Object detection and algorithms for efficient inference

Hyun Oh Song
CS231M - Mobile computer vision
April 29, 2015
Contents

- Sliding window object detection
- Deformable part models
- Cascade DPM
- Sparselets
- Hashing based
Background: Object Detection

Input

Desired output
Sliding window classification
Evaluating a detector

Test image (previously unseen)
Detections

☐ ‘person’ detector predictions
Compared to ground truth

- 'person' detector predictions
- ground truth 'person' boxes
Evaluation metric = AP

Average Precision (AP)
0% is worst
100% is best

mean AP over classes (mAP)
PASCAL VOC Challenge

Dataset: 22k images, 50k objects, 20 classes

Detect: people, horses, sofas, bicycles, pottedplants, ...
Contents

• Sliding window object detection
• Deformable part models
• Cascade DPM
• Sparselets
• Hashing based
Deformable part models

Fig. 4. The matching process at one scale. Responses from the root and part filters are computed at different resolutions in the feature pyramid. The transformed responses are combined to yield a final score for each root location. We show the responses and transformed responses for the "head" and "right shoulder" parts. Note how the "head" filter is more discriminative. The combined scores clearly show two good hypotheses for the object at this scale.

Felzenszwalb et al., PAMI 2010
Star models

test image  part-based deformable model  detection
Object hypothesis score

\[ \Omega \quad \text{set of } (x, y, \text{scale}) \text{ part locations} \]

\[ m_i(\omega) \quad \text{score of } i\text{-th part at } \omega \in \Omega \]

\[ \Delta \quad \text{set of } (dx, dy) \text{ part displacements} \]

\[ d_i(\delta) \quad \text{cost of moving } i\text{-th part by } \delta \in \Delta \]

\[
\text{score}(\omega, \delta_1, \ldots, \delta_n) = m_0(\omega) + \sum_{i=1}^{n} m_i(a_i(\omega) + \delta_i) - d_i(\delta_i)
\]
Object hypothesis score

\[ \Omega \text{ set of } (x, y, \text{scale}) \text{ part locations} \]

\[ m_i(\omega) \text{ score of } i\text{-th part at } \omega \in \Omega \]

\[ \Delta \text{ set of } (dx, dy) \text{ part displacements} \]

\[ d_i(\delta) \text{ cost of moving } i\text{-th part by } \delta \in \Delta \]

\[ \text{score}(\omega, \delta_1, \ldots, \delta_n) = \]

\[ m_0(\omega) + \sum_{i=1}^{n} m_i(a_i(\omega) + \delta_i) - d_i(\delta_i) \]
Object hypothesis score

\[ \text{score}(\omega, \delta_1, \ldots, \delta_n) = m_0(\omega) + \sum_{i=1}^{n} m_i(a_i(\omega) + \delta_i) - d_i(\delta_i) \]

\[ \Omega \quad \text{set of (x, y, scale) part locations} \]

\[ m_i(\omega) \quad \text{score of i-th part at } \omega \in \Omega \]

\[ \Delta \quad \text{set of (dx, dy) part displacements} \]

\[ d_i(\delta) \quad \text{cost of moving i-th part by } \delta \in \Delta \]
Object hypothesis score

\[ \text{score}(\omega, \delta_1, \ldots, \delta_n) = m_0(\omega) + \sum_{i=1}^{n} m_i(a_i(\omega) + \delta_i) - d_i(\delta_i) \]

\[ \Omega \quad \text{set of (x, y, scale) part locations} \]

\[ m_i(\omega) \quad \text{score of } i\text{-th part at } \omega \in \Omega \]

\[ \Delta \quad \text{set of (dx, dy) part displacements} \]

\[ d_i(\delta) \quad \text{cost of moving } i\text{-th part by } \delta \in \Delta \]
Object hypothesis score

\[ \text{score}(\omega, \delta_1, \ldots, \delta_n) = m_0(\omega) + \sum_{i=1}^{n} m_i(a_i(\omega) + \delta_i) - d_i(\delta_i) \]

\( m_i(\omega) \) score of \( i \)-th part at \( \omega \in \Omega \)

