OpenCV Object Detection: Theory and Practice

Vadim Pisarevsky
Senior Software Engineer
Intel Corporation, Software and Solutions Group
Agenda

• Quick introduction to object detection
• Basic theory
• History of the approach
• Object detection functions
• Haartraining workflow and tips
Quick Introduction: Top-level view

"objects"  "non-objects"

opencv/apps/haartraining

cvLoad, cvDetectHaarObjects
Basic Theory of Haar-like Object Detectors
Why is it called “Haar-like”?

The features are similar to the basis functions in Haar wavelets:

Pool of features used in OpenCV implementation:

Can be scaled => ~130,000 features for 24x24 window
How are the features computed?

\[ \text{feature}_{i,k} = w_{i,k,1} \times \text{RectSum}_{i,k,\text{orange+white}}(I) + w_{i,k,2} \times \text{RectSum}_{i,\text{white}}(I) \]

Weights are compensated:

\[ w_{i,k,1} \times \text{Area}_{i,k,\text{orange+white}} + w_{i,k,2} \times \text{Area}_{i,k,\text{white}} = 0 \]

\[ w_{i,k,2} = -3 \times w_{i,k,1} \]

\[ w_{i,k,2} = -9 \times w_{i,k,1} \]
Rapid Computation

First, integral images (SAT, RSAT) are computed, then the features can be computed in $O(1)$

\[ SAT(x, y) = \sum_{x' \leq x, y' \leq y} I(x', y') \]

\[ RSAT(x, y) = \sum_{x' \leq x, x' \leq x - |y - y'|} I(x', y') \]

\[ RecSum(x) = SAT(x-1, y-1) + SAT(x+w-1, y+h-1) - SAT(x-1, y+h-1) - SAT(x+w-1, y-1) \]

\[ RecSum(x) = RSAT(x+w, y+w) + RSAT(x-h, y+h) - RSAT(x, y) - RSAT(x+w-h, y+w+h) \]
Weak classifiers

1-split decision tree (stump)

\[ \text{feature}_{i,k} < t_{i,k,1}? \]

\[ \begin{array}{c}
0 \\
\alpha_{i,k} \\
\beta_{i,k}
\end{array} \]

2-split decision tree

\[ \text{feature}_{i,k} < t_{i,k,1}? \]

\[ \begin{array}{c}
0 \\
\alpha_{i,k} \\
\beta_{i,k}
\end{array} \]

\[ \begin{array}{c}
1 \\
\gamma_{i,k}
\end{array} \]

\[ \text{feature}_{i,k} < t_{i,k,2}? \]

\[ \begin{array}{c}
0 \\
\beta_{i,k}
\end{array} \]

\[ \begin{array}{c}
1 \\
\gamma_{i,k}
\end{array} \]

\( t_{i,k} \) and the values at leaves are found using L.B. Brieman CART™ algorithm
Making weak classifiers stronger: DAB etc.

Discrete Adaboost (DAB) (Freund, Schapire, 1996)

1. Given $N$ examples $(x_1, y_1), \ldots, (x_N, y_N)$ where $x \in \mathbb{R}^k, y_i \in \{-1, 1\}$

2. Start with weights $w_i = 1/N$, $i = 1, \ldots, N$.

3. Repeat for $m = 1, \ldots, M$

   (a) Fit the classifier $f_m(x) \in \{-1, 1\}$ using weights $w_i$ on the training data $(x_1, y_1), \ldots, (x_N, y_N)$.

   (b) Compute $\text{err}_m = E_w[1(y \neq f_m(x))]$, $c_m = \log((1 - \text{err}_m)/\text{err}_m)$.

   (c) Set $w_i \leftarrow w_i \cdot \exp(c_m \cdot 1(y_i \neq f_m(x_i)))$, $i = 1, \ldots, N$, and renormalize weights so that $\sum w_i = 1$.

