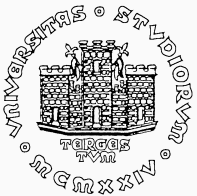
Computer Vision and Pattern Recognition

Course ID: 554SM – Fall 2018

Felice Andrea Pellegrino

University of Trieste
Department of Engineering and Architecture



554SM –Fall 2018

Lecture 8: Recognition

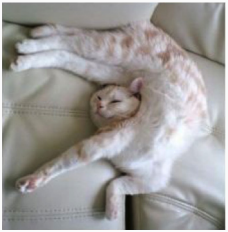
Recognition

- Recognition (in broad sense)
  - *object detection* (find all the regions in an image where a specific kind of object is likely to occur, for instance face detection and pedestrian detection);
  - *instance recognition* (recognize a known specific object potentially being viewed from a novel viewpoint);
  - *category-level recognition* (categorize images as belonging to a general class such as “cat” or “bicycle,” among many possible classes.)

Challenges

Illumination

Occlusion



Deformation

Background

Intraclass variation

slide credit: Fei-Fei Li, Andrej Karpathy and Justin Johnson.

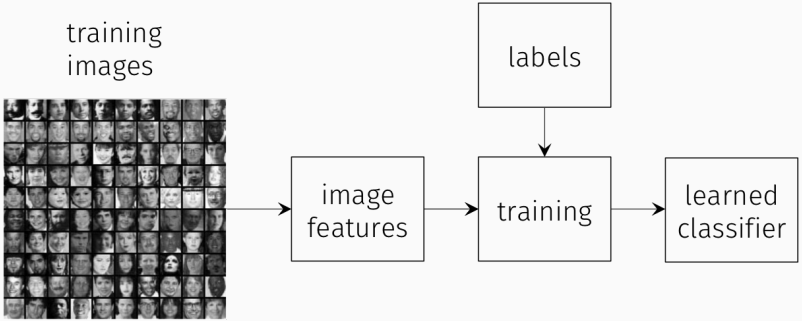
The machine learning framework

- The challenges render practically impossible to realize the recognition through handcrafted and explicit rules.
- The problem of recognition can be cast as a **Machine Learning** problem, where a classifier is designed and trained:

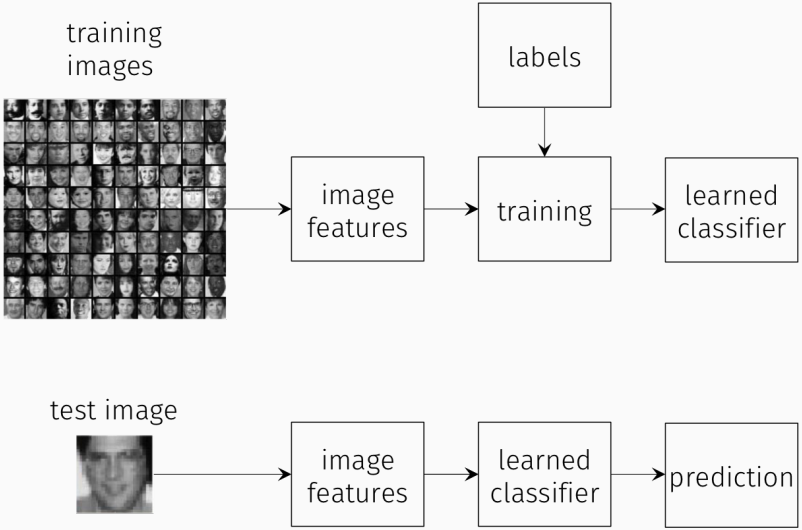
$$y = f(x)$$

  where f is a *prediction function*, x is a *vector of features* extracted from the image (or, possibly, the image itself) and y is a prediction.
- The simplest case is the *supervised learning*, where a set of labeled examples is provided and we distinguish two phases
  - **Training**: given a training set $\mathcal{S} = \{(x_i, y_i),\ i = 1 \dots l\}$, estimate the prediction function f based on the prediction error on the training set;
  - **Testing**: apply f to a never seen before test example x and output the predicted value $y = f(x)$.

Simple pipeline



Simple pipeline



Object detection

Window-based detection

Window-based detection:

- scan the whole image, extracting rectangles (windows) of different size and location;
- evaluate each window through a classifier, possibly after extracting a feature vector.

Window-based detection

Window-based detection:

- scan the whole image, extracting rectangles (windows) of different size and location;
- evaluate each window through a classifier, possibly after extracting a feature vector.

Window-based detection

Window-based detection:

- scan the whole image, extracting rectangles (windows) of different size and location;
- evaluate each window through a classifier, possibly after extracting a feature vector.

Window-based detection

Window-based detection:

- scan the whole image, extracting rectangles (windows) of different size and location;
- evaluate each window through a classifier, possibly after extracting a feature vector.

Strengths and weaknesses

Strengths:

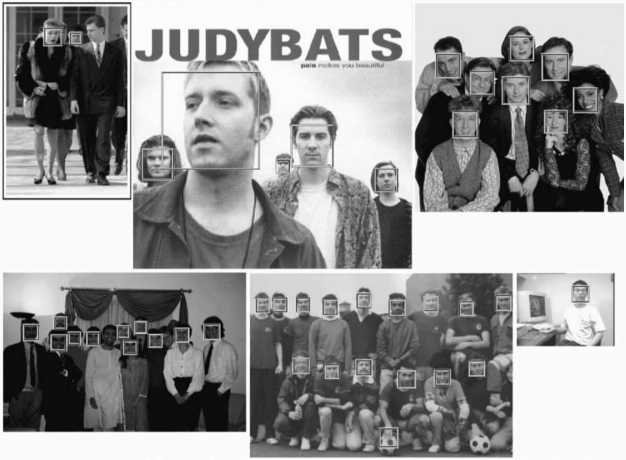
- simple protocol;
- proven to be effective in some important applications, for instance
  - face detection;
  - pedestrian detection.

Weaknesses:

- Computationally expensive (a lot of windows to be evaluated per image, for instance with 250K locations, 16 orientations and 4 scales we get 16M evaluations per image);
- many windows imply that the false positive rate must be very low (for instance, if the false positive rate is 10^{-6} (one per million), we get 16 false positives per image);
- non suitable for non-rigid objects;
- context is lost (due to independent evaluation of each window).

The Viola-Jones face detector

The Viola-Jones face detector (Viola and Jones, 2001) represents a landmark in Computer Vision and provides a time-efficient and accurate window-based detector.

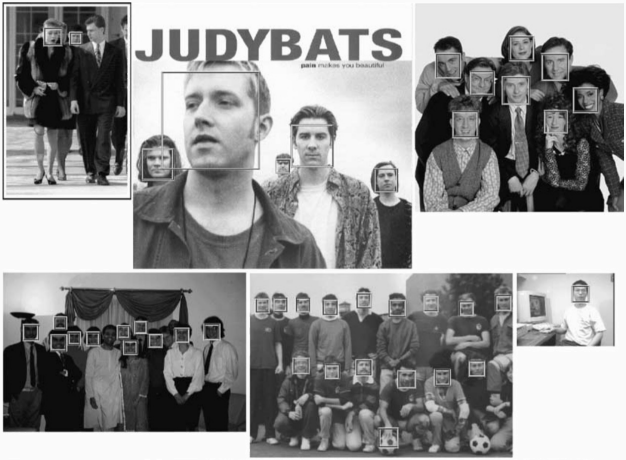


The Viola-Jones face detector

The Viola-Jones face detector (Viola and Jones, 2001) represents a landmark in Computer Vision and provides a time-efficient and accurate window-based detector.

Key ideas

- use **Haar-like features**;
- use **integral images** for fast feature computation;
- use **boosting** (AdaBoost);
- use **elementary weak classifiers** (classifier = feature);
- use a **cascade** of boosted classifiers to **reject negative instances as soon as possible**.

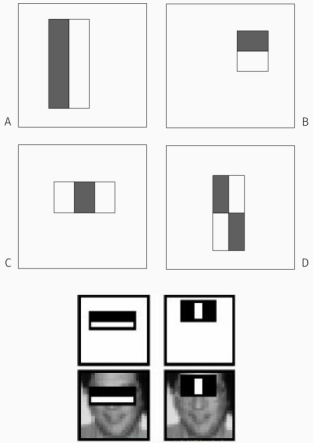


Haar-like features

The features are inspired to Haar wavelets, introduced in Computer Vision by Oren et al. (1997).

In the original formulation, they are a basis for the space of images, while viola and Jones use an overcomplete set of features:

- two-rectangle features (A,B);
- three-rectangle features (C);
- four-rectangle features (D);

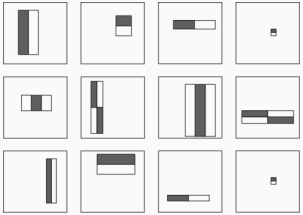


Haar-like features

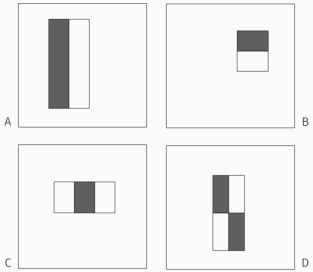
The features are inspired to Haar wavelets, introduced in Computer Vision by Oren et al. (1997).

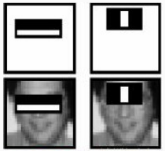
In the original formulation, they are a basis for the space of images, while viola and Jones use an overcomplete set of features:

- two-rectangle features (A,B);
- three-rectangle features (C);
- four-rectangle features (D);
- Viola and Jones use all possible filter parameters: position, scale, and type, within a 24×24 sliding window:

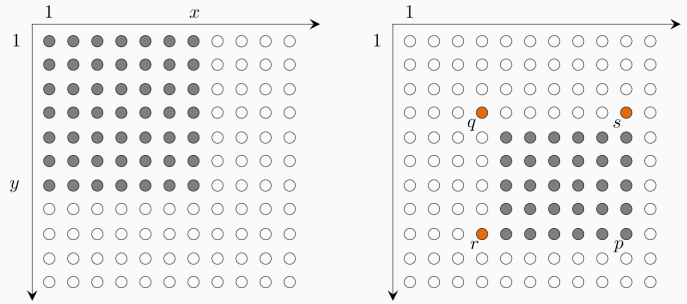


thus the total number of features is $\approx 160K$.





Integral image

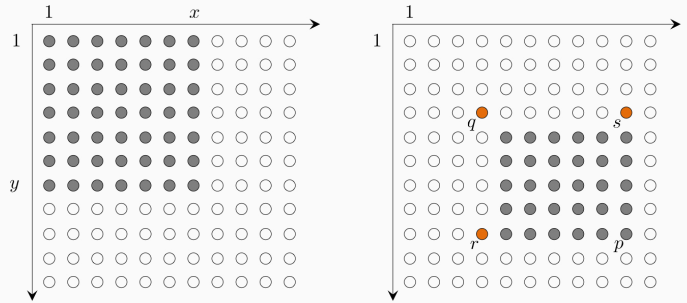


Rectangle features can be computed very rapidly using an intermediate representation for the image called the *integral image*.

For a given image I, the integral image at location x, y contains the sum of the pixels above and to the left of x, y:

$$I_{\text{int}}(x, y) = \sum_{i \leq x, j \leq y} I(i, j).$$

Integral image

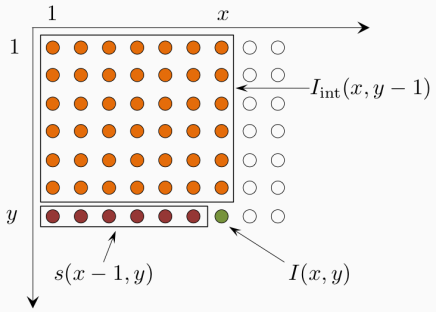


The sum of all pixel values in a rectangular window is a linear combination of four values of the integral image (corresponding to the corners of the rectangle). Consider a window W defined by the four points p, q, r, s (figure above, right). The sum S_W of all the pixel in W is

$$S_W = I_{\text{int}}(p) - I_{\text{int}}(r) - I_{\text{int}}(s) + I_{\text{int}}(q).$$

Thus only three additions are required, independent of the rectangle size. Thus there is **no need for scaling the image**.

Integral image (cont.)



The integral image can be computed in one pass over the original image, using the recurrences:

$$s(x, y) = s(x-1, y) + I(x, y)$$
$$I_{\text{int}}(x, y) = I_{\text{int}}(x, y-1) + s(x, y)$$

where $s(\cdot, y)$ is the cumulative sum of row y.

Boosting

Boosting (Freund, 1995) is a general technique which involves training a series of increasingly discriminating simple classifiers and then blending their outputs.

In boosting, the classifier $h(x)$ is the weighted sum of simple *weak learners*:

$$h(x) = \sum_{1}^{m} \alpha_j h_j(x).$$

Weak learners are extremely simple functions of the input and are not expected to contribute much (in isolation) to the classification performance.

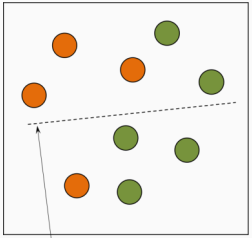
Viola and Jones restrict the weak learner to the set of **classification functions each of which depend on a single feature**, i.e. the generic weak learner takes the form:

$$h_j(x, f_j, p_j, \theta_j) = \begin{cases} 1 & \text{if} \quad p_j f_j(x) < p\theta_j \\ 0 & \text{otherwise} \end{cases}$$

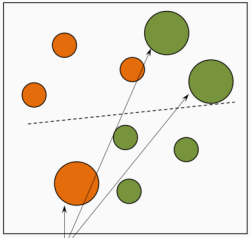
where $f_j(x)$ is the value of feature j in image x, p_j is a polarity and θ_j a threshold.

Weak learners are selected during the training phase by **iteratively re-weighting the training instances**.

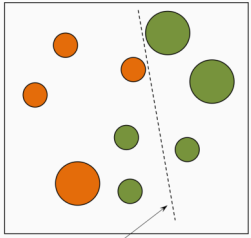
Boosting: intuition



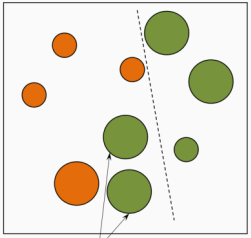
weak classifier 1



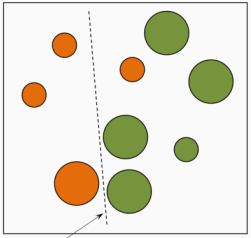
weights increased



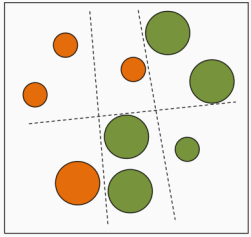
weak classifier 2



weights increased



weak classifier 3



final classifier is a combination of weak classifiers

Boosting: sketch

- Initially, weight each training example equally
- In each boosting round:
  - Find the weak learner that achieves the lowest weighted training error
  - Raise weights of training examples misclassified by current weak learner
- Compute final classifier as linear combination of all weak learners (weight of each learner is directly correlated to its accuracy)
- Exact formulas for re-weighting and combining weak learners depend on the particular boosting scheme (e.g., AdaBoost (Freund and Schapire, 1997))

Algorithm Boosting algorithm (AdaBoost) employed in (Viola and Jones, 2001)

Input: Example images $(x_1, y_1), \ldots (x_n, y_n)$ where $y_i = 0, 1$ for negative and positive example respectively.

Output: A strong classifier composed by T weak classifiers.

1: Initialize weights $w_{1,i} = \frac{1}{2m}, \frac{1}{2l}$ for $y_i = 0, 1$ respectively, where m and l are the number of negatives and positives respectively.