\( \Omega \) set of \((x, y, \text{scale})\) part locations

\( \Delta \) set of \((dx, dy)\) part displacements

\( d_i(\delta) \) cost of moving \( i \)-th part by \( \delta \in \Delta \)

slide credit: Girshick et al

Object hypothesis score
Object hypothesis score

\[ \Omega \quad \text{set of } (x, y, \text{scale}) \text{ part locations} \]

\[ m_i(\omega) \quad \text{score of } i\text{-th part at } \omega \in \Omega \]

\[ \Delta \quad \text{set of } (dx, dy) \text{ part displacements} \]

\[ d_i(\delta) \quad \text{cost of moving } i\text{-th part by } \delta \in \Delta \]

\[
\text{score}(\omega, \delta_1, \ldots, \delta_n) = m_0(\omega) + \sum_{i=1}^{n} m_i(a_i(\omega) + \delta_i) - d_i(\delta_i)
\]

sum over non-root parts
Object hypothesis score

- $\Omega$: set of $(x, y, scale)$ part locations
- $m_i(\omega)$: score of $i$-th part at $\omega \in \Omega$
- $\Delta$: set of $(dx, dy)$ part displacements
- $d_i(\delta)$: cost of moving $i$-th part by $\delta \in \Delta$

$$score(\omega, \delta_1, \ldots, \delta_n) = m_0(\omega) + \sum_{i=1}^{n} m_i(a_i(\omega) + \delta_i) - d_i(\delta_i)$$
Object hypothesis score

\[
\text{score}(\omega, \delta_1, \ldots, \delta_n) = m_0(\omega) + \sum_{i=1}^{n} m_i(a_i(\omega) + \delta_i) - d_i(\delta_i)
\]

- \(\Omega\): set of \((x, y, scale)\) part locations
- \(m_i(\omega)\): score of \(i\)-th part at \(\omega \in \Omega\)
- \(\Delta\): set of \((dx, dy)\) part displacements
- \(d_i(\delta)\): cost of moving \(i\)-th part by \(\delta \in \Delta\)

slide credit: Girshick et al
Object hypothesis score

\[
\text{score}(\omega) = m_0(\omega) + \sum_{i=1}^{n} \text{score}_i(a_i(\omega))
\]

\[
\text{score}_i(\eta) = \max_{\delta_i \in \Delta} (m_i(\eta + \delta_i) - d_i(\delta_i))
\]

Maximize over part displacements
Object hypothesis score

$$\text{score}(\omega) = m_0(\omega) + \sum_{i=1}^{n} \text{score}_i(a_i(\omega))$$

$$\text{score}_i(\eta) = \max_{\delta_i \in \Delta} (m_i(\eta + \delta_i) - d_i(\delta_i))$$

anchor position of $i$-th part
Object hypothesis score

\[ \text{score}(\omega) = m_0(\omega) + \sum_{i=1}^{n} \text{score}_i(a_i(\omega)) \]

\[ \text{score}_i(\eta) = \max_{\delta_i \in \Delta} (m_i(\eta + \delta_i) - d_i(\delta_i)) \]

optimal appearance/displacement tradeoff

Maximize over part displacements
Contents

• Sliding window object detection
• Deformable part models
• Cascade DPM
• Sparselets
Star cascade ingredients

1. A hierarchy of models defined by a part ordering

2. A sequence of thresholds:  
   \[ t = ((t'_1, t_1), \ldots, (t'_n, t_n)) \]

\[ m_0(\omega) \leq t_1 \]

\[ \forall \delta_1 : m_0(\omega) - d_1(a_1(\omega) \oplus \delta_1) \leq t'_1 \]

\[ m_0(\omega) - d_1(a_1(\omega) \oplus \delta_1^*) + m_1(a_1(\omega) \oplus \delta_1^*) \leq t_2 \]

\[ \forall \delta_2 : m_0(\omega) - d_1(a_1(\omega) \oplus \delta_1^*) + m_1(a_1(\omega) \oplus \delta_1^*) - d_2(a_2(\omega) \oplus \delta_2) \leq t'_2 \]

\[ \vdots \]
Star cascade algorithm

test image

object model
+ part ordering
+ thresholds
Star cascade algorithm

HOG pyramid from test image

object model + part ordering + thresholds
Star cascade algorithm

HOG pyramid from test image

object model + part order + thresholds
Star cascade algorithm

filter score tables

Root

\[ m_0(\omega) \]