4. Output the classifier $\text{sign} \left[ \sum_{m=1}^M c_m f_m(x) \right]$.

The Boosting Theorem (paraphrased): “As long as weak classifiers are better than random, with sufficiently large $M$ boosted classifier may become as good as you wish”

There are also Real Adaboost (RAB), Logitboost (LB) and Gentle Adaboost (GAB) implemented in OpenCV, and many other variants.
Cascade of Classifiers

Premise:
Size of feature pool (>100000) exceeds what any reasonable classifier can handle

Cascade of classifiers (special kind of decision tree) can outperform a single stage classifier because it can use more features at the same average computational complexity
Cascade Concept

Background removal in stage 4

Background removal in stage 5

Background removal in stage 6

Background removal in stage 3

Background removal in stage 2

Background removal in stage 1
Tuning global thresholds: ROC curves

Classical boosting algorithms give: $F(x) = \text{sign} \sum_{m=0,M-1} c_m f_m(x)$

We replace it with: $F(x) = \text{sign} \left[ \sum_{m=0,M-1} c_m f_m(x) - T \right] =>$

Instead of a fixed classifier we may choose an optimal balance between the hit-rate and false alarms by varying $T$:
Finding objects of different sizes in an image

\[
\text{window\_size} = \text{window\_size}_0 \\
\text{scale} = 1 \\
\text{faces} = \{\} \\
\text{while}\ \text{window\_size} \leq \text{image\_size} \ \text{do} \\
\quad \text{classifier\_cascade} = \text{classifier\_cascade}_0 \text{ scaled by scale} \\
\quad \text{dX} = \text{scale} \\
\quad \text{dY} = \text{scale} \\
\quad \text{for} \ 0 \leq Y < \text{image\_height} - \text{window\_height} \ \text{do} \\
\quad \quad \text{for} \ 0 \leq X < \text{image\_height} - \text{window\_height} \ \text{do} \\
\quad \quad \quad \text{region\_to\_test} = \{0 \leq x < X + \text{window\_width}; \ 0 \leq y < Y + \text{window\_width}\} \\
\quad \quad \quad \text{if} \ \text{classifier\_cascade}(\text{region\_to\_test}) == 1 \ \text{then} \\
\quad \quad \quad \quad \text{faces} = \text{faces} \cup \{\text{region\_to\_test}\} \\
\quad \quad \quad \quad \text{end if} \\
\quad \quad \quad \quad X = X + \text{dX} \\
\quad \quad \end{for} \\
\quad \quad Y = Y + \text{dY} \\
\quad \quad \end{for} \\
\quad \text{scale} = \text{scale} \times C \quad /\!* \text{C} \quad \text{– some constant, e.g. 1.1 or 1.2} */\! \\
\text{end while}
\]
Algorithm Structure

- Different Object size
- Different Locations
- Cascade Stages
  - Weak classifiers
    - Haar feature
      - (2-3 rectangles)
History and the previous works


----- Recent improvements (under consideration) ------

*Floatboost, more efficient feature selection ...*
Detecting Objects with OpenCV
Object detection within OpenCV package

opencv/

  apps/haartraining/  - haartraining application
  apps/haartraining/doc – haartraining user guide
  cv/include/ - data structures and object detection functions.
  cv/src/cvhaar.cpp – detection algorithm source code
  data/haarcascades – pre-trained classifiers (read the license!)
  samples/c/facedetect.c – object detection demo
Object Detection Sample

```c
#include "cv.h"
#include "highgui.h"
int main( int argc, char** argv )
{
    static CvMemStorage* storage = cvCreateMemStorage(0);
    static CvHaarClassifierCascade* cascade = 0;
    if( argc != 3 || strncmp( argv[1], "--cascade=", 10 ) )
        return -1;
    cascade = (CvHaarClassifierCascade*)cvLoad( argv[1] + 10 );
    CvCapture* capture = cvCaptureFromAVI( argv[2] );
    if( !cascade || !capture ) return -1;
    cvNamedWindow( "Video", 1 );
    for(;;) {
        IplImage* frame = cvQueryFrame( capture ), *img;
        if( !frame )
            break;
        img = cvCloneImage(frame); img->origin = 0;
        if( frame->origin ) cvFlip(img, img);
        cvClearMemStorage( &storage );
        CvSeq* faces = cvHaarDetectObjects( img, cascade, storage,
            1.1, 2, CV_HAAR_DO_CANNY_PRUNING, cvSize(20, 20) );
        for( int i = 0; i < (faces ? faces->total : 0); i++ ) {
            CvRect* r = (CvRect*)cvGetSeqElem( faces, i );
            cvRectangle( img, cvPoint(r->x,r->y),
                cvPoint(r->x+r->width,r->y+r->height),
                CV_RGB(255,0,0), 3 );
        }
        cvShowImage( "Video", img );
        cvReleaseImage( &img );
        if( cvWaitKey(10) >= 0 ) break;
    }
    cvReleaseCapture( &capture );
    return 0;
}
```

./facedetect --cascade=opencv/data/haarcascades/haarcascade_frontalface_alt2.xml screetcar.avi
Object detection functions

- CvHaarClassifierCascade* cascade = (CvHaarClassifierCascade*)cvLoad(<classifier_filename.xml>);

- CvSeq* face_rects = cvHaarDetectObjects(image, cascade, memory_storage, scale_factor, min_neighbors, flags, min_size);
  
  - scale – classifier cascade scale factor. typically, 1.1 or 1.2 (10% and 20%, respectively)
  
  - min_size – starting minimum size of objects. By specifying large enough minimum size one can speedup processing a lot!