2: **for** $t = 1, \ldots, T$ **do**

3: Normalize the weights, $w_{t,i} \leftarrow \frac{w_{t,i}}{\sum_{j=1}^{n} w_{t,j}}$.

4: For each feature j, train a classifier h_j which is restricted to using a single feature. The error is evaluated w.r.t. the current weights:

$$\epsilon_j = \sum_i w_i |h_j(x_i) - y_i|.$$

5: Choose the classifier h_t with the lowest error ϵ_t .

6: Update the weights:

$$w_{t+1,i} = w_{t,i} \beta_t^{1-e_i}$$

where $e_i = 0$ if example x_i is classified correctly, $e_i = 1$ otherwise, and $\beta_t = \frac{\epsilon_t}{1 - \epsilon_t}$.

7: **end for**

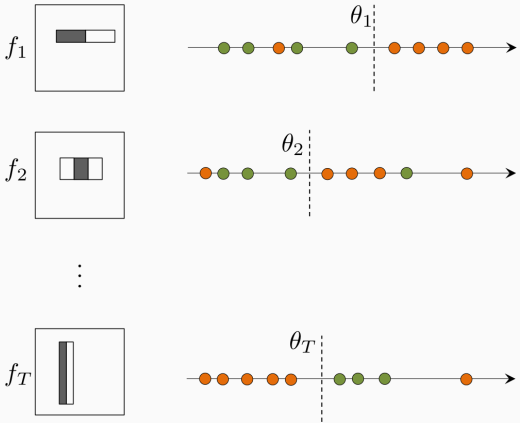
8: The final strong classifier is:

$$C(x) = \begin{cases} 1 & \text{if} \quad \sum_{t=1}^{T} \alpha_t h_t(x) \geq \frac{1}{2} \sum_{t=1}^{T} \alpha_t \\ 0 & \text{otherwise} \end{cases}$$

where $\alpha_t = \log \frac{1}{\beta_t}$.

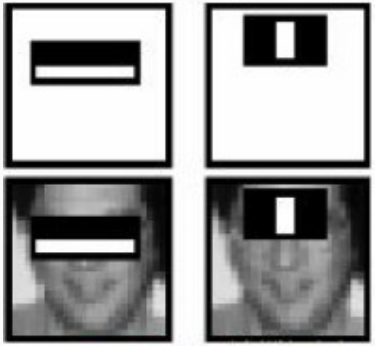
Boosting as a greedy feature selection

- Weak learners are restricted to use single features, thus the recursive selection of classifiers is actually a (greedy) **feature selection**.
- For each t, the goal is to select the single rectangle feature and threshold that best separates positive (faces) and negative (non-faces) training examples, in terms of weighted error.
- Thus, the first feature picked up is the one, among the 160K features, that performs best in the un-weighted training set.
- subsequent features are those that perform best in the weighted training sets.



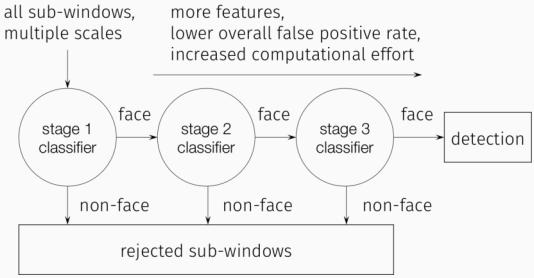
Boosting as a greedy feature selection

- Weak learners are restricted to use single features, thus the recursive selection of classifiers is actually a (greedy) **feature selection**.
- For each t, the goal is to select the single rectangle feature and threshold that best separates positive (faces) and negative (non-faces) training examples, in terms of weighted error.
- Thus, the first feature picked up is the one, among the 160K features, that performs best in the un-weighted training set.
- subsequent features are those that perform best in the weighted training sets.



First two features selected.

Cascade of classifiers



- Goal: speeding up the evaluation of the many sub-windows of an entire image.
- Observation: **most of the windows in an image are non-faces**.
- Idea: build a multi-stage classifier (*cascade*) and tune the stages in order to **reject the negative instances as soon as possible**.

Training the cascade

- The false positive rate of a trained cascade is

$$F = \prod_{i=1}^{K} F_i$$

  where K is the number of stages and F_i is the false positive rate of ith classifier on the examples that get through it.

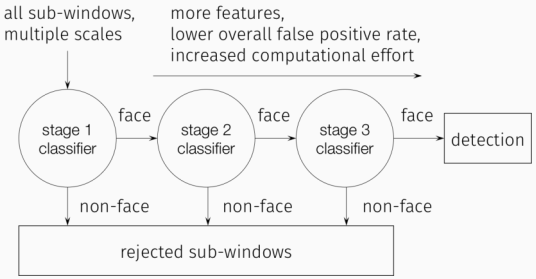
- The detection rate of a trained cascade is

$$D = \prod_{i=1}^{K} D_i$$

  where K is the number of stages and D_i is the detection rate of ith classifier on the examples that get through it.

- Training the cascade requires **choosing F_i and D_i in order to encourage the early rejection of the negative instances** (those that are easier to classify as negative).

Training the cascade (cont.)



Sketch of the training algorithm:

- set target detection and false positive rates for each stage;
- keep adding features to the current stage until its target rates have been met;
  - need to **lower AdaBoost threshold to maximize detection** (instead of minimizing the classification error);
  - evaluate actual rates on a **validation set**;
- if the overall false positive rate is not low enough, then add another stage;
- use **false positives from current stage as the negative training examples for the next stage**;

Some figures

- Training with 4916 positives (of 24×24 pixels) and a maximum of 6000 negatives per stage;
- final classifier is a 38 layer cascade;
- total number of feature of the cascade: 6060;
- the first classifier in the cascade **uses two features and rejects 50% of non-faces** (i.e. has a FPR of 50%) while detecting correctly close to 100% of faces;
- the second classifier uses **ten features and rejects 80% of non-faces** (i.e. has a FPR of 20%) while detecting correctly close to 100% of faces.

Viola-Jones detector: summary

- A milestone of real-time object detection
- Slow training, very fast detection
- Key ideas
  - **Integral images** for fast feature evaluation
  - **Boosting** for feature selection
  - **Attentional cascade** of classifiers for fast rejection of non-face windows

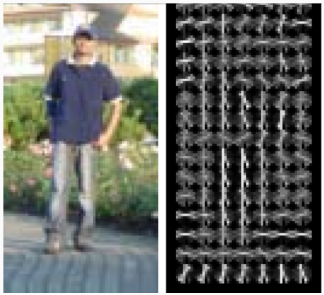
Histograms of oriented gradients (HoG)

Another well known and widely employed window-based approach for object detection is the one proposed by Dalal and Triggs (2005).

- Based on histograms of oriented gradients features;
- employs linear support vector machine as classifier;
- test case of the original paper was pedestrian detection, but since then employed for other objects too.

In brief:

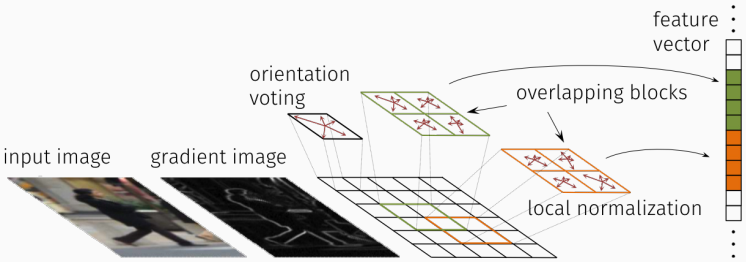
- map each grid cell in the input window to a histogram counting the gradients per orientation.
- train an SVM using training set of pedestrian vs. non-pedestrian windows.

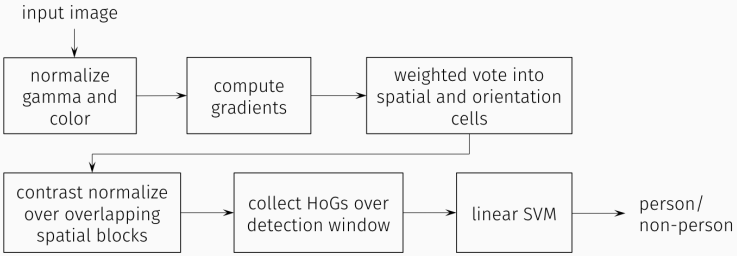


Processing chain

Overview:

- compute gradients in the region to be described;
- put them in bins according to orientation;
- group the cells into large blocks;
- normalize each block;
- train SVM using the obtained descriptors.

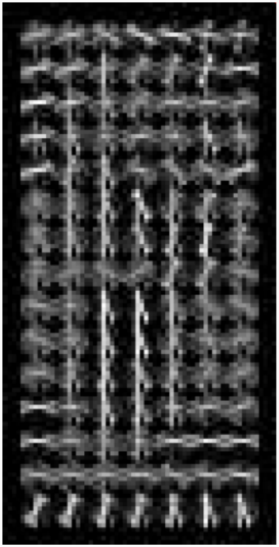




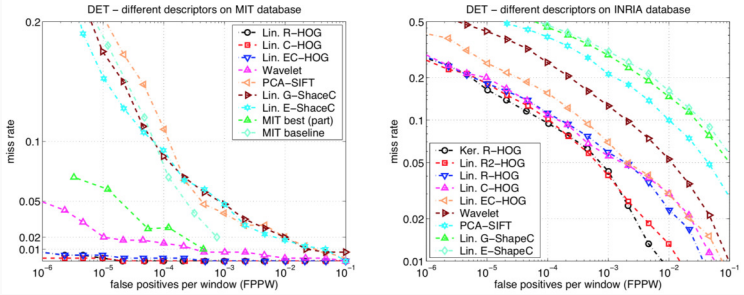
Details

- Deals with color images by taking the maximum gradient over the channels;
- gradient filter $[-1 \;\; 0 \;\; 1]$ with no smoothing;
- 9 orientation bins in 0°-180°;
- 8×8 pixel cells;
- blocks of 4 cells;
- block stride of 8 pixels;
- 64×128 detection window;
- linear SVM.

Notice in the figure that there is a border where no histogram is computed. Thus the number of cells per windows is ×15. Since there are 9 orientations and each histogram is normalized with respect to 4 different regions, the total number of features is 7×15×9×4=3780.

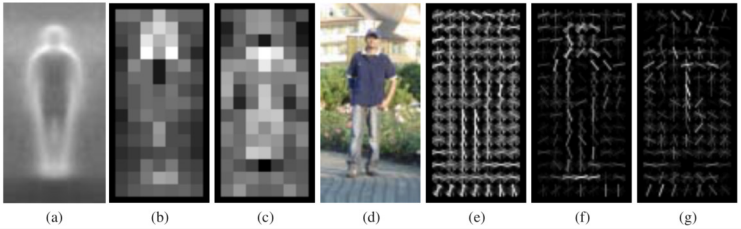


Performance



- HoG features lead to very low false-positive rates compared to other features, for the same detection rate;
- nearly perfect results on the MIT pedestrian dataset;
- detection rate of 90% at 10^{-4} false positives per window (INRIA dataset).

Inspecting the features



(a) average gradient image over training examples;

(b) each pixel shows max positive SVM weight in the block centered on that pixel;

(c) same as (b) for negative SVM weights;

(d) test image;

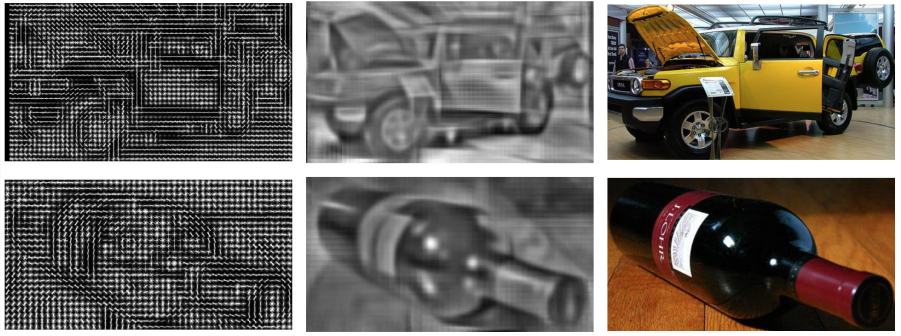
(e) its HoG descriptor;

(f) HoG descriptor weighted by positive SVM weights (head, shoulders, leg silhouettes are the most important cues);

(g) HoG descriptor weighted by negative SVM weights (vertical gradients inside the person count as negative).

Inverting the feature map

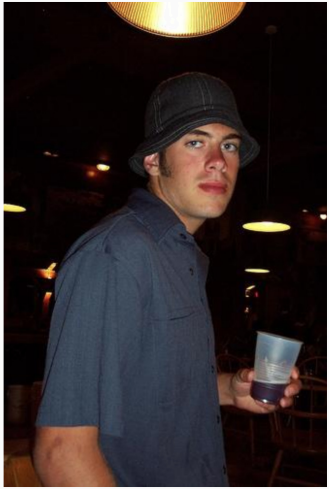
Vondrick et al. (2013) present a method for **inverting the HoG feature map**, i.e. finding images that produce a given HoG descriptor.



HoG descriptor (left), reconstruction (center), original image (right).

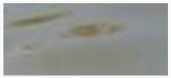
See the HOGgles website for more examples: `http://www.cs.columbia.edu/~vondrick/ihog/`

Inverting the feature map (cont.)

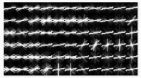


The visualization reveals that HoG can capture invisible (to humans) things.

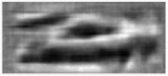
Inverting the feature map (cont.)



original



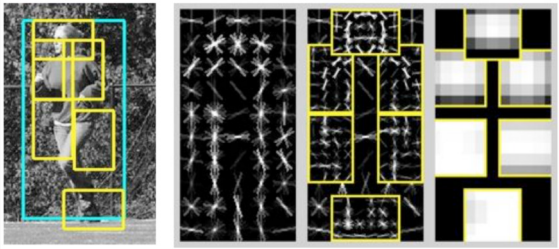
HoG

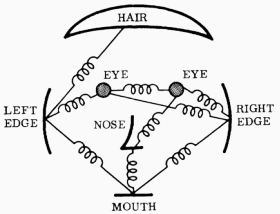


visualization

Sometimes, the visualization provides explanations to an undesirable behavior.

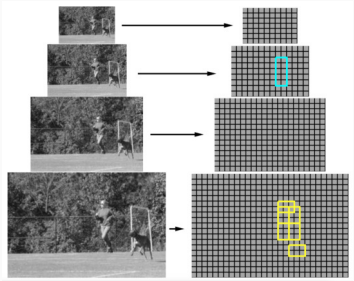
Deformable part models



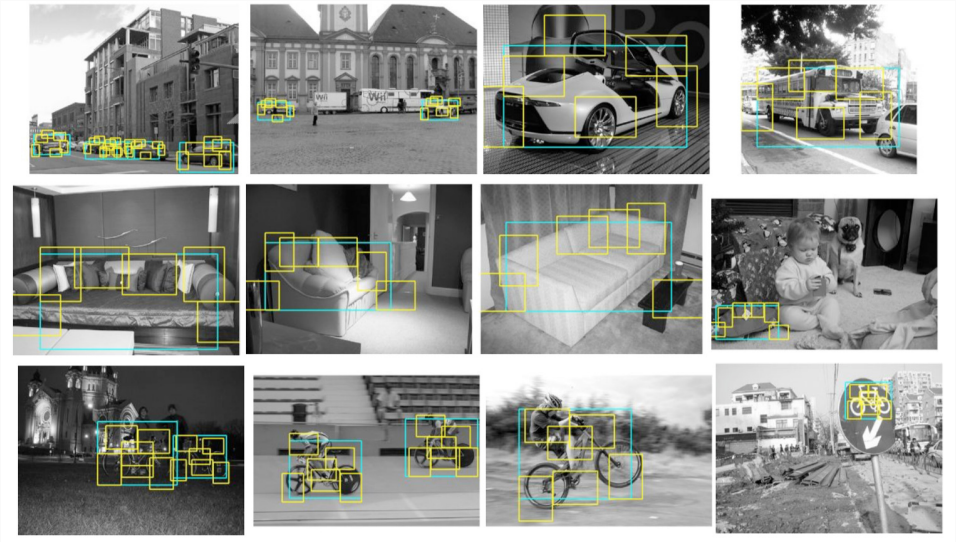


The *deformable part model* approach (Felzenszwalb et al., 2008) exploits HoG and the pictorial structure idea of Fischer and Elschlager (1973).

