Visual Recognition: Combining and Learning Features

Raquel Urtasun

TTI Chicago

Feb 16, 2012
We have a lot of different descriptors focusing on, e.g., shape, gradients, texture.

We have multiple ways to compute similarity (distance) between images (bounding boxes), e.g., histograms, intersection kernels, pyramids.
Combining information

- We have a lot of different descriptors focusing on, e.g., shape, gradients, texture.
- We have multiple ways to computer similarity (distance) between images (bounding boxes), e.g., histograms, intersection kernels, pyramids.
- Which one should we use?
Combining information

- We have a lot of different descriptors focusing on, e.g., shape, gradients, texture.
- We have multiple ways to compute similarity (distance) between images (bounding boxes), e.g., histograms, intersection kernels, pyramids.
- Which one should we use?
- In general there is not a single one that it’s always best.
Combining information

- We have a lot of different descriptors focusing on, e.g., shape, gradients, texture.
- We have multiple ways to compute similarity (distance) between images (bounding boxes), e.g., histograms, intersection kernels, pyramids.
- Which one should we use?
- In general there is not a single one that it’s always best.
- Even if it was, maybe we can perform better by unifying forces ;)

Raquel Urtasun (TTI-C)
Visual Recognition
Feb 16, 2012
2 / 84
Combining information

- We have a lot of different descriptors focusing on, e.g., shape, gradients, texture.

- We have multiple ways to compute similarity (distance) between images (bounding boxes), e.g., histograms, intersection kernels, pyramids.

- Which one should we use?

- In general there is not a single one that it’s always best.

- Even if it was, maybe we can perform better by unifying forces ;)

Multiple ways to combine information

- Stack the feature vectors
- Information fusion
Combining information

Multiple ways to combine information

- Stack the feature vectors
- Information fusion
  - Boosting inherently incorporates multiple features
Combining information

Multiple ways to combine information

- Stack the feature vectors
- Information fusion
- Boosting inherently incorporates multiple features
- Use NN with sum of distances or something more clever
Combining information

Multiple ways to combine information

- Stack the feature vectors
- Information fusion
- Boosting inherently incorporates multiple features
- Use NN with sum of distances or something more clever
- Voting via generalized hough transform, with votes coming from different feature types
Combining information

Multiple ways to combine information

- Stack the feature vectors
- Information fusion
- Boosting inherently incorporates multiple features
- Use NN with sum of distances or something more clever
- Voting via generalized hough transform, with votes coming from different feature types
- Multiple kernel learning
Combining information

Multiple ways to combine information

- Stack the feature vectors
- Information fusion
- Boosting inherently incorporates multiple features
- Use NN with sum of distances or something more clever
- Voting via generalized hough transform, with votes coming from different feature types
- Multiple kernel learning
- Random forest
Combining information

Multiple ways to combine information

- Stack the feature vectors
- Information fusion
- Boosting inherently incorporates multiple features
- Use NN with sum of distances or something more clever
- Voting via generalized hough transform, with votes coming from different feature types
- Multiple kernel learning
- Random forest
  
  etc

Let’s look into some of this strategies.
Combining information

Multiple ways to combine information

- Stack the feature vectors
- Information fusion
- Boosting inherently incorporates multiple features
- Use NN with sum of distances or something more clever
- Voting via generalized hough transform, with votes coming from different feature types
- Multiple kernel learning
- Random forest
- etc

Let’s look into some of this strategies.
NN approaches

NN approaches perform worst than more complex classifiers but [Boiman et al. 08] argue that this is due to

- Quantization of local image descriptors (used to generate bags-of-words, codebooks).
- Computation of Image-to-Image distance, instead of Image-to-Class distance.
NN approaches

NN approaches perform worst than more complex classifiers but [Boiman et al. 08] argue that this is due to

- Quantization of local image descriptors (used to generate bags-of-words, codebooks).
- Computation of Image-to-Image distance, instead of Image-to-Class distance.
- They proposed an effective NN-based classifier NBNN, (Naive-Bayes Nearest-Neighbor), which employs NN distances in the space of the local image descriptors (not images).
NN approaches

NN approaches perform worst than more complex classifiers but [Boiman et al. 08] argue that this is due to

- Quantization of local image descriptors (used to generate bags-of-words, codebooks).
- Computation of Image-to-Image distance, instead of Image-to-Class distance.
- They proposed an effective NN-based classifier NBNN, (Naive-Bayes Nearest-Neighbor), which employs NN distances in the space of the local image descriptors (not images).
- NBNN computes direct Image to- Class distances without descriptor quantization.
NN approaches perform worst than more complex classifiers but [Boiman et al. 08] argue that this is due to

- Quantization of local image descriptors (used to generate bags-of-words, codebooks).
- Computation of Image-to-Image distance, instead of Image-to-Class distance.
- They proposed an effective NN-based classifier NBNN, (Naive-Bayes Nearest-Neighbor), which employs NN distances in the space of the local image descriptors (not images).
- NBNN computes direct Image to- Class distances without descriptor quantization.
- No learning/training phase.
NN approaches

