30Hz Object Detection with DPM V5

Amin Sadeghi
David Forsyth

ALL 20 PASCAL templates in 0.03s per frame
Why speed up DPM?

- Worth doing, because DPM is effective, mature and stable
  - yes, I know that there are more accurate CNN’s - but they keep changing!
- Fast DPM V5 opens a door for applications
  - on mobile devices
  - with large datasets
  - with previously implausible numbers of templates
- Constraint: No weird hardware
  - to avoid being locked out of applications
Review

• What needs to be done?
  • Extract HOG features (used to be thought of as fast)
  • Apply templates (used to be thought of as slow)

• So, make “Apply templates” faster
  • by vector quantizing the HOG cells
  • benefit:
    • replacing multiply+add with lookup
  • cost:
    • less accurate template values (not much; doesn’t seem to matter)
    • construct VQ (offline - no issue)
    • find nearest neighbor for HOG

Sadeghi Forsyth NIPS 13
Estimation error from VQ

Computation Time vs. Estimation Error

- Absolute value of error
- Number of principal components
- Number of centers
Most time is going into HOG computation

Table from Sadeghi Forsyth 13

<table>
<thead>
<tr>
<th></th>
<th>HOG features</th>
<th>per image</th>
<th>per (image×category)</th>
<th>per category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original DPM [2]</td>
<td>40ms</td>
<td>0ms</td>
<td>665ms</td>
<td>0ms</td>
</tr>
<tr>
<td>DPM Cascade [7]</td>
<td>40ms</td>
<td>6ms</td>
<td>84ms</td>
<td>3ms</td>
</tr>
<tr>
<td>FFLD [9]</td>
<td>40ms</td>
<td>7ms</td>
<td>91ms</td>
<td>43ms</td>
</tr>
<tr>
<td>Our+rescoring</td>
<td>40ms</td>
<td>76ms</td>
<td>21ms</td>
<td>6ms</td>
</tr>
<tr>
<td>Our-rescoring</td>
<td>40ms</td>
<td>76ms</td>
<td>9ms</td>
<td>6ms</td>
</tr>
</tbody>
</table>

Image feature times | Template work

PASCAL’07 Detection Challenge, Intel Xeon E5-1650 processor, All algorithms using 6 cores
Desirable Features of any Approach

- Can work with legacy templates
  - No retraining required

- Random access to image
  - otherwise we’re locked out of pruning/cascade strategies

- Can trade accuracy vs. speed
  - so application developers can look for a sweet spot

- Anytime property
  - so that plausible/tolerable results are returned whenever interrupted
Strategies to speed up

- Accept some loss of numerical precision
  - quantize cells hierarchically
- Manage Scale carefully
  - More templates, fewer HOG features
- Prioritize
  - Avoid evaluating templates at all locations
  - By “peeking” and hashing
Hierarchical Quantization

- 8-times faster computation, loss of 0.001 in mAP

Original: NN to 256 centers

Current: NN to 16 centers, then NN again to 16 centers dep. on first
Strategies to speed up

• Accept some loss of numerical precision
  • quantize cells hierarchically

• Manage Scale carefully
  • More templates, fewer HOG features

• Prioritize
  • Avoid evaluating templates at all locations
  • By “peeking” and hashing
Pyramid of Templates

Conventional Approach

Proposed Approach

Interpolated Templates

Single Template

Pyramid of Templates

Sparse Pyramid of Features
Strategies to speed up

• Accept some loss of numerical precision
  • quantize cells hierarchically

• Manage Scale carefully
  • More templates, fewer HOG features

• Prioritize
  • Avoid evaluating templates at all locations
  • By “peeking” and hashing
“Peeking” to speed up template evaluation
Object Proposal
Any-time Property

- Prioritize tasks
- Halt when time is up

Right now, this block is split in half, but this might not be optimal?
Speed up vs Accuracy

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>mAP</th>
<th>Time</th>
<th>Authors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original DPM</td>
<td>0.33</td>
<td>13.3</td>
<td>Felzenszwalb et al. ‘10</td>
</tr>
<tr>
<td>DPM Cascade</td>
<td>0.331</td>
<td>1.7</td>
<td>Felzenszwalb et al. ‘10</td>
</tr>
<tr>
<td>FFLD</td>
<td>0.323</td>
<td>1.8</td>
<td>Dubout and Fleuret ‘12</td>
</tr>
<tr>
<td>Sparse Kernel</td>
<td>0.277</td>
<td>7</td>
<td>Vedaldi and Zisserman ‘12</td>
</tr>
<tr>
<td>WTA</td>
<td>0.24</td>
<td>26</td>
<td>Felzenszwalb et al.</td>
</tr>
<tr>
<td>FTVQ</td>
<td>0.331</td>
<td>0.53</td>
<td>Sadeghi, Forsyth ‘13</td>
</tr>
<tr>
<td>Ours</td>
<td>0.261</td>
<td>0.03</td>
<td>Sadeghi, Forsyth ‘14</td>
</tr>
</tbody>
</table>

Time = time to complete the detection of all 20 categories starting at raw image

PASCAL’07 Object detection
Intel Xeon E5-1650 Processor (6-cores)
Typical PR curves
Results

Time = time to complete the detection of all 20 categories starting at raw image
AP against time
Desirable Features of any Approach

- Can work with legacy templates
  - No retraining required

- Random access to image
  - otherwise we’re locked out of pruning/cascade strategies

- Can trade accuracy vs. speed
  - so application developers can look for a sweet spot

- Anytime property
  - so that plausible/tolerable results are returned whenever interrupted
## Desirable Features

<table>
<thead>
<tr>
<th>Year</th>
<th>Algorithm</th>
<th>mAP</th>
<th>Time (s)</th>
<th>Legacy Templates</th>
<th>Random Access</th>
<th>Trade Accuracy vs Speed</th>
<th>Anytime Property</th>
</tr>
</thead>
<tbody>
<tr>
<td>2006</td>
<td>Original DPM</td>
<td>0.33</td>
<td>13.3</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>2010</td>
<td>DPM Cascade</td>
<td>0.331</td>
<td>1.7</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>2012</td>
<td>FFLD</td>
<td>0.323</td>
<td>1.8</td>
<td>Yes</td>
<td>Impossible</td>
<td>Impossible</td>
<td>Impossible</td>
</tr>
<tr>
<td>2012</td>
<td>Sparse Kernel</td>
<td>0.277</td>
<td>7</td>
<td>Impossible</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>2013</td>
<td>WTA</td>
<td>0.24</td>
<td>26</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>2013</td>
<td>FTVQ</td>
<td>0.331</td>
<td>0.53</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>2014</td>
<td>Ours</td>
<td>0.261</td>
<td>0.03</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Future Work

• More systematic resource allocation
  • Autotune:
    • hashing
    • priority values
    • internal parameters
    • to get best behavior for fixed set of templates

• Lazy HOG evaluation
• Apply similar speedups to Conv-net

• What should we do with many templates?
Final comments

- It’s fast, for quite simple reasons
  - Code URL on poster

- All fast codes should:
  - admit legacy templates
  - allow random access
  - allow speed/accuracy tradeoffs
  - be anytime

- As well as fast detection, our code is likely good for:
  - proposal algorithms
  - high precision regimes
  - Mobile-applications