# CS 554
# Computer Vision

## Shape, Scene, and Object Recognition


**Hamdi Dibeklioğlu**


Slide Credits: L. van der Maaten

# Shape recognition

# Shape recognition

*Region-based descriptors* describe region occupied by the shape

*Contour-based descriptors* describe the contour of the shape

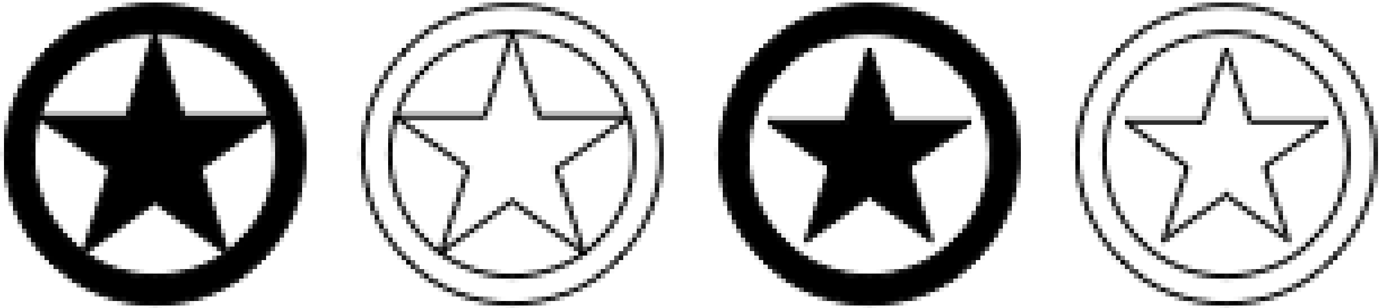
How do the two types of descriptors differ on the following two shapes?

# Shape recognition

*Region-based descriptors* describe region occupied by the shape

*Contour-based descriptors* describe the contour of the shape

How do the two types of descriptors differ on the following two shapes?



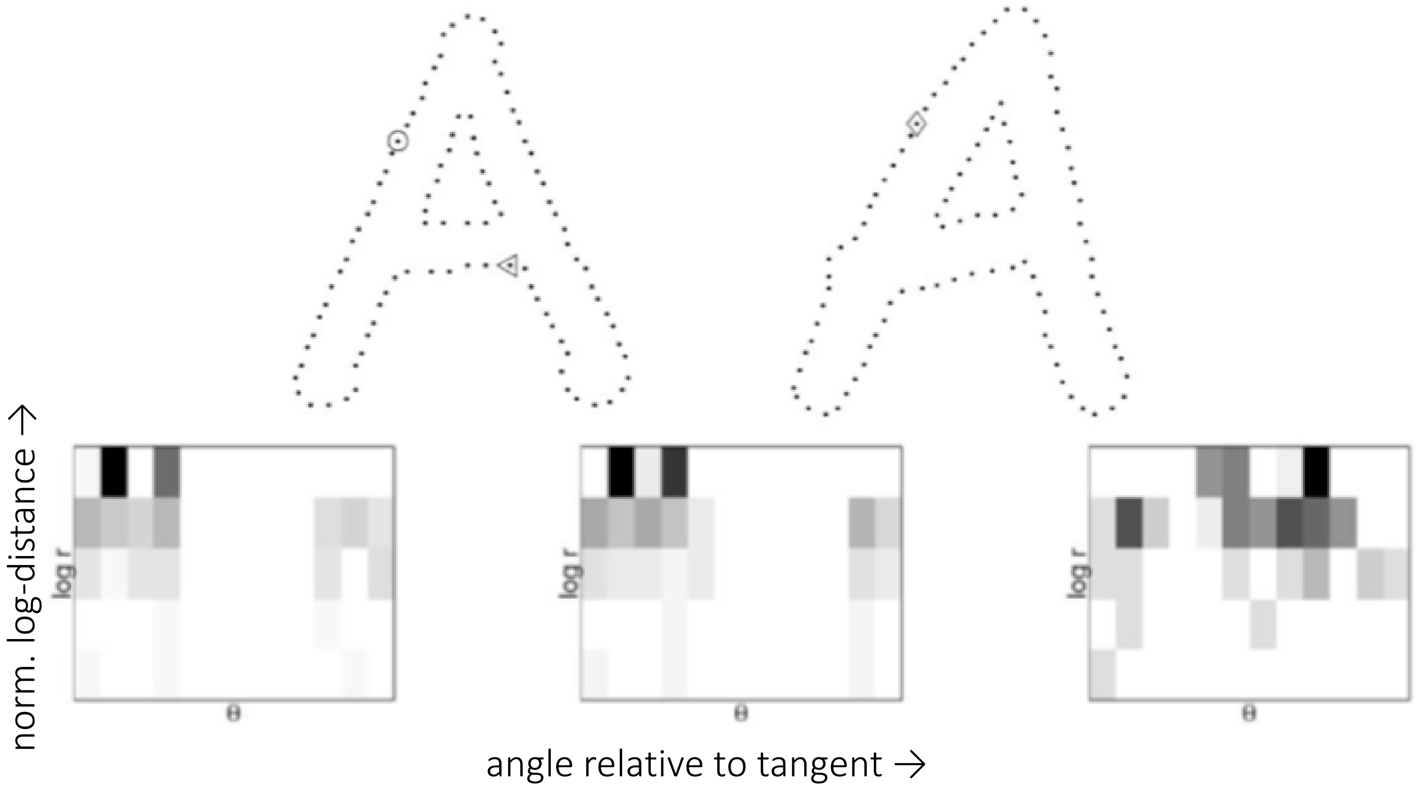
# Shape contexts

Contour-based method to measure shape dissimilarity that works as follows:

1) Sample points from the contours of two shapes
2) Describe points using distance-angle histograms
3) Match points by solving an assignment problem between descriptors
4) Warp target shape using the assignments; return to step 3)
5) Final shape dissimilarity is the energy of the warp plus the residual error

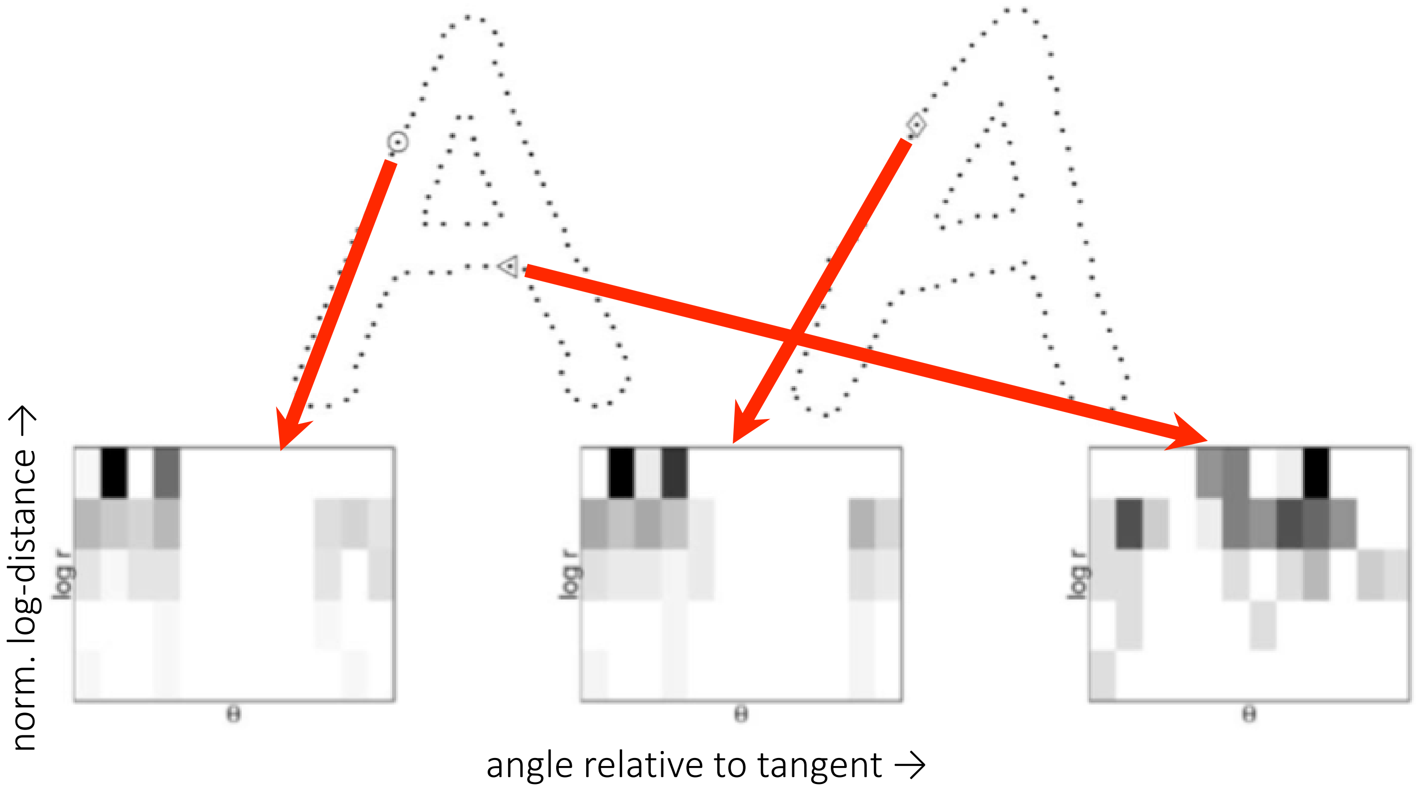
# Shape contexts

Histogram of *log-distance* and *relative angle* to all other points:



# Shape contexts

Histogram of *log-distance* and *relative angle* to all other points:



# Shape contexts

Compute distances between shape-context descriptors (Euclidean)

Solve assignment problem between the two point sets
(using Hungarian algorithm):

| | Bathroom | Floor | Windows |
|---|---|---|---|
| Jim | 1$ | 2$ | 3$ |
| Steve | 3$ | 3$ | 3$ |
| Allan | 3$ | 3$ | 2$ |

# Shape contexts

Compute distances between shape-context descriptors (Euclidean)

Solve assignment problem between the two point sets
(using Hungarian algorithm):

| | Bathroom | Floor | Windows |
|---|---|---|---|
| Jim | $\boxed{1\$}$ | 2\$ | 3\$ |
| Steve | 3\$ | $\boxed{3\$}$ | 3\$ |
| Allan | 3\$ | 3\$ | $\boxed{2\$}$ |

Optimal: Jim cleans bathroom, Steve cleans floor, and Allan cleans windows

# Shape contexts

Compute distances between shape-context descriptors (Euclidean)

Solve assignment problem between the two point sets
(using Hungarian algorithm):

| | SC2.1 | SC2.2 | SC2.3 |
|---|---|---|---|
| SC1.1 | 1 | 2 | 3 |
| SC1.2 | 3 | 3 | 3 |
| SC1.3 | 3 | 3 | 2 |

We now know which points correspond to each other

# Shape contexts

Compute distances between shape-context descriptors (Euclidean)

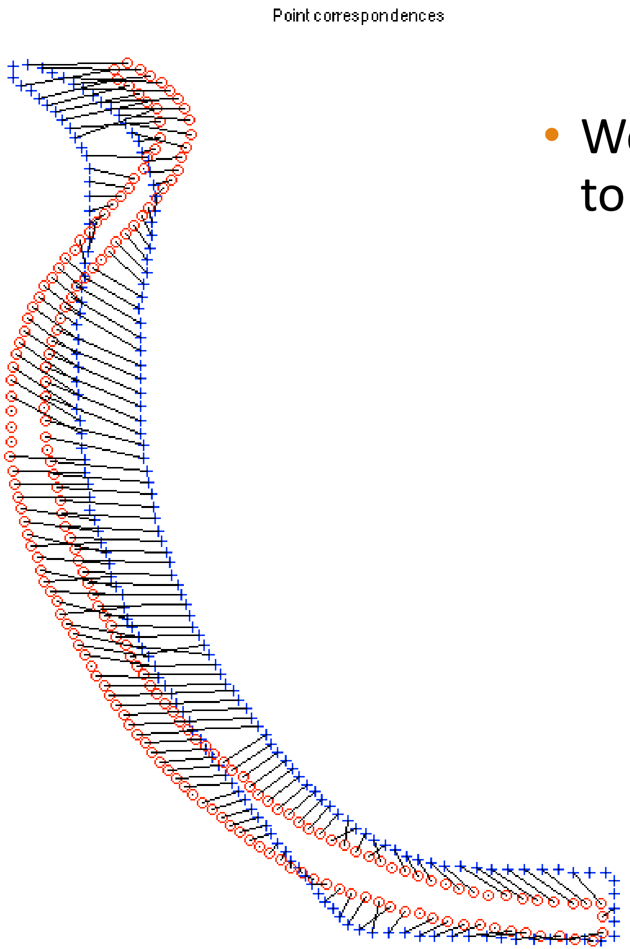
Solve assignment problem between the two point sets
(using Hungarian algorithm):

| | SC2.1 | SC2.2 | SC2.3 |
|---|---|---|---|
| SC1.1 | 1 | 2 | 3 |
| SC1.2 | 3 | 3 | 3 |
| SC1.3 | 3 | 3 | 2 |

We now know which points correspond to each other

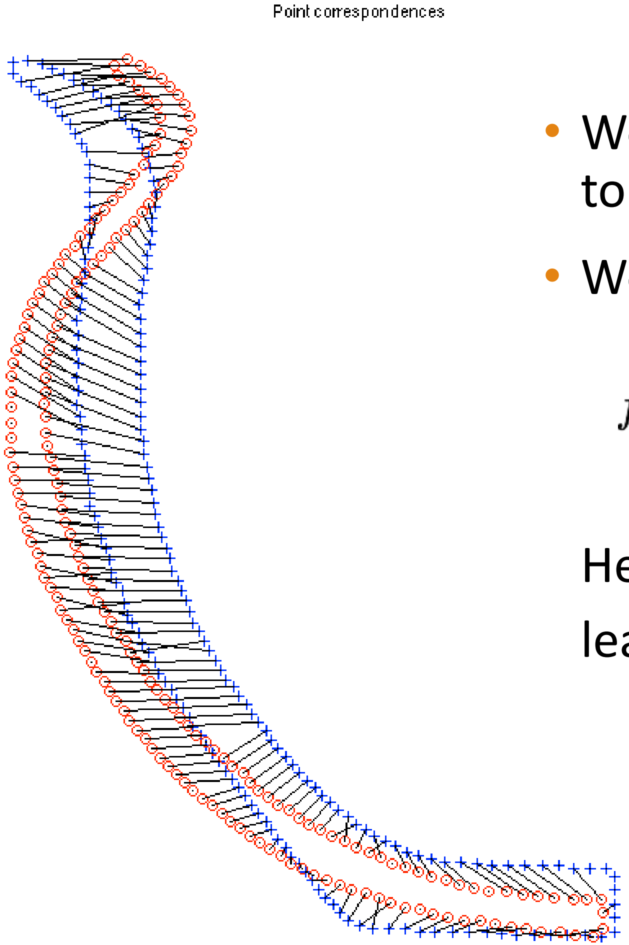
- We could have used RANSAC to find the correspondences

# Shape contexts




- We can use the correspondences as *control points* to find a *warp*

# Shape contexts




- We can use the correspondences as *control points* to find a *warp*
- We use a *thin-plate spline warp*:

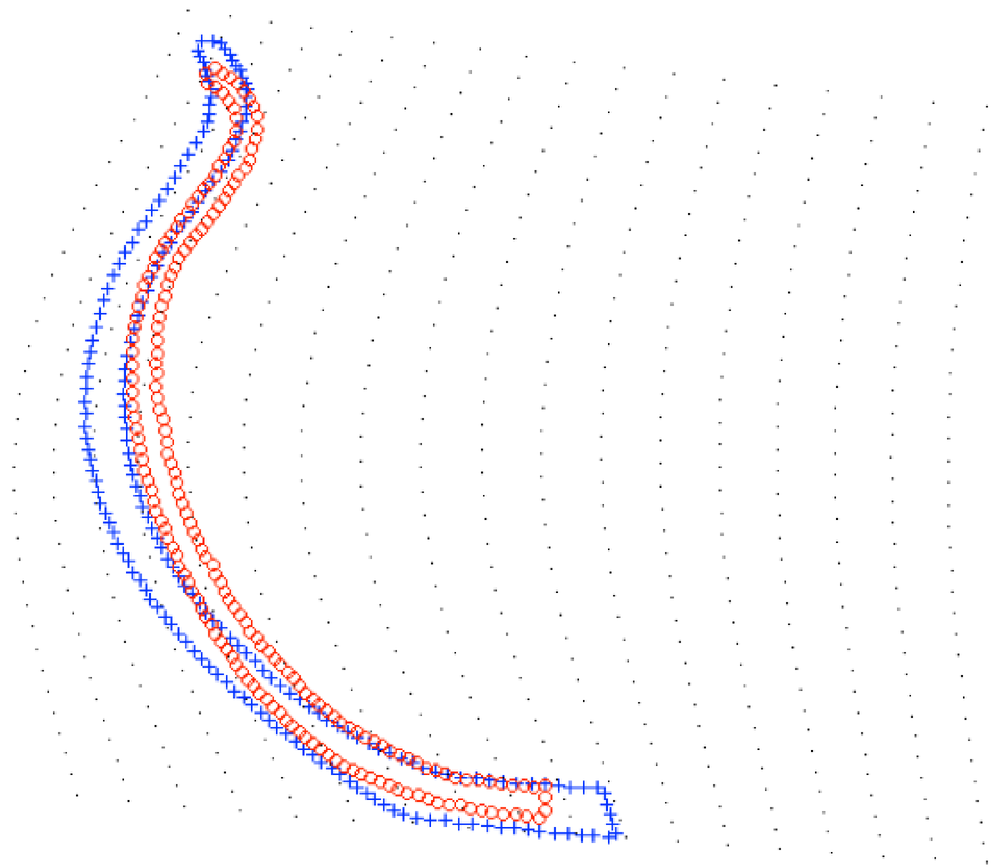
$$f(x,y) = a_1 + a_x x + a_y y + \sum_{i=1}^{K} w_i U(\|(x_i,y_i)-(x,y)\|)$$

Herein, $U(r) = r^2 \log r^2$ and the parameters are learned via LLS:

$$\min \sum_{i=1}^{K} (x_i' - f_x(x_i,y_i))^2$$

$$\min \sum_{i=1}^{K} (y_i' - f_y(x_i,y_i))^2$$

# Shape contexts



# Shape contexts

Chicken-and-egg problem, the warp changed the point correspondences:

- Iterate for a few iterations or until convergence

# Shape contexts

Chicken-and-egg problem, the warp changed the point correspondences:

- Iterate for a few iterations or until convergence

The final dissimilarity between the two shapes is now a sum of:

- The sum of distances between corresponding points after the final warp
- The bending energy of the final warp

# Shape contexts

Chicken-and-egg problem, the warp changed the point correspondences:

- Iterate for a few iterations or until convergence

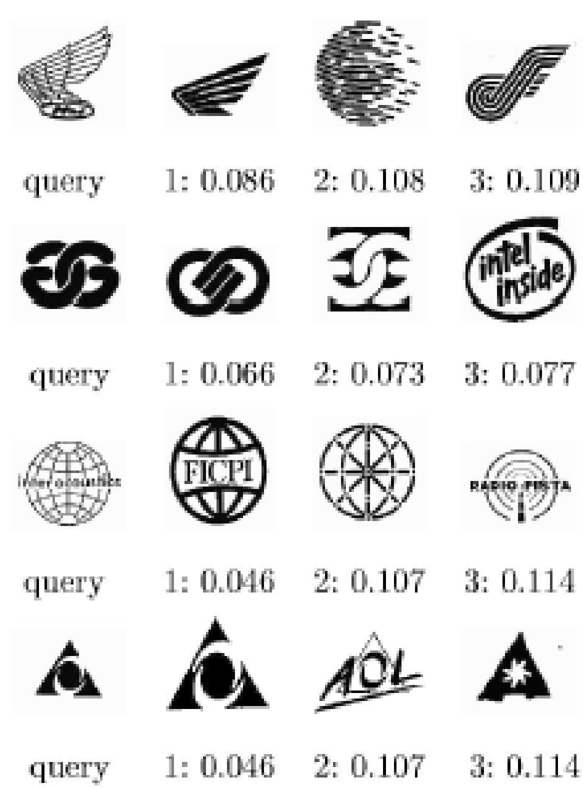
The final dissimilarity between the two shapes is now a sum of:

- The sum of distances between corresponding points after the final warp
- The bending energy of the final warp

Note that shape contexts are translation-, rotation-, and scale-invariant

# Shape contexts

Example of using shape contexts to retrieve logos:




- Attendance will be tracked through Zoom.
  - Please set your participant ID as your NAME & SURNAME.
- Attendance requirement has been lifted for this course, however, in-class participation and attendance still have a 5% impact on your grade.
- Midterm exam has been postponed indefinitely.
- Details of the homework will be announced today.
  - 2-people groups; maybe 1-person if required (needs my approval).
  - Topic: Image Stitching & Depth Estimation

## CS 554 – Computer Vision

| Task | Report Submission Deadline | Presentation Date |
|---|---|---|
| Homework | 15 April 2020, 23.59 (TR) | n/a |
| Survey | 26 April 2020, 23.59 (TR) | 27/30 April 2020 |
| Project Progress | 22 April 2020, 23.59 (TR) | n/a |
| Project | 8 May 2020, 23.59 (TR) | 11/14 May 2020 |

## CS 559 – Deep Learning

| Task | Report Submission Deadline | Presentation Date |
|---|---|---|
| Homework | 13 April 2020, 23.59 (TR) | n/a |
| Survey | 3 May 2020, 23.59 (TR) | 4/6 May 2020 |
| Project Progress | 20 April 2020, 23.59 (TR) | 20 April 2020 |
| Project | 10 May 2020, 23.59 (TR) | 11/13 May 2020 |

# Scene recognition

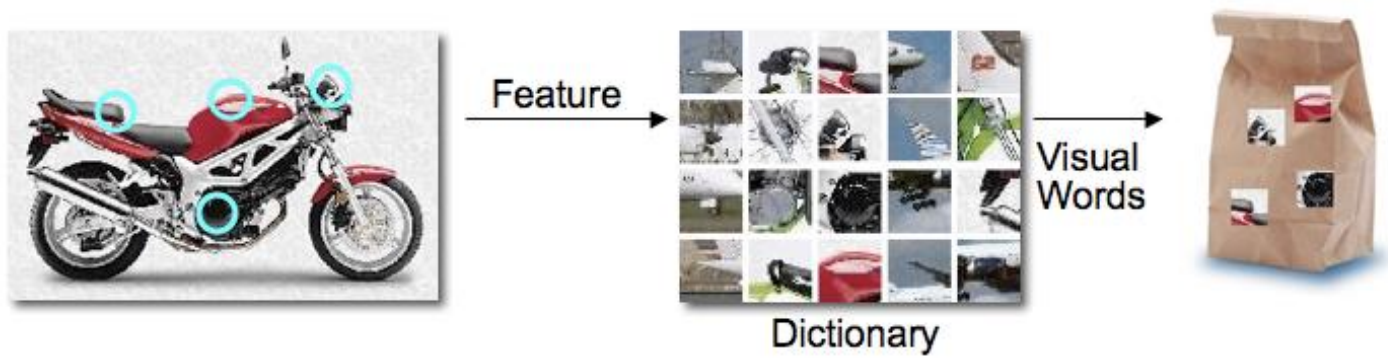
# Scene recognition

*Scenes* are constellations of different objects in somewhat arbitrary locations:

# Bag of visual words

We have already seen descriptors for local image patches (*e.g.*, SIFT)

Bag-of-visual-words features consider an image as a *bag* of image patches, *ignoring* the spatial relations between those images:




How do we obtain the *dictionary* of visual words?

# Bag of visual words

Construction of bag-of-visual-words features proceeds in four main steps:

1. Gather a collection of randomly selected image patches
2. Perform k-means clustering on the image patches (using some descriptor for the patch) to obtain a *codebook* or *dictionary* of visual words
3. Gather image patches from each image (using keypoint detector or dense sampling)
4. Describe each image by counting how often each word appears in the image

# Bag of visual words

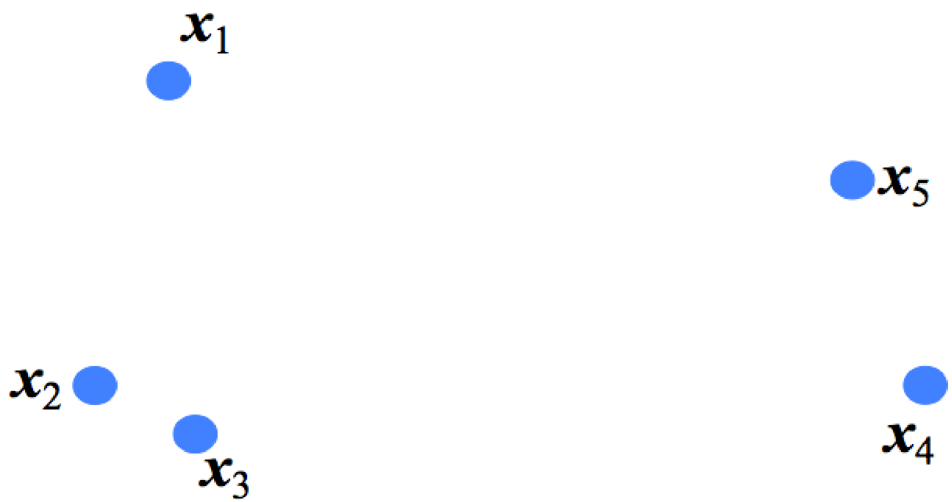
The *k*-means algorithms finds clusters via an iterative process:

- Given *K* cluster centers, determine current assignments
- Given cluster assignments, compute the *K* cluster centers

This procedure minimizes the sum of squared Euclidean distances between each point and its corresponding cluster center ( $c_n \in \{1, \ldots, K\}$):

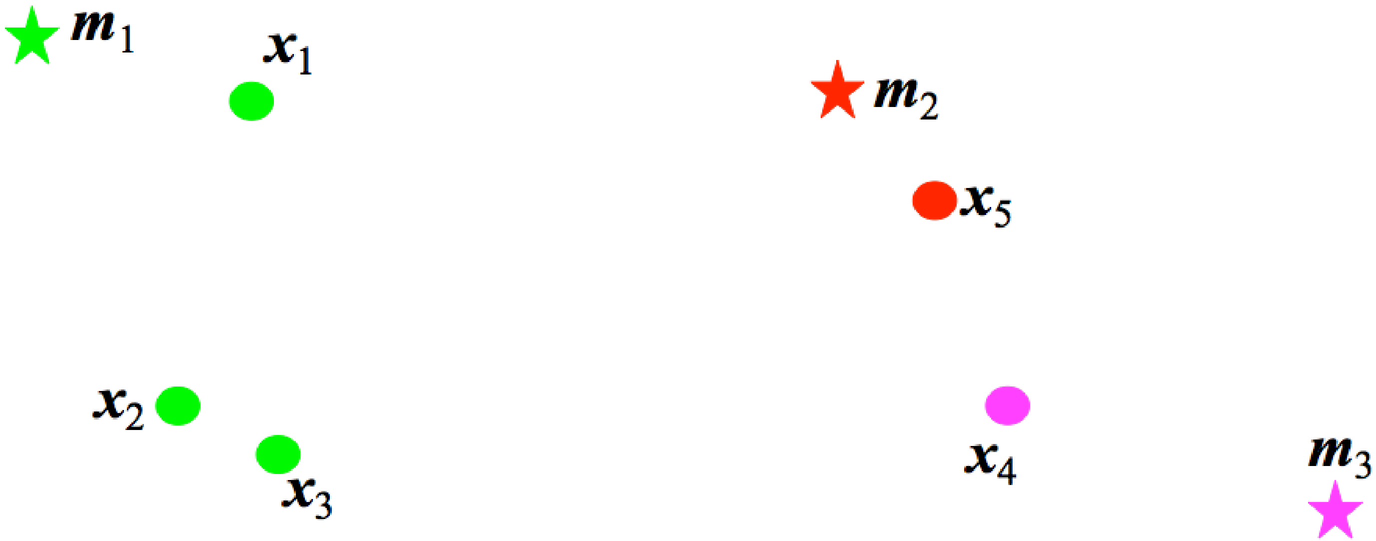
$$\sum_{n=1}^{N} \|\mathbf{x}_n - \boldsymbol{\mu}_{c_n}\|^2$$