- build a HoG feature pyramid;
- detect root object at coarse scale;
- match parts at finer scale (by minimizing an energy function);
- score of the detection depends on:
  - detection score of the root;
  - detection score of the parts;
  - deformation cost.
- parts are learned during training (no need for labeling parts).

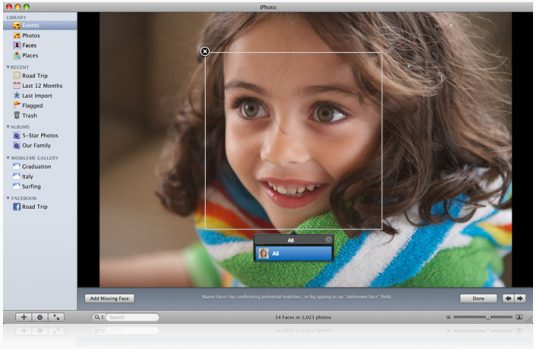


Deformable part models (cont.)



Instance recognition

Face recognition



Face recognition is one of the recognition tasks where computers had the most success (Szeliski, 2010). In the following we describe two simple but effective approaches:

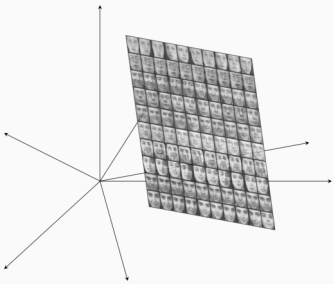
- Eigenfaces (Turk and Pentland, 1991);
- Fisherfaces (Belhumeur et al., 1997).

The space of faces

- Take a set of frontal faces images, each of size, say, 256×256;
- each image can be thought of a point in a space $\mathcal{I}$ of dimension $256 \times 256 = 65536$;
- however, **relatively few high dimensional points correspond to valid face images**;

The space of faces

- Take a set of frontal faces images, each of size, say, 256×256;
- each image can be thought of a point in a space $\mathcal{I}$ of dimension $256 \times 256 = 65536$;
- however, **relatively few high dimensional points correspond to valid face images**;
- both the approaches (Eigenfaces and Fisherfaces) model the subset $\mathcal{F}$ of $\mathcal{I}$ corresponding to faces as a low dimensional affine subspace of $\mathcal{I}$.



The space of faces

- Take a set of frontal faces images, each of size, say, 256×256;
- each image can be thought of a point in a space $\mathcal{I}$ of dimension 256×256=65536;
- however, **relatively few high dimensional points correspond to valid face images**;
- both the approaches (Eigenfaces and Fisherfaces) model the subset $\mathcal{F}$ of $\mathcal{I}$ corresponding to faces as a low dimensional affine subspace of $\mathcal{I}$.
- provided that we find the affine subspace we can:
  - **detect** faces by computing the **distance from the query image to the subspace**;
  - **recognize** faces by **projecting onto the subspace** the query image and **computing the distances along the subspace** from the sample faces.

Eigenfaces

- Take a collection of training images (in the following, treat an image as a 1D vector of $\mathbb{R}^d$):

$$\mathcal{S} = \{x_j \in \mathbb{R}^d, \quad j = 1, \ldots, N\}.$$

- Compute the mean face $m = \dfrac{\sum_j x_j}{N}$.
- Assumption: the difference from the mean face belongs to the subspace spanned by the first $k \ll d$ directions of maximum variance.
- Apply Principal Component Analysis to find the directions of maximum variance: the covariance matrix:

$$S = \frac{1}{N} \sum_{j=1}^{N} (x_j - m)(x_j - m)^\top,$$

can be factorized using SVD (which is also an eigenvalue decomposition, being S symmetric and positive semidefinite):

$$S = U \Lambda U^\top = \sum_{i=1}^{N} \lambda_i u_i u_i^\top,$$

where λ_i and u_i are eigenvalues and eigenvectors of S, respectively.

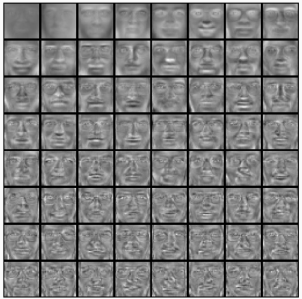
Eigenfaces (cont.)

- By taking the top k eigenvalues (i.e. the k directions of maximum variance), each image x of the training set can be approximated as:

$$\tilde{x} = m + \sum_{i=1}^{k} \phi_i u_i, \qquad \phi_i = (x - m)^\top u_i.$$

mean image

top eigenvectors



Example of reconstruction from a basis of eigenfaces: original image (left), top 8 eigenfaces (center), reconstructed image (right). From (Moghaddam and Pentland, 1997).

Distances

We may define some distances:

- *distance in face space* (DIFS) which is the Euclidean distance between the projections $\tilde{x}$ and $\tilde{x}'$ of two images x and x' onto the face space:

$$\text{DIFS}(x, x') = \left\|\tilde{x} - \tilde{x}'\right\| = \left\|\sum_{i=1}^{k} \phi_i u_i - \sum_{i=1}^{k} \phi_i' u_i\right\| = \left\|\sum_{i=1}^{k} (\phi_i - \phi_i') u_i\right\| = \sqrt{\sum_{i=1}^{k} (\phi_i - \phi_i')^2}$$

  where the last equality follows by the fact that the u_i vectors are orthogonal and of unit norm.

- *distance from face space* (DFFS) which is the Euclidean distance between the original image x and its projection $\tilde{x}$ onto the face space:

$$\text{DFFS}(x) = \|x - \tilde{x}\|,$$

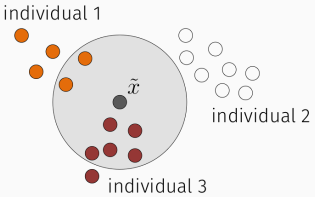
  and can be computed directly in pixel space. It represents the "faceness" of a particular image.

Recognizing a face

1. Take a query image x;
2. Project onto eigenface space and compute the coefficients ϕ_i:

$$x \longrightarrow [(x-m)^\top u_1, (x-m)^\top u_2, \ldots, (x-m)^\top u_k]^\top = [\phi_1(x), \phi_2(x), \ldots, \phi_k(x)]^\top = \phi.$$

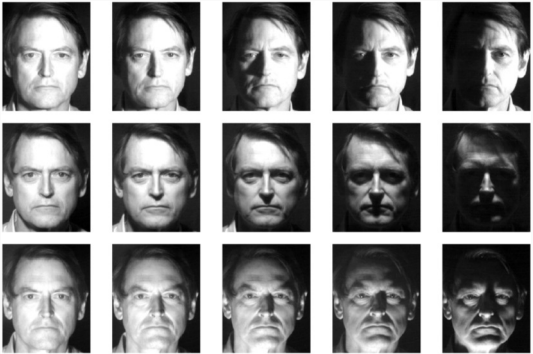
3. compare ϕ with all the projections of N training images;
   - metric: Euclidean;
   - decision rule: k-nearest neighbor.



Fisherfaces

A problem with Eigenfaces is that principal component analysis is optimal for reconstruction from a low dimensional basis but **may not be optimal from a discrimination standpoint** (Belhumeur et al., 1997).

- PCA finds the directions of maximal variance in the training set (i.e. **across all classes**);
- thus, **PCA retains also the unwanted variations**, due to lighting and facial expressions;
- those variations can actually be dramatic and be more significant than the variations due to change of face identity.

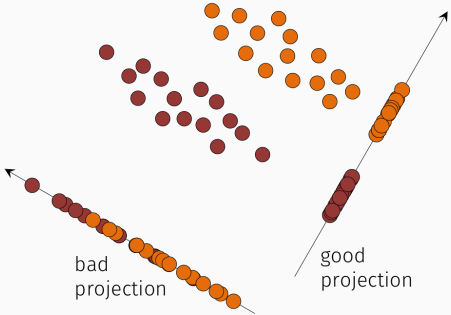


Example of intrapersonal variability, due to lighting.

Fisherfaces

A problem with Eigenfaces is that principal component analysis is optimal for reconstruction from a low dimensional basis but **may not be optimal from a discrimination standpoint** (Belhumeur et al., 1997).

- PCA finds the directions of maximal variance in the training set (i.e. **across all classes**);
- thus, **PCA retains also the unwanted variations**, due to lighting and facial expressions;
- those variations can actually be dramatic and be more significant than the variations due to change of face identity.

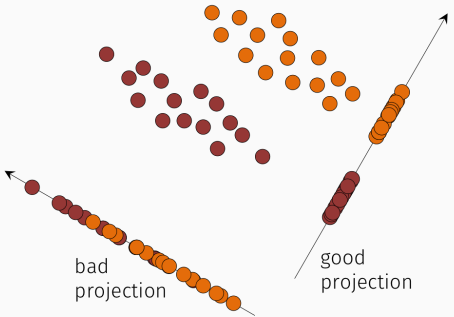


Projecting onto the direction of maximal variance may damage discriminability.

Fisherfaces

A problem with Eigenfaces is that principal component analysis is optimal for reconstruction from a low dimensional basis but **may not be optimal from a discrimination standpoint** (Belhumeur et al., 1997).

- PCA finds the directions of maximal variance in the training set (i.e. **across all classes**);
- thus, **PCA retains also the unwanted variations**, due to lighting and facial expressions;
- those variations can actually be dramatic and be more significant than the variations due to change of face identity.



Projecting onto the direction of maximal variance may damage discriminability.

Belhumeur et al. (1997) propose to apply Fisher's Linear Discriminant method, to **maximize the ratio of between-class scatter to that of within-class scatter**.

As opposite to the Eigenfaces, in the Fisherfaces, **the training labels do affect the computation of the low dimensional projection space**.

Linear discriminant analysis

- Let C be the number of classes and $\mathcal{C}_i$ the set of example belonging to the ith class;
- define the total *within-class* scatter matrix as

$$S_W = \sum_{j=1}^{C} S_j = \sum_{j=1}^{C} \sum_{i \in \mathcal{C}_j} (x_i - m_j)(x_i - m_j)^\top,$$

  where m_j is the mean of class j ans S_j is its within-class scatter matrix;
- define the *between-class* scatter as

$$S_B = \sum_{j=1}^{C} N_j (m_j - m)(m_j - m)^\top,$$

  where m is the overall mean and N_j is the number of example of class j (thus the total number of examples is $N = \sum N_j$).
- we begin finding the linear combination $z = u^\top x$ such that the between-class scatter is maximized relative to the within-class scatter.

Linear discriminant analysis (cont.)

- the problem can be formulated as:

$$\underset{u}{\text{argmax}} \quad \frac{u^\top S_B u}{u^\top S_W u},$$

or, equivalently:

$$\begin{aligned} &\underset{u}{\text{argmax}} && u^\top S_B u \\ &\text{s. t.} && \\ &u^\top S_W u = 1 && \end{aligned} .$$

- introducing the Lagrange multiplier λ, the Lagrangian becomes

$$L(u, \lambda) = u^\top S_B u - \lambda(u^\top S_W u - 1) = u^\top (S_B - \lambda S_W) u + \lambda, \tag{1}$$

to be maximized w.r.t. both u and λ. By imposing stationarity w.r.t. u we get

$$\nabla_u L(u, \lambda) = 2(S_B - \lambda S_W) u = 0.$$

Linear discriminant analysis (cont.)

- Thus the sought u, λ must solve the generalized eigenvalue problem

$$S_B u = \lambda S_W u,$$

which, if S_W is invertible, becomes

$$(S_W^{-1} S_B) u = \lambda u.$$

Thus the solution is an eigenvalue-eigenvector pair of the matrix $F = S_W^{-1} S_B$.

- Since we need to maximize (1), the solution is the eigenvector u_1 associated to the maximum eigenvalue λ_1 of F.
- Similarly, we can find the next direction u_2 orthogonal to u_1 such that $u_2^\top S_B u_2 / u_2^\top S_W u_2$ is maximized. This is the eigenvector associated to the maximum of the remaining eigenvalues of F. Notice that being F symmetric by construction, its eigenvectors are orthogonal.
- By iterating m times, we get a basis $U = [u_1, \dots, u_m]$ and the projection takes the form

$$\tilde{x} = U^\top x, \qquad U \in \mathbb{R}^{d \times m},$$

where d is the dimension of the original space (the number of pixels, in the case of images).

Linear discriminant analysis (cont.)

- How many non-zero generalized eigenvalues can we find? Since $F = S_W^{-1} S_B$,

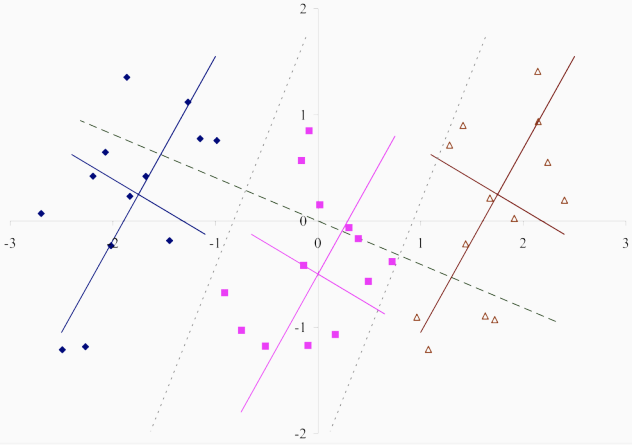
$$\text{rank } F \leq \text{rank } S_B \leq C,$$

where C is the number of classes and the last inequality holds because S_B is the sum of C matrices of rank one or less. It actually can be shown (Duda et al., 2012) that

$$\text{rank } S_B \leq C - 1,$$

thus at most $C - 1$ eigenvalues are non-zero.

Example



Example of Fisher linear discriminant analysis. The samples come from three different classes, shown in different colors along with their principal axes (the intersections of the tilted axes are the class meansT. The dashed line is the dominant Fisher linear discriminant direction. The PCA directions are parallel to the x, y axes. From Szeliski (2010).

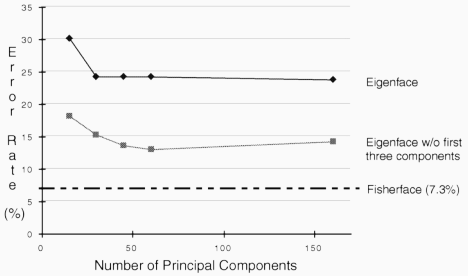
Singularity of the within class scatter matrix

- It can be shown (Duda et al., 2012) that:

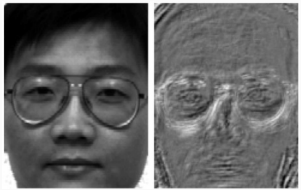
$$\text{rank } S_W \leq N - C.$$

- As a consequence, if $N < d + C$ (the number of examples is less than the number of pixels in each image plus the number of classes), the matrix $S_W \in \mathbb{R}^{d \times d}$ is certainly singular.
- Two-step solution proposed by (Belhumeur et al., 1997):
  1. reduce dimension of the feature space from d to $N - C$ using PCA;
  2. apply standard LDA to reduce the dimension to $C - 1$.