NN approaches perform worst than more complex classifiers but [Boiman et al. 08] argue that this is due to

- Quantization of local image descriptors (used to generate bags-of-words, codebooks).
- Computation of Image-to-Image distance, instead of Image-to-Class distance.
- They proposed an effective NN-based classifier NBNN, (Naive-Bayes Nearest-Neighbor), which employs NN distances in the space of the local image descriptors (not images).
- NBNN computes direct Image to- Class distances without descriptor quantization.
- No learning/training phase.
- Similarities with ISM but now for classification.
NN approaches

NN approaches perform worst than more complex classifiers but [Boiman et al. 08] argue that this is due to

- Quantization of local image descriptors (used to generate bags-of-words, codebooks).
- Computation of Image-to-Image distance, instead of Image-to-Class distance.
- They proposed an effective NN-based classifier NBNN, (Naive-Bayes Nearest-Neighbor), which employs NN distances in the space of the local image descriptors (not images).
- NBNN computes direct Image to- Class distances without descriptor quantization.
- No learning/training phase.
- Similarities with ISM but now for classification.
Algorithm of NBNN

- Given a query image, compute all its local image descriptors $d_1, \cdots, d_n$.
- Search for the class $C$ which minimizes
  \[
  \sum_{i=1}^{n} \| d_i - NN_C(d_i) \|^2
  \]
  with $NN_C(d_i)$ the NN descriptor of $d_i$ in class $C$.
- Requires fast NN search.
Why quantization is bad

- When densely sampled image descriptors are divided into fine bins, the bin-density follows a power-law.
- There are almost no clusters in the descriptor space.
- Therefore, any clustering to a small number of clusters (even thousands) will inevitably incur a very high quantization error.
- Informative descriptors have low database frequency, leading to high quantization error.
Image-to-Image vs. Image-to-Class distance

\[ KL(p_Q \mid p_1) = 17.54 \]

\[ KL(p_Q \mid p_2) = 18.20 \]

\[ KL(p_Q \mid p_3) = 14.56 \]

\[ KL(p_Q \mid p_c) = 8.35 \]
Multiple descriptors by summing weighted distances.
Effects of Quantization

Impact of introducing descriptor quantization or Image-to-Image distance into NBNN (using SIFT descriptor on Caltech-101, nlabel = 30).

<table>
<thead>
<tr>
<th></th>
<th>No Quant.</th>
<th>With Quant.</th>
</tr>
</thead>
<tbody>
<tr>
<td>“Image-to-Class”</td>
<td>70.4%</td>
<td>50.4% (-28.4%)</td>
</tr>
<tr>
<td>“Image-to-Image”</td>
<td>58.4% (-17%)</td>
<td>-</td>
</tr>
</tbody>
</table>
Randomized Decision Forests

- Very fast tools for classification, clustering and regression
- Good generalization through randomized training
- Inherently multi-class: automatic feature sharing
- Simple training / testing algorithms
Randomized Forests in Vision

[Source: Shotton et al., 08] object segmentation

[Shotton et al., 08] object segmentation

[Lepetit et al., 06] keypoint recognition

[Moosmann et al., 06] visual word clustering

[Source: Shotton et al.]
Is the grass wet?

[Source: Shotton et al.]
Binary Decision Trees

- feature vector $\mathbf{v} \in \mathbb{R}^N$
- split functions $f_n(\mathbf{v}) : \mathbb{R}^N \rightarrow \mathbb{R}$
- thresholds $t_n \in \mathbb{R}$
- classifications $P_n(c)$

[Source: Shotton et al.]
double[] ClassifyDT(node, v)
  if node.IsSplitNode then
    if node.f(v) >= node.t then
      return ClassifyDT(node.right, v)
    else
      return ClassifyDT(node.left, v)
  end
  else
    return node.P
  end
end

[Source: Shotton et al.]
Toy Example

• Try several lines, chosen at random

• Keep line that best separates data
  – information gain

• Recurse

- feature vectors are $x$, $y$ coordinates: $\mathbf{v} = [x, y]^T$
- split functions are lines with parameters $a$, $b$: $f_n(\mathbf{v}) = ax + by$
- threshold determines intercepts: $t_n$
- four classes: purple, blue, red, green

[Source: Shotton et al.]
Toy Example

• Try several lines, chosen at random

• Keep line that best separates data
  – information gain

• Recurse

- feature vectors are x, y coordinates: \( \mathbf{v} = [x, y]^T \)
- split functions are lines with parameters \( a, b \): \( f_n(\mathbf{v}) = ax + by \)
- threshold determines intercepts: \( t_n \)
- four classes: purple, blue, red, green

[Source: Shotton et al.]
Toy Example

- Try several lines, chosen at random
- Keep line that best separates data
  - information gain
- Recurse

- feature vectors are x, y coordinates: \( \mathbf{v} = [x, y]^T \)
- split functions are lines with parameters a, b: \( f_n(\mathbf{v}) = ax + by \)
- threshold determines intercepts: \( t_n \)
- four classes: purple, blue, red, green

[Source: Shotton et al.]
Toy Example

- Try several lines, chosen at random
- Keep line that best separates data
  - information gain
- Recurse

- feature vectors are $x, y$ coordinates: $\mathbf{v} = [x, y]^T$
- split functions are lines with parameters $a, b$: $f_n(\mathbf{v}) = ax + by$
- threshold determines intercepts: $t_n$
- four classes: purple, blue, red, green

[Source: Shotton et al.]
Randomized Learning

- Recursively split examples at node $n$: set $I_n$ indexes labeled training examples $(v_i, l_i)$

$$I_1 = \{ i \in I_n \mid f(v_i) < t \}$$

$$I_r = I_n \setminus I_1$$

- At node $n$, $P_n(c)$ is histogram of example labels $l_i$.

