Visual Recognition: Combining and Learning Features

Raquel Urtasun

TTI Chicago

Feb 16, 2012

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.

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 computer 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 computer 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 ;)

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?
- 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 ;)

Combining information

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

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.

Algortithm 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.

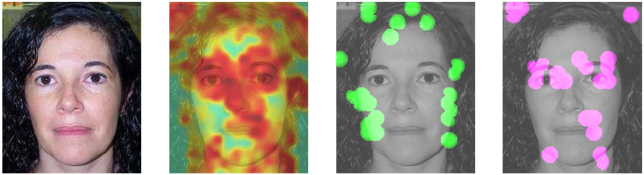
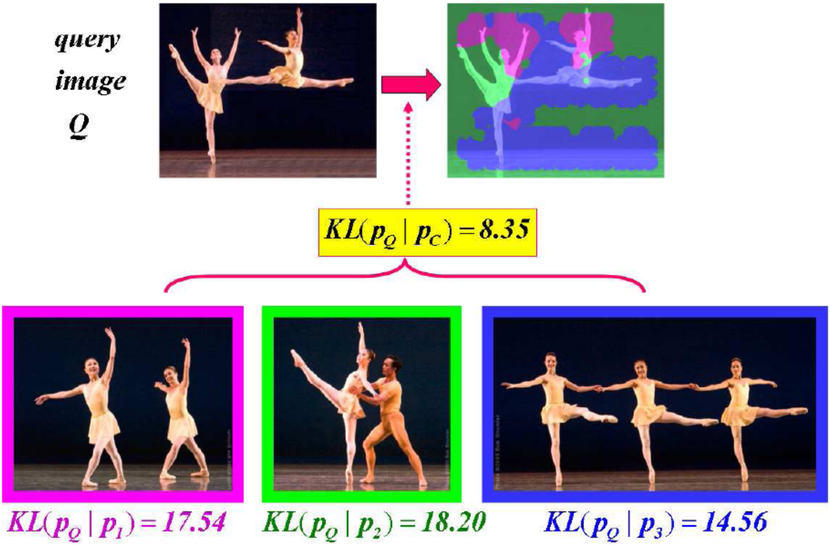
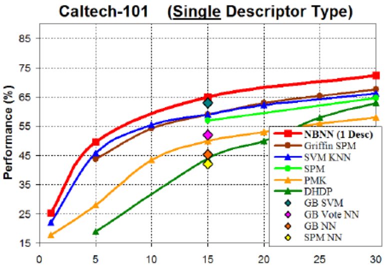
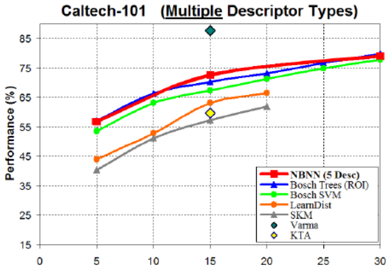


Image-to-Image vs. Image-to-Class distance



Results Caltech 101





Multiple descriptors by summing weighted distances.

Effects of Quantization

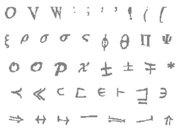
Impact of introducing descriptor quantization or Image-to-Image distance into NBNN (using SIFT descriptor on Caltech- 101, nlabel $= 30$).

	No Quant.	**With Quant.**
"Image-to-Class"	**70.4%**	50.4% (-28.4%)
"Image-to-Image"	58.4% (-17%)	-

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



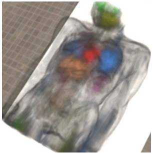
[Amit & Geman, 97]
digit recognition

[Lepetit *et al.*, 06]
keypoint recognition

[Moosmann *et al.*, 06]
visual word clustering

[Shotton *et al.*, 08]
object segmentation

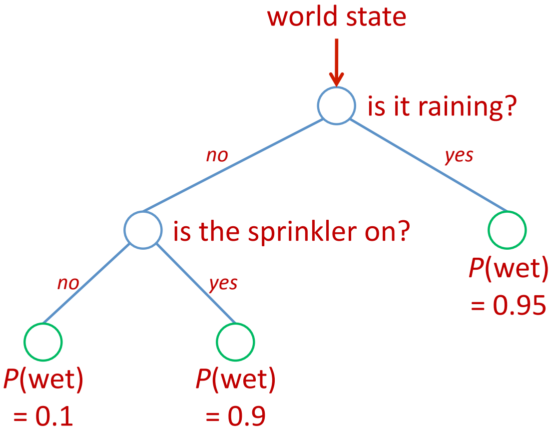
[Rogez *et al.*, 08]
pose estimation



[Criminisi *et al.*, 09]
organ detection

[Source: Shotton et al.]

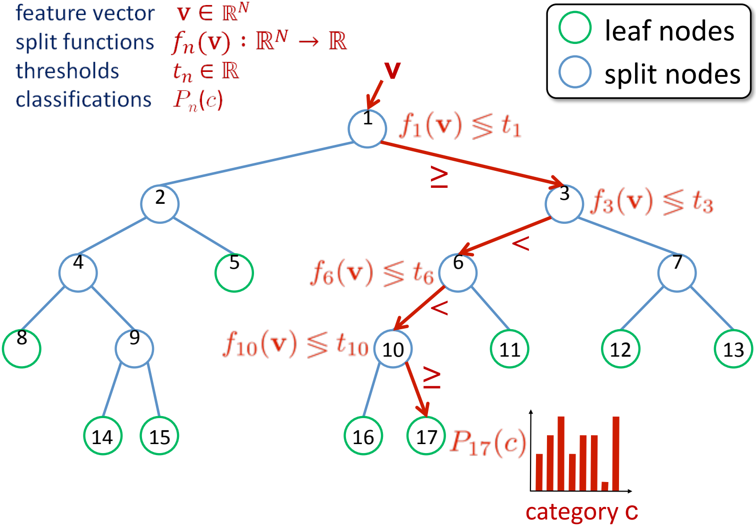
Is the grass wet?



[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.]

