Detecting objects at 100 Hz with rigid templates

Markus Mathias
Radu Timofte
Marco Pedersoli
Shanshan Zhang
Mohamed Omran
Jan Hosang
Piotr Dollar
Luc Van Gool
Tinne Tuytelaars
Bernt Schiele
(and other authors)
1) Who is familiar with the “HOG descriptor”?

2) Who is already familiar with the “ICF detectors”?

3) Who has **needed** to use a detector?

4) Who has **trained** a detector him/herself?

5) Who has **written a paper** about detection?

6) Any burning question? Shoot now!
The next 45 minutes:

How to design a detector running at 100 Hz (CPU only), step by step

https://youtu.be/scDyiDl7D7c
Why care about 100 Hz detection?

- When detection is only a component of a larger system
- When computational power matters (e.g. cell phones !)
- When latency matters

- For certain categories, can be both fast and reach competitive detection quality (e.g. pedestrian, face, and traffic signs)
- Used as class-specific object proposals for convnets
How to reach high detection speed?

I only know three ways:

1) Wait 6 months

2) Make a better fit between algorithm and hardware

3) Design an algorithm that computes less
How to reach high detection quality?

- “A + B features”; boring, and slower than A or B.
- Find the core ingredient, then optimize it.
- Do not hand-design, arbitrary choices are most likely wrong.
Running example: Pedestrian detection
Pedestrian detection is harder than you might think

INRIA training examples
Pedestrian detection is harder than you might think.

Which of these is not a pedestrian? (Caltech test set)
Pedestrian detection is an interesting problem

- Large variance for intra-class appearance
- Strong illumination changes
- Deformations
- Occlusions
- (Interest on small instances)
- No structural variations
  (number of wings in an airplane)
Part I: Quality (first)

Part II: Speed (given a quality target)
Part I
Quality
aka “what makes strong rigid templates?”

Viola & Jones 2001
Dollar et al. 2009, 2014
HOG: Histogram of oriented gradients

- Good features are key
- Gradient orientation is a good feature
- Fully hand-crafted

[Dalal & Triggs 2005]
DPM: Deformable part models

- Good (pre-convnet) generic detector
- Components
- Latent variables estimation
- Many free parameters (despite best effort from authors)
- Are parts good to handle deformation? [Hariharan et al. 2014]
- Components do not share training samples [Zhu et al. 2014]
  [Felzenszwalb et al. 2008; 30 Hz Sadeghi & Forsyth 2014]
In this talk, we focus on rigid templates

- Simpler
- Faster
- Deformations modeled via components
Everything old is new again

Best rigid templates (I know), are Viola&Jones++

[Viola & Jones 2001; Dollar et al. 2009; Benenson et al. 2013, 2015]
Strong detection with (shallow) boosted decision trees
Strong detection with (shallow) boosted decision trees

\[ \text{score} = w_1 \cdot h_1 + \]
Strong detection with (shallow) boosted decision trees

\[ \text{score} = w_1 \cdot h_1 + \]

\[ w_2 \cdot h_2 + \]
Strong detection with (shallow) boosted decision trees

\[ \text{score} = w_1 \cdot h_1 + w_2 \cdot h_2 + \cdots + w_N \cdot h_N \]

[ChnFtrs, Dollar et al. 2009; Viola & Jones 2001] (\sim 4 \text{ Hz on GPU})
Less is more

- Sophisticated features
- Deformable parts
- Deeper architectures
- Non-linear classifiers
- Richer training data
- Geometric priors
- Motion information
Using oriented gradients channels is key

\[
\text{score} = w_1 \cdot h_1 + w_2 \cdot h_2 + \cdots + w_N \cdot h_N
\]
Using oriented gradients channels is key

\[ \text{score} = w_1 \cdot h_1 + w_2 \cdot h_2 + \cdots + w_N \cdot h_N \]
Using oriented gradients channels is key.

\[ \text{score} = w_1 \cdot h_1 + w_2 \cdot h_2 + \ldots + w_N \cdot h_N \]
HOG != ICF

- No clipping
- No fix cells nor blocks
- No cell nor block normalization
- Not only oriented gradient
- More machine learning, fewer hand-crafted choices

VS
HOG is dead, long live the HOG!

Dalal & Triggs
2005

(ICF variants)
Additional ingredients for detection quality
The feature pool matters

Which feature pool use?
All rectangles > all squares > random rectangles
Irregular cells are better than regular ones
One template cannot detect at multiple scales
One template cannot detect at multiple scales
One model for all scales is sub-optimal

Scale 1
(64x128 pixels)

VS

Scales
1,2,4,8
Multi-scales model is better than single scale

- Strong baseline (18.21%)
- +Better feature pool (17.87%)
- +Multi-scales (15.55%)
- HOG (45.18%)
Multi-scales model is better than single scale
We improve over previous best

INRIA dataset

ETH dataset

Caltech dataset

[Benenson et al. 2013]
Pooling $\Leftrightarrow$ Filtering + response reading

Plenty of possible filter banks:

[Zhang et al. 2015]
Filtered channels further push quality
Normalized gradient is better than raw gradient

For datasets with difficult illumination, gradient normalization is crucial (37 → 23 MR, without → with normalization)

\[
\tilde{M}_{i,j} = \frac{M_{i,j}}{\left(\overline{M}_{i,j} + \epsilon\right)}
\]

\(\overline{M}_{i,j}\), gradient magnitude

\(\overline{M}_{i,j}\), 10x10 local average
More and better training data is a game changer.
Additional ingredients for detection quality

1. More and better training data
2. Filtered feature channels (one more layer)
3. Use a large features pool
4. Use more than one model size
5. Do local features normalization
Before global normalization

After global normalization
Only pedestrians?
[Mathias et al. 2014]
For multi-view detection we use components mirrored.

(-100°,-60°)  (-60°,-20°)  (+20°,-20°)  (+20°,+60°)  (+60°,+100°)
Part II
Speed, speed, speed
aka “techniques to make your processor happier”

Viola & Jones 2001
Bourdev & Brandt 2005
Benenson et al. 2012
Sadeghi & Forsyth 2013
Costea et al. 2015
(Fast != Scalable)

[Branch&Rank, Lehmann et al. 2014]
CPU != GPU
How to make detection faster?
What slows down fastHOG?

GPU Time (Total, %)

- Block histogram computation: 64.75%
- Linear SVM evaluation: 11.69%
- Gradient magnitudes and orientations: 16.22%
- Block histogram normalization: 3.44%
- Image downscale: 3.14%
- Others: 0.76%

[Prisacariu and Reid 2009]
How to make features computation faster?
1. Features aggregation
2. Scales handling
3. Cascades
4. Geometric prior
1. Features aggregation
2. Scales handling
3. Cascades
4. Geometric prior

\{ Faster
\{ Fewer
1. Features aggregation
2. Scales handling
3. Cascades
4. Geometric prior

\{ Faster \\
\{ Fewer \}
How to quickly compute the input to the trees?

\[ \text{score} = w_1 \cdot h_1 + w_2 \cdot h_2 + \cdots + w_N \cdot h_N \]
How to quickly compute the input to the trees?
How to quickly compute the input to the trees?
Single scale features are a fix bottleneck

- Use minimal set of feature channels
- Show-off your low-level optimization skills!
- This stage is very CPU and GPU friendly
Integral channels allow to handle any rectangle

Also known as “integral images”, “summed area table”.

Computing table is $O(N)$, but CPU and GPU unfriendly

Reading a feature is $O(1)$
Integral channels allow to handle any rectangle.

Also known as “integral images”, “summed area table”.

Computing table is O(N), but CPU and GPU unfriendly.

Will pay-off when handling scales.
We can go faster is we only use few shapes

Small drop in quality, for high speed gains
(3 square sizes already good enough)

Computing table is $O(N)$, but CPU and GPU friendly

Reading a feature is $O(1)$
(and 4x faster than integral channels)
1. Features aggregation
2. Scales handling
3. Cascades
4. Geometric prior
One template cannot detect at multiple scales
Traditionally, features are computed many times ~50 scales
Traditionally, features are computed many times ~50 scales
We can invert the relation

1 model, 50 image scales

50 models, 1 image scale
Training one model per scale is too expensive

~50 scales
We propose a method to reduce training time $10x$

5 models, 1 image scale

$\approx$

50 models, 1 image scale
Features can be approximated across scales

\[ f(I, s) \approx f(I, 0)e^{-\lambda s} \]

[Dollar et al. 2010]

\[ \sim 5 \text{ scales} \]

\[ \sim 50 \text{ scales} \]
We transfer test time computation to training time

1 model, 5 image scales

5 models, 1 image scale

(3x reduction in features computation)
At runtime, we use as many models as scales

5 models, 1 image scale  ≈  50 models, 1 image scale

[Benenson et al. 2012]
Detecting without resizing improves quality
One can also use hybrids

5 models,
3 image scale
= 15 effective scales

[Costea et al. 2015]
1. Features aggregation  
2. Scales handling  
3. Cascades  
4. Geometric prior

\[ \text{Faster} \quad \text{Fewer} \]
There are many cascade types:
1. Soft cascade
2. Hard cascade
3. Cross-talk cascade
4. NMS cascade
(and many others not covered here)
Cascades are all about detecting dead ends early.