Fisherfaces: results



- Task: recognition;
- input: 160 images of 16 people;
- evaluation: leave-one-out (train with 159 and test with 1);
- metric: Euclidean;
- decision rule: nearest neighbor;
- removed first 3 eigenfaces because had been reported depending mostly on lighting.

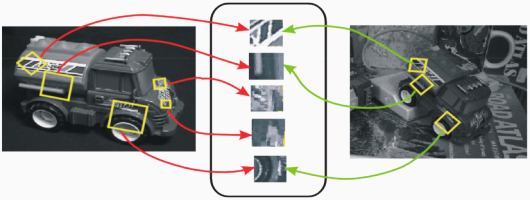


COMPARATIVE RECOGNITION ERROR RATES FOR GLASSES/NO GLASSES RECOGNITION USING THE YALE DATABASE

Glasses Recognition		
Method	Reduced Space	Error Rate (%)
PCA	10	52.6
Fisherface	1	5.3

- Task: "wearing glasses" vs "not wearing glasses";
- input: 36 images, half with glasses;
- since $C = 2$, a single Fisherface is obtained (shown on top, right);
- Eigenfaces rate is near chance.

Instance recognition from local features



A possible approach to instance recognition is based on local features (Lowe, 1999).

At training time:

- build a model as a **collection of local features and their relative location**; the features are invariant to translation, rotation and scale.

At testing time:

- detect local features in the image;
- find candidate matches, based on Euclidean distance;
- check geometric consistency.

Checking geometric consistency

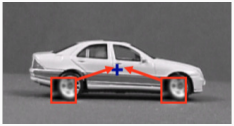
- Geometric consistency can be checked for instance using RANSAC or Generalized Hough transform (GHT).
- Basic assumptions:
  - objects are planar OR
  - the observed deformation is explained by an affine transformation.

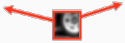
Lowe (1999) suggest using GHT (because the percent of outliers may be as high as 99%) and rather broad bin sizes:

- 30° for orientation;
- a factor 2 for scale;
- a quarter of the maximum object size for location.

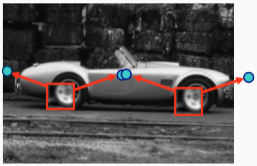
Generalized Hough transform with local features

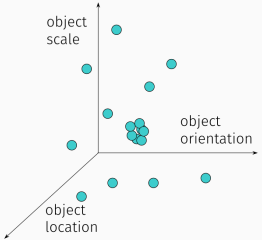
- For every feature, store possible occurrences (there may be many in the same model);





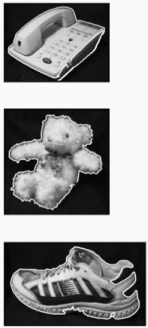
- For new image, let the matched features vote for possible object location, orientation and scale;





slide credit: Kristen Grauman and Bastian Leibe

Some results



Recognition result for 3D objects (center) and recognition result for 3D object with occlusion (right). From Lowe (1999).

Some results (cont.)

Recognition result for 3D objects under severe occlusion. A parallelogram is drawn around each recognized object showing the boundaries of the original training image under the affine transformation solved for during recognition. Smaller squares indicate the keypoints that were used for recognition. From Lowe (2004).

Instance recognition from large databases

Query images.

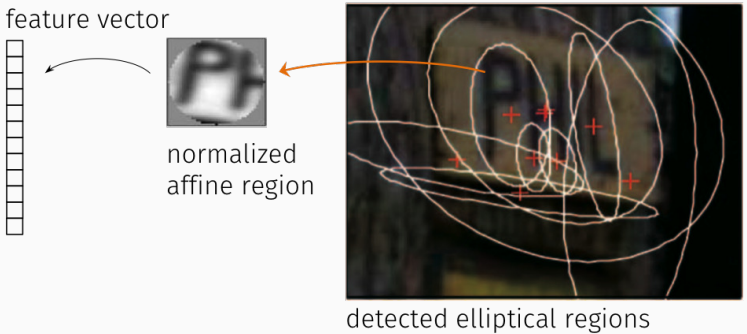
Highest-ranked results. From Philbin et al. (2007).

- Assume that the database contains millions of objects (for example, in *location recognition*, the task is finding out the current location based on an cell-phone image, and a large collection of photographs from Internet);
- the time required to match a new image against each database image can become prohibitive, thus efficient searching techniques are needed;
- Sivic and Zisserman (2003) borrow ideas from *text retrieval*, where:
  1. a set of words ("**vocabulary**") appearing in the database documents is collected;
  2. an **inverted index** is pre-computed between individual words and the documents where they occur;
  3. the **frequency** of occurrence of particular words in a document is used to quickly find documents that match a given query.
- to apply the same strategy to images, a *visual vocabulary* is needed. How can we build it?

Visual vocabulary

- The basic idea of Sivic and Zisserman (2003) is that of using the local feature descriptors as the analogous of words;
- however, problems arise because words are inherently discrete, while image descriptors are high-dimensional and real-valued feature points;
- solution: **quantize the feature space of local image descriptors**, and obtain a set of regions in the feature space, each corresponding to a *visual word*.
- in the following we describe the first and simplest of many possible techniques for building a visual vocabulary.

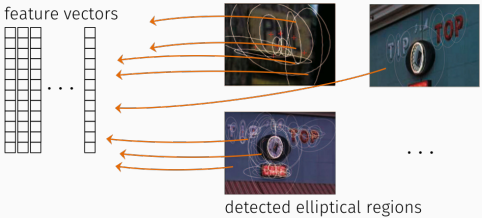
Visual vocabulary construction



- As a first step, Sivic and Zisserman (2003) detect Harris keypoints and maximally stable extremal regions (Matas et al., 2004);
- then, they compute 128-dimensional SIFT descriptors from each normalized region (i.e. elliptical regions are affinely transformed to circular regions);

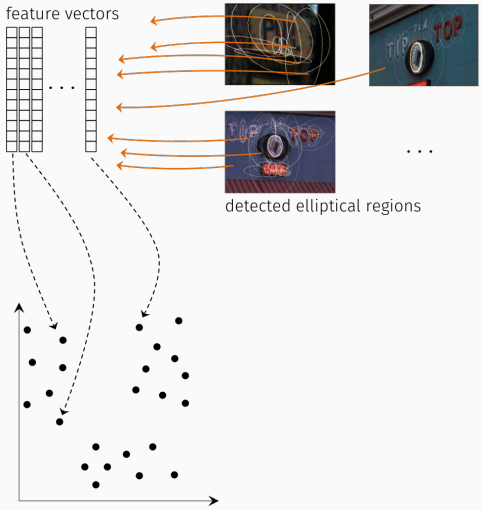
Visual vocabulary construction (cont.)

- feature vectors are collected from each database image;



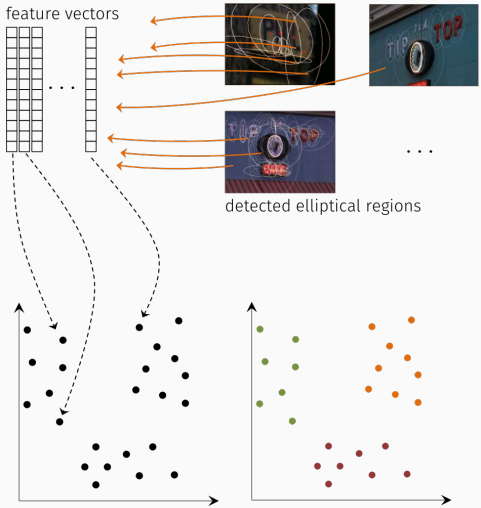
Visual vocabulary construction (cont.)

- feature vectors are collected from each database image;
- each feature vector is a point in the 128-dimensional feature space;



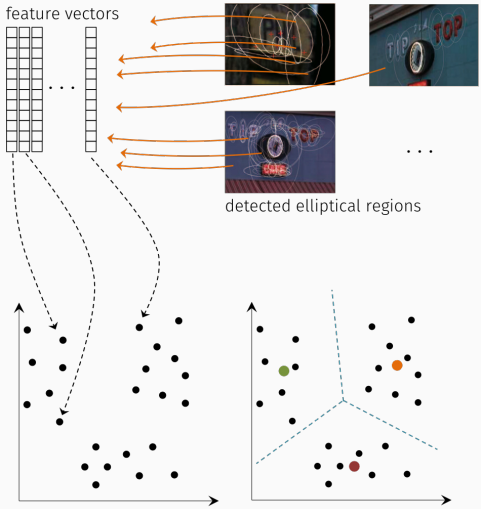
Visual vocabulary construction (cont.)

- feature vectors are collected from each database image;
- each feature vector is a point in the 128-dimensional feature space;
- k-means clustering is applied (the number k of clusters is a design parameter);



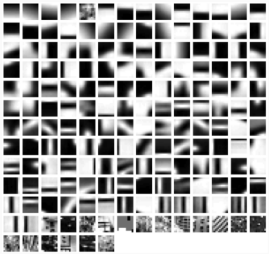
Visual vocabulary construction (cont.)

- feature vectors are collected from each database image;
- each feature vector is a point in the 128-dimensional feature space;
- k-means clustering is applied (the number k of clusters is a design parameter);
- the visual words are cluster centers;
- a partition of the feature space is thus obtained according to the distance from the cluster centers.



Observations

1. There exist alternatives to collecting descriptors of keypoints neighborhoods. For instance, in Fei-Fei and Perona (2005) patches are collected on a regular grid and employed as visual words (after k-means clustering).



"Visual codebook" from Fei-Fei and Perona (2005).

2. In principle, the database employed for constructing the visual vocabulary could be different from the set of images we want retrieve. If the corpus of images is sufficiently representative, the obtained vocabulary will be "universal".
3. Very common visual words are not informative (as, for instance, the word "and" in English documents), thus they are put in a *stop list* and dropped from further consideration.

Inverted file index

- To speed-up the search of potentially matching instances, discarding those instances that are certainly non-relevant, an *inverted file index* is employed;
- the inverted file index stores, for each visual word, the list of the images containing that word;
- retrieval via the inverted file index is faster then searching every image, assuming that not all images contain every word.

visual word	image list
1	4,8,15,16,23,42, ...
2	3,14,15,92, ...
⋮	⋮

Term frequency-inverse document frequency

- Information retrieval systems are based on the relative frequency of words in documents;
- let n_{id} be the number of occurrences of word i in document d and n_d be the total number of words in document d;
- thus, we can define the *term frequency*

$$\frac{n_{id}}{n_d};$$

- in order to downweight words that occur frequently and focus on rarer (thus, more informative) terms, an *inverse document frequency* weighting is applied

$$\log \frac{N}{N_i},$$

where N_i is the number of documents containing word i and N is the total number of documents.

- Thus we obtain the *term frequency-inverse document frequency (tf-idf)* measure:

$$t_i = \frac{n_{id}}{n_d} \log \frac{N}{N_i}.$$

- At match time, each document is represented by a vector

$$t = [t_1, \ldots, t_i, \ldots, t_m]^\top$$

- the similarity $s(a, b)$ between two documents a and b is measured by the dot product between their corresponding normalized vectors:

$$s(a, b) = \frac{t_a}{\|t_a\|} \cdot \frac{t_b}{\|t_b\|}$$

- The search is restricted to potentially relevant images, using inverted index.
- After retrieving the top 500 documents based on word frequencies, Sivic and Zisserman (2009) re-rank these results using spatial consistency;
- in particular, they take every matching feature and count the number of 15 nearest adjacent features that also match between the two documents.

Summary Vocabulary construction (off-line)

1: Extract affine covariant regions from each database image.

2: Compute descriptors.

3: Cluster the descriptors into visual words, either using k-means or other techniques such as hierarchical clustering (Nister and Stewenius, 2006), and randomized k-d trees (Philbin et al., 2007).

4: Decide which words are too common and put them in the stop list.

Summary (cont.)

Summary Database construction (off-line)

1: Extract affine covariant regions from each database image.
2: Compute descriptors.
3: For each descriptor, find the closest visual word in the vocabulary.
4: Compute term frequencies for the visual word in each image, document frequencies for each word, and normalized tf-idf vectors for each document.
5: Compute inverted indices from visual words to images.
6: Optionally keep track of the location x, y of each occurrence of each visual word in each image (for checking spatial consistency).

Steps 1 and 2 are only needed if the vocabulary is extracted from a different set of images.

Summary (cont.)

Summary Image retrieval (on-line)

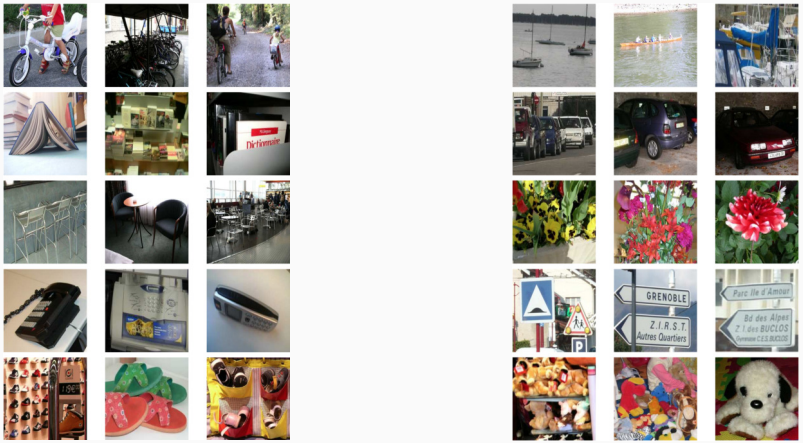
1: Extract regions, descriptors, and visual words, and compute a tf-idf vector for the query image or region.

2: Retrieve the top image candidates, either by exhaustively comparing sparse tf-idf vectors or by using inverted indices to examine only a subset of the images.

3: Optionally re-rank or verify all the candidate matches, using spatial consistency.

Category recognition

Category-level recognition



In the figures above (Csurka et al., 2006), images of the same row belong to the same category. **Category-level recognition is extremely challenging**, and remains still largely unsolved. However, dramatic progresses have been made, especially in the last decade.

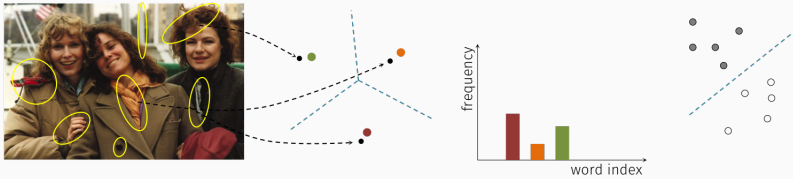
Category-level recognition (cont.)

In the following we discuss two approaches to category-level recognition:

- **Bag of words** (BOW).
- **Convolutional neural networs** (ConvNets).

Category recognition with bag of words

Bag of words



Typical processing pipeline for a bag of words category recognition system.

- The *bag of words* approach (first proposed in Csurka et al. (2004) with the name of *bag of keypoints*), simply computes the histogram of visual words found in the query image and compares this distribution to those found in the training images.
- The histogram is known as **bag of (visual) words** and is a compact description of the whole image in terms of its elementary tokens (the visual words).
- SVM classification on bag of words descriptors (see figure) is a standard paradigm for image classification.
- The main difference w.r.t. instance recognition is the **absence of a geometric consistency check**.

Spatial consistency

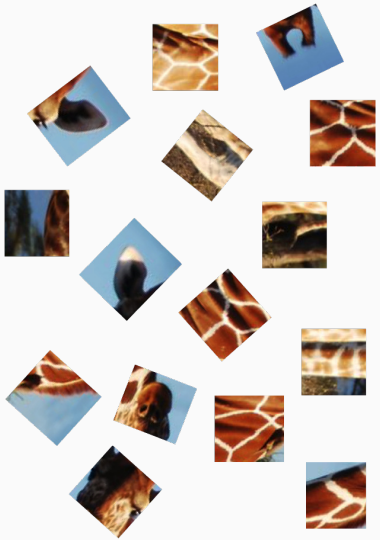
Pablo Picasso, *Guitar* (1913)

Spatial consistency (cont.)