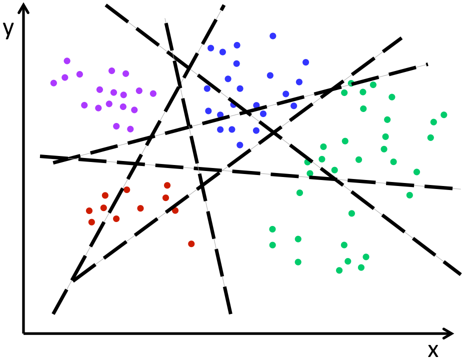
Decision Tree Pseudo-Code

```
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**

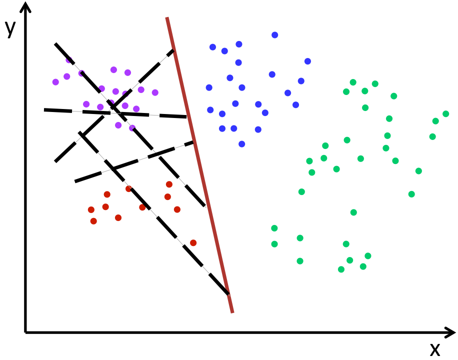


- feature vectors are x, y coordinates: $\mathbf{v} = [x, y]^T$
- split functions are lines with parameters a, b: $f_n(\mathbf{v}) = ax + by$
- threshold determines intercepts: t_n
- four classes: purple, blue, red, green

[Source: Shotton et al.]

Toy Example

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

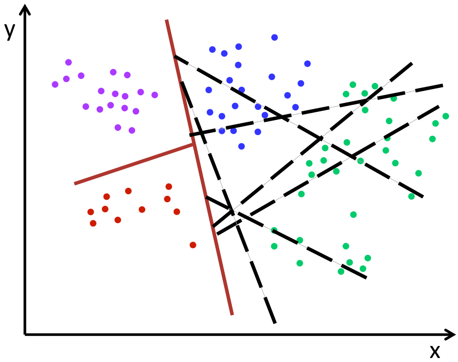


- feature vectors are x, y coordinates: $\mathbf{v} = [x, y]^T$
- split functions are lines with parameters a, b: $f_n(\mathbf{v}) = ax + by$
- threshold determines intercepts: t_n
- four classes: purple, blue, red, green

[Source: Shotton et al.]

Toy Example

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

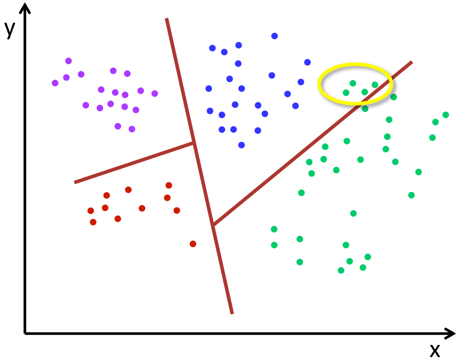


- feature vectors are x, y coordinates: $\mathbf{v} = [x, y]^T$
- split functions are lines with parameters a, b: $f_n(\mathbf{v}) = ax + by$
- threshold determines intercepts: t_n
- four classes: purple, blue, red, green

[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 $(\mathbf{v}_i, l_i)$

left split

$$I_l = \{i \in I_n \mid f(\mathbf{v}_i) < t\}$$

$$I_r = I_n \setminus I_l$$

right split

threshold

function of example i's feature vector

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

[Source: Shotton et al.]

Randomized Learning

$$\begin{array}{rrcl} \text{left split} & I_{\mathrm{l}} & = & \{i \in I_n \mid f(\mathbf{v}_i) < t\} \\ \text{right split} & I_{\mathrm{r}} & = & I_n \setminus I_{\mathrm{l}} \end{array}$$

- **Features** f(**v**) **chosen at random from feature pool** f **2** F
- **Thresholds** t **chosen in range** $t \in (\min_i f(\mathbf{v}_i), \max_i f(\mathbf{v}_i))$
- **Choose** f **and** t **to maximize gain in information**

$$\Delta E = -\frac{|I_{\mathrm{l}}|}{|I_n|} E(I_{\mathrm{l}}) - \frac{|I_{\mathrm{r}}|}{|I_n|} E(I_{\mathrm{r}})$$

Entropy E calculated from histogram of labels in I

[Source: Shotton et al.]

Details

How many features and thresholds to try?

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

When to stop growing the tree?

- maximum depth
- minimum entropy gain
- pruning

[Source: Shotton et al.]

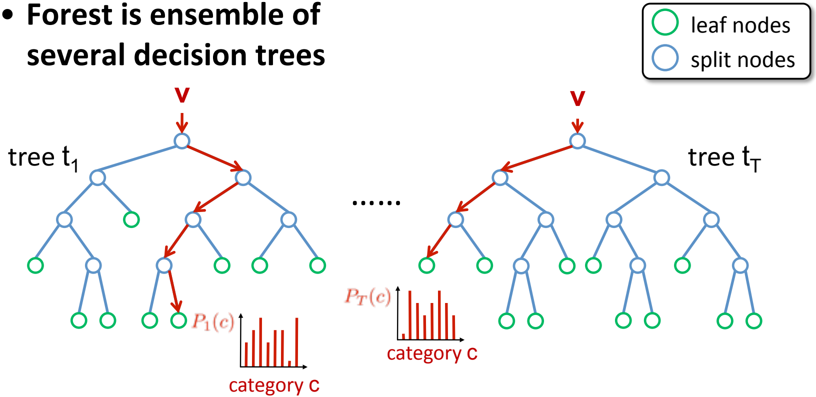
Randomized Learning Pseudo Code

```
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 forests of trees

- **Forest is ensemble of several decision trees**



– classification is $P(c|\mathbf{v}) = \frac{1}{T}\sum_{t=1}^{T} P_t(c|\mathbf{v})$

[Amit & Geman 97]
[Breiman 01]
[Lepetit *et al.* 06]

[Source: Shotton et al.]

Forest Pseudo

```
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
```

[Source: Shotton et al.]

Learning

- **Divide training examples into T subsets I_t μ I**
  - **improves generalization**
  - reduces **memory requirements** & **training time**

- **Train each decision tree t on subset I_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

