Fast, Accurate Detection of 100k Object Classes on a Single Machine

Tom Dean
Mark Ruzon
Mark Segal
Jonathon Shlens
Sudheendra Vijayanarasimhan
Jay Yagnik

Google Research
(tld, ruzon, segal, shlens, svnaras, jyagnik)@google.com

Max Jaderberg
24 June 2013
Main Ideas

- Use standard Deformable Parts Model (DPM) method.
- Replace dot product of parts filters with hashing.
- No root filter - loss in detection performance.
- But speed of detection does not depend on number of classes - can scale to 100k classes.
Object Detection

Given an image, determine what object categories are present and where?

Challenge: How do we scale it to a large number of categories?
Overview

Object Detection with Deformable Parts Model (DPM)

1. Object and part templates are learned.

2. An image feature pyramid is computed.

3. Part templates are convolved with the feature pyramid.

4. Convolved part responses are grouped geometrically to locate the object.

- Complexity of $O(\#classes \ast \#parts \ast \#locations)$
Recent work considers various ways to reduce object detection complexity

- **Cascade based architectures**
  - reduce number of locations to evaluate [Viola et al. IJCV 2002, Felzenszwalb et al. CVPR 2010, Pedersoli et al. CVPR 2011]

- **Saliency based methods**
  - jumping windows [Vedaldi et al. CVPR 2009]
  - objectness [Alexe et al. CVPR 2010]
  - segmentations [van de Sande CVPR 2011]

- **Learn a shared basis of parts**
  - steerable filters [Pirsiavash et al. CVPR 2012]
  - sparselets [Song et al. CVPR 2012]

- **GPUs** [Song et al. CVPR 2012]

But overall complexity is still $\propto \#classes$
Our Approach

Our Approach: Locality Sensitive Hashing

- Convolutions cost only $O(\#\text{locations})$!
- Detection time independent of number of classes
Hashing

- Input vector $x$ (e.g. flattened HOG window)
- Hash $h(x)$ produced by hash function $h$.
- A Locality-sensitive Hash (LSH) means similar input will give similar hash e.g. if $x_1 \approx x_2$ then $h(x_1) = h(x_2)$. 

$x$: 

$h(x)$:
Hash Table

• Hash the output of training and store reference to them.
• Hash test input and lookup “nearest neighbour”.
• Complexity $O(1)$

\[ x \rightarrow h(x) = 2 \]

<table>
<thead>
<tr>
<th>Hash value</th>
<th>Part</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>dog2</td>
</tr>
<tr>
<td>1</td>
<td>bike4</td>
</tr>
<tr>
<td>2</td>
<td>person1</td>
</tr>
<tr>
<td>3</td>
<td>dog3</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Winner-Take-All (WTA) Hashing

- WTA hashing provides a way to convert arbitrary feature vectors into compact binary codes.
- Preserves the rank correlation - not sensitive to absolute values of each dimension, but implicit ordering of values (ordinal space).
- Distance between WTA hashes approximates the rank correlation.

<table>
<thead>
<tr>
<th>$x$</th>
<th>$10,5,2,6,12,3$</th>
<th>$4,5,10,2,3,1$</th>
<th>$22,12,6,14,26,8$</th>
<th>$11,4,3,7,13,2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>WTA($x$)</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Ordinal measures provide a different meaning of 'similarity' 

- robust to variations in individual filter values

- capture differences not reflected in linear measures
Winner-Take-All (WTA) Hashing

- WTA provides a way to convert arbitrary feature vectors into compact binary codes [Yagnik et al. ICCV 2011]
- Dot product in the binary space approximates rank correlation between the feature vectors

Illustration

<table>
<thead>
<tr>
<th>Dot product</th>
<th>Ordinal dot product</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5 1.2 -1.2 -1.3 0.9</td>
<td>00101110101001000101</td>
</tr>
<tr>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>0.6 -0.5 -0.3 1.1 0.7</td>
<td>00101110101001000101</td>
</tr>
</tbody>
</table>
## WTA Hashing

### WTA code computation

<table>
<thead>
<tr>
<th>0.5</th>
<th>1.2</th>
<th>-1.2</th>
<th>-1.3</th>
<th>0.9</th>
</tr>
</thead>
</table>

MinHash can be shown to be a special case of WTA hashing.
WTA Hashing

WTA code computation

\[ H = 4 \]

- Generate \( H \) permutations
- Select the first \( K \) items
- Select the highest scoring item
- Record its index using \( d \log_2(K) \) bits
- MinHash can be shown to be a special case of WTA hashing
WTA Hashing