Part 1

\[ m_1(\omega) \]

Part 2

\[ m_2(\omega) \]

cascade test:

model:

operation:

slide credit: Girshick et al
Star cascade algorithm

<table>
<thead>
<tr>
<th>Part 1</th>
<th>Part 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>(m_0(\omega))</td>
<td>(m_1(\omega))</td>
</tr>
<tr>
<td>(m_2(\omega))</td>
<td></td>
</tr>
</tbody>
</table>

filter score tables

cascade test:

model: [image]

operation: [image]
Star cascade algorithm

filter score tables

Root
\( m_0(\omega) \)

Part 1
\( m_1(\omega) \)

Part 2
\( m_2(\omega) \)

cascade test:

model:

operation: test root locations

slide credit: Girshick et al
Star cascade algorithm

filter score tables

Root

\[ m_0(\omega) \]

Part 1

\[ m_1(\omega) \]

Part 2

\[ m_2(\omega) \]

cascade test: \[ m_0(\omega) \geq t_1 \]

model:

operation: test root locations
result: fail
Star cascade algorithm

<table>
<thead>
<tr>
<th>Root</th>
<th>filter score tables</th>
</tr>
</thead>
<tbody>
<tr>
<td>( m_0(\omega) )</td>
<td>![Table Image]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Part 1</th>
<th>filter score tables</th>
</tr>
</thead>
<tbody>
<tr>
<td>( m_1(\omega) )</td>
<td>![Table Image]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Part 2</th>
<th>filter score tables</th>
</tr>
</thead>
<tbody>
<tr>
<td>( m_2(\omega) )</td>
<td>![Table Image]</td>
</tr>
</tbody>
</table>

cascade test: \( m_0(\omega) \geq t_1 \)

model: ![Model Image]

operation: test root locations

result: fail

slide credit: Girshick et al
Star cascade algorithm

Filter score tables

cascade test: $m_0(\omega) \geq t_1$

model:

operation: test root locations

result: fail
Star cascade algorithm

<table>
<thead>
<tr>
<th></th>
<th>filter score tables</th>
</tr>
</thead>
<tbody>
<tr>
<td>Root</td>
<td></td>
</tr>
<tr>
<td>$m_0(\omega)$</td>
<td></td>
</tr>
<tr>
<td>Part 1</td>
<td></td>
</tr>
<tr>
<td>$m_1(\omega)$</td>
<td></td>
</tr>
<tr>
<td>Part 2</td>
<td></td>
</tr>
<tr>
<td>$m_2(\omega)$</td>
<td></td>
</tr>
</tbody>
</table>

cascade test: $m_0(\omega) \geq t_1$

model: 

test root locations

result: fail
**Star cascade algorithm**

<table>
<thead>
<tr>
<th>Part 1</th>
<th>Part 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>(m_0(\omega))</td>
<td>(m_1(\omega))</td>
</tr>
<tr>
<td>(m_2(\omega))</td>
<td></td>
</tr>
</tbody>
</table>

**filter score tables**

**cascade test:** \(m_0(\omega) \geq t_1\)

**model:**

**operation:** test root locations

**result:** fail
Star cascade algorithm

- **Root**: $m_0(\omega)$
- **Part 1**: $m_1(\omega)$
- **Part 2**: $m_2(\omega)$

**Filter score tables**

**Cascade test**: $m_0(\omega) \geq t_1$

**Model**: Test root locations

**Result**: Pass

*Slide credit: Girshick et al*
Star cascade algorithm

filter score tables

Root
\[ m_0(\omega) \]

Part 1
\[ m_1(\omega) \]

Part 2
\[ m_2(\omega) \]

cascade test: \[ m_0(\omega) - d_1(\delta_1) \geq t'_1 \]

model:

operation: displacement search

slide credit: Girshick et al
Star cascade algorithm

filter score tables

Root
$m_0(\omega)$

Part 1
$m_1(\omega)$

Part 2
$m_2(\omega)$

cascade test: $m_0(\omega) - d_1(\delta_1) \geq t'_1$

model:

operation: displacement search
Star cascade algorithm

Filter score tables

Root
$m_0(\omega)$

Part 1
$m_1(\omega)$

Part 2
$m_2(\omega)$

cascade test: $m_0(\omega) - d_1(\delta_1) \geq t'_1$

model:

operation: displacement search

result: pass
Star cascade algorithm

- **Root**: $m_0(\omega)$
- **Part 1**: $m_1(\omega)$
- **Part 2**: $m_2(\omega)$