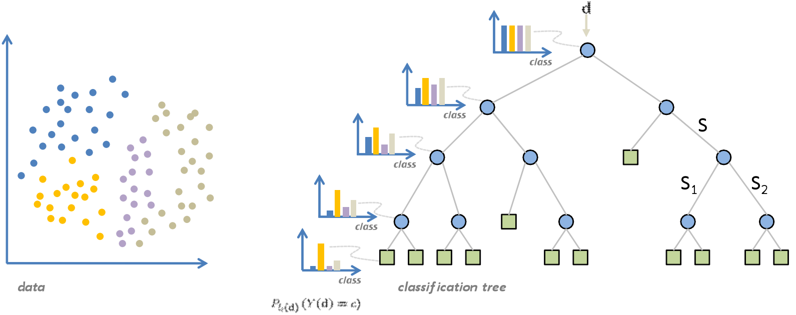
```
Forest LearnDF(countTrees, I)
   // allocate memory
   let forest = Forest(countTrees)

   // 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.]

Classification

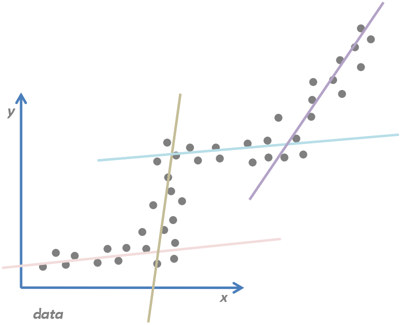
- **Trees can be trained for**
  - classification, regression, or clustering

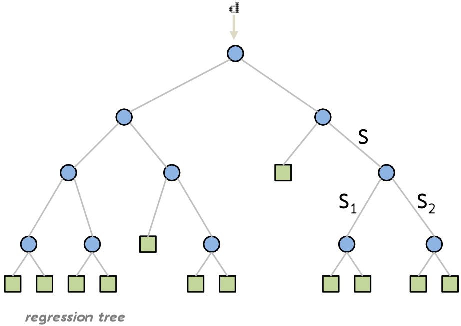
- **Change the object function**
  - information gain for classification: $I = H(S) - \sum_{i=1}^{2} \frac{|S_i|}{|S|} H(S_i)$ measure of distribution purity



[Source: Shotton et al.]

Regression





- Real-valued output *y*
- Object function: maximize $Err(S) - \sum_{i=1}^{2} \frac{|S_i|}{|S|} Err(S_i)$

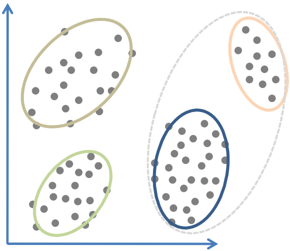
measure of fit of model

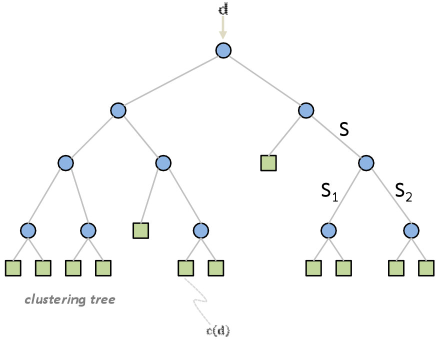
$$Err(S) = \sum_{j \in S} \left(y_j - y(x_j)\right)^2$$

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.]

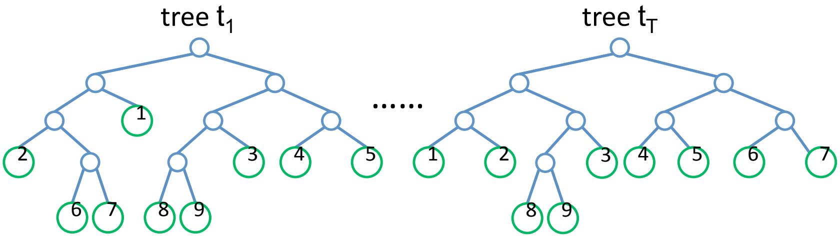
Clustering example [Moosmann et al. 06]

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

[Sivic *et al.* 03]
[Csurka *et al.* 04]

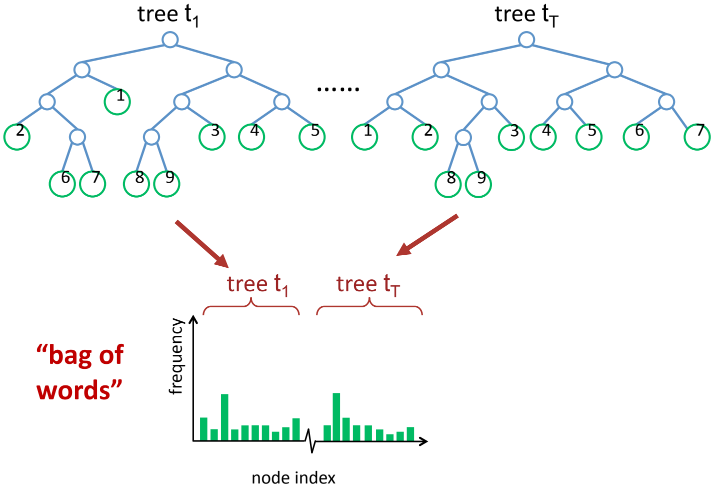
- **Randomized forests for clustering descriptors**
  - e.g. SIFT, texton filter-banks, etc.

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



[Source: Shotton et al.]

Clustering example [Moosmann et al. 06]

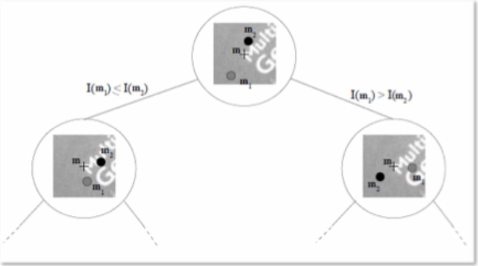


[Source: Shotton et al.]

Applications: keypoint detection [LePetit 06]

- **Wide-baseline matching as classification problem**

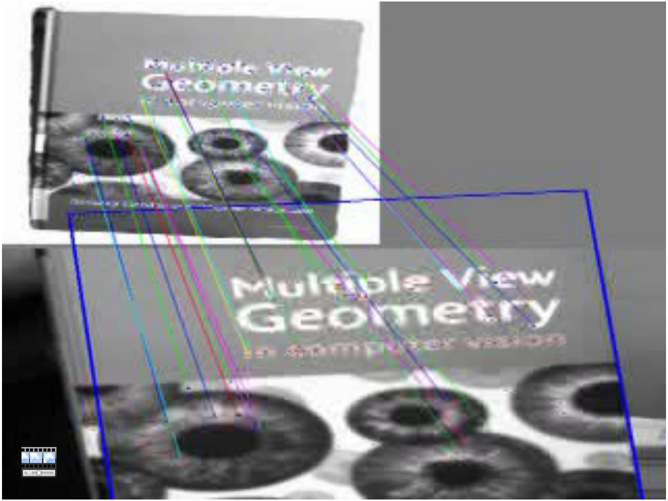
- **Extract prominent key-points in training images**



- **Forest classifies**
  - patches -> keypoints
- **Features**
  - pixel comparisons
- **Augmented training set**
  - gives robustness to patch scaling, translation, rotation

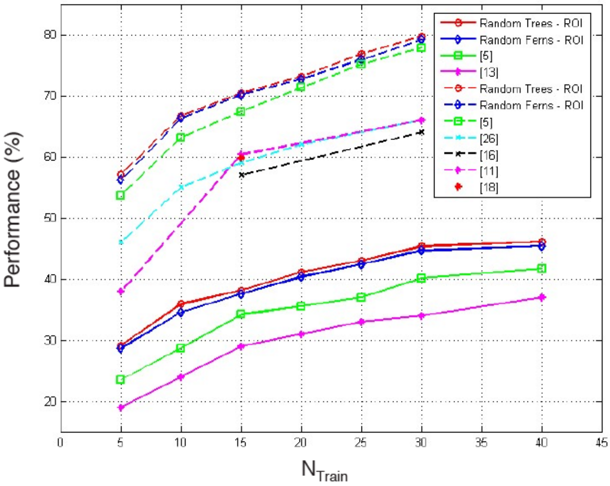
[Source: Shotton et al.]

Fast Keypoint Recognition

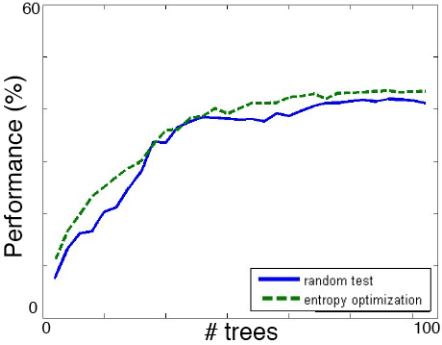


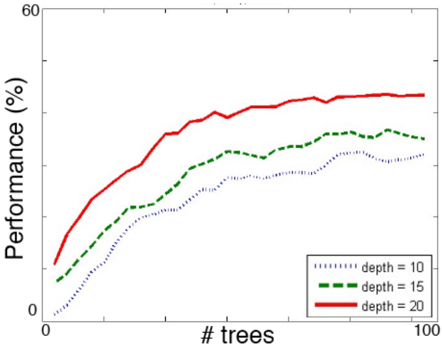
[Source: Shotton et al.]

Classification



Classification





Object Recognition Pipeline

extract features → **clustering** → **assignment** → **classification algorithm**

extract features	clustering	assignment
SIFT, filter bank	*k*-means	nearest neighbour
hand-crafted	unsupervised	