We want to stop wasting computation in doomed candidates as early as possible (CPU, GPU).
Cascades are all about stopping at an early stage

\[
\text{score} = w_1 \cdot h_1 + w_2 \cdot h_2 + \cdots + w_N \cdot h_N
\]
Cascades are all about stopping at an early stage

score = \( w_1 \cdot h_1 + w_2 \cdot h_2 + \cdots + w_N \cdot h_N \)
Cascades are all about stopping at an early stage

\[ \text{score} = w_1 \cdot h_1 + w_2 \cdot h_2 + \cdots + w_N \cdot h_N \]
Cascades are all about stopping at an early stage

\[
\text{score} = w_1 \cdot h_1 + w_2 \cdot h_2 + \cdots + w_N \cdot h_N
\]

Can we stop now?
Soft-cascades are post-hoc training

[Bourdev & Brandt 2005]
**Soft-cascades are post-hoc training**

2k → 30: For a detector with 2k weak classifier using soft-cascade, the average number of evaluated weak classifiers is \(~30\).
Soft-cascades are inconvenient, but:

- Needs a good validation set to adjust properly
- Are conceptually wrong, it asks too much of the later weak classifiers

[Bourdev & Brandt 2005]
Hard-cascades are created during training

Tree 1

-1 -1 +1 -1 -1 -1 +1 +1 -1 -1 -1 +1 -1

Tree N
Hard-cascades are created during training

Tree 1

Tree N
Hard-cascades are created during training

- Number of stages needs to be decided at training time
- The second stage only sees the difficult cases that pass the first stage, thus focuses its resources on the problem of interest
Soft-cascade $\neq$ Hard-cascade

Classifier evaluation traces

- **Cumulative sum**
- **Feature index**

- **Face samples**
- **Non-face samples**
There are many cascade types:

1. **Soft cascade**
2. **Hard cascade**
3. **Cross-talk cascade**
4. **NMS cascade**

(and many others not covered here)

Soft- and Hard-cascades *ignore the neighbor* detections
Neighbor windows score trajectories are very correlated both across space and scale, one detection tells a lot about its neighbors.
Crosstalk cascades exploits neighbors to minimize computation

Crosstalk is designed around a simple logic:

1) Start with coarse grid of active detections (the rest is dormant)
2) If a detection is under-performing, do early stop
3) If a detection is promising, trigger is dormant neighbors
4) If a detection is less promising than neighbors, do early stop

(Neighbors are across space and scales)

Crosstalk detector runs at \(~30\text{Hz}\) on CPU (GPU).

[Dollar et al. 2012]
NMS cascades are simplified Crosstalk cascades

Crosstalk is designed around a simple logic:

1) Start with coarse grid of active detections (the rest is dormant)
2) If a detection is under-performing, do early stop
3) If a detection is promising, trigger is dormant neighbors
4) If a detection is less promising than neighbors, do early stop
NMS cascades are simplified Crosstalk cascades
NMS cascades are simplified Crosstalk cascades

Tree 1

Tree N

Lets do NMS now!

~10x reduction

$S_1$ $S_2$

[Costea et al. 2015]
1. Soft cascade
2. Hard cascade
3. Cross-talk cascade
4. NMS cascade

(and many others not covered here)

Soft + NMS cascade is easy to implement, and provides large speed gains (10x~30x).
1. Features aggregation
2. Scales handling
3. Cascades
4. Geometric prior
One row $\rightarrow$ one person scale
One row → one person scale
One row → one person scale
One row $\rightarrow$ one person scale

When ground plane assumption holds, easy 5x reduction in search space

[Park et al. 2010; Sudowe & Leibe 2011]
But we still slide along each row, can we do better?
But we still slide along each row, can we do better?

Yes, if stereo images are available
Stereo data can be leveraged for fast detection.

Using stereo input, object regions can be estimated directly from the stereo pair, without computing a pixel-wise depth map.

Object regions (stixels) estimation runs at 200Hz on CPU.

30x reduction in search space compared to full image search.

[Benenson et al. 2011, 2012]
1. Features aggregation
2. Scales handling
3. Cascades
4. Geometric prior
<table>
<thead>
<tr>
<th>Aspect</th>
<th>Relative speed</th>
<th>Absolute speed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline detector</td>
<td>1×</td>
<td>1.4 Hz</td>
</tr>
<tr>
<td>+ Scales handling</td>
<td>2×</td>
<td>2.8 Hz</td>
</tr>
<tr>
<td>+ Soft-cascade</td>
<td>20×</td>
<td>50 Hz</td>
</tr>
<tr>
<td>+ Geometric prior (ground plane)</td>
<td>2×</td>
<td>100 Hz</td>
</tr>
<tr>
<td>+ Geometric prior (stixels)</td>
<td>1.6×</td>
<td>160 Hz</td>
</tr>
<tr>
<td>Final CPU + GPU ICF detector</td>
<td>-</td>
<td>160 Hz</td>
</tr>
</tbody>
</table>

[Benenson et al. 2012]
Fast ICF detectors are implementable and re-implementable

- **VeryFast, Roerei**, [https://bitbucket.org/rodrigob/doppia](https://bitbucket.org/rodrigob/doppia) (GPU)
- **ICF** [http://opencv.org](http://opencv.org) (CPU), also [http://libccv.org](http://libccv.org) (CPU)

- 100 fps results also reached by:
  Costea et al. 2015 CPU, and Runia et al. 2015 GPU
Four lessons to remember

1. The right design of features is key for high quality
2. The right design minimizes hand-crafting
3. Fast features computation is 80% of the effort for high speed detection
4. Proper cascades and search space tuning will do the rest
Rodrigo Benenson
http://rodrigob.github.com