**Filter score tables**

**Cascade test**: $m_0(\omega) - d_1(\delta^*_1) + m_1(\omega \oplus \delta^*_1) \geq t_2$

**Model**: test partial score

**Result**: fail
Star cascade algorithm

Filter score tables

Root
\[ m_0(\omega) \]

Part 1
\[ m_1(\omega) \]

Part 2
\[ m_2(\omega) \]

cascade test: \[ m_0(\omega) \geq t_1 \]

model:

operation: test root locations

result: pass

slide credit: Girshick et al.
Star cascade algorithm

filter score tables

Root
\[ m_0(\omega) \]

Part 1
\[ m_1(\omega) \]

Part 2
\[ m_2(\omega) \]

cascade test: \[ m_0(\omega) - d_1(\delta_1) \geq t'_1 \]

model:

operation: displacement search
Star cascade algorithm

filter score tables

Root
$m_0(\omega)$

cached!

Part 1
$m_1(\omega)$

Part 2
$m_2(\omega)$

cascade test: $m_0(\omega) - d_1(\delta_1) \geq t'_1$

model:

operation: displacement search

result: pass
Star cascade algorithm

filter score tables

Root

\[ m_0(\omega) \]

Part 1

\[ m_1(\omega) \]

cascade test: \[ m_0(\omega) - d_1(\delta^*_1) + m_1(\omega \oplus \delta^*_1) \geq t_2 \]

model:

operation: test partial score

result: pass

slide credit: Girshick et al
Star cascade algorithm

filter score tables

Root

\[ m_0(\omega) \]

Part 1

\[ m_1(\omega) \]

cascade test: \[ m_0(\omega) - d_1(\delta_1^*) + m_1(\omega \oplus \delta_1^*) - d_2(\delta_2) \geq t_3 \]

model:

operation: displacement search
Star cascade algorithm

Filter score tables

Root
$m_0(\omega)$

Part 1
$m_1(\omega)$

Part 2
$m_2(\omega)$

cascade test: $m_0(\omega) - d_1(\delta_1^*) + m_1(\omega \oplus \delta_1^*) - d_2(\delta_2) \geq t_3$

model: [image]

operation: displacement search

result: pass
Star cascade algorithm

**Part 1**
$m_1(\omega)$

**Part 2**
$m_2(\omega)$

**Root**
$m_0(\omega)$

---

**filter score tables**

---

**cascade test:**
$m_0(\omega) - d_1(\delta_1^*) + m_1(\omega \oplus \delta_1^*) - d_2(\delta_2^*) + m_2(\omega \oplus \delta_2^*) \geq t_3$

**model:**

**operation:** test partial score

**result:** pass

*slide credit: Girshick et al*
Star cascade algorithm

filter score tables

Root
\[ m_0(\omega) \]

Part 1
\[ m_1(\omega) \]

Part 2
\[ m_2(\omega) \]

cascade test: ...

model:

operation: continue testing remaining parts
Star cascade algorithm

filter score tables

Root
\[ m_0(\omega) \]

Part 1
\[ m_1(\omega) \]

cascade test: all tests passed => detection!

model:

operation: report object hypothesis

slide credit: Girshick et al
## Star cascade algorithm

<table>
<thead>
<tr>
<th>Filter score tables</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td>Root</td>
</tr>
<tr>
<td>$m_0(\omega)$</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Part 1</td>
</tr>
<tr>
<td>$m_1(\omega)$</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Part 2</td>
</tr>
<tr>
<td>$m_2(\omega)$</td>
</tr>
</tbody>
</table>

### Cascade test:
- **Model:**
- **Operation:** continue with root locations...

*Slide credit: Girshick et al.*
Threshold selection

We want *safe and effective* thresholds

- don’t prune many true positives
- but do prune lots of true negatives

slide credit: Girshick et al
PAA threshold

\[ X = \text{IID set of positive examples } \sim D \]

\[ \text{error}(t) = P_{x \sim D}(\text{cascade-score}(t, \omega) \neq \text{score}(\omega)) \]