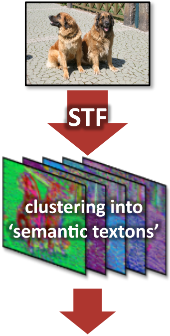
classification algorithm

SVM, decision forest, boosting

supervised

[Source: Shotton et al.]

Object Recognition Pipeline



classification algorithm

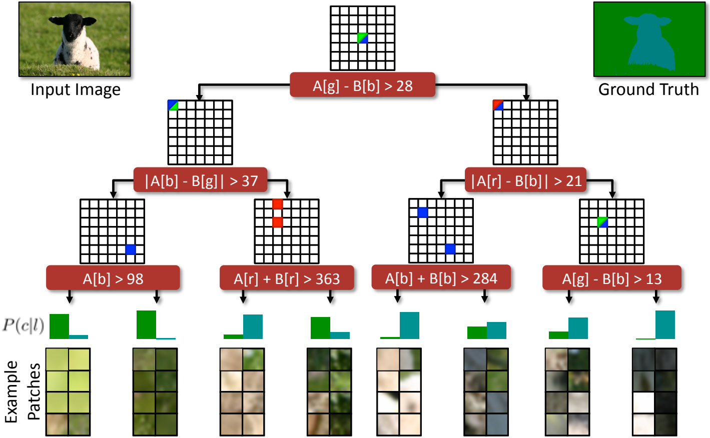
SVM, decision forest, boosting

Semantic Texton Forest (STF)

- decision forest for **clustering** & **classification**
- tree nodes have learned object category associations

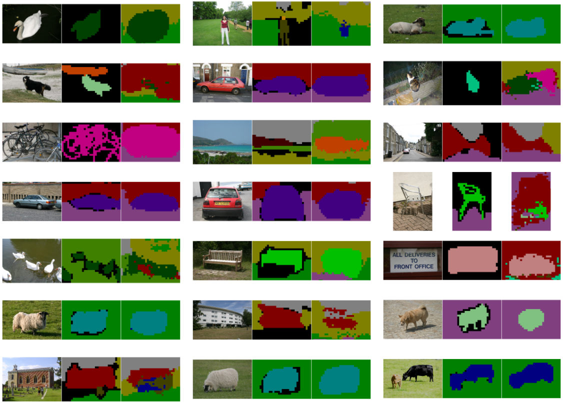
[Source: Shotton et al.]

Example Semantic Texton Forest



[Source: Shotton et al.]

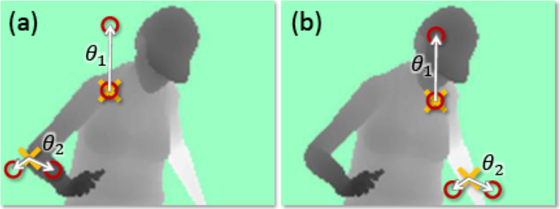
MSRC Dataset Results



building	grass	tree	cow	sheep	sky	airplane	water	face	car	boat
bicycle	flower	sign	bird	book	chair	road	cat	dog	body	

[Source: Shotton et al.]

Microsoft Kinect



$$P(c|I,\mathbf{x}) = \frac{1}{T}\sum_{t=1}^{T} P_t(c|I,\mathbf{x}) \,. \quad (2)$$

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 θ and thresholds τ).
2. Partition the set of examples $Q = \{(I,\mathbf{x})\}$ into left and right subsets by each ϕ:

$$Q_{\mathrm{l}}(\phi) = \{\, (I,\mathbf{x}) \mid f_\theta(I,\mathbf{x}) < \tau \,\} \quad (3)$$
$$Q_{\mathrm{r}}(\phi) = Q \setminus Q_{\mathrm{l}}(\phi) \quad (4)$$

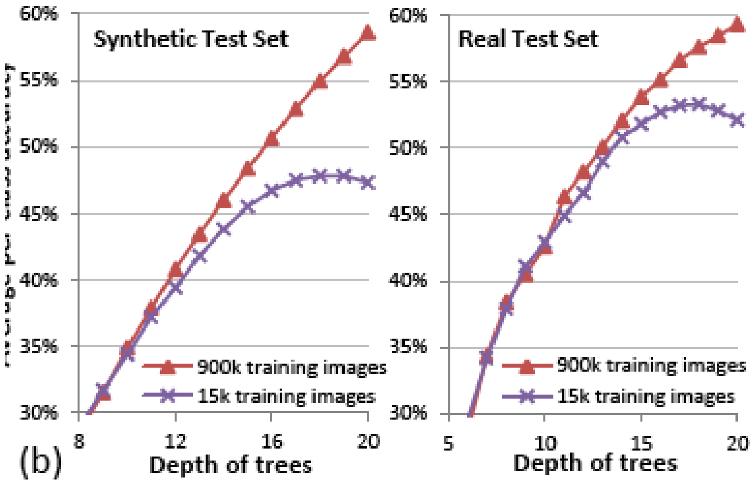
3. Compute the ϕ giving the largest gain in information:

$$\phi^\star = \operatorname*{argmax}_{\phi} G(\phi) \quad (5)$$
$$G(\phi) = H(Q) - \sum_{s\in\{\mathrm{l},\mathrm{r}\}} \frac{|Q_s(\phi)|}{|Q|} H(Q_s(\phi)) \quad (6)$$

   where Shannon entropy $H(Q)$ is computed on the normalized histogram of body part labels $l_I(\mathbf{x})$ for all $(I,\mathbf{x}) \in Q$.

4. If the largest gain $G(\phi^\star)$ is sufficient, and the depth in the tree is below a maximum, then recurse for left and right subsets $Q_{\mathrm{l}}(\phi^\star)$ and $Q_{\mathrm{r}}(\phi^\star)$.

Microsoft Kinect

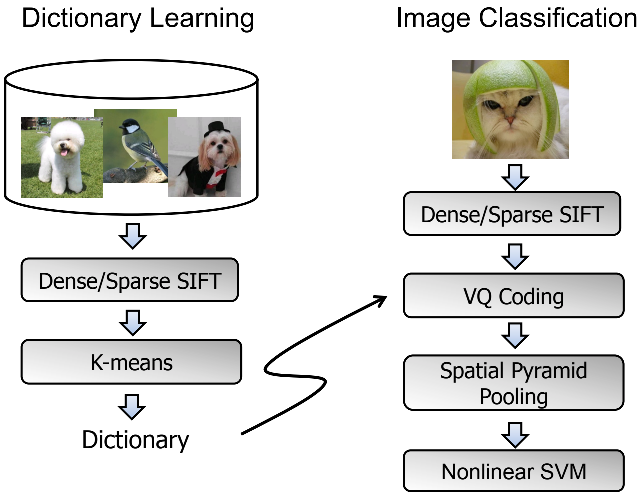


Learning Representations

Learning Representations

- Sparse coding
- Deep architectures
- Topic models

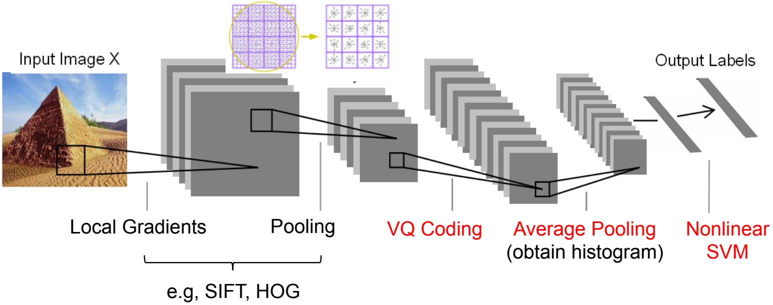
Image Classification using BoW



[Source: K. Yu]

BoW+SPM: the architecture

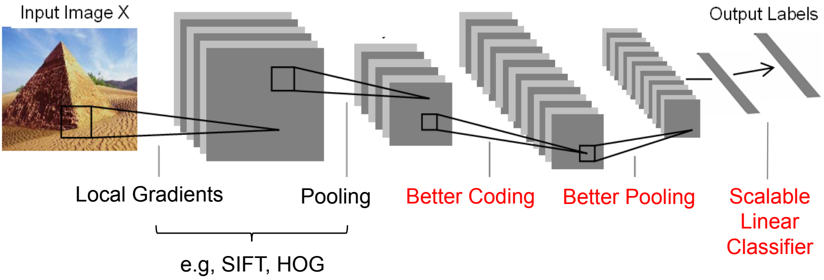
- Nonlinear SVM is not scalable
- VQ coding may be too coarse
- Average pooling is not optimal
- Why not learn the whole thing?