[Source: Shotton et al.]
Randomized Learning

Left split \( I_1 = \{ i \in I_n \mid f(v_i) < t \} \)
Right split \( I_r = I_n \setminus I_1 \)

- Features \( f(v) \) chosen at random from feature pool \( f \in F \)
- Thresholds \( t \) chosen in range \( t \in (\min_i f(v_i), \max_i f(v_i)) \)
- Choose \( f \) and \( t \) to maximize gain in information

\[
\Delta E = -\frac{|I_1|}{|I_n|} E(I_1) - \frac{|I_r|}{|I_n|} E(I_r)
\]
Entrophy \( E \) calculated from histogram of labels in \( I \)

[Source: Shotton et al.]
How many features and thresholds to try?

- just one = extremely randomized
- few → fast training, may under-fit, maybe too deep
- many → slower training, may over-fit

When to stop growing the tree?

- maximum depth
- minimum entropy gain
- pruning

[Source: Shotton et al.]
Randomized Learning Pseudo Code

```
TreeNode LearnDT(I)

    repeat featureTests times
    let f = RndFeature()
    let r = EvaluateFeatureResponses(I, f)

    repeat threshTests times
    let t = RndThreshold(r)
    let (I_l, I_r) = Split(I, r, t)
    let gain = InfoGain(I_l, I_r)
    if gain is best then remember f, t, I_l, I_r
    end
    end

    if best gain is sufficient
    return SplitNode(f, t, LearnDT(I_l), LearnDT(I_r))
    else
    return LeafNode(HistogramExamples(I))
    end
end
```

[Source: Shotton et al.]
A forest of trees

- Forest is ensemble of several decision trees

\[ P(c|v) = \frac{1}{T} \sum_{t=1}^{T} P_t(c|v) \]

[Source: Shotton et al.]

[Source: Shotton et al.]
double[] ClassifyDF(forest, v)
    // allocate memory
    let P = double[forest.CountClasses]

    // loop over trees in forest
    for t = 1 to forest.CountTrees
        let P' = ClassifyDT(forest.Tree[t], v)
        P = P + P' // sum distributions
    end

    // normalise
    P = P / forest.CountTrees
end
• **Divide training examples into** $T$ **subsets** $l_t \in L$
  
  – improves generalization
  – reduces **memory requirements** & **training time**

• **Train each decision tree** $t$ **on subset** $l_t$
  
  – same decision tree learning as before

• **Multi-core friendly**

  - Subsets can be chosen at random or hand-picked
  - Subsets can have overlap (and usually do)
  - Can enforce subsets of **images** (not just examples)
  - Could also divide the feature pool into subsets

[Source: Shotton et al.]
Learning

```plaintext
Forest LearnDF(countTrees, I)
    // allocate memory
    let forest = Forest(countTrees)

    // loop over trees in forest
    for t = 1 to countTrees
        let I_t = RandomSplit(I)
        forest[t] = LearnDT(I_t)
    end

    // return forest object
    return forest
end

[Source: Shotton et al.]
```
• Trees can be trained for
  – classification, regression, or clustering

• Change the object function
  – information gain for classification:
    \[ I = H(S) - \sum_{i=1}^{2} \frac{|S_i|}{|S|} H(S_i) \]
    measure of distribution purity

[Source: Shotton et al.]
- Real-valued output $y$
- Object function: maximize

$$Err(S) - \sum_{i=1}^{2} \frac{|S_i|}{|S|} Err(S_i)$$

Measure of fit of model:

$$Err(S) = \sum_{j \in S} (y_j - y(x_j))^2$$

E.g. linear model $y = ax + b$
Or just constant model

[Source: Shotton et al.]
Clustering

- Output is cluster membership
- Option 1 – minimize imbalance:  \[ B = | \log |S_1| - \log |S_2| | \]  [Moosmann et al. 06]
- Option 2 – maximize Gaussian likelihood:
  \[ T = |\Lambda_S| - \sum_{i=1}^{2} \frac{|S_i|}{|S|} |\Lambda_{S_i}| \]  measure of cluster tightness (maximizing a function of info gain for Gaussian distributions)

[Source: Shotton et al.]
Clustering example [Moosmann et al. 06]

- Visual words good for e.g. matching, recognition but $k$-means clustering very slow

- Randomized forests for clustering descriptors
  - e.g. SIFT, texton filter-banks, etc.