Let’s look at the other parameters …
min_neighbors: clustering output rectangles

min_neighbors=0

min_neighbors=2
flags

CV_HAAR_DO_CANNY_PRUNING (reject regions with too few or too many edges inside (parameters are tuned for faces!)):

<table>
<thead>
<tr>
<th>Image size</th>
<th>Scale factor</th>
<th>Classifier calls (naïve algorithm)</th>
<th>Classifier calls (with pruning)</th>
</tr>
</thead>
<tbody>
<tr>
<td>160x120</td>
<td>1.2</td>
<td>320000</td>
<td>8000</td>
</tr>
<tr>
<td>320x240</td>
<td>1.2</td>
<td>180000</td>
<td>50000</td>
</tr>
<tr>
<td>640x480</td>
<td>1.2</td>
<td>850000</td>
<td>200000</td>
</tr>
<tr>
<td>160x120</td>
<td>1.1</td>
<td>55000</td>
<td>20000</td>
</tr>
<tr>
<td>320x240</td>
<td>1.1</td>
<td>300000</td>
<td>90000</td>
</tr>
<tr>
<td>640x480</td>
<td>1.1</td>
<td>1500000</td>
<td>390000</td>
</tr>
</tbody>
</table>

So, the pruning techniques decrease number of calls to classifier by 60-80%.

CV_HAAR_FIND_BIGGEST_OBJECT:

Decreases the processing time by factor of 10(!) when you need to find at most 1 face (the biggest one)
Haartraining
Haartraining use:

1. Put all the positive samples in a directory, prepare textual description (info file) in a special format, e.g.:

**Directory with positive samples:**

mydir/positive/
   - face1.jpg
   - face2.jpg
   - my_family.png
   ...

**Info file (e.g. my_info.dat):**

```
mydir/positive/face1.jpg  1  140  100  45  45
mydir/positive/face2.jpg  1  10  20  50  50
mydir/positive/my_family.png  4  100  200  50  50   50  30  25  25  ...
```

1. Run opencv/bin/createsamples.exe:

```
createsamples –vec pos_samples.vec –info my_info.dat –w <width> -h <height>
```

createsamples can also generate a set of positive samples out of a single image. See the reference in opencv/apps/haartraining/doc.
Haartraining use:

1. Now prepare collection of negative samples and another corresponding text file:

**Directory with negative samples:**

mydir/negative/
   my_house.jpg
   beijing_view.jpg
   riverside.png
   ...

Background info file *(e.g. bg.txt)*:

mydir/negative/my_house.jpg
mydir/negative/beijing_view.jpg
mydir/negative/riverside.jpg

1. Now run opencv/bin/haartraining.exe:

   `haartraining --vec pos_samples.vec --bg bg.txt --w <width> -h <height> -data my_classifier_dir --nsplits 1 --nstages 15 --npos N1 --nneg N2 --mem <mem_buf_size>`

See the reference for detailed description of haartraining parameters
Haartraining tips

- Get the fastest machine with a lot of memory (few gig’s), and specify large enough buffer size using –mem option of haartraining.

- Build OpenMP-enabled haartraining (or use precompiled one from OpenCV distribution).

- Haartraining resumes training automatically starting from the last trained stage.

- Positive samples: take care of proper alignment, avoid a lot of background; the smaller is standard deviation => easier for classifier to learn; consider training several classifiers instead of a single almighty one.

- Negative samples: make sure you have enough large-resolution background images (a big percentage of background images is rejected by first few stages => those images can not be used on later stages).

- Choose the optimal object size for haartraining. Play with the other parameters (set of haar features, type of boosting algorithm, number of splits in weak classifier etc.) too. See “Empirical Analysis of Detection Cascades of Boosted Classifiers for Rapid Object Detection” technical report by R. Lienhart et al for empirical study of face detection classifier.