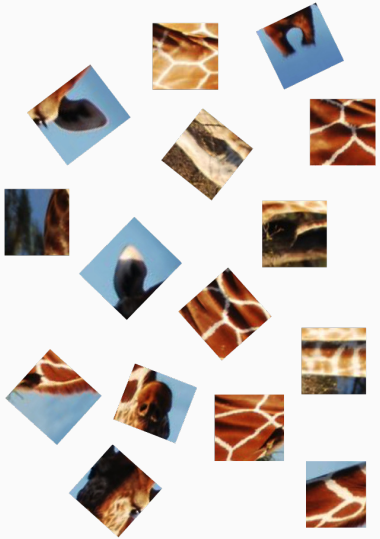
Spatial consistency (cont.)

Leo Fender, *Telecaster* (1952)

Spatial consistency (cont.)



Spatial consistency (cont.)



Giraffe in Samburu National Reserve, Kenya (photo by Julian Fennessy)

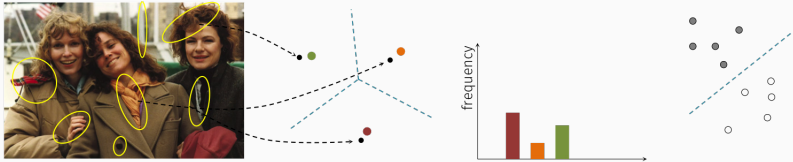
Training process

Steps of the training process for bag of words classification, from Grauman and Leibe (2011):

- Collect training image examples from each category of interest.
- Detect or sample local features in all training images. For sparser local features, use one or more interest operators (e.g., DoG, Harris, etc). For denser local features, sample uniformly across the image at multiple scales.
- Extract a descriptor (e.g. SIFT) at each interest point or sampled point.
- Form a visual vocabulary: sample a corpus of descriptors from the training images, quantize the descriptor space (typically using k-means), and store the resulting visual words. These words have the same dimensionality as the descriptors.
- Map each training image's local descriptors to their respective visual words, yielding a single k-dimensional histogram for each training image.
- Select a kernel function, and use it to compute the pairwise similarities between all training images. For N total training images, we now have an $N \times N$ kernel matrix.
- Use the kernel matrix and labels to train an SVM.

The last two steps will be discussed next.

Recognition process



local feature detection and description | quantization to visual vocabulary | bag-of-words histogram | classification

Steps of the recognition process for bag of words classification, from Grauman and Leibe (2011):

- Given the novel test image, repeat the feature detection, description, and histogram steps as during training, using the same visual vocabulary.
- Classify the image using the SVM. For non-linear kernels, this entails computing kernel values between the novel bag-of-words histogram and the training instances involved in the decision function, i.e. the support vectors.

Observe that although the BOW is typically used for image-level classification, it is straightforward to instead train with bounding box annotations and test with a sliding window scan.

Recall that the formulation of the non-linear binary SVM is:

$$\max_{\alpha} W(\alpha) = \sum_{i=1}^{l} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{l} \alpha_i \alpha_j y_i y_j k(x_i, x_j)$$

s.t.

$$\sum_{i=1}^{l} \alpha_i y_i = 0$$

$$0 \leq \alpha_i \leq C, \; \forall i = 1, \ldots, l$$

,

and the corresponding decision function is

$$h(x) = \Theta[\sum_{\text{SV}} y_i \alpha_i k(x_i, x) + b].$$

The function $k(\cdot, \cdot)$ is the kernel function and the matrix $[K_{ij}] = [k(x_i, x_j)], \; i, j = 1, \ldots, l$ is the kernel matrix (or Gram matrix). For training, we need to evaluate the kernel for all pairs of training examples, while for testing, we need to compute the kernel for all the pairs (x_i, x) where x_i is a support vector and x is the instance to be classified.

Specialized kernels (cont.)

As it is well known, the Gaussian kernel, which is a common choice for SVM, is defined as

$$k(x, x') = \exp(-\frac{\|x - x'\|^2}{\sigma^2}).$$

We can define a *generalized Gaussian kernel* as

$$k(x, x') = \exp(-\frac{D(x, x')}{A}),$$

where $A > 0$ is a scaling parameter and $D(x, x')$ is some distance between input vectors x and x'. We thus can replace the usual Euclidean distance with other distances, more appropriate to comparing the descriptors.

Specialized kernels (cont.)

Suppose we are given two descriptors (actually, signatures) x, x' of the form:

$$x = \{(t_1, w_1), \ldots (t_m, w_m)\} \qquad \text{and} \qquad x' = \{(t'_1, w'_1), \ldots (t'_{m'}, w'_{m'})\},$$

where t_i is the relative frequency of the word w_i. In Zhang et al. (2007), two distance metrics are compared:

- earth mover's distance (Rubner et al., 1998):

$$\text{EMD}(x, x') = \frac{\sum_{i=1}^{l} \sum_{j=1}^{m} d(w_i, w'_j) f_{ij}}{\sum_{i=1}^{l} \sum_{j=1}^{m} f_{ij}},$$

  where $d(w_i, w'_j)$ is the ground distance (possibly, Euclidean) between x_i and x'_j and f_{ij} is a flow that can be computed by solving a linear programming problem.

- χ^2 ("chi squared") distance:

$$\chi^2(x, x') = \frac{1}{2} \sum_i \frac{(t_i - t'_i)^2}{t_i + t'_i},$$

(that can be employed only if $m = m'$ and $w_i = w'_i, \quad \forall i$).

In their experiments, they find that the EMD works best for visual category recognition and the χ^2 distance is best for texture recognition.

A widely adopted alternative to the previous is the *histogram intersection kernel (IK)*:

$$\mathrm{IK}(x, x') = \sum_i \mathsf{min}(t_i, t'_i).$$

The histogram intersection kernel can be evaluated very quickly (Maji et al., 2008).

Pyramid match kernel

The pyramid match kernel (PMK, Grauman and Darrell (2007)) is a technique for directly computing an approximate distance between two collections of feature vectors, say

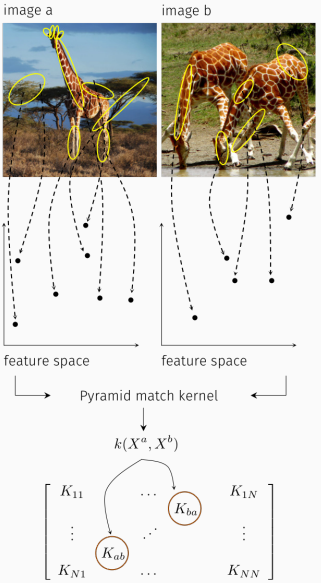
$$X^a = \{x_i^a \in \mathbb{R}^d,\ i = 1 \dots m^a\}$$

and

$$X^b = \{x_i^b \in \mathbb{R}^d,\ i = 1 \dots m^b\},$$

possibly having a different number of elements.

PMK satisfies the Mercer condition for kernel functions (i.e. it produces positive-definite Gram matrices that can be used for training SVMs).



Pyramid match kernel (cont.)

Given two collections of feature vectors $X^a = \{x_i^a \in \mathbb{R}^d,\ i = 1 \dots m^a\}$ and $X^b = \{x_i^b \in \mathbb{R}^d,\ i = 1 \dots m^b\}$, let the optimal matching be

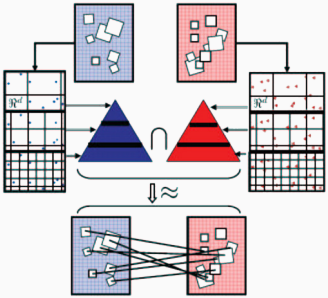
$$\pi* = \underset{\pi}{\text{argmin}} \sum_{x_i^a \in X^a} \left\| x_i^a - x_{\pi_i}^b \right\|_1,$$

where the vector $\pi = [\pi_1 \dots \pi_{m^a}]$, $1 \leq \pi_i \leq m^b$ defines the assignments (we assume $m^a \leq m^b$).

The similarity between X^a and X^b is then defined as

$$\sum_{x_i^a \in X^a} \frac{1}{\left\| x_i^a - x_{\pi_i}^b \right\|_1 + 1}.$$

The PMK computes, efficiently, an approximation of such similarity (that can be used as a kernel).



The pyramid match kernel approximates the similarity measure corresponding to the optimal match between two sets of features of variable cardinality. From Grauman and Darrell (2007).

Pyramid match kernel (cont.)

In essence, the PMK approach consists of:

- binning the feature vectors into a multiresolution pyramid defined **in the feature space**;
- counting the number of features that land in corresponding bins B^a_{il}, B^b_{il} where i is the index of bin and l the level of the pyramid;
- computing a distance from the two sets at each level as a histogram intersection:

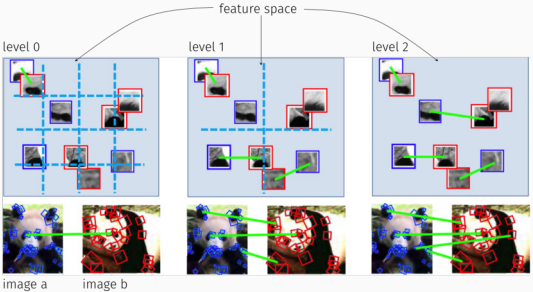
$$C_l = \sum_i \min(B^a_{il}, B^b_{il});$$

- summing up the per-level counts in a weighted fashion:

$$k(X^a, X^b) = \sum_l w_l N_l,$$

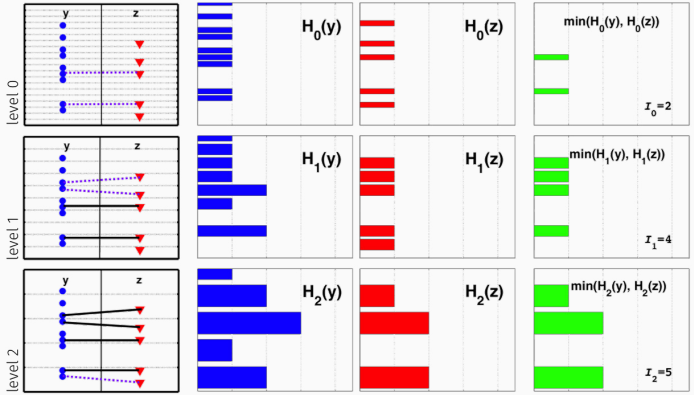
where

$$N_l = \underbrace{C_l - C_{l-1}}_{\text{new matches}} \quad \text{and} \quad \frac{1}{d2^l}.$$



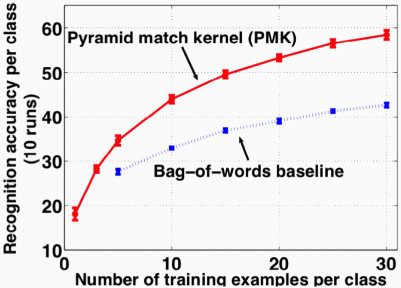
Pictorial representation of pyramid match kernel, adapted from Grauman and Leibe (2011).

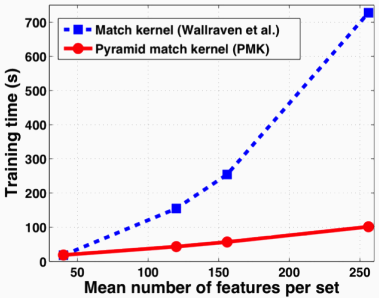
Pyramid match kernel (cont.)



A one-dimensional illustration of comparing collections of feature vectors using the pyramid match kernel: (a) distribution of feature vectors (point sets) into the pyramidal bins; (b) histogram of point counts in bins B_{il}^a and B_{il}^b for the two images; (c) histogram intersections (minimum values); (d), new matches and weights, that are used to compute the similarity metric. From Grauman and Darrell (2007).

Pyramid match kernel (cont.)

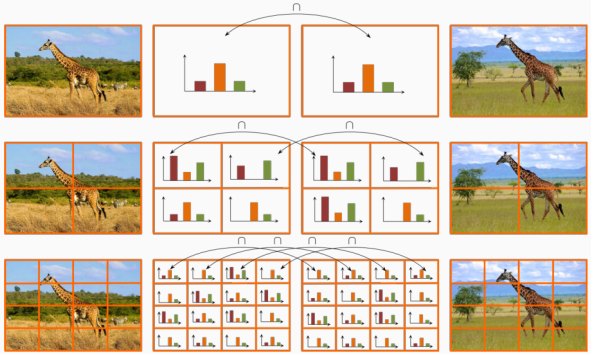




Benefits of pyramid matching kernel:

- Using PMK, there is no need to quantize feature vectors to visual words;
- Sometimes this is sufficient to get better accuracy than the bag of words approach;
- PMK is much more efficient (linear in the number of elements to be matched) w.r.t. other approaches based on direct matching, which are polynomial in the number of elements.

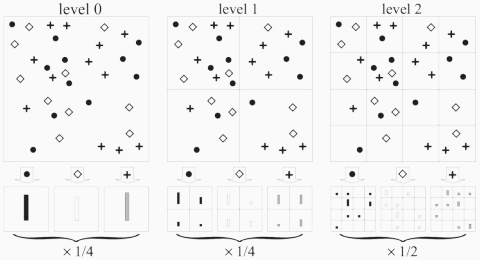
Spatial pyramid kernel



Spatial pyramid kernel is based on the intersection of multi-level histograms.

The spatial pyramid kernel (Lazebnik et al., 2006) is inspired by the pyramid match kernel but **operates in the image space**. The goal is to **augment bag of words with loose notions of 2D spatial location**. This is achieved by repeatedly subdividing an image and computing histograms of image features over the resulting subregions. The similarity measure between two images is the **weighted sum of histogram intersections of corresponding subregions**.

Spatial pyramid kernel (cont.)

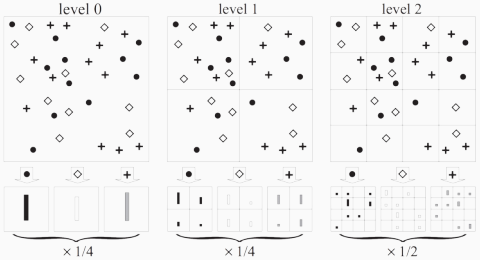


Toy example of constructing a three-level pyramid, from Lazebnik et al. (2006).

The essential steps of the approach are

- extract affine regions descriptors (on a regular grid);
- quantize the descriptions into visual words;
- form a spatial pyramid of bins containing word counts (histograms) as in figure;
- combine weighted histograms intersections in a hierarchical fashion.

Spatial pyramid kernel (cont.)

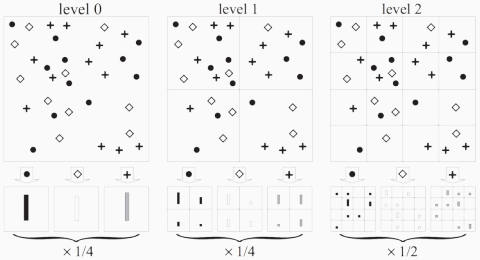


Toy example of constructing a three-level pyramid, from Lazebnik et al. (2006).

- Suppose two images are given, $\mathcal{I}^a$ and $\mathcal{I}^b$ and let l denote the level of the pyramid, corresponding to a grid;
- denote with $H^a_{t,l}(i)$ the number of features of type t in the ith box of grid l of image $\mathcal{I}^a$;
- the number of elements of type t that can be matched within the corresponding ith boxes in grid l of the two images is

$$\min(H^a_{t,l}(i), H^b_{t,l}(i));$$

Spatial pyramid kernel (cont.)