- Leaf nodes in forest are clusters
  - concatenate histograms from trees in forest

[Source: Shotton et al.]
Clustering example [Moosmann et al. 06]

```
<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td></td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

“bag of words”

[Source: Shotton et al.]
Applications: keypoint detection [LePetit 06]

- Wide-baseline matching as classification problem

- Extract prominent key-points in training images

- Forest classifies
  - patches -> keypoints

- Features
  - pixel comparisons

- Augmented training set
  - gives robustness to patch scaling, translation, rotation

[Source: Shotton et al.]
Fast Keypoint Recognition

[Source: Shotton et al.]
Classification

![Graph](image)

- **Performance (%)**
- **# trees**

**Legend**
- Blue line: random test
- Green line: entropy optimization

![Graph](image)

- **Performance (%)**
- **# trees**

**Legend**
- Blue dotted line: depth = 10
- Green line: depth = 15
- Red line: depth = 20
Object Recognition Pipeline

- **extract features**: SIFT, filter bank
- **clustering**: k-means
- **assignment**: nearest neighbour

- **classification algorithm**: SVM, decision forest, boosting

[Source: Shotton et al.]
Object Recognition Pipeline

Semantic Texton Forest (STF)
- decision forest for clustering & classification
- tree nodes have learned object category associations

Semantic Texton Forest (STF)

[Source: Shotton et al.]
Example Semantic Texton Forest

[Source: Shotton et al.]
MSRC Dataset Results

[Source: Shotton et al.]
Microsoft Kinect
\[ P(c|I, x) = \frac{1}{T} \sum_{t=1}^{T} P_t(c|I, x) . \] (2)

**Training.** Each tree is trained on a different set of randomly synthesized images. A random subset of 2000 example pixels from each image is chosen to ensure a roughly even distribution across body parts. Each tree is trained using the following algorithm [20]:

1. Randomly propose a set of splitting candidates \( \phi = (\theta, \tau) \) (feature parameters \( \theta \) and thresholds \( \tau \)).

2. Partition the set of examples \( Q = \{(I, x)\} \) into left and right subsets by each \( \phi \):
   \[
   Q_1(\phi) = \{(I, x) \mid f_\theta(I, x) < \tau\} \quad (3)
   \]
   \[
   Q_r(\phi) = Q \setminus Q_1(\phi) \quad (4)
   \]

3. Compute the \( \phi^* \) giving the largest gain in information:
   \[
   \phi^* = \arg \max_{\phi} G(\phi) \quad (5)
   \]
   \[
   G(\phi) = H(Q) - \sum_{s \in \{l, r\}} \frac{|Q_s(\phi)|}{|Q|} H(Q_s(\phi)) \quad (6)
   \]
   where Shannon entropy \( H(Q) \) is computed on the normalized histogram of body part labels \( l_I(x) \) for all \((I, x) \in Q\).

4. If the largest gain \( G(\phi^*) \) is sufficient, and the depth in the tree is below a maximum, then recurse for left and right subsets \( Q_1(\phi^*) \) and \( Q_r(\phi^*) \).
Microsoft Kinect

![Graph showing performance vs. depth of trees for synthetic and real test sets with different training image counts.]

- Synthetic Test Set
  - 900k training images
  - 15k training images

- Real Test Set
  - 900k training images
  - 15k training images

(b) Depth of trees
Learning Representations
Learning Representations

- Sparse coding
- Deep architectures
- Topic models
Image Classification using BoW

Dictionary Learning

Dictionary

K-means

Dense/Sparse SIFT

Dense/Sparse SIFT

Spatial Pyramid Pooling

VQ Coding

Nonlinear SVM

Image Classification

[Source: K. Yu]
BoW+SPM: the architecture

- Nonlinear SVM is not scalable
- VQ coding may be too coarse
- Average pooling is not optimal
- Why not learn the whole thing?

[Source: K. Yu]
Sparse Architecture

- Nonlinear SVM is not scalable → Scalable linear classifier
- VQ coding may be too coarse → Better coding methods
- Average pooling is not optimal → Better pooling methods
- Why not learn the whole → Deep learning

[Source: A. Ng]
Feature learning problem

- Given a $14 \times 14$ image patch $x$, can represent it using 196 real numbers.