**Probably Approximately Admissible thresholds**

\[ P(\text{error}(t) > \epsilon) \leq \delta \]

min of partial scores over examples in \( X \)

Theorem: \[ |X| \geq 2n/\epsilon \ln(2n/\delta) \implies (\epsilon, \delta)-\text{PAA thresholds} \]
Example results

**high recall**

PASCAL 2007 comp3 class: motorbike

- Baseline (AP 48.7)
- Cascade (AP 48.9)

23.2x faster
(618ms per/image)

**less recall ⇒ faster**

PASCAL 2007 comp3 class: motorbike

- Baseline (AP 48.7)
- Cascade (AP 41.8)

31.6x faster
(454ms per/image)

slide credit: Girshick et al
Discussion
Contents

• Sliding window object detection
• Deformable part models
• Cascade DPM
• Sparselets
• Hashing based
Generalized Sparselet Models for Real-Time Multiclass Object Recognition

Hyun Oh Song, Ross Girshick, Stefan Zickler, Christopher Geyer, Pedro Felzenszwalb, Trevor Darrell

ECCV12, ICML13, TPAMI14
Goal

- Shared predictive model with sparse activation vectors
- Efficient inference for linear structured output predictors
- Example application: realtime object recognition in CV, faster retrieval in IR, etc.
Related works

• Learning shared low dimensional predictive structure (e.g., Ando and Zhang, JMLR05)
• Shared part models (Steerable part models, Pirsiavash et al)
Deformable part models

Fig. 4. The matching process at one scale. Responses from the root and part filters are computed at different resolutions in the feature pyramid. The transformed responses are combined to yield a final score for each root location. We show the responses and transformed responses for the "head" and "right shoulder" parts. Note how the "head" filter is more discriminative. The combined scores clearly show two good hypotheses for the object at this scale.

Felzenszwalb et al, PAMI 2010
Sparselet review

Set of model filters

\[ \mathcal{W} = \{ w_1, \ldots, w_K \} \]

Set of sparselet filters

\[ \mathcal{S} = \{ s_1, \ldots, s_d \} \]

\[ \begin{align*}
\min_{\alpha_{ij}, s_j} & \sum_{i=1}^{K} \| w_i - \sum_{j=1}^{d} \alpha_{ij} s_j \|_2^2 \\
\text{subject to} & \| \alpha_i \|_0 \leq \epsilon \quad \forall i = 1, \ldots, K \\
& \| s_j \|_2^2 \leq 1 \quad \forall j = 1, \ldots, d
\end{align*} \]
Sparse reconstruction of filter response

\[ \Psi \star w_i \approx \Psi \star \left( \sum_{j=1}^{d} \alpha_{ij} s_j \right) = \sum_{j=1}^{d} \alpha_{ij} (\Psi \star s_j) \]

Cached

Sparsity
Matrix factorization point of view

\[
\begin{bmatrix}
\Psi \ast w_1 \\
\Psi \ast w_2 \\
\vdots \\
\Psi \ast w_K
\end{bmatrix} \approx
\begin{bmatrix}
\alpha_1 \\
\alpha_2 \\
\vdots \\
\alpha_K
\end{bmatrix}
\begin{bmatrix}
\Psi \ast s_1 \\
\Psi \ast s_2 \\
\vdots \\
\Psi \ast s_d
\end{bmatrix}
\]

80 ~ 99 % Sparse
System concept

The paper is structured as follows. In Sec. 3, we provide a brief overview of sparselets [8] and formulate a discriminative sparselet activation training framework. Also, we introduce the notion of generalized sparselets in structured output prediction [8]. The system described here differs from the one in [5], [6], [7], and so we can optimize the objective in a coordinate descent fashion by iterating between updating column estimates the optimal matching projections of the input signal onto the dictionary and so we can optimize the objective in a coordinate descent fashion by iterating between updating column estimates the optimal matching projections of the input signal onto the dictionary.

Formally, a sparselet model is completely defined by a dictionary $W$ and a set of sparse weights $s$. The intermediate representation is obtained as the inner product between the input image and the dictionary:

$$X = Ws$$

where each $w_i$ is a dictionary column and $s_i$ is the corresponding sparse weight.

3.1 Sparse reconstruction of object models

The system described here differs from the one in [13], [14], [15], and (3) modeling and learning with low-rank approximations [3], [16], [17]. None of these methods, however, simultaneously exploit shared interclass information of these methods, however, simultaneously exploit shared interclass information.

