Histograms of Sparse Codes for Object Detection

Xiaofeng Ren (Amazon), Deva Ramanan (UC Irvine)

Presented by Hossein Azizpour
What does the paper do?

- (learning) a new representation
- local histograms of sparse encodings
- replaces HOG...
- (sliding window) detection
- and improves result
How is the feature extracted? (summary)

- **Offline part**
- randomly select a set of $n$ local image intensity patches $Y = [y_1, y_2, \ldots, y_n]$
- generate a codebook $D = [d_1, d_2, \ldots, d_m]$ able to reconstruct $Y$:
- $Y = DX$ where $X = [x_1, x_2, \ldots, x_n]$ is the new encoding of the patches
How is the feature extracted? (summary)

- **Online part**
  - compute the encodings for patches around each *pixel*.
  - do average pooling over fixed regions of pixels
  - (optionally) reduce dimensions using a projection matrix
  - post process the extracted feature (e.g. L2 normalization)

Use instead of HOG in training a detector e.g. DPM
Generating the Codebook

- Given a set of image patches \( Y = [y_1, \cdots, y_n] \)
- Jointly find
  - dictionary \( D = [d_1, \cdots, d_m] \)
  - \textit{sparse} code matrix \( X = [x_1, \cdots, x_n] \)
- \( K \) is a predefined sparsity level
- \textit{minimize residual} rather than a complete reconstruction \( (Y = DX + \epsilon) \)
- solve for \( X \) and \( D \) in the following objective using K-SVD
  \[
  \min_{D,X} \| Y - DX \|_F^2 \quad \text{s.t.} \quad \forall i, \| x_i \|_0 \leq K
  \]
Generating the Codebook (K-SVD)

- alternate between the estimation of $X$ and $D$
- given a dictionary $D$, use Orthogonal Matching Pursuit (OMP) to find the codes $X$
  - a greedy method to select $K$ codes
- given the codes $X$, update dictionary $D$ using SVD

\[
\min_{D,X} \|Y - DX\|_F^2 \quad s.t. \quad \forall i, \|x_i\|_0 \leq K
\]
Generating the Codebook (K-SVD)

- K-SVD as generalization of K-means
- Alternates between estimation of D and X
- A good approximate solver of the optimization

Task: Find the best possible codebook to represent the data samples \( \{y_i\}_{i=1}^{N} \) by nearest neighbor, by solving

\[
\min_{D, X} \|Y - DX\|_F^2 \quad \text{s.t.} \quad \forall i, \|x_i\|_0 \leq K
\]

Initialization: Set the codebook matrix \( C^{(0)} \in \mathbb{R}^{N \times K} \). Set \( J = 1 \).
Repeat until convergence (use stop rule):

- **Sparse Coding Stage:** Partition the training samples \( Y \) into \( K \) sets
  
  \[
  \mathcal{R}_k^{(i-1)} = \{ i \in \{1, \ldots, N\} : \|y_i - c_k^{(i-1)}\|_2 \leq \|y_i - c_k^{(i-1)}\|_2 \}
  \]

  each holding the sample indices most similar to the column \( c_k^{(i-1)} \).

- **Codebook Update Stage:** For each column \( k \) in \( C^{(i-1)} \), update it by

  \[
  c_k^{(i)} = \frac{1}{|\mathcal{R}_k|} \sum_{i \in \mathcal{R}_k} y_i
  \]

  Set \( J = J + 1 \).

Fig. 1. The K-means algorithm.
Generating the Codebook (OMP)

- Greedily selects the codes for the encodings
- Updates all the coefficients
- There is a fast version Called **Batch OMP**

\[
\min_{\alpha \in \mathbb{R}^p} \| y - D\alpha \|_2^2 \quad \text{s.t.} \quad \|\alpha\|_0 \leq L
\]