[Source: K. Yu]

Sparse Architecture

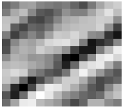
- Nonlinear SVM is not scalable → Scalable linear classifier
- VQ coding may be too coarse → Better coding methods
- Average pooling is not optimal → Better pooling methods
- Why not learn the whole → Deep learning



[Source: A. Ng]

Feature learning problem

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



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

[Source: A. Ng]

Learning an image representation

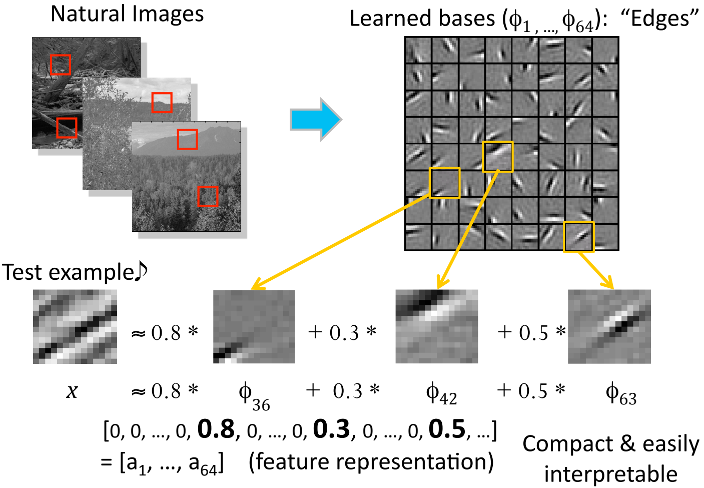
- Sparse coding [Olshausen & Field,1996]
- Input: Images $x^{(1)}, x^{(2)}, \cdots, x^{(m)}$ (each in $\Re^{n \times n}$)
- Learn: Dictionary of bases $\phi_1, \cdots, \phi_k$ (also $\Re^{n \times n}$), so that each input x can be approximately decomposed as:

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

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

[Source: A. Ng]

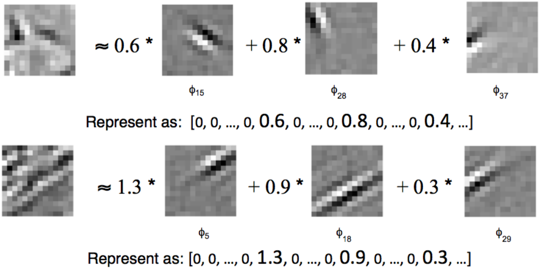
Sparse Coding Illustration



[Source: A. Ng]

More examples

- 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.



[Source: A. Ng]

Sparse Coding details

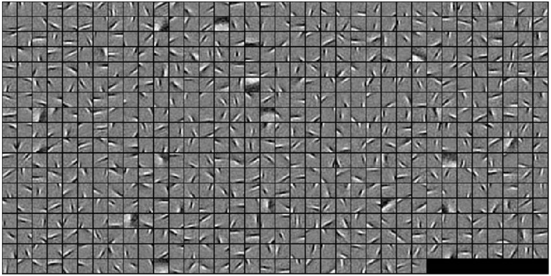
- Input: Images $x^{(1)}, x^{(2)}, \cdots, x^{(m)}$ (each in $\Re^{n \times n}$)
- Obtain dictionary elements and weights by

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

- Alternating minimization with respect to ϕ_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 $x^{(1)}, x^{(2)}, \cdots, x^{(m)}$ (each in $\Re^{n \times n}$)
- Learn: Dictionary of bases $\phi_1, \cdots, \phi_k$ (also $\Re^{n \times n}$),

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

Test time:

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

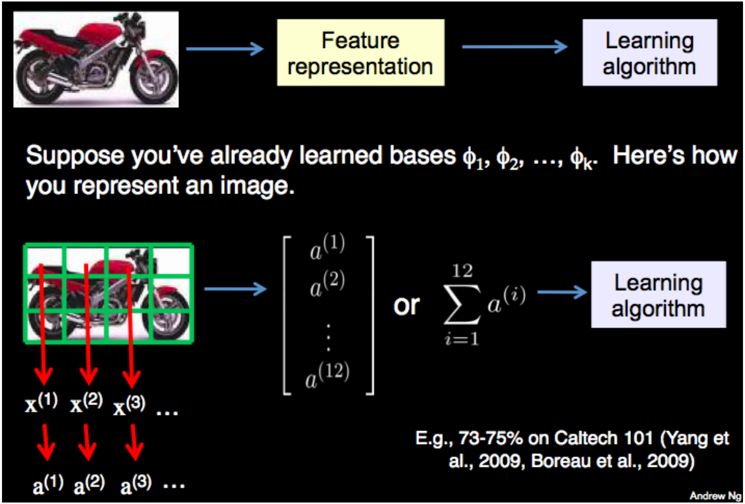
$$\min_{a} \sum_{i=1}^{m} \left(||x - \sum_{j=1}^{k} a_j \phi_j||^2 + \lambda \sum_{j=1}^{k} |a_j| \right)$$

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.

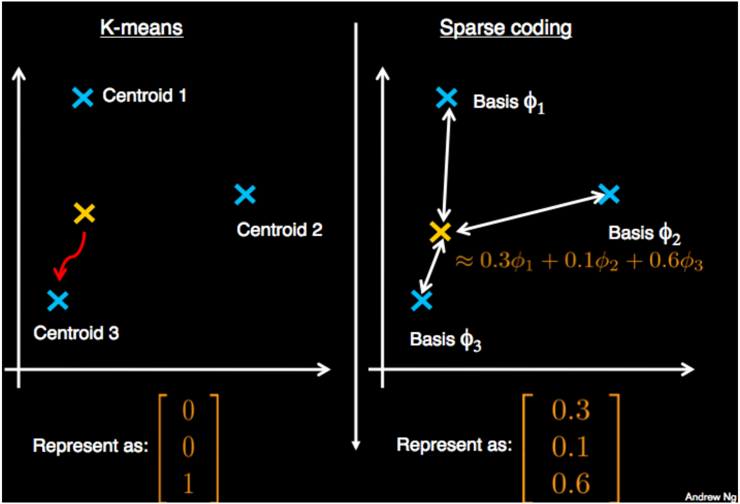
[Source: A. Ng]

Sparse Classification



[Source: A. Ng]

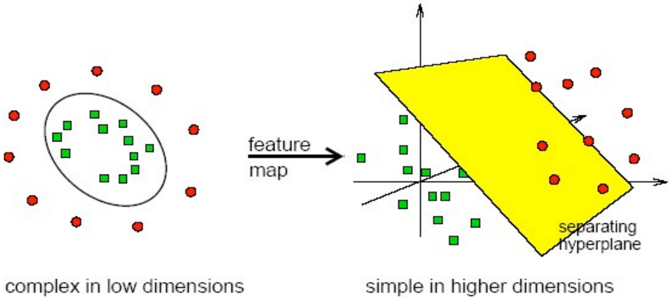
K-means vs sparse coding



[Source: A. Ng]

Why sparse coding helps classification?

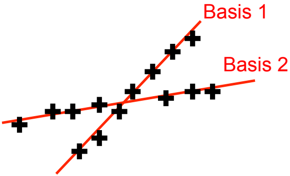
- The coding is a nonlinear feature mapping
- Represent data in a higher dimensional space