3.2 Precomputation and efficient reconstruction

Although the above optimization is NP-hard, although the above optimization is NP-hard, we use the online dictionary learning algorithm from [20] to approximate the convolution response we would have obtained from convolution with the original filters, and by linearity of convolution we can then use the approximate solution. OMP is efficient, and so we can optimize the objective in a coordinate descent fashion by iterating between updating column estimates the optimal matching projections of the input signal onto the dictionary.

$$\text{approximate solution} = OMP(X, W)$$

The $X$ is obtained by inner product between the input image and the dictionary $W$ times $s$.

$$\text{OMP}$$

OMP iteratively estimates the optimal matching projections of the input signal onto the dictionary.

$$\text{OMP}(X, W) = \arg\min_{s} \|X - WS\|_2^2 \text{ subject to } \|s\|_0 \leq \text{fixed threshold}$$

OMP is a greedy algorithm that iteratively selects the dictionary column that makes the largest contribution to the reconstruction error.

$$\text{signal} = \arg\min_{s} \|X - WS\|_2^2 \text{ subject to } \|s\|_0 \leq \text{fixed threshold}$$

OMP is a greedy algorithm that iteratively selects the dictionary column that makes the largest contribution to the reconstruction error.

$$\text{OMP}(X, W) = \arg\min_{s} \|X - WS\|_2^2 \text{ subject to } \|s\|_0 \leq \text{fixed threshold}$$

OMP is a greedy algorithm that iteratively selects the dictionary column that makes the largest contribution to the reconstruction error.

$$\text{OMP}(X, W) = \arg\min_{s} \|X - WS\|_2^2 \text{ subject to } \|s\|_0 \leq \text{fixed threshold}$$

OMP is a greedy algorithm that iteratively selects the dictionary column that makes the largest contribution to the reconstruction error.

$$\text{OMP}(X, W) = \arg\min_{s} \|X - WS\|_2^2 \text{ subject to } \|s\|_0 \leq \text{fixed threshold}$$

OMP is a greedy algorithm that iteratively selects the dictionary column that makes the largest contribution to the reconstruction error.

$$\text{OMP}(X, W) = \arg\min_{s} \|X - WS\|_2^2 \text{ subject to } \|s\|_0 \leq \text{fixed threshold}$$

OMP is a greedy algorithm that iteratively selects the dictionary column that makes the largest contribution to the reconstruction error.

$$\text{OMP}(X, W) = \arg\min_{s} \|X - WS\|_2^2 \text{ subject to } \|s\|_0 \leq \text{fixed threshold}$$

OMP is a greedy algorithm that iteratively selects the dictionary column that makes the largest contribution to the reconstruction error.

$$\text{OMP}(X, W) = \arg\min_{s} \|X - WS\|_2^2 \text{ subject to } \|s\|_0 \leq \text{fixed threshold}$$

OMP is a greedy algorithm that iteratively selects the dictionary column that makes the largest contribution to the reconstruction error.

$$\text{OMP}(X, W) = \arg\min_{s} \|X - WS\|_2^2 \text{ subject to } \|s\|_0 \leq \text{fixed threshold}$$

OMP is a greedy algorithm that iteratively selects the dictionary column that makes the largest contribution to the reconstruction error.

$$\text{OMP}(X, W) = \arg\min_{s} \|X - WS\|_2^2 \text{ subject to } \|s\|_0 \leq \text{fixed threshold}$$

OMP is a greedy algorithm that iteratively selects the dictionary column that makes the largest contribution to the reconstruction error.

$$\text{OMP}(X, W) = \arg\min_{s} \|X - WS\|_2^2 \text{ subject to } \|s\|_0 \leq \text{fixed threshold}$$

OMP is a greedy algorithm that iteratively selects the dictionary column that makes the largest contribution to the reconstruction error.

$$\text{OMP}(X, W) = \arg\min_{s} \|X - WS\|_2^2 \text{ subject to } \|s\|_0 \leq \text{fixed threshold}$$

OMP is a greedy algorithm that iteratively selects the dictionary column that makes the largest contribution to the reconstruction error.
Blocked representation

• Intuition: model weights might be composed of shared building blocks/tiles
Blocked representation

Reconstruction error for all 20 object categories from PASCAL 2007 dataset as sparselet parameters are varied. The precomputation time is fixed in the top figure and the representation space is fixed on the bottom. Object categories are sorted by the reconstruction error by 6 × 6 in the top figure and by 1 × 1 in the bottom figure.