1: $\Gamma = \emptyset$.
2: for \( \text{iter} = 1, \ldots, L \) do
3: \hspace{1em} Select the atom which most reduces the objective
4: \hspace{2em} $\hat{i} \leftarrow \arg\min_{i \in \Gamma^c} \left\{ \min_{\alpha'} \| y - D_{\Gamma \cup \{i\}} \alpha' \|_2^2 \right\}$
5: \hspace{1em} Update the active set: $\Gamma \leftarrow \Gamma \cup \{\hat{i}\}$.
6: \hspace{1em} Update the residual (orthogonal projection)
7: \hspace{2em} $r \leftarrow (I - D_{\Gamma}(D_{\Gamma}^T D_{\Gamma})^{-1} D_{\Gamma}^T) y$.
8: \hspace{1em} Update the coefficients
9: \hspace{2em} $\alpha_{\Gamma} \leftarrow (D_{\Gamma}^T D_{\Gamma})^{-1} D_{\Gamma}^T y$.
10: end for
Feature Extraction (Binning)

- 8x8 cells
- soft binning (bilinear interpolation)
- 4 – neighborhood
- average pooling on 16x16 neighborhood
- absolute value of sparse codes
- \[ F = (|x_1|, |x_2|, \ldots, |x_m|) \]
- contrast sensitive features \([|x_i|, \max(x_i, 0), \max(-x_i, 0)]\)
Feature Extraction (post processing)

- L2 normalization
- power transform $\bar{F} = F^\alpha$ (element-wise)
Feature Extraction (example)

- 3m dimensional feature
- HOG can be replaced

Figure 3: Visualizing HSC vs HOG: (a) image; (b) dominant orientation in HOG, weighted by gradient magnitude; (c) dominant codeword in HSC, weighted by histogram value; (d) per-cell responses of HSC features when multiplied with a linear SVM model trained on INRIA (colors are on the same scale).
Feature Extraction (dim reduction)

- Too long feature vectors -> slow training and testing
- (somewhat) supervised dimensionality reduction
- Train root filters \((w_1, w_2, \ldots, w_q)\) for different classes/subclasses using original 3m dimensional features.

\[
\begin{align*}
  w_i &= w_i^1 || w_i^2 || \ldots || w_i^C \\
  &\text{where } w_i^c \text{ is the corresponding part of cell } c.
\end{align*}
\]

- stack all cells of all weight vectors to produce matrix \(W = \begin{bmatrix}
  w_1^{1T} \\
  \vdots \\
  w_i^{cT} \\
  \vdots \\
  w_q^{cT}
\end{bmatrix}\)

- Do PCA dimensionality reduction on \(W' = WP, \quad W = USP^T\),

- Use the projection matrix to transform original cell features to lower dimensions \(F' = FP\)
Training Detector

- Deformable Parts Model (DPM)
- Root only
- Root with Parts
- Fixing part latency
- Using originally trained DPMs
- To make training faster!

\[
\sum_{i \in V} w_i^m \phi(x, p_i) + \sum_{ij \in E} w_{ij}^m \psi(p_i, p_j) + b_m
\]

\[
\arg\min_{\beta, \xi_n \geq 0} \frac{1}{2} \beta \cdot \beta + C \sum_n \xi_n
\]

s.t.  \( \forall n \in \text{pos} \)  \( \beta \cdot \Phi(I_n, z_n) \geq 1 - \xi_n \)

\( \forall n \in \text{neg}, \forall z \)  \( \beta \cdot \Phi(I_n, z) \leq -1 + \xi_n \)
Experiments (Different Parameters)

- INRIA pedestrian
- root-only
- HOG AP = 80.2%
- sparsity level vs Dictionary Size
- fix K = 1
- histogram of sparse codes
Experiments (Different Parameters)

- Patch size vs Dictionary size
- K-medoid clustering wouldn’t gain performance larger than 3x3
- Fixed to 5x5
Experiments (Different Parameters)

- K-SVD vs K-means
- Activated code can have a weight other than 1
- Possible change of sign
Experiments (Different Parameters)

- Power transform
- Fixed at 0.25
- double helinger kernel!
Experiments (Different Parameters)

- Supervised PCA (on models) vs PCA on data
- PASCAL 4 classes
  - bus
  - cat
  - diningtable
  - Motorbike
- More effective for person
- fixed at 100
Final Experiments