- Sparsity makes prominent patterns more distinctive



[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.

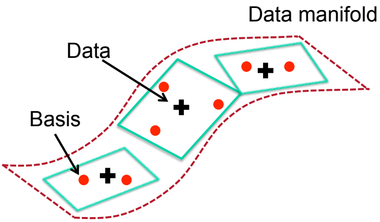


Both figures adapted from CVPR10 tutorial by F. Bach, J. Mairal, J. Ponce and G. Sapiro

[Source: K. Yu]

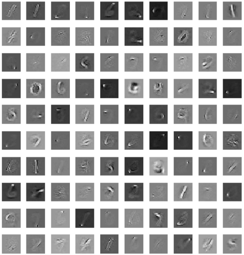
A geometric view to sparse coding

- 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

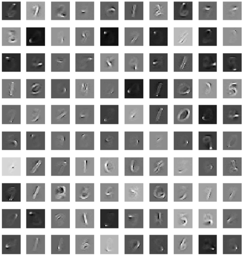


[Source: K. Yu]

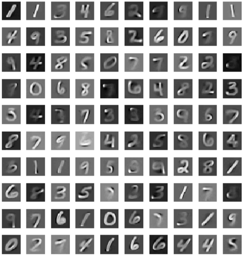
- When SC achieves the best classification accuracy, the learned bases are like digits – each basis has a clear local class association.



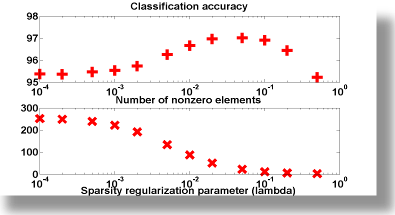
Error: 4.54%



Error: 3.75%



Error: 2.64%



The image classification setting for analysis

- Learning an image classifier is a matter of learning nonlinear functions on patches

$$\underbrace{f(x)}_{f.images} = w^T x = \sum_{i=1}^{m} a_i (w^T \phi^{(i)}) = \sum_{i=1}^{m} a_i \underbrace{f(\phi^{(i)})}_{f.patches}$$

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

Dense local feature

⇩

Sparse Coding

⇩

Linear Pooling

⇩

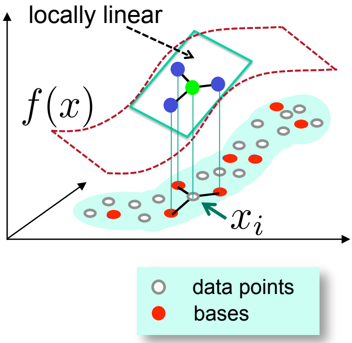
Linear SVM

[Source: K. Yu]

Nonlinear learning via local coding

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

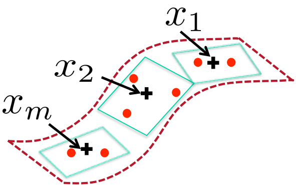
$$f(x_i) = \sum_{j=1}^{k} a_{i,j} f(\phi_j)$$



[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 $x_i \to a_{i,1}, \cdots, a_{i,k}$



3. Estimate the parameters

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

Nonlinear local learning via learning a global linear function

Local Coordinate Coding (LCC):

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

$$\underbrace{\left| f(x_i) - \sum_{j=1}^{k} a_{i,j} f(\phi_j) \right|}_{\text{Function approximation error}} \leq \alpha \underbrace{\left\| x_i - \sum_{j=1}^{k} a_{i,j} \phi_j \right\|}_{\text{Coding error}} + \beta \sum_{j=1}^{k} |a_{i,j}| \underbrace{\|x_i - \phi_j\|^2}_{\text{Locality term}}$$

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

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

[Source: K. Yu]

Results

- 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.

	Codebook size	256	512	1024
30 train	ScSPM	68.26	71.20	**73.20**