Increase dictionary size

Fix dictionary size
Blocked representation

• Empirically, filter reconstruction error always decreases as we decrease sparselet size (@ fixed computation time)

• However, the space required to store the intermediate representation is proportional to the sparselet dictionary size $|S|$. This means we have computation time VS memory bandwidth tradeoff.
Visualized sparselet blocks on HOG

(Left) Sparselet dictionary of size 128
(Right) Top 16 activated sparselets for PASCAL motorcycle class
Blocked representation

\[ f_w(x) = \arg \max_{k \in \{1, \ldots, K\}} w_k^T x \]

Model parameterization

\[ w_k = (b_{k1}^T, \ldots, b_{kp}^T)^T \]

Data parameterization

\[ x = (c_1^T, \ldots, c_p^T)^T \]

Sparselets approximation of model blocks

\[ b \approx S \alpha = \sum_{i=1}^{d} \alpha_i s_i \]

Sparselets: \( S = [s_1, \ldots, s_d] \)
Sparselet Demo
Demo specifications

• Alienware laptop with NVIDIA GeForce GTX580 with 3GB memory

• Runs all 20 PASCAL category detection @ 5 Hz (frames per second)

• Full specs and quantitative average precision results in Song et al, TPAMI115

• CPU version of the source code available at https://github.com/rksltnl/sparselet-release1
Potential mobile implementation

- NVIDIA Shield supports CUDA with < 2GB memory
- ARM NEON optimizations on CPU side
Discriminative sparselet activation

\[ f_w(x) = \arg\max_{k \in \{1, \ldots, K\}} w_k^T x \]

Original \( w_k \)

Sparselet approximation \( w_k \)

(i) Reconstructive ECCV 12

(ii) Discriminative ICML 13
Learning parameterization

\[ w_k^T x = (b_{k1}^T, \ldots, b_{kp}^T)(c_1^T, \ldots, c_p^T)^T \]
\[ = \sum_{i=1}^{p} b_{ki}^T c_i \approx \sum_{i=1}^{p} (S\alpha_{ki})^T c_i = \sum_{i=1}^{p} \alpha_{ki}^T (S^T c_i) \]

Model parameter: sparse activation vector

Feature: sparselet response
Structural SVM for DAS

Parameter vector
\[ \beta = (\alpha^T, \tilde{w}^T)^T \]

Transformed features
\[ \tilde{\Phi}_k(x, y) = (c_1^T S, \ldots, c_{p_k}^T S)^T \]

Aggregate feature vector
\[ \tilde{\Phi}(x, y) = (\tilde{\Phi}_1^T(x, y), \ldots, \tilde{\Phi}_s^T(x, y), \tilde{\Phi}_{s+1}^T(x, y), \ldots, \tilde{\Phi}_K^T(x, y))^T \]

- projected feature slot
- remainder feature slot
Discriminative activation of sparselets

\[
\beta^* = \arg\min_{\beta} R(\alpha) + \frac{\lambda}{2} \| \hat{w} \|^2 + \frac{1}{M} \sum_{i=1}^{M} \max_{\hat{y} \in Y} \left( \beta^T \tilde{\Phi}(x_i, \hat{y}) + \Delta(y_i, \hat{y}) \right) - \beta^T \tilde{\Phi}(x_i, y_i)
\]

Sparsity inducing norm
Sparsity enforcing norms

I. Lasso penalty \( R_{\text{Lasso}}(\alpha) = \lambda_1 \| \alpha \|_1 \)

II. Elastic net penalty \( R_{\text{EN}}(\alpha) = \lambda_1 \| \alpha \|_1 + \lambda_2 \| \alpha \|_2^2 \)

III. Combined \( \ell_0 \) and \( \ell_2 \) penalty \( R_{0,2}(\alpha) = \lambda_2 \| \alpha \|_2^2 \) subject to \( \| \alpha \|_0 \leq \lambda_0 \)

III-A. Overshoot, rank, and threshold (ORT)