# K-means algorithm

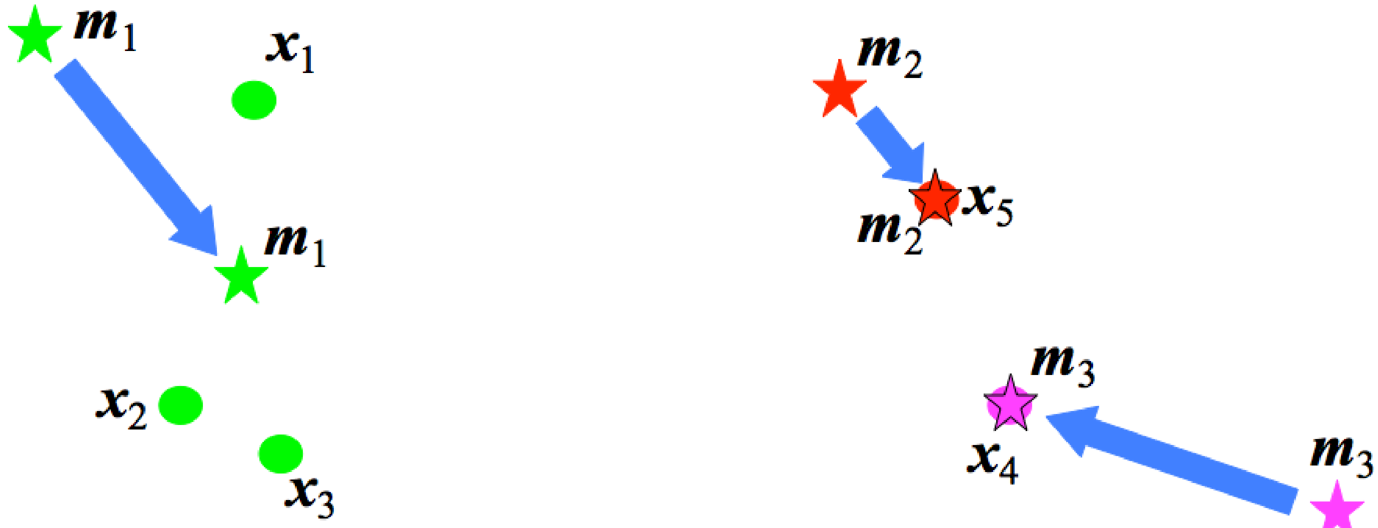


# K-means algorithm

# K-means algorithm



# K-means algorithm



# K-means algorithm

# K-means algorithm

Overall objective: $$\min_{(\boldsymbol{\mu}_1,\ldots,\boldsymbol{\mu}_K,\mathbf{c})} \sum_{n=1}^{N} \|\mathbf{x}_n - \boldsymbol{\mu}_{c_n}\|^2$$

# K-means algorithm

Overall objective: $$\min_{(\boldsymbol{\mu}_1,\ldots,\boldsymbol{\mu}_K,\mathbf{c})} \sum_{n=1}^{N} \|\mathbf{x}_n - \boldsymbol{\mu}_{c_n}\|^2$$

centers of
all *K* clusters

# K-means algorithm

Overall objective: $$\min_{(\boldsymbol{\mu}_1,\ldots,\boldsymbol{\mu}_K,\mathbf{c})} \sum_{n=1}^{N} \|\mathbf{x}_n - \boldsymbol{\mu}_{c_n}\|^2$$

centers of
all *K* clusters

cluster assignments:
vector of length *N* with
values between *1* and *K*

# K-means algorithm

Overall objective: $$\min_{(\boldsymbol{\mu}_1,\ldots,\boldsymbol{\mu}_K,\mathbf{c})} \sum_{n=1}^{N} \|\mathbf{x}_n - \boldsymbol{\mu}_{c_n}\|^2$$

centers of
all *K* clusters

minimize squared Euclidean distance
between each data point and its
corresponding cluster center

cluster assignments:
vector of length *N* with
values between *1* and *K*

# K-means algorithm

Overall objective: $$\min_{(\boldsymbol{\mu}_1,\ldots,\boldsymbol{\mu}_K,\mathbf{c})} \sum_{n=1}^{N} \|\mathbf{x}_n - \boldsymbol{\mu}_{c_n}\|^2$$

<u>Step 1</u>: Assign data to clusters: $$\min_{\mathbf{c}} \sum_{n=1}^{N} \|\mathbf{x}_n - \boldsymbol{\mu}_{c_n}\|^2$$

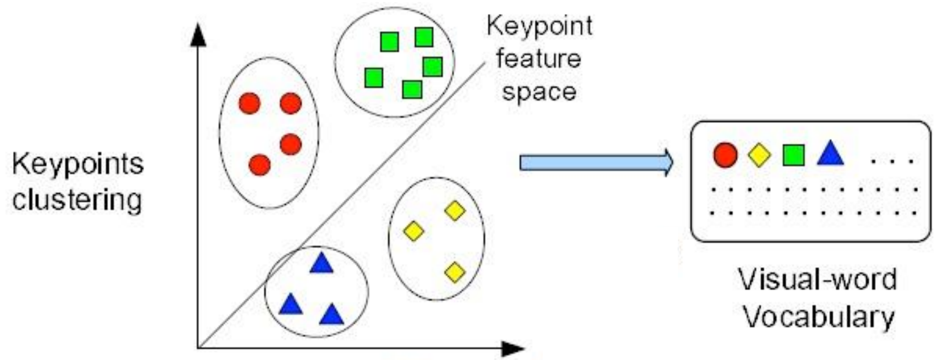
# K-means algorithm

Overall objective: $$\min_{(\boldsymbol{\mu}_1,\ldots,\boldsymbol{\mu}_K,\mathbf{c})} \sum_{n=1}^{N} \|\mathbf{x}_n - \boldsymbol{\mu}_{c_n}\|^2$$

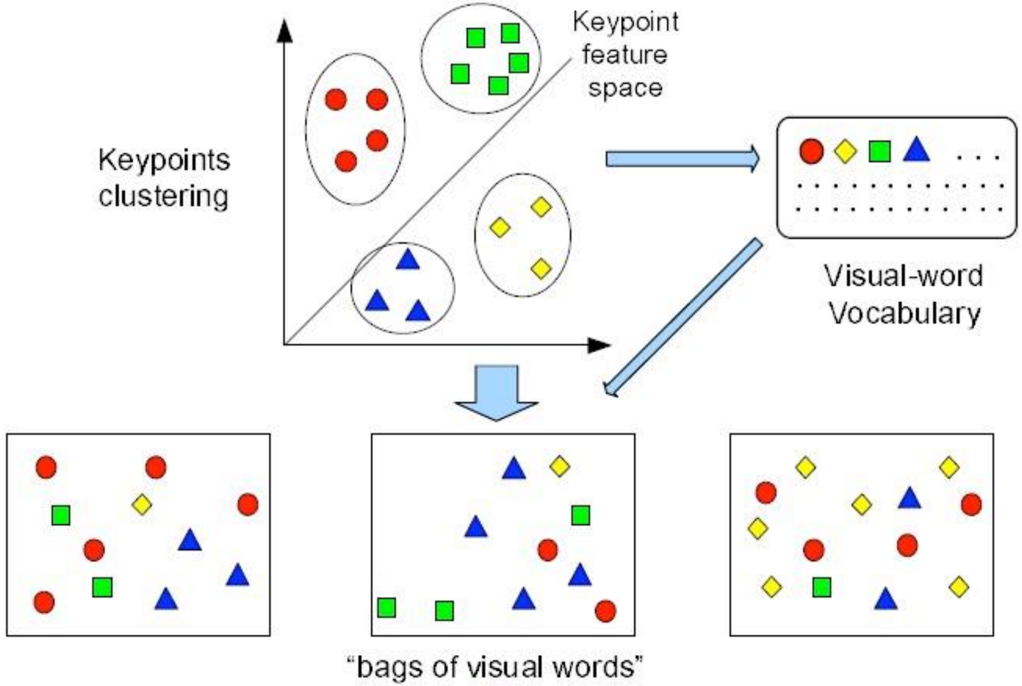
<u>Step 1:</u> Assign data to clusters: $$\min_{\mathbf{c}} \sum_{n=1}^{N} \|\mathbf{x}_n - \boldsymbol{\mu}_{c_n}\|^2$$