- **INRIA root only**

- **PASCAL root only**

- **PASCAL with parts**

<table>
<thead>
<tr>
<th></th>
<th>HOG</th>
<th>HSC (_{3\times3})</th>
<th>HSC (_{5\times5})</th>
<th>HSC (_{7\times7})</th>
<th>[14]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>80.2%</td>
<td>80.7%</td>
<td>84.0%</td>
<td>84.9%</td>
<td>84.9%</td>
</tr>
</tbody>
</table>

|                | aero | bike | bird | boat | bttl | bus  | car  | cat  | chair | cow  | dog  | hors | mbik | prsn | plnt | shep | sofa | train | tv  | avg  |
|----------------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|-----|
| **HOG**        | 20.5 | 47.7 | 9.2  | 11.3 | 18.3 | 35.4 | 40.8 | 4.0  | 12.2 | 23.4 | 11.2 | 2.6  | 41.0 | 30.3 | 21.0 | 6.6  | 11.8 | 16.0 | 31.5 |
| **HSC**        | 25.3 | 49.2 | 6.2  | 15.4 | 24.0 | 44.3 | 45.6 | 12.0 | 15.6 | 27.7 | 16.1 | 10.8 | 43.3 | 42.7 | 28.5 | 10.8 | 20.9 | 25.1 | 34.4 |
| **Δ\(_{HSC}\)** | +4.7 | +1.5 | -3.0 | +4.0 | +5.6 | +8.9 | +4.9 | +8.0 | +3.4 | +4.3 | +4.9 | +8.2 | +2.3 | +12.5 | +7.5 | +4.2 | +9.1 | +9.2 | +2.8 | +7.3 | +5.5 |
| [14]           | 25.2 | 50.2 | 5.8  | 11.8 | 17.2 | 41.4 | 43.6 | 3.5  | 15.9 | 21.0 | 15.6 | 7.9  | 44.1 | 34.8 | 30.3 | 9.9  | 14.6 | 18.4 | 36.4 |

(a) Root-only models: HOG, HSC, their difference \(Δ_{HSC}\) (HSC-HOG); and DPM [14]

|                | aero | bike | bird | boat | bttl | bus  | car  | cat  | chair | cow  | dog  | hors | mbik | prsn | plnt | shep | sofa | train | tv  | avg  |
|----------------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|-----|
| **HOG**        | 30.3 | 56.4 | 9.7  | 15.6 | 23.2 | 49.1 | 51.1 | 14.9 | 19.6 | 21.6 | 19.6 | 10.7 | 56.0 | 47.3 | 40.0 | 12.8 | 16.7 | 27.9 | 41.0 |
| **HSC**        | 32.2 | 58.3 | 11.5 | 16.3 | 30.6 | 49.9 | 54.8 | 23.5 | 21.5 | 27.7 | 34.0 | 13.7 | 58.1 | 51.6 | 39.9 | 12.4 | 23.5 | 34.4 | 47.4 |
| **Δ\(_{HSC}\)** | +1.9 | +1.9 | +0.7 | +7.4 | +0.8 | +3.7 | +8.7 | +1.9 | +6.1 | +14.3 | +3.0 | +2.2 | +4.2 | -0.1 | -0.4 | +6.8 | +6.5 | +6.4 | +5.7 | +4.2 |
| [14]           | 30.7 | 58.9 | 10.4 | 14.4 | 24.8 | 49.0 | 54.1 | 11.1 | 20.6 | 25.3 | 25.2 | 11.0 | 58.5 | 48.4 | 41.3 | 12.1 | 15.5 | 34.4 | 43.4 |

(b) Part-based models, with dimension reduction
Visualizing HSC with Reconstructions

(1) Image
(2) HSC
(3) HOG

Courtesy of Carl Voldrick et al 2012 “Inverting and Visualizing Features”
Some detections...

Figure 6: A few examples of HOG (left) vs HSC (right) based detection (root-only), showing top three candidates (in the order of red, green, blue). HSC behaves differently than HOG and tends to have different modes of success (and failure).
Conclusions

- we can easily go beyond hand crafted HOG by a sort of feature learning
- deep learning
- primitive shape codes might work better than simple gradient orientation
- PCA on model instead of data seems promising