III-B. Orthogonal matching pursuit (OMP)
Joint feature map: multiclass classification

\[ \mathbf{w} = (\mathbf{w}_1^\top, \ldots, \mathbf{w}_K^\top)^\top \]

\[ \Phi (\mathbf{x}, k) = (0, \ldots, 0, \mathbf{x}^\top, 0, \ldots, 0)^\top \]

Inference \( f_{\mathbf{w}} (\mathbf{x}) = \arg\max_k \mathbf{w}^\top \Phi (\mathbf{x}, k) \)
Joint feature map: multiclass classification with sparselets

\[
\beta = (\alpha_1^T, \ldots, \alpha_K^T, \tilde{w}_1^T, \ldots, \tilde{w}_K^T)^T
\]

\[
\tilde{\Phi}(x, k) = (0, \ldots, 0, (c_1^T S, \ldots, c_{p_k}^T S)^T, 0, \ldots, 0, 0, \ldots, 0, 1, 0, \ldots, 0)^T
\]

Inference \[
f_{\beta}(x) = \arg\max_k \beta^T \tilde{\Phi}(x, k)
\]
Object detection with HOG+SVM
Joint feature map: object detection

\[ w = (w_1^T, \ldots, w_K^T)^T \]

\[ \Phi(x, (k, y)) = (0, \ldots, 0, x_{y:n}^T, 0, \ldots, 0)^T \]

- length n window at position y in slot k
- position in the pyramid
- class index
- feature pyramid

Inference \[ f_w(x) = \arg\max_{k,y} \ w^T \Phi(x, (k, y)) \]
Joint feature map: object detection with sparselets

\[ \beta = (\alpha_1^T, \ldots, \alpha_K^T, \tilde{w}_1^T, \ldots, \tilde{w}_K^T)^T \]

\[ \tilde{\Phi}(x, (k, y)) = (0, \ldots, 0, (c_{y,1}^TS, \ldots, c_{y,p_k}^TS)^T, 0, \ldots, 0, 0, \ldots, 0, 1, 0, \ldots, 0)^T \]

Inference \[ f_\beta(x) = \underset{k,y}{\arg\max} \beta^T \tilde{\Phi}(x, (k, y)) \]
Computational cost analysis

Speedup = \[
\frac{\text{Original classifier cost}}{\text{Sparselet shared cost} + \text{Sparse reconstruction}} = \frac{Qm}{dm + Q\lambda_0}
\]

- To achieve speedup, number of sparselets should be small. \( Q > d \)

- Activation sparsity \( \lambda_0 \) dominates the speedup as \( Q \) grows.
Experiment 1 - Run Time

Run time comparison for DPM implementation on GPU, reconstructive sparselets and discriminatively activated sparselets in contrast to CPU cascade.
Experiment 2 - PASCAL detection

PASCAL VOC 2007 object detection

mAP (%) vs. Sparsity (%)

- Original
- Reconstructive sparselets
- R−Lasso
- R−EN
- R−0,2 ORT
- R−0,2 OMP
Experiment 3 - ImageNet detection

ImageNet object detection (9 classes)

Sparsity (%)

mAP (%)

Original
Reconstructive sparselets
R−0,2 OMP

Sparsity (%)
Experiment 4 - Caltech 101 Classification

![Graph showing the relationship between Speedup factor and Accuracy (%) for different methods. The graph compares the original method with reconstructive sparselets and OMP for different values of m. The legend includes symbols for Original, Reconstructive sparselets m=100, Reconstructive sparselets m=200, R−0,2 OMP m=100, and R−0,2 OMP m=200.]

- Original
- Reconstructive sparselets m=100
- Reconstructive sparselets m=200
- R−0,2 OMP m=100
- R−0,2 OMP m=200
Experiment 5 - Caltech 256 Classification

Caltech-256

Accuracy (%) vs. Speedup factor
Discussion
Contents

- Sliding window object detection
- Deformable part models
- Cascade DPM
- Sparselets
- Hashing based
Hashing part filters

**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

Fast, Accurate Detection of 100k object classes on a single machine, Dean et al, CVPR'13
Conclusion

• Surveyed sliding window object detection

• Various methods exist for speeding up the inference time (not training time)

• For fast training, LDA HOG (Hariharan, ECCV12) works well.