- Problem: Can we find a better representation for this?

- Given a set of images, learn a better way to represent image than pixels.

[Source: A. Ng]
Sparse coding [Olshausen & Field, 1996]

Input: Images $x^{(1)}, x^{(2)}, \cdots, x^{(m)}$ (each in $\mathbb{R}^{n \times n}$)

Learn: Dictionary of bases $\phi_1, \cdots, \phi_k$ (also $\mathbb{R}^{n \times n}$), so that each input $x$ can be approximately decomposed as:

$$x \approx \sum_{j=1}^{k} a_j \phi_j$$

such that the $a_j$'s are mostly zero, i.e., sparse

[Source: A. Ng]
Sparse Coding Illustration

Natural Images

Learned bases ($\phi_1, \ldots, \phi_{64}$): “Edges”

Test example

\[
x \approx 0.8 * \phi_{36} + 0.3 * \phi_{42} + 0.5 * \phi_{63}
\]

\[
(0, 0, \ldots, 0, 0.8, 0, \ldots, 0, 0.3, 0, \ldots, 0, 0.5, \ldots)
\]

= [a_1, \ldots, a_{64}] (feature representation)

Compact & easily interpretable

[Source: A. Ng]
Method hypothesizes that edge-like patches are the most basic elements of a scene, and represents an image in terms of the edges that appear in it.

Use to obtain a more compact, higher-level representation of the scene than pixels.

Represent as: \([0, 0, ..., 0, 0.6, 0, ..., 0, 0.8, 0, ..., 0, 0.4, ...]\)

Represent as: \([0, 0, ..., 0, 1.3, 0, ..., 0, 0.9, 0, ..., 0, 0.3, ...]\)

[Source: A. Ng]
Sparse Coding details

- **Input**: Images $x^{(1)}, x^{(2)}, \ldots, x^{(m)}$ (each in $\mathbb{R}^{n \times n}$)

- Obtain dictionary elements and weights by

$$
\min_{a, \phi} \sum_{i=1}^{m} \left( \| x^{(i)} - \sum_{j=1}^{k} a^{(i)}_{j} \phi_{j} \|_2^2 + \lambda \sum_{j=1}^{k} |a^{(i)}_{j}|_1 \right)
$$

- Alternating minimization with respect to $\phi_j$’s and a’s.

- The second is harder, the first one is closed form.
Fast algorithms

- Solving for $a$ is expensive.
- Simple algorithm that works well
  - Repeatedly guess sign (+, -, or 0) of each of the $a_i$’s.
  - Solve for $a_i$’s in closed form. Refine guess for signs.
- Other algorithms such as projective gradient descent, stochastic subgradient descent, etc

[Source: A. Ng]
Recap of sparse coding for feature learning

Training:

- Input: Images $x^{(1)}, x^{(2)}, \ldots, x^{(m)}$ (each in $\mathbb{R}^{n \times n}$)
- Learn: Dictionary of bases $\phi_1, \ldots, \phi_k$ (also $\mathbb{R}^{n \times n}$),

$$\min_{a, \phi} \sum_{i=1}^{m} \left( \|x^{(i)} - \sum_{j=1}^{k} a_j^{(i)} \phi_j \|^2 + \lambda \sum_{j=1}^{k} |a_j^{(i)}| \right)$$

Test time:

- Input: novel $x^{(1)}, x^{(2)}, \ldots, x^{(m)}$ and learned $\phi_1, \ldots, \phi_k$.
- Solve for the representation $a_1, \ldots, a_k$ for each example

$$\min_{a} \sum_{i=1}^{m} \left( \|x - \sum_{j=1}^{k} a_j \phi_j \|^2 + \lambda \sum_{j=1}^{k} |a_j| \right)$$
Sparse coding recap

- Much better than pixel representation, but still not competitive with SIFT, etc.
- Three ways to make it competitive:
  - Combine this with SIFT.
  - Advanced versions of sparse coding, e.g., LCC.
  - Deep learning.

[Source: A. Ng]
Sparse Classification

Suppose you’ve already learned bases $\phi_1, \phi_2, \ldots, \phi_k$. Here’s how you represent an image.

E.g., 73-75% on Caltech 101 (Yang et al., 2009, Boreau et al., 2009)

[Source: A. Ng]
K-means vs sparse coding

[Source: A. Ng]
Why sparse coding helps classification?

- The coding is a nonlinear feature mapping
- Represent data in a higher dimensional space
- Sparsity makes prominent patterns more distinctive

[Source: K. Yu]
A topic model view to sparse coding

- Each basis is a direction or a topic.
- Sparsity: each datum is a linear combination of only a few bases.
- Applicable to image denoising, inpainting, and super-resolution.

Both figures adapted from CVPR10 tutorial by F. Bach, J. Mairal, J. Ponce and G. Sapiro

[Source: K. Yu]
A geometric view to sparse coding

- Each basis is somewhat like a pseudo data point anchor point.
- Sparsity: each datum is a sparse combination of neighbor anchors.