Toy example of constructing a three-level pyramid, from Lazebnik et al. (2006).

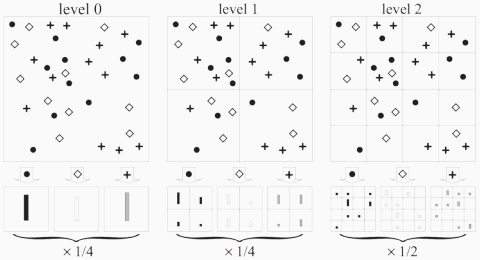
- the similarity between $\mathcal{I}^a$ and $\mathcal{I}^b$, measured at grid level l, is

$$\sum_{i \in \text{grid boxes}} \min(H^a_{t,l}(i), H^b_{t,l}(i)).$$

- the overall similarity is a weighted sum of the similarities at all feature types and all levels, where more weights is placed on matches in fine grids:

$$\sum_{t \in \text{feature types}} \sum_{l \in \text{levels}} \sum_{i \in \text{grid boxes}} w_l \min(H^a_{t,l}(i), H^b_{t,l}(i)).$$

Spatial pyramid kernel (cont.)

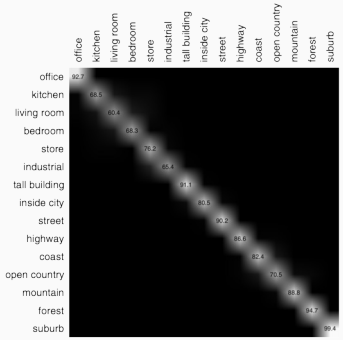


Toy example of constructing a three-level pyramid, from Lazebnik et al. (2006).

- the resulting similarity estimates can be used either to rank image similarity, or as a kernel for a kernel-based classifier.

Results

Sample categories.



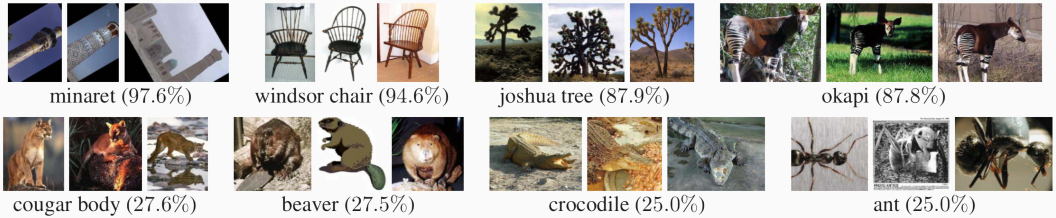
Confusion matrix.

- 15 categories;
- 200 to 400 images of about 300×250 per category;
- 100 images per class for training, the rest for test;
- multiclass classification via one-vs-all SVM;
- using levels from 0 to 2, leads to a substantial improvement w.r.t. standard bag of words.

Results (cont.)

Example of similarity-based retrieval (query image on left).

Results (cont.)

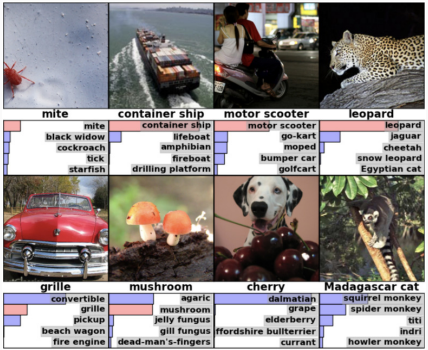


Example of classes and classification rates on the Caltech-101 dataset.

- Caltech-101 dataset;
- 101 categories;
- 31 to 800 images of about 300×300 per category;
- train on 30 images per class;
- successful classes are either dominated by rotation artifacts (like minaret), have very little clutter (like windsor chair), or represent coherent natural “scenes” (like joshua tree and okapi);
- least successful classes are either textureless animals (like beaver and cougar), animals that camouflage well in their environment (like crocodile), or “thin” objects (like ant).

Category recognition using convolutional neural networks

A categorization challenge

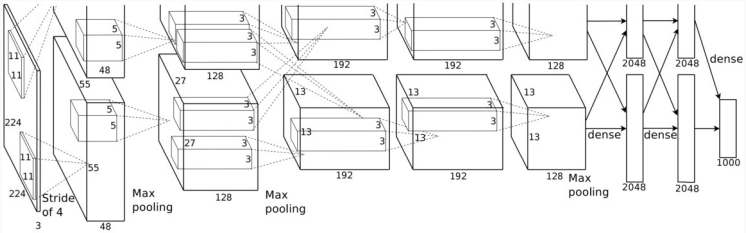


ImageNet (`www.image-net.org`)

- 14M images
- 22K categories
  - Animal (fish, bird, mammal, invertebrate)
  - Plant (tree, flower, vegetable)
  - Instrumentation (utensile, appliance, tool, musical instrument)
  - ...
- annual ImageNet Large Scale Visual Recognition Challenge (ILSVRC)
- object detection within an image may be obtained by applying image categorization to many sub-images.

Convolutional Neural Networks

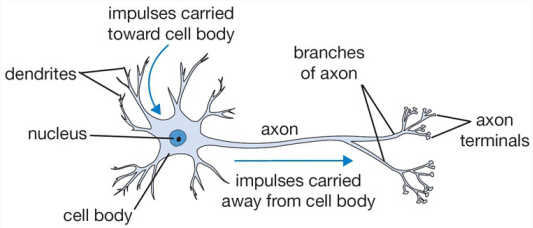
- In 2012, a Convolutional Neural Network (CNN) won the ImageNet challenge by a wide margin:

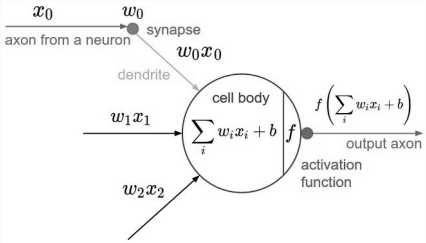


Imagenet classification with deep convolutional neural networks, (Krizhevsky et al., 2012)

- a CNN is an Artificial Neural Networks (ANN) having a particular architecture, developed specifically for dealing with images
- nowadays, CNNs are employed even for dealing with information other than images
- in the last few years, the term “Deep Learning”” has become popular to identify the approach to machine learning based on ANN having many layers (“deep”)

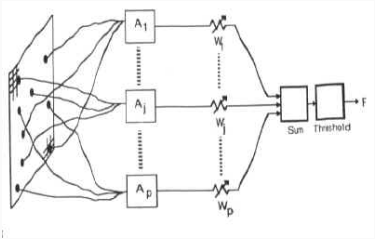
A step backward: Artificial Neural Networks I





- a network of artificial neurons (McCulloch and Pitts, 1943)
- biological analogy (with caution)
- the output of a neuron is obtained by applying an *activation function* to the weighted sum of its inputs

A step backward: Artificial Neural Networks II

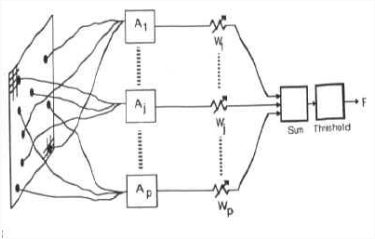


The perceptron

Frank Rosenblatt (1928-1971)

- first trainable Artificial Neural Network: Rosenblatt's Perceptron (Rosenblatt, 1958)

A step backward: Artificial Neural Networks II

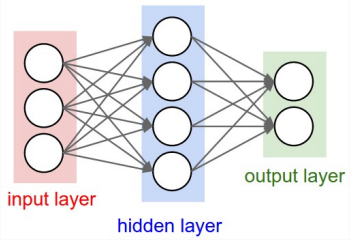


The perceptron

Frank Rosenblatt (1928-1971)

- first trainable Artificial Neural Network: Rosenblatt's Perceptron (Rosenblatt, 1958)
- basic procedure:
  - design the *architecture* (# layers, topology of connections)
  - choose the (possibly different) *activation functions*
  - learn the *weights* (based on the provided examples)

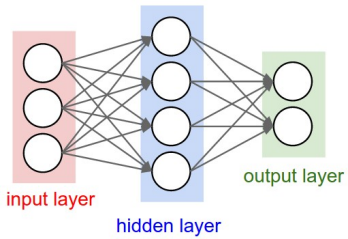
Architectures



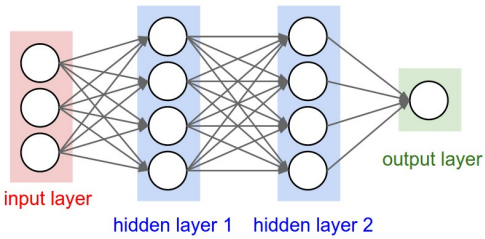
Fully connected, 1 hidden layer

- left:
  - $4 + 2 = 6$ neurons (input layer is disregarded)
  - $3 \times 4 + 4 \times 2$ weights (w) and $4 + 2$ biases (b) $\Rightarrow$ 26 parameters

Architectures



Fully connected, 1 hidden layer



Fully connected, 2 hidden layers

- left:
  - $4 + 2 = 6$ neurons (input layer is disregarded)
  - $3 \times 4 + 4 \times 2$ weights (w) and $4 + 2$ biases (b) $\Rightarrow$ 26 parameters
- right:
  - $4 + 4 + 1 = 9$ neurons
  - $3 \times 4 + 4 \times 4 + 4 \times 1$ weights (w) and $4 + 4 + 1$ biases (b) $\Rightarrow$ 41 parameters
- a typical CNN has million of parameters (e.g. VGG ha 138M parameters)

Activation functions

- output of a neuron

$$f(\sum_i w_i x_i + b)$$

- $f(\cdot) : \mathbb{R} \longrightarrow \mathbb{R}$ is the *activation function*
- many possible choices, e.g.
  - sigmoid

$$f(x) = \frac{1}{1 + e^{-x}}$$

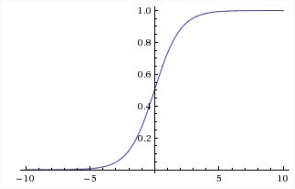
  - hyperbolic tangent

$$f(x) = \tanh(x)$$

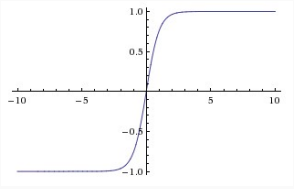
  - rectified linear unit (ReLU)

$$f(x) = \max\{0, x\}$$

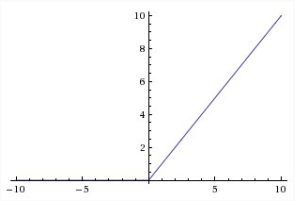
- ReLU is currently preferred because
  - is fast to compute
  - guarantees faster convergence



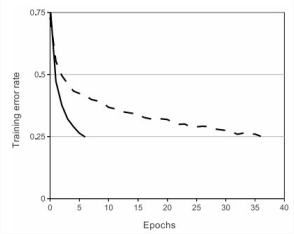
sigmoid



hyperbolic tangent



ReLU



ReLU vs hyperbolic tangent (Krizhevsky et al., 2012)

Training as an optimization problem

- N instances, each of dimension D, and K classes
- the training set is

$$\mathcal{T} = \left\{(x_i, y_i), x_i \in \mathbb{R}^D, y_i \in 1 \dots K, i = 1 \dots N\right\}$$

- let the output of the network be

$$f(x, \theta) : \mathbb{R}^D \longrightarrow \mathbb{R}^K$$

- define the loss as

$$J(\theta) = \frac{1}{N} \sum_i L(f(x_i, \theta), y_i) + \lambda R(\theta)$$

- many possible choices for L and R, e.g.

$$\begin{aligned} L(f(x_i, \theta), y_i) &= \sum_{j \neq y_i} \max(0, f_j(x_i, \theta) - f_{y_i}(x_i, \theta) + \Delta) \\ R(\theta) &= \sum_k \sum_l W_{kl}^2 \end{aligned}$$

Training as an optimization problem

- N instances, each of dimension D, and K classes
- the training set is

$$\mathcal{T} = \Big\{(x_i, y_i), x_i \in \mathbb{R}^D, y_i \in 1 \dots K, i = 1 \dots N\Big\}$$

- let the output of the network be

$$f(x, \theta) : \mathbb{R}^D \longrightarrow \mathbb{R}^K$$

- define the loss as

$$J(\theta) = \frac{1}{N} \sum_i L(f(x_i, \theta), y_i) + \lambda R(\theta)$$

- many possible choices for L and R, e.g.

$$\begin{aligned} L(f(x_i, \theta), y_i) &= \sum_{j \neq y_i} \max(0, f_j(x_i, \theta) - f_{y_i}(x_i, \theta) + \Delta) \\ R(\theta) &= \sum_k \sum_l W_{kl}^2 \end{aligned}$$

- minimize the loss by a proper choice of $\theta = (W, b)$
- iterative minimization
  - the instances are presented repeatedly to the network
  - each cycle of presentation is called an "epoch"
  - the weights and biases are adapted in such a way to modify the actual output toward the desired output
- the most popular training methods are based on the so called *back-propagation*

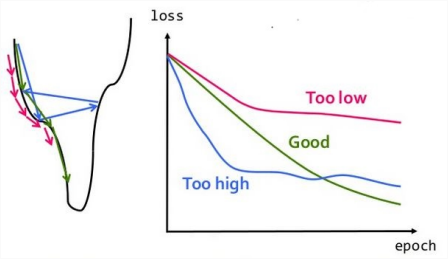
Weight update by gradient descent

Weight update by gradient descent

- at each step of iteration:
  - compute an estimate $\hat{g}$ of the gradient $\nabla J(\theta)$
  - update the weight vector:

$$\theta \leftarrow \theta - \epsilon \hat{g}$$

- ϵ is called *learning rate* and is one of the most important learning parameters



Stochastic Gradient Descent

- the gradient of the loss function is

$$\nabla J(\theta) = \frac{1}{N} \nabla \sum_i L(f(x_i, \theta), y_i)$$

- at each iteration, for computing $\nabla J(\theta)$, the evaluation of the whole training data is required $\rightarrow$ very slow for large N

Stochastic Gradient Descent

- the gradient of the loss function is

$$\nabla J(\theta) = \frac{1}{N} \nabla \sum_i L(f(x_i, \theta), y_i)$$

- at each iteration, for computing $\nabla J(\theta)$, the evaluation of the whole training data is required $\rightarrow$ very slow for large N
- Stochastic Gradient Descent (SGD): at each iteration
  - sample a *minibatch* of m examples $\{x_1, \ldots, x_m\}$ and the corresponding y_i
  - *estimate* the gradient as

$$\hat{g} = \frac{1}{m} \nabla \sum_i L(f(x_i, \theta), y_i)$$

  - apply the update

$$\theta \leftarrow \theta - \epsilon \hat{g}$$

Stochastic Gradient Descent

- the gradient of the loss function is

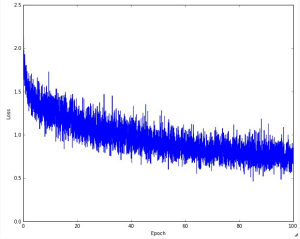
$$\nabla J(\theta) = \frac{1}{N} \nabla \sum_i L(f(x_i, \theta), y_i)$$

- at each iteration, for computing $\nabla J(\theta)$, the evaluation of the whole training data is required $\rightarrow$ very slow for large N
- Stochastic Gradient Descent (SGD): at each iteration
  - sample a *minibatch* of m examples $\{x_1, \ldots, x_m\}$ and the corresponding y_i
  - *estimate* the gradient as

$$\hat{g} = \frac{1}{m} \nabla \sum_i L(f(x_i, \theta), y_i)$$

  - apply the update

$$\theta \leftarrow \theta - \epsilon \hat{g}$$



- Size of minibatch:
  - $m = 1$: gradient direction is unreliable (based on a single example)
  - $m = N$: precise gradient direction, but slow computation
- typical choices for m are 32, 64, 128, to exploit the GPU