<u>Step 2:</u> Compute cluster means: $$\min_{(\boldsymbol{\mu}_1,\ldots,\boldsymbol{\mu}_K)} \sum_{n=1}^{N} \|\mathbf{x}_n - \boldsymbol{\mu}_{c_n}\|^2$$

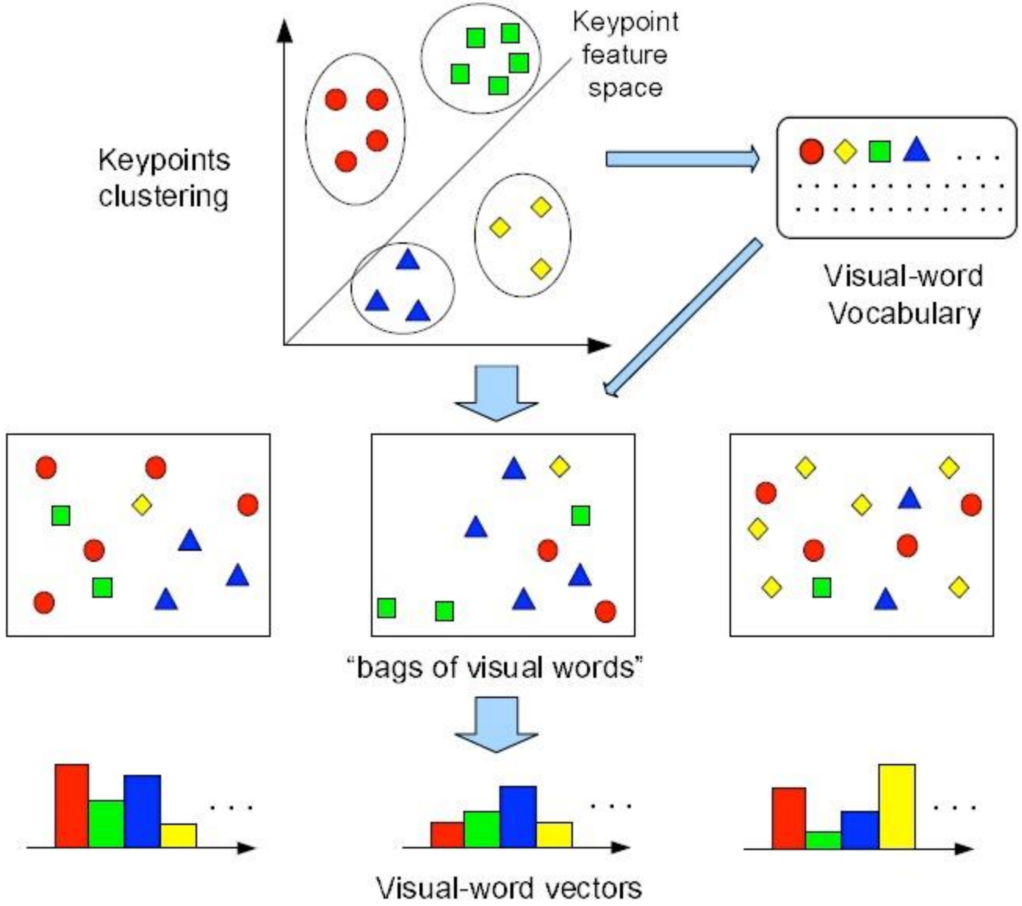
# Bag of visual words



# Bag of visual words

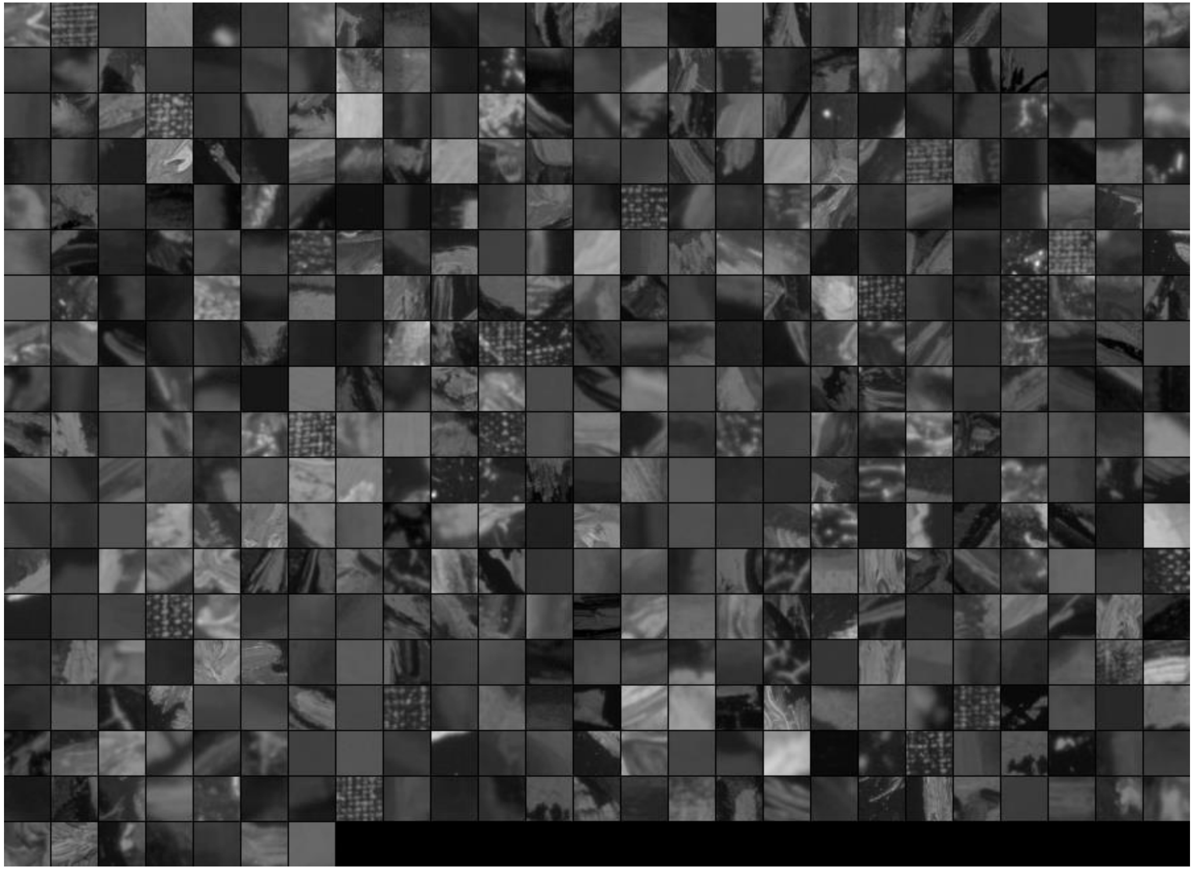


# Bag of visual words



# Bag of visual words

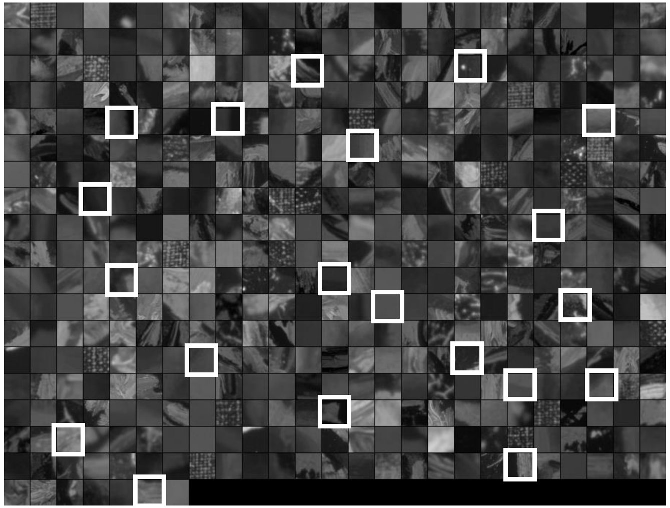
Example of a *(texton) codebook* build using k-means on small patches:



# Bag of visual words

RHS: *Texton codebook* build by performing k-means on small patches

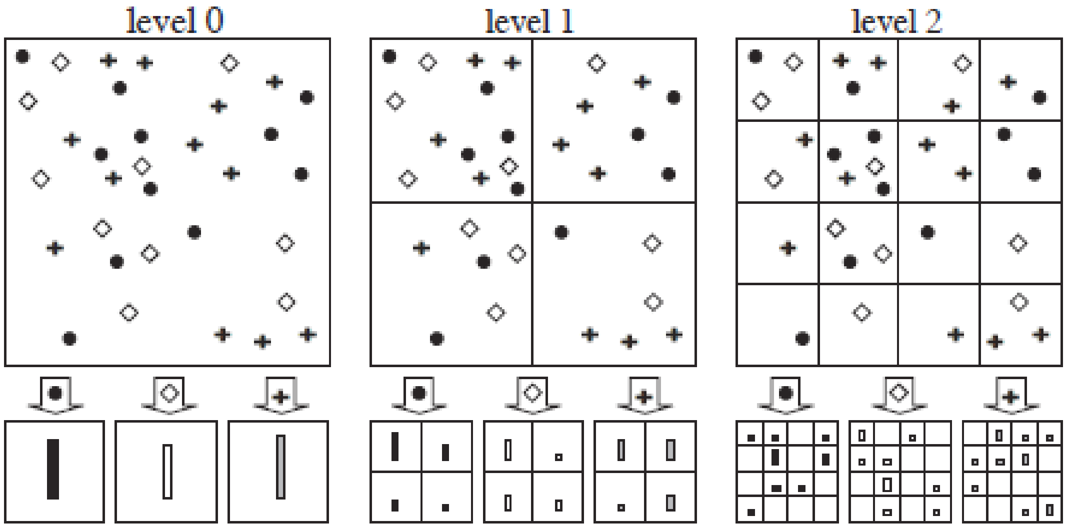
LHS: Find nearest neighbor in codebook for every image patch, and bin result



etcetera...