- The coding scheme explores the manifold structure of data.

[Source: K. Yu]
Influence of Sparsity

- When SC achieves the best classification accuracy, the learned bases are like digits – each basis has a clear local class association.
Learning an image classifier is a matter of learning nonlinear functions on patches:

\[ f(x) = w^T x = \sum_{i=1}^{m} a_i (w^T \phi(i)) = \sum_{i=1}^{m} a_i f(\phi(i)) \]

where \( x = \sum_{i=1}^{m} a_i \phi(i) \)

[Source: K. Yu]
Nonlinear learning via local coding

- We assume \( x_i \approx \sum_{j=1}^{k} a_{i,j} \phi_j \) and thus

\[
f(x_i) = \sum_{j=1}^{k} a_{i,j} f(\phi_j)
\]

[Source: K. Yu]
How to learn the non-linear function

1. Learning the dictionary $\phi_1, \cdots, \phi_k$ from unlabeled data

2. Use the dictionary to encode data $x_i \rightarrow a_{i,1}, \cdots, a_{i,k}$

3. Estimate the parameters

$$
\begin{bmatrix}
  f(x_1) \\
  f(x_2) \\
  f(x_m)
\end{bmatrix} \approx
\begin{bmatrix}
a_{1,1} & a_{1,2} & \cdots & a_{1,k} \\
a_{2,1} & a_{2,2} & \cdots & a_{2,k} \\
a_{m,1} & a_{m,2} & \cdots & a_{m,k}
\end{bmatrix}
\begin{bmatrix}
f(\phi_1) \\
f(\phi_2) \\
f(\phi_k)
\end{bmatrix}
$$

Nonlinear local learning via learning a global linear function
Local Coordinate Coding (LCC):

- If $f(x)$ is $(\alpha, \beta)$-Lipschitz smooth

\[
 f(x_i) - \sum_{j=1}^{k} a_{i,j} f(\phi_j) \leq \alpha \left\| x_i - \sum_{j=1}^{k} a_{i,j} \phi_j \right\| + \beta \sum_{j=1}^{k} |a_{i,j}| \left\| x_i - \phi_j \right\|^2
\]

Function approximation error
Coding error
Locality term

A good coding scheme should [Yu et al. 09]

- have a small coding error,
- and also be sufficiently local

[Source: K. Yu]
The larger dictionary, the higher accuracy, but also the higher comp. cost

Table 2: Error rates (%) of MNIST classification with different $|C|$.

| $|C|$                        | 512   | 1024  | 2048  | 4096  |
|-----------------------------|-------|-------|-------|-------|
| Linear SVM with sparse coding | 2.96  | 2.64  | 2.16  | 2.02  |
| Linear SVM with local coordinate coding | 2.64  | 2.44  | 2.08  | 1.90  |

(Yu et al. 09)

Table 5. The effects of codebook size on ScSPM and LSPM respectively on Caltech 101 dataset.

<table>
<thead>
<tr>
<th>Codebook size</th>
<th>256</th>
<th>512</th>
<th>1024</th>
</tr>
</thead>
<tbody>
<tr>
<td>30 train</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ScSPM</td>
<td>68.26</td>
<td>71.20</td>
<td>73.20</td>
</tr>
<tr>
<td>LSPM</td>
<td>57.42</td>
<td>58.81</td>
<td>58.56</td>
</tr>
<tr>
<td>15 train</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ScSPM</td>
<td>61.97</td>
<td>63.23</td>
<td>69.70</td>
</tr>
<tr>
<td>LSPM</td>
<td>51.84</td>
<td>53.23</td>
<td>51.74</td>
</tr>
</tbody>
</table>

(Yang et al. 09)

The same observation for Caltech256, PASCAL, ImageNet

[Source: K. Yu]
Locality-constrained linear coding

- A fast implementation of LCC [Wang et al. 10]
- Dictionary learning using k-means and code for \( x \) based on

**Step 1** ensure locality: find the K nearest bases \([\phi_j]_{j \in \mathcal{J}(x)}\)

**Step 2** ensure low coding error:

\[
\min_a ||x - \sum_{j \in \mathcal{J}(x)} a_{i,j} \phi_j||^2, \quad \text{s.t.} \quad \sum_{j \in \mathcal{J}(x)} a_{i,j} = 1
\]

[Source: K. Yu]
Table 1. Image classification results on Caltech-101 dataset

<table>
<thead>
<tr>
<th>training images</th>
<th>5</th>
<th>10</th>
<th>15</th>
<th>20</th>
<th>25</th>
<th>30</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zhang [25]</td>
<td>46.6</td>
<td>55.8</td>
<td>59.1</td>
<td>62.0</td>
<td>-</td>
<td>66.20</td>
</tr>
<tr>
<td>Lazebnik [15]</td>
<td>-</td>
<td>-</td>
<td>56.40</td>
<td>-</td>
<td>-</td>
<td>64.60</td>
</tr>
<tr>
<td>Griffin [11]</td>
<td>44.2</td>
<td>54.5</td>
<td>59.0</td>
<td>63.3</td>
<td>65.8</td>
<td>67.60</td>
</tr>
<tr>
<td>Boiman [2]</td>
<td>-</td>
<td>-</td>
<td>65.00</td>
<td>-</td>
<td>-</td>
<td>70.40</td>
</tr>
<tr>
<td>Jain [12]</td>