	LSPM	57.42	**58.81**	58.56
15 train	ScSPM	61.97	63.23	**69.70**
	LSPM	51.84	**53.23**	51.74

(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 $[\phi_j]_{j \in J(x)}$
Step 2 ensure low coding error:

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

[Source: K. Yu]

Results

Table 1. Image classification results on Caltech-101 dataset

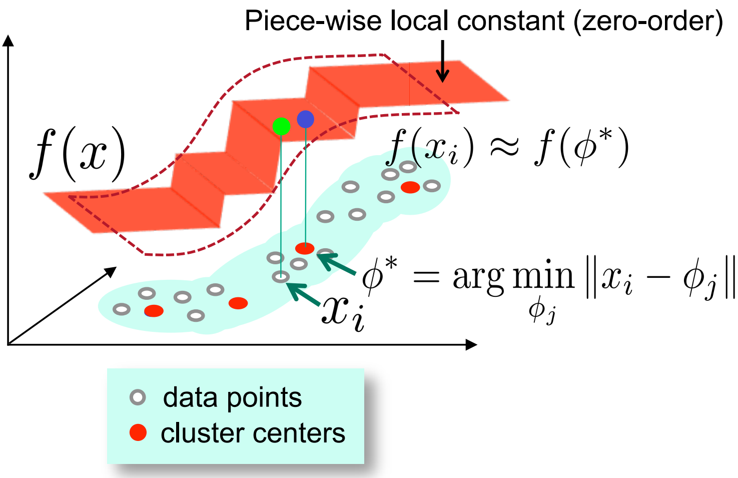
training images	5	10	15	20	25	30
Zhang [25]	46.6	55.8	59.1	62.0	-	66.20
Lazebnik [15]	-	-	56.40	-	-	64.60
Griffin [11]	44.2	54.5	59.0	63.3	65.8	67.60
Boiman [2]	-	-	65.00	-	-	70.40
Jain [12]	-	-	61.00	-	-	69.10
Gemert [8]	-	-	-	-	-	64.16
Yang [22]	-	-	**67.00**	-	-	73.20
Ours	**51.15**	**59.77**	65.43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

Table 2. Image classification results using Caltech-256 dataset

training images	15	30	45	60
Griffin [11]	28.30	34.10	-	-
Gemert [8]	-	27.17	-	-
Yang [22]	27.73	34.02	37.46	40.14
Ours	**34.36**	**41.19**	**45.31**	**47.68**

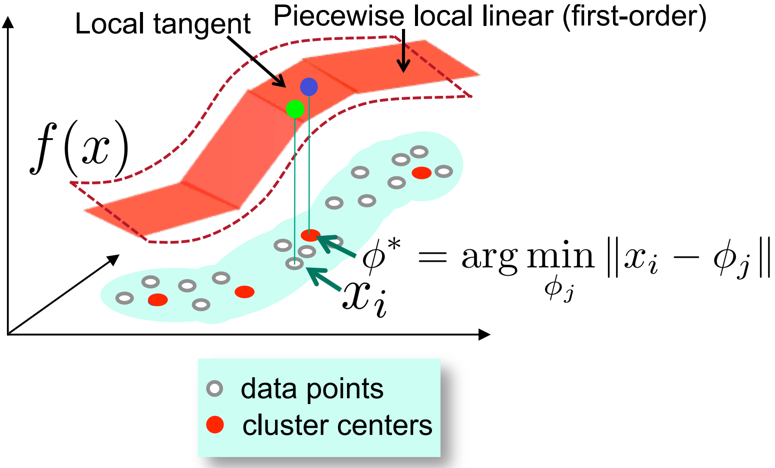
[Source: K. Yu]

Interpretation of BoW + linear classifier



[Source: K. Yu]

Support vector coding [Zhou et al, 10]



[Source: K. Yu]

Details...

- Let $[a_i, 1, \cdots, a_{i,k}]$ be the VQ coding of x_i

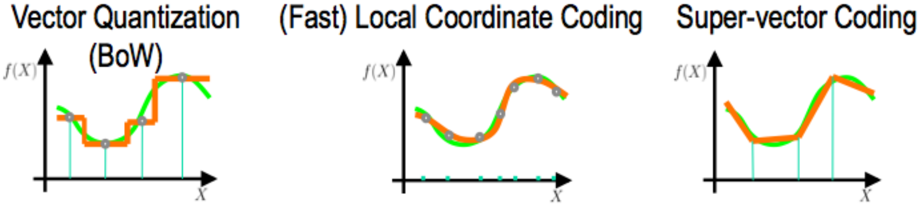
$$f(x_i) \approx \underbrace{\left[a_{i,1}(1, x_i - \phi_1), \ldots a_{i,k}(1, x_i - \phi_k)\right]}_{\text{Super-vector codes of data}} \times \underbrace{\begin{bmatrix} f(\phi_1) \\ \nabla f(\phi_1) \\ \vdots \\ f(\phi_k) \\ \nabla f(\phi_k) \end{bmatrix}}_{\text{Global linear weights to be learned}}$$

- 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

Classes	Ours	Best of Other Teams	Difference
Aeroplane	88.1	86.6	1.5
Bicycle	68.6	63.9	**4.7**
Bird	68.1	66.7	1.4
Boat	72.9	67.3	**5.6**
Bottle	44.2	43.7	0.5
Bus	79.5	74.1	**5.4**
Car	72.5	64.7	**7.8**
Cat	70.8	64.2	**6.6**
Chair	59.5	57.4	2.1
Cow	53.6	46.2	**7.4**
Diningtable	57.5	54.7	2.8
Dog	59.3	53.5	**5.8**
Horse	73.1	68.1	**5.0**
Motorbike	72.3	70.6	1.7
Person	85.3	85.2	0.1
Pottedplant	36.6	39.1	-2.5
Sheep	56.9	48.2	**8.7**
Sofa	57.9	50.0	**7.9**
Train	86.0	83.4	2.6
Tvmonitor	68.0	68.6	-0.6

ImageNet Large-scale Visual Recognition Challenge 2010

ImageNet: 1000 categories, 1.2 million images for training

[Source: K. Yu]