WTA code computation

<table>
<thead>
<tr>
<th>0.5</th>
<th>1.2</th>
<th>-1.2</th>
<th>-1.3</th>
<th>0.9</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1.2</td>
<td>-1.3</td>
<td>0.5</td>
<td>1.2</td>
<td>0.9</td>
</tr>
<tr>
<td>1.2</td>
<td>-1.2</td>
<td>0.9</td>
<td>0.5</td>
<td>-1.3</td>
</tr>
<tr>
<td>-1.3</td>
<td>0.9</td>
<td>-1.2</td>
<td>1.2</td>
<td>0.5</td>
</tr>
<tr>
<td>0.5</td>
<td>1.2</td>
<td>-1.3</td>
<td>-1.2</td>
<td>0.9</td>
</tr>
</tbody>
</table>

$H = 4$

$K = 3$

- generate $H$ permutations
- select first $K$ items

MinHash can be shown to be a special case of WTA hashing.
WTA Hashing

WTA code computation

<table>
<thead>
<tr>
<th>0.5</th>
<th>1.2</th>
<th>-1.2</th>
<th>-1.3</th>
<th>0.9</th>
</tr>
</thead>
</table>

-1.2 -1.3 0.5 1.2 0.9

1.2 -1.2 0.9 0.5 -1.3

-1.3 0.9 -1.2 1.2 0.5

0.5 1.2 -1.3 -1.2 0.9

- generate $H$ permutations
- select first $K$ items
- select highest scoring item

$H = 4$

$K = 3$
WTA Hashing

WTA code computation

-1.2 -1.3 0.5 1.2 0.9

-1.2 -1.3 0.5 1.2 0.9

1.2 -1.2 0.9 0.5 -1.3

-1.3 0.9 -1.2 1.2 0.5

0.5 1.2 -1.3 -1.2 0.9

H = 4

K = 3

- generate $H$ permutations
- select first $K$ items
- select highest scoring item
- record its index using $\lceil \log_2(K) \rceil$ bits

as $H \rightarrow \infty$ dot product in WTA space \rightarrow rank correlation
MinHash can be shown to be a special case of WTA hashing
% theta = permutation vector
% e.g. theta = randperm(length(x));
[~, c] = max(x(theta(1:K)));
Locality-sensitive Hashing with WTA

Fast approximate lookup through hash tables

1111 0101 0011
Locality-sensitive Hashing with WTA

Fast approximate lookup through hash tables

P bands of WTA code

1111 0101 0011
Locality-sensitive Hashing with WTA

Fast approximate lookup through hash tables

P bands of WTA code

\[
\begin{array}{c}
1111 \\
0101 \\
0011
\end{array}
\]

\[
\begin{array}{cccc}
0101 & & & \\
0110 & & & \\
1111 & & & \\
\end{array}
\]

P hash tables
Locality-sensitive Hashing with WTA

Fast approximate lookup through hash tables

- number of matches provides lower bound on rank correlation
Locality-sensitive Hashing with WTA

Fast approximate lookup through hash tables

- Number of matches provides lower bound on rank correlation
- Threshold on matches to select the top candidates
WTA hashing has a different computational profile than dot products

**Advantages**
- single WTA hash is very fast
- comparisons only (no floating-point arithmetic)
- allows fast lookup through hash tables

**Disadvantages**
- need many hashes to achieve certainty
- percentage of matches is too noisy to be useful later
- all filters must be the same size
Summary of Approach

Training
- **learn** part filters using latent SVM
- **compute** WTA code of each filter and split into M keys
- **store** index of each filter in M hash tables

Detection
- **compute** WTA for filter-sized windows in image
- **lookup** in hash tables to retrieve matching filters
- **detect** objects using sparse filter scores
Results

Experiments

- Evaluate and compare against state-of-art on benchmark
- Determine how hashing parameters affect accuracy, speed and memory
- Determine how accuracy scales with increasing number of objects
- Train 100,000 object classes and evaluate with humans

Datasets

<table>
<thead>
<tr>
<th>PASCAL VOC 2007</th>
<th>ImageSearch-100k</th>
</tr>
</thead>
<tbody>
<tr>
<td>standard benchmark for object detection</td>
<td>100k Freebase entities of visual type</td>
</tr>
<tr>
<td>20 categories, 5000 training and test images</td>
<td>top 500 image search results</td>
</tr>
<tr>
<td>mixture models of 3 aspect ratios</td>
<td>25 million training and 7 million validation images</td>
</tr>
</tbody>
</table>
Comparison with baseline [Felzenszwalb et al.]