<td>-</td>
<td>-</td>
<td>61.00</td>
<td>-</td>
<td>-</td>
<td>69.10</td>
</tr>
<tr>
<td>Gemert [8]</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>64.16</td>
</tr>
<tr>
<td>Yang [22]</td>
<td>-</td>
<td>-</td>
<td>67.00</td>
<td>-</td>
<td>-</td>
<td>73.20</td>
</tr>
<tr>
<td>Ours</td>
<td><strong>51.15</strong></td>
<td><strong>59.77</strong></td>
<td>65.43</td>
<td><strong>67.74</strong></td>
<td><strong>70.16</strong></td>
<td><strong>73.44</strong></td>
</tr>
</tbody>
</table>

Table 2. Image classification results using Caltech-256 dataset

<table>
<thead>
<tr>
<th>training images</th>
<th>15</th>
<th>30</th>
<th>45</th>
<th>60</th>
</tr>
</thead>
<tbody>
<tr>
<td>Griffin [11]</td>
<td>28.30</td>
<td>34.10</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Gemert [8]</td>
<td>-</td>
<td>27.17</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Yang [22]</td>
<td>27.73</td>
<td>34.02</td>
<td>37.46</td>
<td>40.14</td>
</tr>
<tr>
<td>Ours</td>
<td><strong>34.36</strong></td>
<td><strong>41.19</strong></td>
<td><strong>45.31</strong></td>
<td><strong>47.68</strong></td>
</tr>
</tbody>
</table>

[Source: K. Yu]
Interpretation of BoW + linear classifier

Piece-wise local constant (zero-order)

\[ f(x_i) \approx f(\phi^*) \]

\[ \phi^* = \arg \min_{\phi_j} \| x_i - \phi_j \| \]

- data points
- cluster centers

[Source: K. Yu]
Support vector coding [Zhou et al, 10]

\[ f(x) \]

Local tangent  
Piecewise local linear (first-order)

\[ \phi^* = \arg \min_{\phi_j} \| x_i - \phi_j \| \]

- data points
- cluster centers

[Source: K. Yu]
Let \([a_i, 1, \cdots, a_i, k]\) be the VQ coding of \(x_i\)

\[
f(x_i) \approx \begin{bmatrix} a_{i,1}(1, x_i - \phi_1), & \cdots & a_{i,k}(1, x_i - \phi_k) \end{bmatrix} \begin{bmatrix} f(\phi_1) \\ \nabla f(\phi_1) \\ \vdots \\ f(\phi_k) \\ \nabla f(\phi_k) \end{bmatrix}
\]

Super-vector codes of data

Global linear weights to be learned

No.1 position in PASCAL VOC 2009

[Source: K. Yu]
Summary of coding algorithms

- All lead to higher-dimensional, sparse, and localized coding
- All explore geometric structure of data
- New coding methods are suitable for linear classifiers.
- Their implementations are quite straightforward.

[Source: K. Yu]
**PASCAL VOC 2009**

- No.1 for 18 of 20 categories
- PASCAL: 20 categories, 10,000 images
- They use only HOG feature on gray images

<table>
<thead>
<tr>
<th>Classes</th>
<th>Ours</th>
<th>Best of Other Teams</th>
<th>Difference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aeroplane</td>
<td>88.1</td>
<td>86.6</td>
<td>1.5</td>
</tr>
<tr>
<td>Bicycle</td>
<td>68.6</td>
<td>63.9</td>
<td>4.7</td>
</tr>
<tr>
<td>Bird</td>
<td>68.1</td>
<td>66.7</td>
<td>1.4</td>
</tr>
<tr>
<td>Boat</td>
<td>72.9</td>
<td>67.3</td>
<td>5.6</td>
</tr>
<tr>
<td>Bottle</td>
<td>44.2</td>
<td>43.7</td>
<td>0.5</td>
</tr>
<tr>
<td>Bus</td>
<td>79.5</td>
<td>74.1</td>
<td>5.4</td>
</tr>
<tr>
<td>Car</td>
<td>72.5</td>
<td>64.7</td>
<td>7.8</td>
</tr>
<tr>
<td>Cat</td>
<td>70.8</td>
<td>64.2</td>
<td>6.6</td>
</tr>
<tr>
<td>Chair</td>
<td>59.5</td>
<td>57.4</td>
<td>2.1</td>
</tr>
<tr>
<td>Cow</td>
<td>53.6</td>
<td>46.2</td>
<td>7.4</td>
</tr>
<tr>
<td>Diningtable</td>
<td>57.5</td>
<td>54.7</td>
<td>2.8</td>
</tr>
<tr>
<td>Dog</td>
<td>59.3</td>
<td>53.5</td>
<td>5.8</td>
</tr>
<tr>
<td>Horse</td>
<td>73.1</td>
<td>68.1</td>
<td>5.0</td>
</tr>
<tr>
<td>Motorbike</td>
<td>72.3</td>
<td>70.6</td>
<td>1.7</td>
</tr>
<tr>
<td>Person</td>
<td>85.3</td>
<td>85.2</td>
<td>0.1</td>
</tr>
<tr>
<td>Pottedplant</td>
<td>36.6</td>
<td>39.1</td>
<td>-2.5</td>
</tr>
<tr>
<td>Sheep</td>
<td>58.9</td>
<td>48.2</td>
<td>8.7</td>
</tr>
<tr>
<td>Sofa</td>
<td>57.9</td>
<td>50.0</td>
<td>7.9</td>
</tr>
<tr>
<td>Train</td>
<td>86.0</td>
<td>83.4</td>
<td>2.6</td>
</tr>
<tr>
<td>Tvmonitor</td>
<td>68.0</td>
<td>68.6</td>
<td>-0.6</td>
</tr>
</tbody>
</table>

(Source: K. Yu)
