed stage). If the object is symmetric (like the bowl in our example) the parameter "`-nonsym`" is not needed. This saves feature calculation time and memory usage in each stage.

The system you should use for haar training should have a fast processor and enough RAM installed. The machine used for training here has 1.5GB of RAM and a P4 2.4GHz without "Hyper Threading". Using Windows 2000 Advanced Server for better memory management and paging file behaviour, we can use 1,300 MB of RAM for "haartraining.exe". It's important not to use all system RAM because otherwise it will result in a considerable training slow down.

The training of our bowl will be started by the following call:

```
haartraining.exe -data data/cascade -vec data/positives.vec -bg negatives/train.txt
-npos 1300 -nneg 5350 -nstages 30 -mem 1300 -mode ALL -w 20 -h 20
```

While training is running, you already can get a "feeling" whether it will be suitable classifier or something has to be improved in your training set and/or training parameters.
The line starting with "POS:" shows the hitrate in the set of training samples. The next line starting with "NEG" indicates the falsealarm rate. The rate of the positives should be equal or near 1.0 (as it is in "stage 0"). The falsealarm rate should reach at least $5*10^{-6}$ (five zeros) until it is a usable classifier [Fig 3-1]. Otherwise the falsalarm is be too high for real world application. If one of these values gets below zero ["Stage 18", Fig 3-1] there's just an overflow. This means that the falsealarm rate is so low that is can be stopped, no further training would make sense.

```
[…]
STAGE TRAINING TIME: 5037.31
STAGE: 17
POS: 1293 1293 1.000000
NEG: 5000 1308777143 0.000004
BACKGROUND PROCESSING TIME: 26671.78
PRECALCULATION TIME: 108.59

[…]
STAGE TRAINING TIME: 5389.59
STAGE: 18
POS: 1293 1293 1.000000
NEG: 5000 -1465156860 -0.000003
BACKGROUND PROCESSING TIME: 58371.50
PRECALCULATION TIME: 108.56
```

Fig. 3-1: Example of bowl training: In "STAGE 17" five zeros (red coloured number) indicate to possibly become a suitable classifier. In 1.3 billion backgrounds might be 5000 backgrounds in which an object is detected falsly.

Step 4 - Testing

A classifier can be tested with the performance tool mentioned under “utilies” or directly via a “live” test if a detailed report is not necessary.

If you want a report test you must have a different set of positives and negatives as mentioned in “Step 1 – Preparation”.
The info file for this performance utility must not contain a path to the image. Only the filename itself is allowed. Otherwise the *cvSaveImage*() function throws an error because it cannot save the image where the rectangles are drawn into.
To avoid this error you can also use the option “`-ni` and no detection result is saved to an image.

In our example the test of hitrate and falsealarm will be done by calling

```
performance.exe -data data/cascade -info positives/testing/testing.txt -w 20 -h 20 -rs 30
```

It will go through all images and tries detect the object. If one object is found and option “`-ni`” is not specified, it will save the current image

The results of this performance utility should only be seen as one possible result and don’t reflect the possible detection behaviour of your application [Fig 4-1].

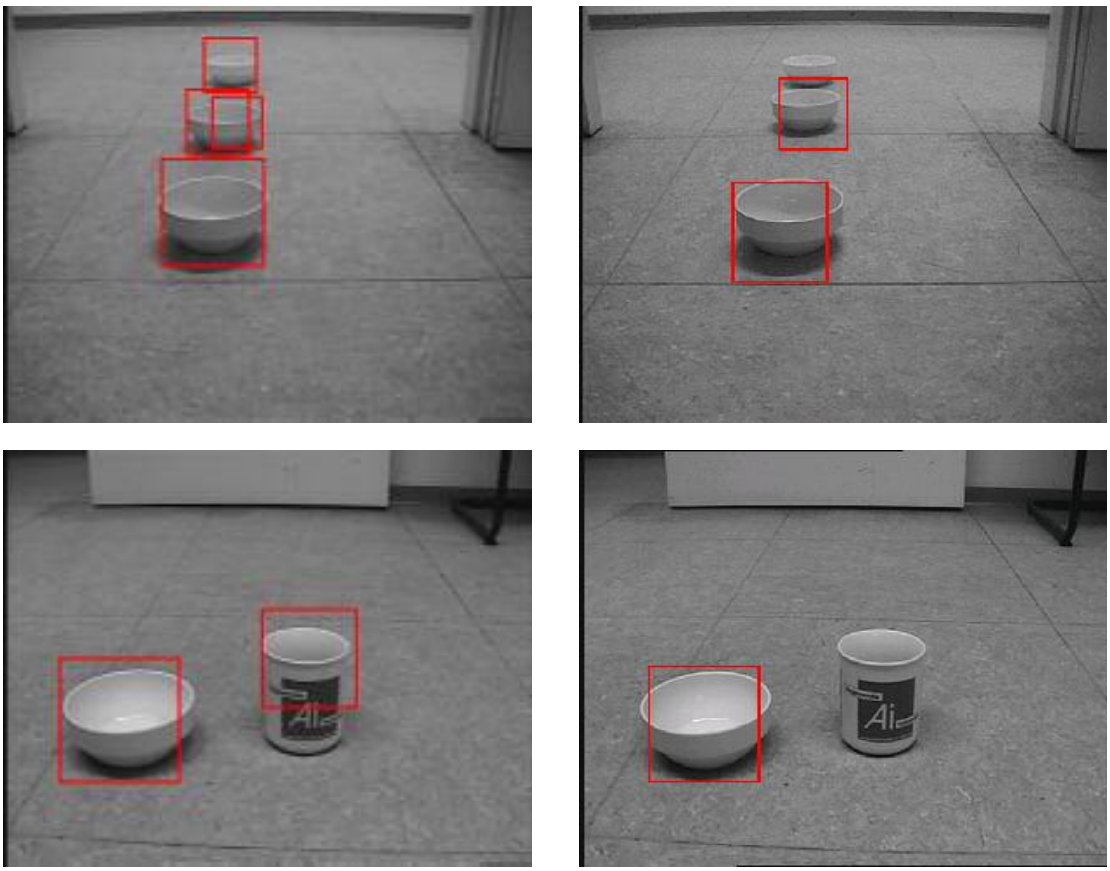

Fig. 4-1: Different detection results for the same classifier base: performance tool (left column) and example application from OpenCV documenation (right column)