<table>
<thead>
<tr>
<th></th>
<th>arp</th>
<th>bike</th>
<th>bird</th>
<th>boat</th>
<th>btll</th>
<th>bus</th>
<th>car</th>
<th>cat</th>
<th>chr</th>
<th>cow</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ours</td>
<td>0.19</td>
<td>0.48</td>
<td>0.03</td>
<td>0.10</td>
<td>0.16</td>
<td>0.41</td>
<td>0.44</td>
<td>0.09</td>
<td>0.15</td>
<td>0.19</td>
</tr>
<tr>
<td>Base&lt;sup&gt;a&lt;/sup&gt;</td>
<td>0.29</td>
<td>0.55</td>
<td>0.01</td>
<td>0.13</td>
<td>0.26</td>
<td>0.39</td>
<td>0.46</td>
<td>0.16</td>
<td>0.16</td>
<td>0.17</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>tbl</th>
<th>dog</th>
<th>hrs</th>
<th>mbke</th>
<th>prsn</th>
<th>plnt</th>
<th>shp</th>
<th>sofa</th>
<th>trn</th>
<th>tv</th>
<th>Mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ours</td>
<td>0.23</td>
<td>0.10</td>
<td>0.52</td>
<td>0.34</td>
<td>0.20</td>
<td>0.10</td>
<td>0.16</td>
<td>0.28</td>
<td>0.34</td>
<td>0.34</td>
<td>0.24</td>
</tr>
<tr>
<td>Base&lt;sup&gt;a&lt;/sup&gt;</td>
<td>0.25</td>
<td>0.05</td>
<td>0.44</td>
<td>0.38</td>
<td>0.35</td>
<td>0.09</td>
<td>0.17</td>
<td>0.22</td>
<td>0.34</td>
<td>0.39</td>
<td>0.26</td>
</tr>
</tbody>
</table>

<sup>a</sup>Felzenszwalb et al PAMI 2010

- no bounding box prediction or context rescoring
- no root filter
- 8-bit hash keys with $K = 4$, $W = 4$, $M = 3000$ hash tables
- absence of a root filter affects harder (low precision) examples
- weaker classes do not have distinctive parts
Results - Accuracy vs. Memory

- Memory is a function of the number of hash tables.
- 5 KB per filter achieves 95% of the best accuracy.
- 100,000 classes with 10 filters each could be stored in 5 GB.
Results - Accuracy vs. Time

- baseline requires 100 secs for 20 classes
- at 5 secs we obtain 90% accuracy with a speed-up of 20x
- comparable to cascade based approaches
Test accuracy with increasing number of distractor classes competing for computational budget

- exhaustive search of 1000 hashing model parameters
- accuracy gracefully decreases with more classes for a fixed set of parameters
Results - 100k object classes

- 100k Freebase entities of visual type
- 500 images per object from image search
- 80% for training and 20% for validation
- Bootstrapping
  - Initialize using easy images
  - Detect and replace in subsequent iterations
- Training: 24 hrs with 2000 machines

<table>
<thead>
<tr>
<th>time</th>
<th>t = 8</th>
<th>t = 28</th>
<th>t = 76</th>
</tr>
</thead>
<tbody>
<tr>
<td>mAP</td>
<td>0.07</td>
<td>0.11</td>
<td>0.16</td>
</tr>
<tr>
<td>Speed-up</td>
<td>62,500×</td>
<td>17,857×</td>
<td>6,580×</td>
</tr>
</tbody>
</table>

Summary of mAP over the 100,000 objects for three different parameter settings.
### Results - Example detections

<table>
<thead>
<tr>
<th>Ariane</th>
<th>Bandurria</th>
<th>Gherkin</th>
<th>Liberty Bell</th>
<th>Overcoat</th>
<th>Jefferson Memorial</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.png" alt="Image" /></td>
<td><img src="image2.png" alt="Image" /></td>
<td><img src="image3.png" alt="Image" /></td>
<td><img src="image4.png" alt="Image" /></td>
<td><img src="image5.png" alt="Image" /></td>
<td><img src="image6.png" alt="Image" /></td>
</tr>
<tr>
<td><img src="image7.png" alt="Image" /></td>
<td><img src="image8.png" alt="Image" /></td>
<td><img src="image9.png" alt="Image" /></td>
<td><img src="image10.png" alt="Image" /></td>
<td><img src="image11.png" alt="Image" /></td>
<td><img src="image12.png" alt="Image" /></td>
</tr>
<tr>
<td><img src="image13.png" alt="Image" /></td>
<td><img src="image14.png" alt="Image" /></td>
<td><img src="image15.png" alt="Image" /></td>
<td><img src="image16.png" alt="Image" /></td>
<td><img src="image17.png" alt="Image" /></td>
<td><img src="image18.png" alt="Image" /></td>
</tr>
<tr>
<td><img src="image19.png" alt="Image" /></td>
<td><img src="image20.png" alt="Image" /></td>
<td><img src="image21.png" alt="Image" /></td>
<td><img src="image22.png" alt="Image" /></td>
<td><img src="image23.png" alt="Image" /></td>
<td><img src="image24.png" alt="Image" /></td>
</tr>
</tbody>
</table>
Results - 100k human evaluation

Top 50 detections from 545 objects sent to MTurk for human evaluation

- close to 200 out of 545 objects have a precision greater than 50%
Conclusions and Future Work

Key Contributions

- scalable approach to replace convolution with an efficient LSH scheme
- applicable to a variety of recognition methods that use convolution
- demonstrated efficacy of approach by scaling part-based object detection to hundred thousand object classes

Future Work

- learning in WTA hash space to improve accuracy
- scaling convolutional and deep nets with similar approach
Questions