Improved weight update

- simple update:

$$\theta \leftarrow \theta - \epsilon \hat{g}$$

Improved weight update

- simple update:

$$\theta \leftarrow \theta - \epsilon \hat{g}$$

- momentum:

$$\begin{cases} v & \leftarrow & \alpha v - \epsilon \hat{g} \\ \theta & \leftarrow & \theta + v \end{cases}$$

Improved weight update

- simple update:

$$\theta \leftarrow \theta - \epsilon \hat{g}$$

- momentum:

$$\begin{cases} v & \leftarrow & \alpha v - \epsilon \hat{g} \\ \theta & \leftarrow & \theta + v \end{cases}$$

- global adaption of learning rate ("annealing"):
  - step decay: Reduce the learning rate by some factor every few epochs.
  - exponential decay:

$$\epsilon = \epsilon_0 e^{-kt}$$

Improved weight update

- simple update:

$$\theta \leftarrow \theta - \epsilon \hat{g}$$

- momentum:

$$\begin{cases} v & \leftarrow & \alpha v - \epsilon \hat{g} \\ \theta & \leftarrow & \theta + v \end{cases}$$

- global adaption of learning rate ("annealing"):
  - step decay: Reduce the learning rate by some factor every few epochs.
  - exponential decay:

$$\epsilon = \epsilon_0 e^{-kt}$$

- per-parameter learning rate adaption:
  - AdaGrad
  - RMSProp
  - Adam (the most recommended, to date)

Improved weight update

- simple update:

$$\theta \leftarrow \theta - \epsilon \hat{g}$$

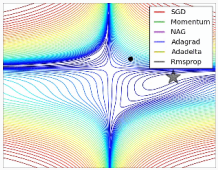
- momentum:

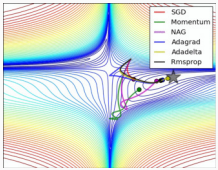
$$\begin{cases} v & \leftarrow & \alpha v - \epsilon \hat{g} \\ \theta & \leftarrow & \theta + v \end{cases}$$

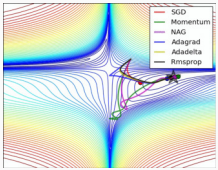
- global adaption of learning rate (“annealing”):
  - step decay: Reduce the learning rate by some factor every few epochs.
  - exponential decay:

$$\epsilon = \epsilon_0 e^{-kt}$$

- per-parameter learning rate adaption:
  - AdaGrad
  - RMSProp
  - Adam (the most recommended, to date)







Computing the gradient $\nabla J(\theta)$

- how can we actually compute the gradient $\nabla J(\theta)$?

Computing the gradient $\nabla J(\theta)$

- how can we actually compute the gradient $\nabla J(\theta)$?
- the ANN can be thought of as the composition of many functions:

$$\text{output} = H(\text{input}) = [h_n \circ h_{n-1} \circ \cdots \circ h_0](\text{input})$$

Computing the gradient $\nabla J(\theta)$

- how can we actually compute the gradient $\nabla J(\theta)$?
- the ANN can be thought of as the composition of many functions:

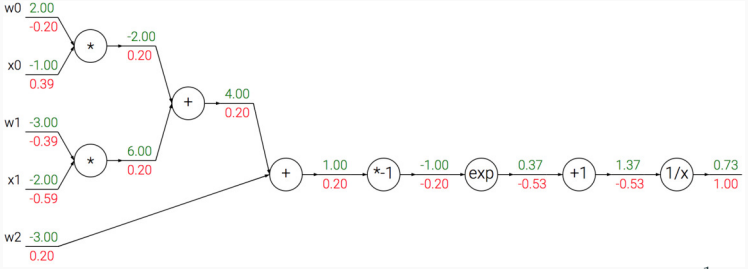
$$\text{output} = H(\text{input}) = [h_n \circ h_{n-1} \circ \cdots \circ h_0](\text{input})$$

- apply the *chain rule*:

$$(f \circ g)'(x) = f'(g(x))g'(x)$$

"the derivative of a composition of functions is the product of the derivatives"

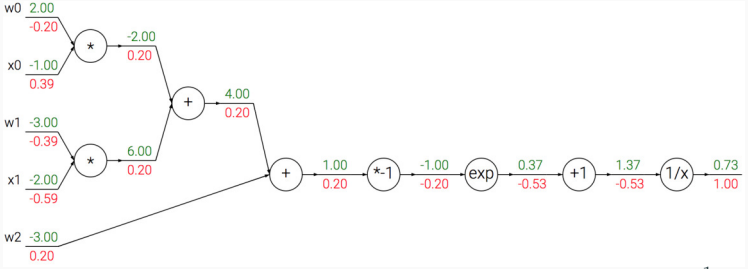
Backpropagation



Example of backpropagation for a neuron with sigmoid activation function: $\sigma = \frac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + w_2)}}$

- forward step (from input to output, green): sets the "state" and output of the network corresponding to a particular input

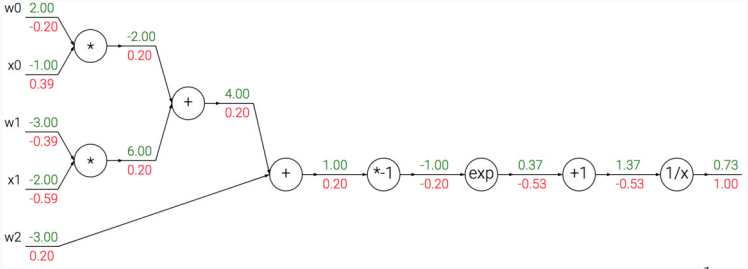
Backpropagation



Example of backpropagation for a neuron with sigmoid activation function: $\sigma = \frac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + w_2)}}$

- forward step (from input to output, green): sets the "state" and output of the network corresponding to a particular input
- backward step (reverse order, red): by applying the chain rule, computes the *derivative of the output w.r.t. the input for the given state*

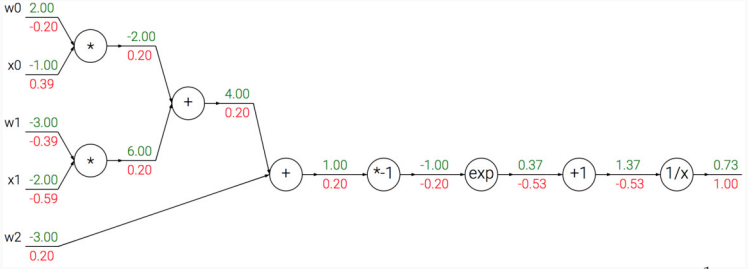
Backpropagation



Example of backpropagation for a neuron with sigmoid activation function: $\sigma = \dfrac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + w_2)}}$

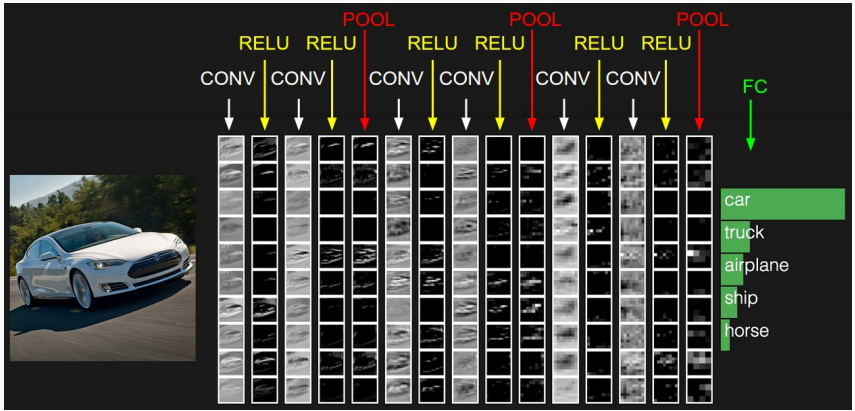
- forward step (from input to output, green): sets the "state" and output of the network corresponding to a particular input
- backward step (reverse order, red): by applying the chain rule, computes the *derivative of the output w.r.t. the input for the given state*
- $s = 2 \times (-1) + (-3) \times (-2) + (-3) = 1 \Rightarrow (1 + e^{-s})^{-1} = 0.731$

Backpropagation



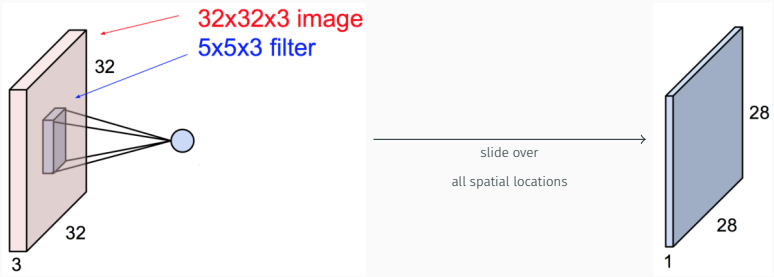
Example of backpropagation for a neuron with sigmoid activation function: $\sigma = \dfrac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + w_2)}}$

- forward step (from input to output, green): sets the "state" and output of the network corresponding to a particular input
- backward step (reverse order, red): by applying the chain rule, computes the *derivative of the output w.r.t. the input for the given state*
- $s = 2 \times (-1) + (-3) \times (-2) + (-3) = 1 \Rightarrow (1 + e^{-s})^{-1} = 0.731$
- $s = (2{-}0.2) \times (-1) + (-3{-}0.39) \times (-2) + (-3{+}0.2) = 2.18 \Rightarrow (1 + e^{-s})^{-1} = 0.898 \leftarrow$ increased



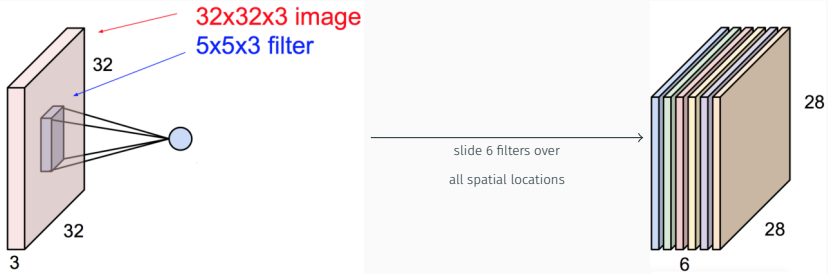
Example of CNN: a sequence of convolutional layers, pool layers and a final fully connected layer

Convolution Layers



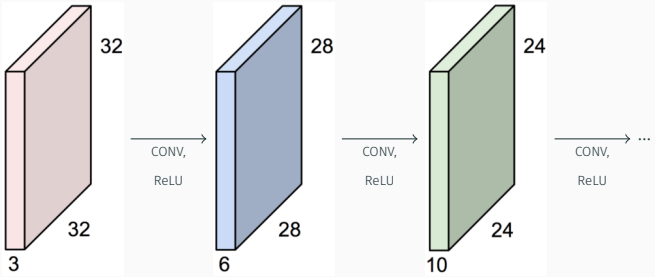
- the $28 \times 28 \times 1$ "image" on the right is called *activation map*
- each element of the activation map is obtained by computing $w^T x + b$ where w and b are the weights and bias of the filter, and x is a $5 \times 5 \times 3$ chunk of the image

Convolution Layers (cont.)



- typically, a *bank of filters* is employed
- using e.g. 6 filters, leads to a $28 \times 28 \times 6$ activation map

Convolution Layers (cont.)



- a ConvNet is a sequence of convolution layers, interspersed with activation functions

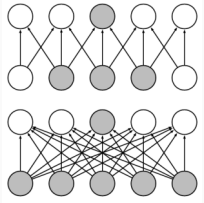
Convolution Layers (cont.)

Benefits due to convolutional layers:

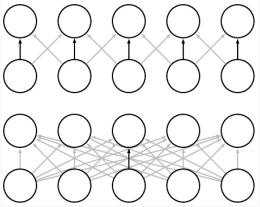
- sparse connectivity
- parameter sharing
- equivariance to translation

$$f(T(x)) = T(f(x))$$

"activation maps of translated image are translated activation maps of the original image"

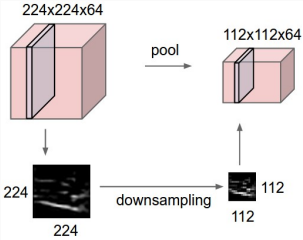


Sparse connectivity (top) vs full connectivity (bottom)

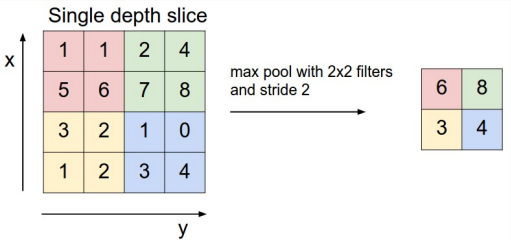


Parameter sharing (top) vs individual parameters (bottom)

Pooling layer



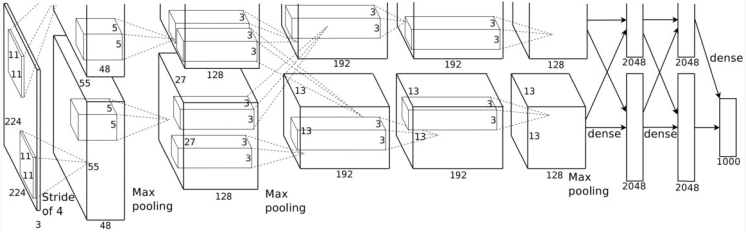
Pooling principle



Max pooling

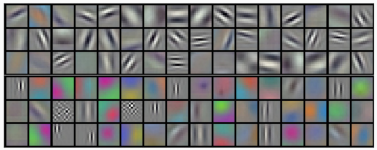
- pooling layer *downsamples the volume spatially*, independently in each depth slice of the input volume
- the most common pooling operator is **max**
- also used: average pooling

AlexNet



Imagenet classification with deep convolutional neural networks, (Krizhevsky et al., 2012)

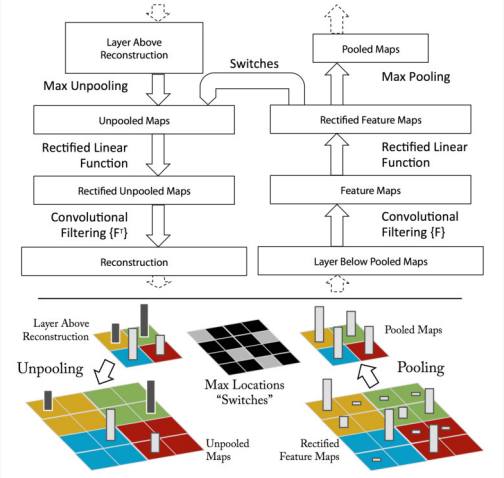
- ILSVRC 2012 winner (15.3% top five error)



Convolutional kernels learned from the first convolutional layer of AlexNet

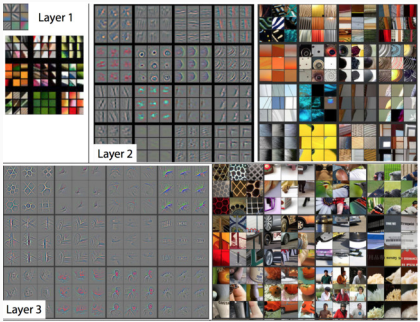
Feature visualization

- aim: understanding how a Convnet works
- are the activation maps and layers “tuned” to specific patterns?
- which input pattern activates a specific activation map of a specific layer?
- “Deconvnet” (Zeiler and Fergus, 2014) maps features to pixels (while a convolutional network does the opposite)

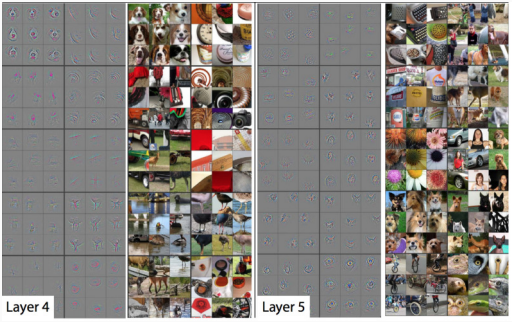


A deconvnet layer (left) attached to a convnet layer (right), (Zeiler and Fergus, 2014)

Feature visualization (cont.)