ImageNet: 1000 categories, 1.2 million images for training

[Source: K. Yu]
150 teams registered worldwide, resulting 37 submissions from 11 teams.

SC achieved 52% for 1000 class classification.

<table>
<thead>
<tr>
<th>Teams</th>
<th>Top 5 Hit Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Our team: NEC-UIUC</td>
<td>72.8%</td>
</tr>
<tr>
<td>Xerox European Lab, France</td>
<td>66.4%</td>
</tr>
<tr>
<td>Univ. of Tokyo</td>
<td>55.4%</td>
</tr>
<tr>
<td>Univ. of California, Irvine</td>
<td>53.4%</td>
</tr>
<tr>
<td>MIT</td>
<td>45.6%</td>
</tr>
<tr>
<td>NTU, Singapore</td>
<td>41.7%</td>
</tr>
<tr>
<td>LIG, France</td>
<td>39.3%</td>
</tr>
<tr>
<td>IBM T. J. Waston Research Center</td>
<td>30.0%</td>
</tr>
<tr>
<td>National Institute of Informatics, Tokyo</td>
<td>25.8%</td>
</tr>
<tr>
<td>SRI (Stanford Research Institute)</td>
<td>24.9%</td>
</tr>
</tbody>
</table>

[Source: K. Yu]
Learning Codebooks for Image Classification

Replacing Vector Quantization by Learned Dictionaries

- unsupervised: [Yang et al., 2009]
- supervised: [Boureau et al., 2010, Yang et al., 2010]

[Source: Mairal]
“Discriminative” training

[Mairal, Bach, Ponce, Sapiro, and Zisserman, 2008a]

$$\min_{D_{-}, D_{+}} \sum_{i} C \left( \lambda z_{i} (R^{*}(x_{i}, D_{-}) - R^{*}(x_{i}, D_{+})) \right),$$

where $z_{i} \in \{-1, +1\}$ is the label of $x_{i}$. 

[Source: Mairal]
Discriminative Dictionaries

Figure: Top: reconstructive, Bottom: discriminative, Left: Bicycle, Right: Background

[Source: Mairal]
Application: Edge detection

[Source: Mairal]
Application: Edge detection

[Source: Mairal]
Application: Authentic from fake

[Source: Mairal]
Predictive Sparse Decomposition (PSD)

- Feed-forward predictor function for feature extraction

\[
\min_{a, \phi, K} \sum_{i=1}^{m} \left( \|x^{(i)} - \sum_{j=1}^{k} a^{(i)} j \phi_j \|^2 + \lambda \sum_{j=1}^{k} |a^{(i)} j| + \beta \|a - C(X, K)\|^2 \right)
\]

with e.g., \( C(X, K) = g \cdot \tanh(x \ast k) \)

Learning is done by

1) Fix \( K \) and \( a \), minimize to get optimal \( \phi \)

2) Update \( a \) and \( K \) using optimal \( \phi \)

3) Scale elements of \( \phi \) to be unit norm.

[Source: Y. LeCun]
12 x12 natural image patches with 256 dictionary elements

Figure: (Left) Encoder, (Right) Decoder

[Source: Y. LeCun]
Train layer wise [Hinton, 06]

\[ C(X, K^1) \]
\[ C(f(K^1), K^2) \]
\[ \ldots \]

Each layer is trained on the output \( f(x) \) produced from previous layer.

\( f \) is a series of non-linearity and pooling operations
Object Recognition - Caltech 101

<table>
<thead>
<tr>
<th></th>
<th>R</th>
<th>RR</th>
<th>U</th>
<th>UU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unsupervised</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Random</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Supervised</td>
<td>R⁺</td>
<td>R⁺ R⁺</td>
<td>U⁺</td>
<td>U⁺ U⁺</td>
</tr>
</tbody>
</table>

Legend:
- Pa
- N-Pa
- N-Pm
- Rabs-Pa
- Rabs-N-Pa
- Conv Rabs-N-Pa
Sparse coding produces filters that are shifted versions of each other.

It ignores that it’s going to be used in a convolutional fashion.

Inference in all overlapping patches independently.

Problems with sparse coding:

1) The representations are redundant, as the training and inference are done at the patch level.

2) Inference for the whole image is computationally expensive.
Solutions

Problems

1) the representations are redundant, as the training and inference are done at the patch level

2) inference for the whole image is computationally expensive

Solutions

1) Apply sparse coding to the entire image at once, with the dictionary as a convolutional filter bank

\[ \ell(x, z, D) = \frac{1}{2} \| x - \sum_{k=1}^{K} D_k * z_k \|^2_2 + |z|_1 \]

2) Use feed-forward, non-linear encoders to produce a fast approx. to the sparse code

\[ \ell(x, z, D, W) = \frac{1}{2} \| x - \sum_{k=1}^{K} D_k * z_k \|^2_2 + \sum_{k=1}^{K} \| z_k - f(W^k * x) \|^2_2 + |z|_1 \]