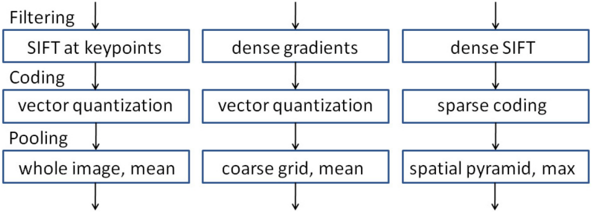
ImageNet Large-scale Visual Recognition Challenge 2010

- 150 teams registered worldwide, resulting 37 submissions from 11 teams
- SC achieved 52% for 1000 class classification.

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

[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]

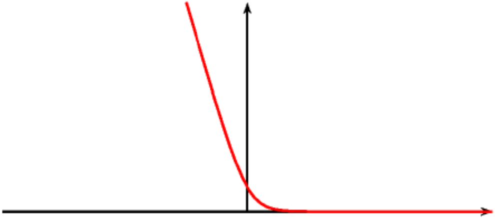
Discriminative Training

"Discriminative" training

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

$$\min_{\mathbf{D}_-,\mathbf{D}_+} \sum_i \mathcal{C}\Big(\lambda z_i \big(\mathbf{R}^\star(\mathbf{x}_i,\mathbf{D}_-) - \mathbf{R}^\star(\mathbf{x}_i,\mathbf{D}_+)\big)\Big),$$

where $z_i \in \{-1,+1\}$ is the label of $\mathbf{x}_i$.



Logistic regression function

[Source: Mairal]

Discriminative Dictionaries

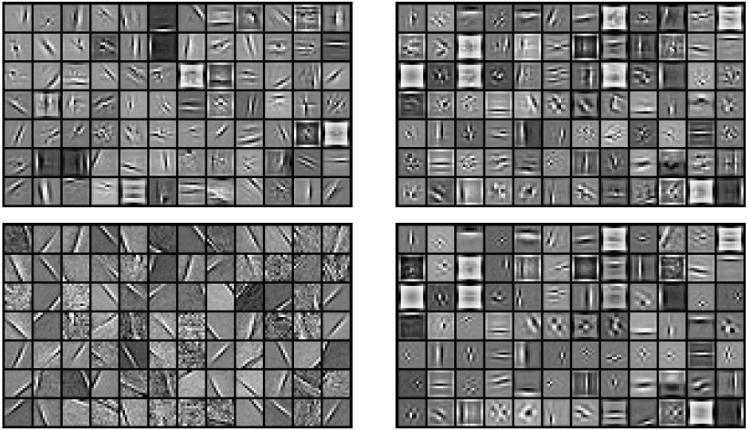


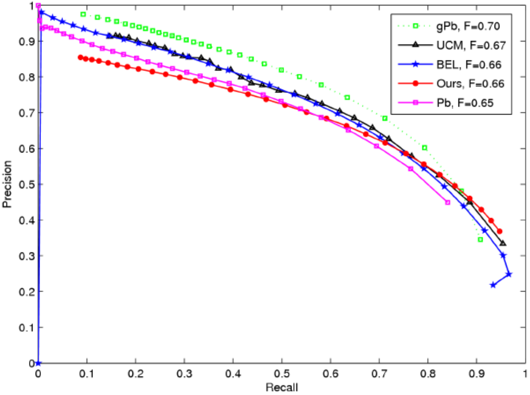
Figure: Top: reconstructive, Bottom: discriminative, Left: Bicycle, Right: Background

[Source: Mairal]



[Source: Mairal]

Application:Edge detection



[Source: Mairal]

Authentic

Fake

Fake

[Source: Mairal]

Predictive Sparse Decomposition (PSD)

- Feed-forward predictor function for feature extraction

$$\min_{a,\phi,K} \sum_{i=1}^{m} \left(||x^{(i)} - \sum_{j=1}^{k} a_j^{(i)} \phi_j||^2 + \lambda \sum_{j=1}^{k} |a_j^{(i)}| + \beta ||a - C(X, K)||_2^2 \right)$$

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

Learning is done by

1) Fix K and a, minimize to get optimal ϕ
2) Update a and K using optimal ϕ
3) Scale elements of ϕ to be unit norm.

[Source: Y. LeCun]

Predictive Sparse Decomposition (PSD)

- 12 x12 natural image patches with 256 dictionary elements

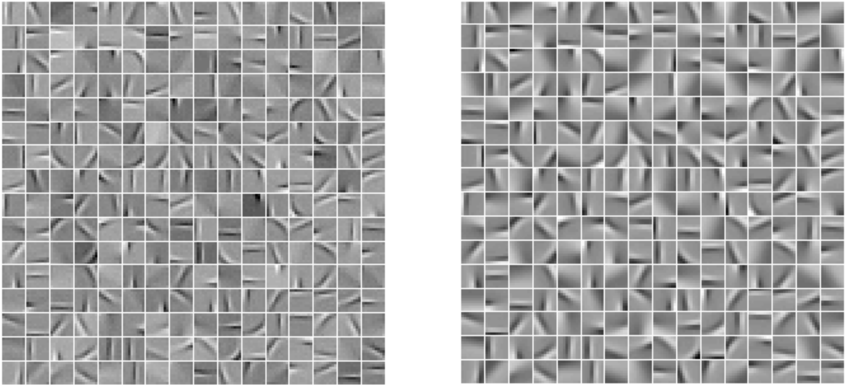


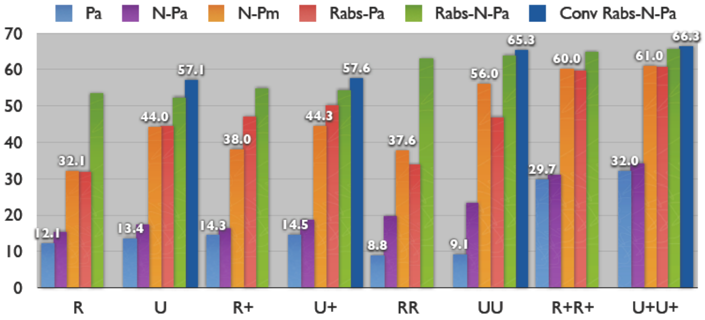
Figure: (Left) Encoder, (Right) Decoder

[Source: Y. LeCun]

Training Deep Networks with PSD

- Train layer wise [Hinton, 06]
  - $C(X, K^1)$
  - $C(f(K^1), K^2)$
  - $\cdots$
- 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



	R	RR	U	UU
Unsupervised	✘	✘	✓	✓
Random	✓	✓	✘	✘
Supervised	R^+	$R^+ R^+$	U^+	$U^+ U^+$

Convolutional Nets

- Sparse coding produces filters that are shifted version 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

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