Layers 1-3

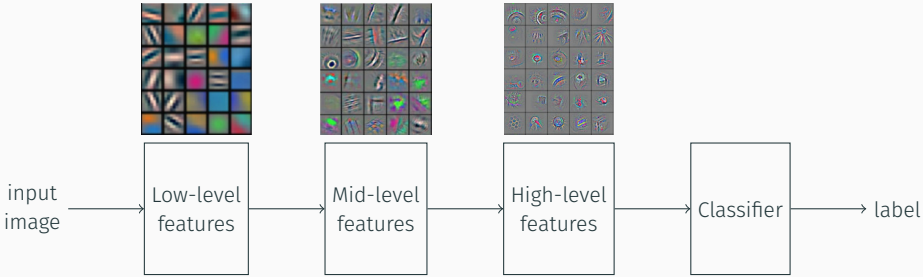


Layers 4-5

Random subsets of feature maps and corresponding image patches

- layer 2: low level features
- layer 3: textures (mainly)
- layer 4: more class-specific
- layer 5: entire objects

Feature visualization (cont.)



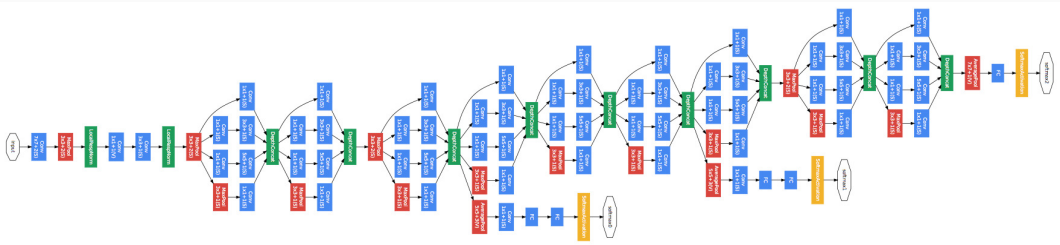
Case study: VGGNet

- systematic evaluation of many structures
- winner:
  - only 3×3 convolution kernels
  - *stride* 1
  - *pad* 1
  - only 2×2 max pool layers

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

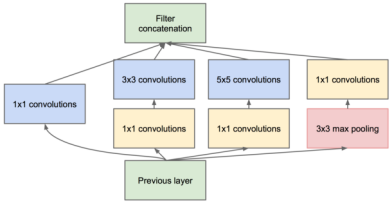
VGGNet (Simonyan and Zisserman, 2014)

Case study: GoogLe Net



GoogLe Net (Szegedy et al., 2015)

- ILSVRC 2014 winner (6.7% top five error)
- inception modules:
  - "network within a network"
  - filters with different spatial support in the same layer
- auxiliary intermediate classifiers (ignored at inference time)



Inception module

Case study: ResNet

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
		3×3 max pool, stride 2				
conv2_x	56×56	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

ResNet (He et al., 2016)

ILSVRC 2015 winner (3.6% top five error)

Training deep networks

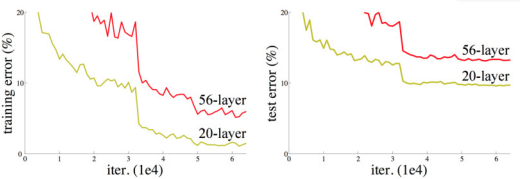


Figure 1. Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer "plain" networks. The deeper network has higher training error, and thus test error. Similar phenomena on ImageNet is presented in Fig. 4.

- network *depth* is of crucial importance
- results on ImageNet competition seem to suggest that "the deeper, the better" BUT
  - when the net is "too deep" a *degradation* phenomen occurs
  - the degradation is NOT caused by overfitting

Training deep networks

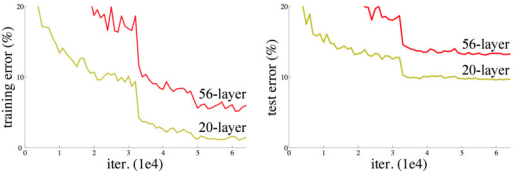
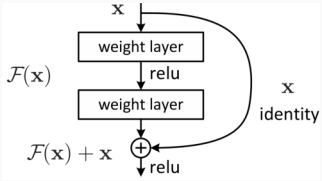


Figure 1. Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer "plain" networks. The deeper network has higher training error, and thus test error. Similar phenomena on ImageNet is presented in Fig. 4.

- network *depth* is of crucial importance
- results on ImageNet competition seem to suggest that "the deeper, the better" BUT
  - when the net is "too deep" a *degradation* phenomen occurs
  - the degradation is NOT caused by overfitting

- deeper structures are inherently more difficult to train
- "proof":
  - take a shallow network
  - add some layers having an equal amount of inputs and outputs (i.e. layers that could implement an *identity mapping*)
  - train both the networks on the same data
  - the deeper network should produce no higher training error than its shallower counterpart, but experiments with current solvers show that this is not the case

Learning the residuals

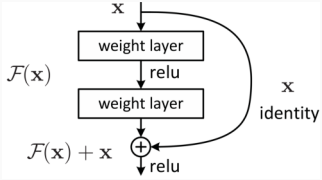


- let $\mathcal{H}(x)$ be the whole transformation from top to bottom
- instead of learning $\mathcal{H}(x)$, write

$$\mathcal{H}(x) = x + \mathcal{F}(x)$$

and learn the *residuals* $\mathcal{F}(x)$

Learning the residuals

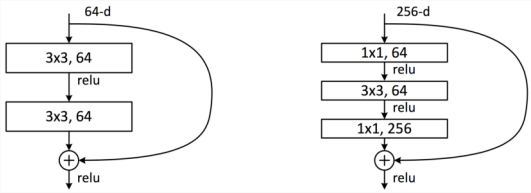


- let $\mathcal{H}(x)$ be the whole transformation from top to bottom
- instead of learning $\mathcal{H}(x)$, write

$$\mathcal{H}(x) = x + \mathcal{F}(x)$$

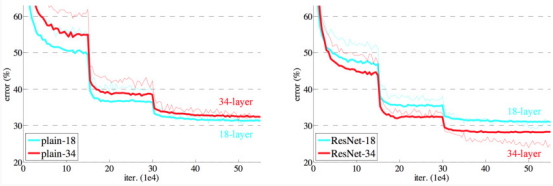
and learn the *residuals* $\mathcal{F}(x)$

- ResNet blocks learn **variations from the identity**

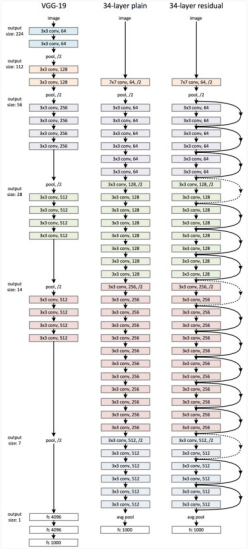


Typical blocks. With a single layer the authors did not observe advantages.

Results

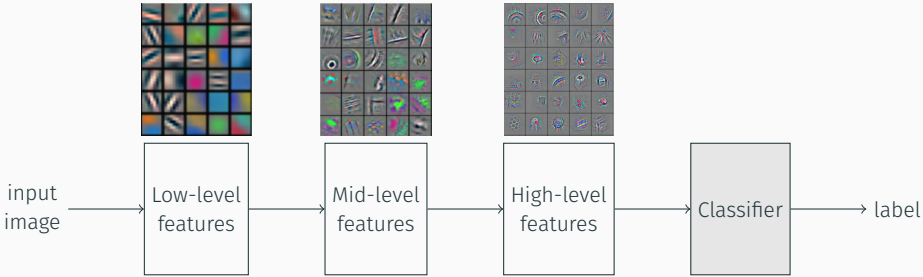


method	top-5 err. (**test**)
VGG (ILSVRC'14)	7.32
GoogLeNet (ILSVRC'14)	6.66
VGG	6.8
PReLU-net	4.94
BN-inception	4.82
ResNet (ILSVRC'15)	**3.57**



- thin curves: training error, bold curves: validation error.
- for ResNet, deeper networks show better performance as desired
- ResNet-34 has 3.6 billion FLOPS
- VGG-19 has 19.6 billion FLOPS

Transfer learning I



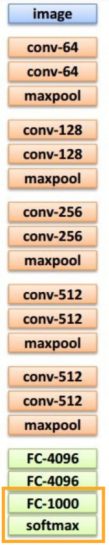
- features may be significant *per se* (especially features at lower levels)
- idea: take a pre-trained network and *retrain the classifier only*

Transfer learning II

1. train on ImageNet

Transfer learning II

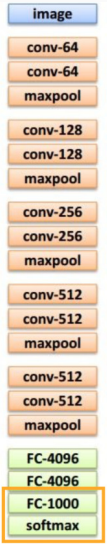
1. train on ImageNet



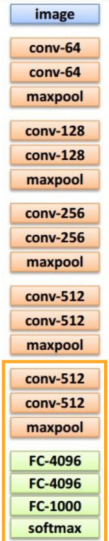
2. for small dataset: treat the CNN as a fixed *feature extractor* and train the classifier only

Transfer learning II

1. train on ImageNet

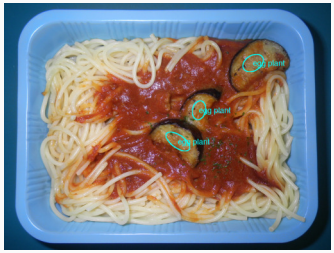


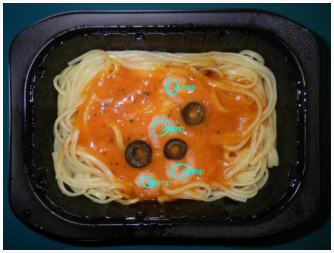
2. for small dataset: treat the CNN as a fixed *feature extractor* and train the classifier only



3. for medium-sized dataset: use the old weights as initialization and retrain a bigger portion of the net

Example: transfer learning with VGG





- transfer learning based on VGG net
- took the output of the last hidden layer (4096 elements)
- trained a linear Support Vector Machine

References

References

Belhumeur, P. N., Hespanha, J. P., and Kriegman, D. J. (1997). Eigenfaces vs. fisherfaces: recognition using class specific linear projection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(7):711–720.

Csurka, G., Dance, C., Fan, L., Willamowski, J., and Bray, C. (2004). Visual categorization with bags of keypoints. In *Workshop on statistical learning in computer vision, ECCV*, volume 1, pages 1–2. Prague.

Csurka, G., Dance, C. R., Perronnin, F., and Willamowski, J. (2006). Generic visual categorization using weak geometry. In *Toward Category-Level Object Recognition*, pages 207–224. Springer.

Dalal, N. and Triggs, B. (2005). Histograms of oriented gradients for human detection. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 886–893. IEEE.

Duda, R. O., Hart, P. E., and Stork, D. G. (2012). *Pattern classification*. John Wiley & Sons.

Fei-Fei, L. and Perona, P. (2005). A bayesian hierarchical model for learning natural scene categories. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 2, pages 524–531. IEEE.

Felzenszwalb, P., McAllester, D., and Ramanan, D. (2008). A discriminatively trained, multiscale, deformable part model. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8. IEEE.

Fischer, M. and Elschlager, R. (1973). The representation and matching of pictorial structure. *IEEE Trans. Comput*, 1:67–92.

Freund, Y. (1995). Boosting a weak learning algorithm by majority. *Information and computation*, 121(2):256–285.

References (cont.)

Freund, Y. and Schapire, R. E. (1997). A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences*, 55(1):119–139.

Grauman, K. and Darrell, T. (2007). The pyramid match kernel: Efficient learning with sets of features. *Journal of Machine Learning Research*, 8(Apr):725–760.

Grauman, K. and Leibe, B. (2011). *Visual object recognition*. Morgan & Claypool Publishers.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105.

Lazebnik, S., Schmid, C., and Ponce, J. (2006). Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In *null*, pages 2169–2178. IEEE.

Lowe, D. G. (1999). Object recognition from local scale-invariant features. In *Computer Vision, 1999. The proceedings of the seventh IEEE International Conference on*, volume 2, pages 1150–1157. Ieee.

Lowe, D. G. (2004). Distinctive Image Features from Scale-Invariant Keypoints. *International Journal of Computer Vision*, 60(2):91–110.

Maji, S., Berg, A. C., and Malik, J. (2008). Classification using intersection kernel support vector machines is efficient. In *2008 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8.

References (cont.)

Matas, J., Chum, O., Urban, M., and Pajdla, T. (2004). Robust wide-baseline stereo from maximally stable extremal regions. *Image and vision computing*, 22(10):761–767.

McCulloch, W. S. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133.

Moghaddam, B. and Pentland, A. (1997). Probabilistic visual learning for object representation. *IEEE Transactions on pattern analysis and machine intelligence*, 19(7):696–710.

Nister, D. and Stewenius, H. (2006). Scalable recognition with a vocabulary tree. In *Computer vision and pattern recognition, 2006 IEEE computer society conference on*, volume 2, pages 2161–2168. Ieee.

Oren, M., Papageorgiou, C., Sinha, P., Osuna, E., and Poggio, T. (1997). Pedestrian detection using wavelet templates. In *Computer Vision and Pattern Recognition, 1997. Proceedings., 1997 IEEE Computer Society Conference on*, pages 193–199. IEEE.

Philbin, J., Chum, O., Isard, M., Sivic, J., and Zisserman, A. (2007). Object retrieval with large vocabularies and fast spatial matching. In *Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on*, pages 1–8. IEEE.

Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386.

Rubner, Y., Tomasi, C., and Guibas, L. J. (1998). A metric for distributions with applications to image databases. In *Computer Vision, 1998. Sixth International Conference on*, pages 59–66. IEEE.

References (cont.)

Simonyan, K. and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556.

Sivic and Zisserman (2003). Video google: a text retrieval approach to object matching in videos. In *Proceedings Ninth IEEE International Conference on Computer Vision*, pages 1470–1477 vol.2.

Sivic, J. and Zisserman, A. (2009). Efficient visual search of videos cast as text retrieval. *IEEE transactions on pattern analysis and machine intelligence*, 31(4):591–606.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2015). Going deeper with convolutions. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Szeliski, R. (2010). *Computer vision: algorithms and applications*. Springer Science & Business Media.

Turk, M. and Pentland, A. (1991). Eigenfaces for recognition. *Journal of cognitive neuroscience*, 3(1):71–86.

Viola, P. and Jones, M. (2001). Rapid object detection using a boosted cascade of simple features. In *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, volume 1, pages I–I. IEEE.

Vondrick, C., Khosla, A., Malisiewicz, T., and Torralba, A. (2013). Hoggles: Visualizing object detection features. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1–8.

Zeiler, M. D. and Fergus, R. (2014). Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833. Springer.

Zhang, J., Marszałek, M., Lazebnik, S., and Schmid, C. (2007). Local features and kernels for classification of texture and object categories: A comprehensive study. *International journal of computer vision*, 73(2):213–238.

554SM –Fall 2018

Lecture 8
Recognition

END