# Bag of visual words

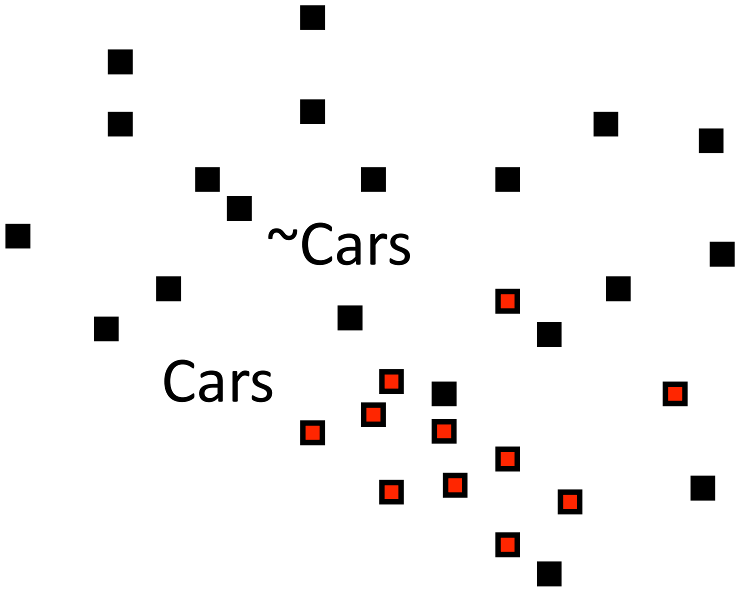
Spatial *pyramid matching* incorporates spatial structure in bag-of-words:




Histograms are constructed per segment (on multiple scales)

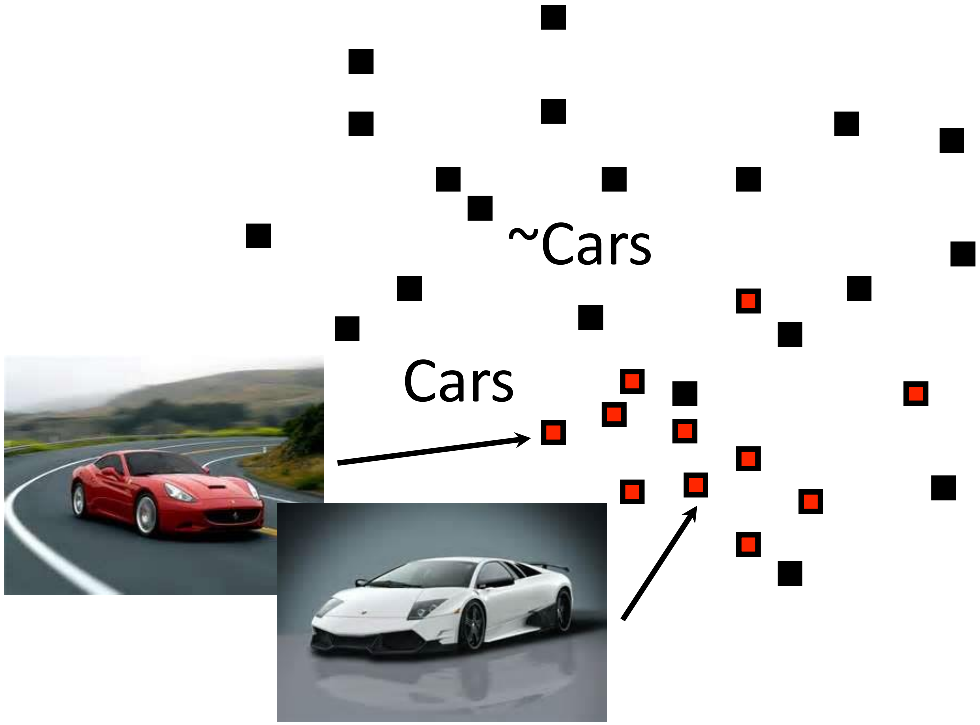
# Object / scene recognition

Bag-of-words features can be used as input into a classifier:



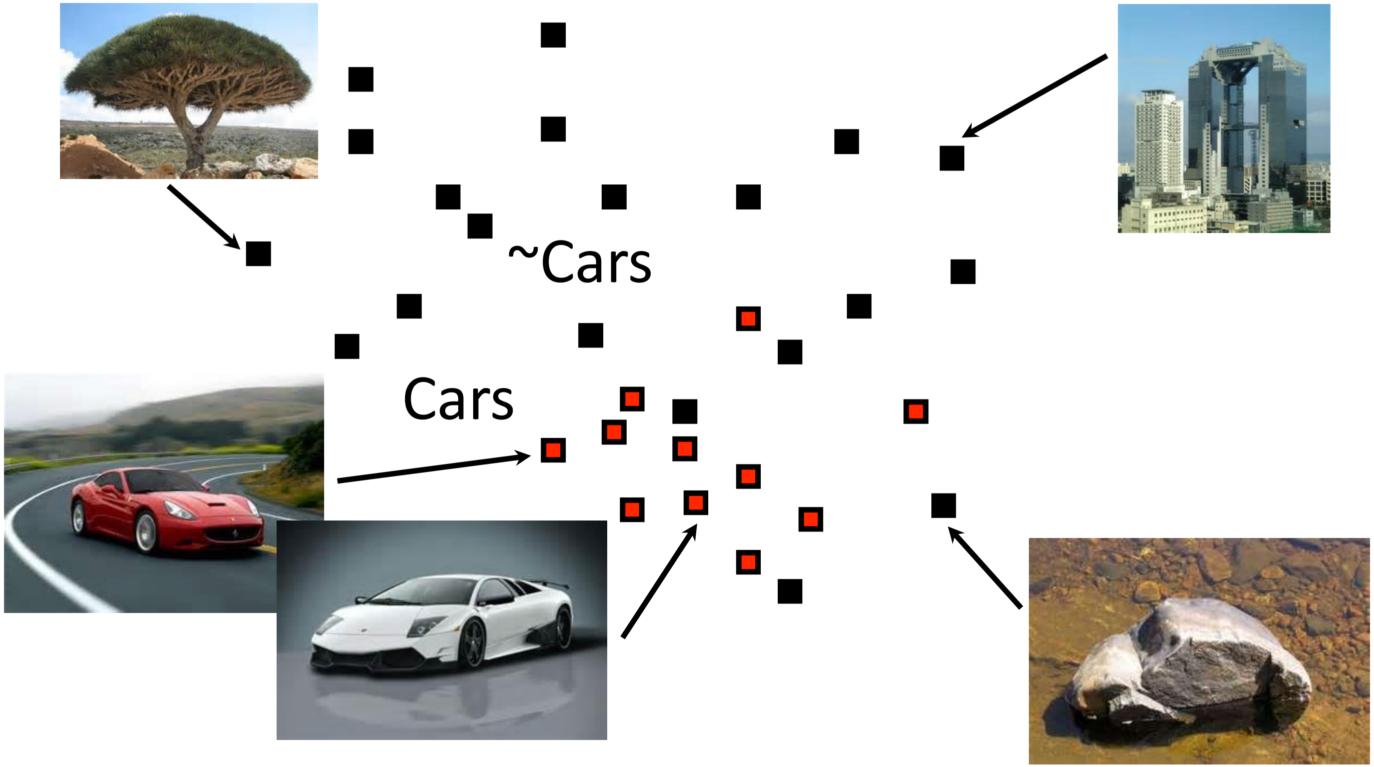
# Object / scene recognition

Bag-of-words features can be used as input into a classifier:



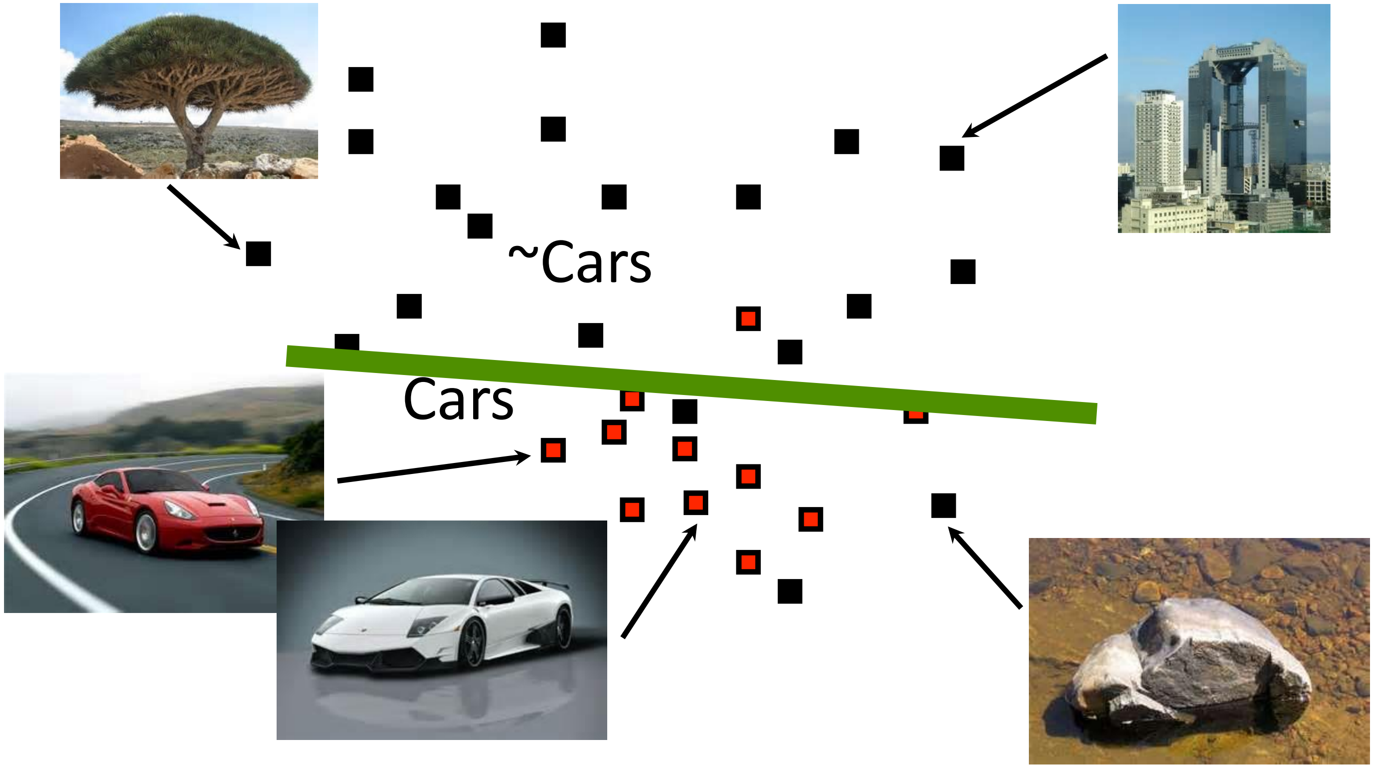
# Object / scene recognition

Bag-of-words features can be used as input into a classifier:



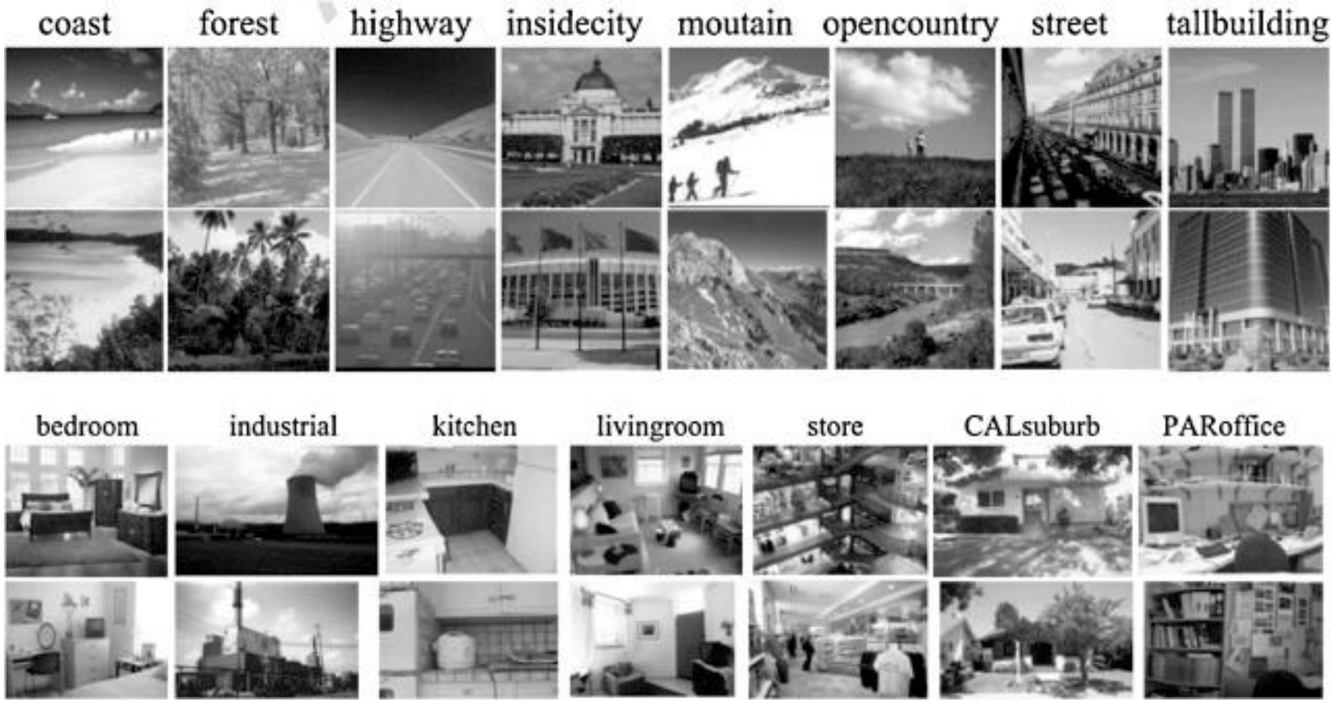
# Object / scene recognition

Bag-of-words features can be used as input into a classifier:



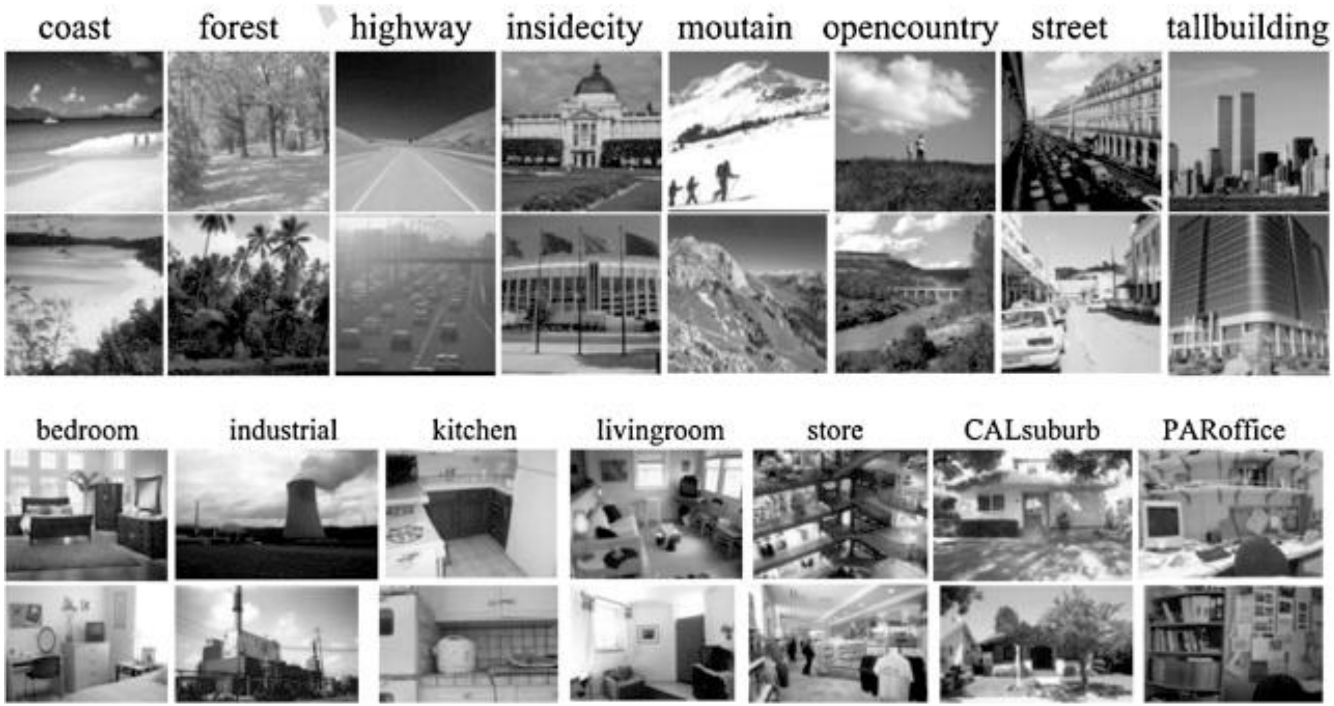
# Example: Scene understanding

Bag-of words representations are particularly good for *scene understanding*:



# Example: Scene understanding

Bag-of words representations are particularly good for *scene understanding*:




Scene is not determined by specific objects, but by a *constellation* of features

# Object recognition

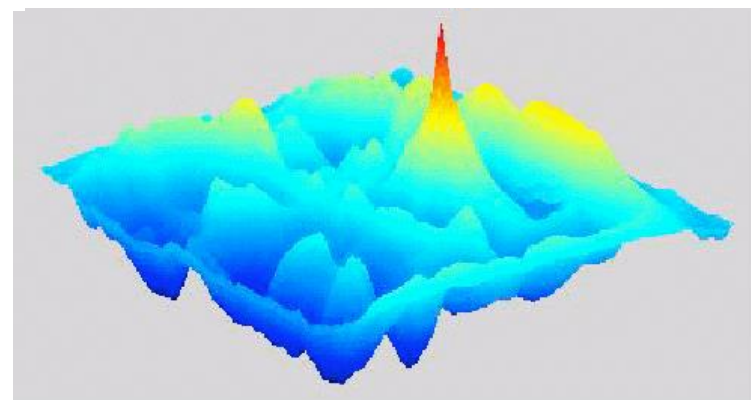
# Object recognition

Simple approach matches a *template* with the image to find its location:

- E.g., use *linear filter* or *normalized cross-correlation*: $\frac{1}{N}\sum_{x,y}\frac{(I(x,y)-\overline{I})(T(x,y)-\overline{T})}{\sigma_I \sigma_T}$

+

=



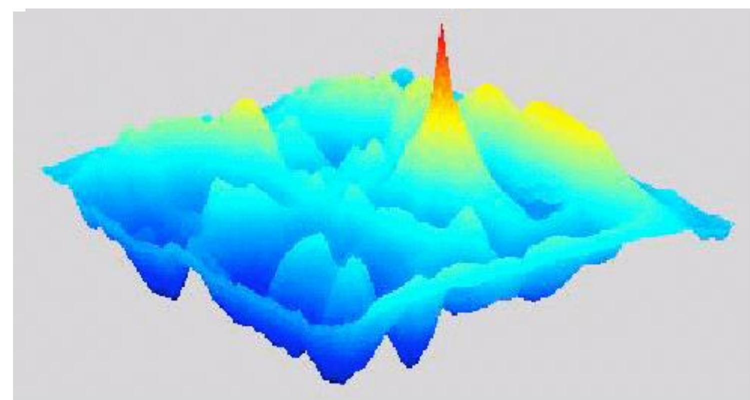
# Object recognition

Simple approach matches a *template* with the image to find its location:

- E.g., use *linear filter* or *normalized cross-correlation*: $\frac{1}{N}\sum_{x,y}\frac{(I(x,y)-\overline{I})(T(x,y)-\overline{T})}{\sigma_I\sigma_T}$

+

=

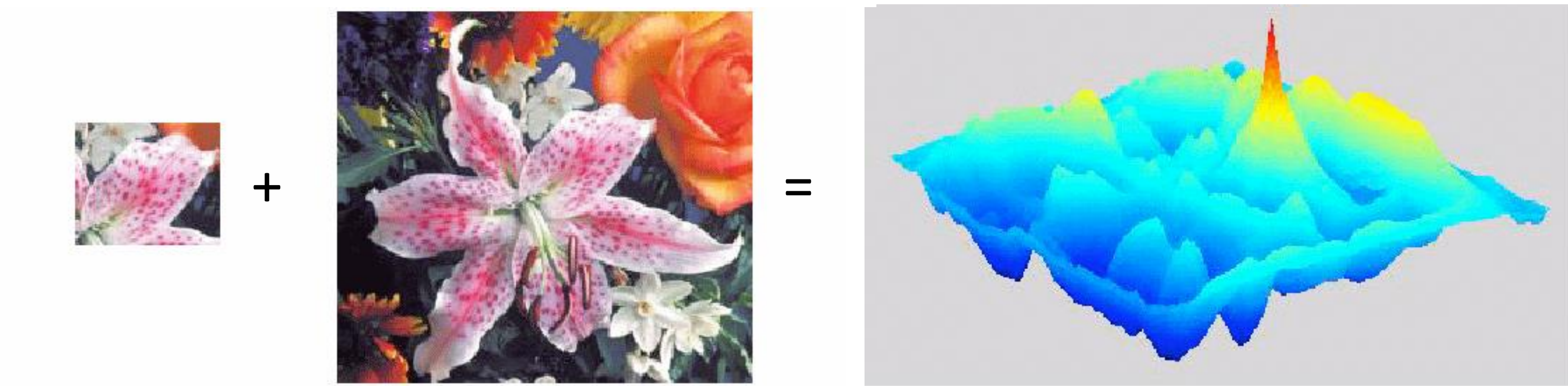


- How do we obtain the template?

# Object recognition

Simple approach matches a *template* with the image to find its location:

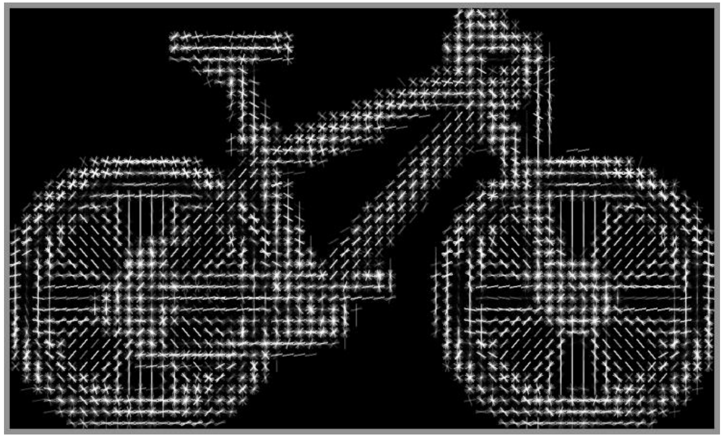
- E.g., use *linear filter* or *normalized cross-correlation*: $\frac{1}{N}\sum_{x,y}\frac{(I(x,y)-\overline{I})(T(x,y)-\overline{T})}{\sigma_I \sigma_T}$



- How do we obtain the template? Pattern recognition: train a classifier!

# Dalal-Triggs detector

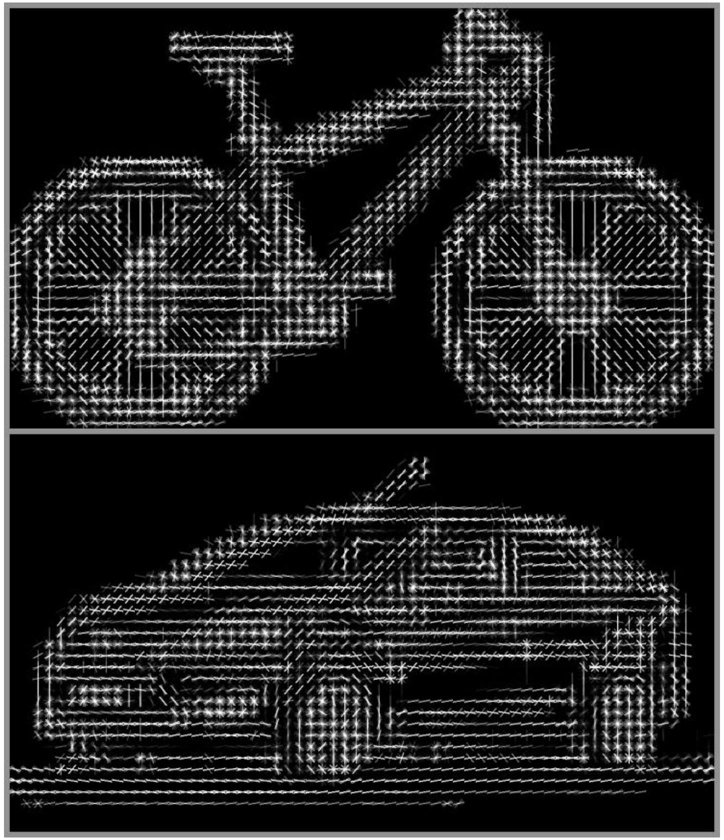
Extract histogram of oriented gradients (HOG) features from the image patch:



HOG features divide an image into small (8x8) *blocks*, and measure the *gradient orientations* in each of the blocks using a histogram (almost like SIFT)

# Dalal-Triggs detector

Different objects have different HOG features:

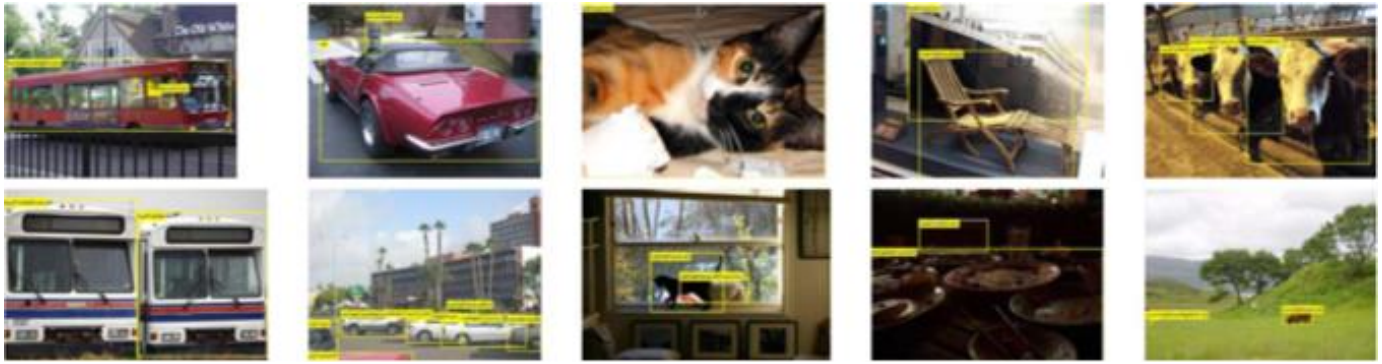


# Dalal-Triggs detector

Train a linear SVM on annotated images to predict object presence:

Training: $$\mathbf{w}^* = \underset{\mathbf{w}}{\operatorname{argmin}} \max\left(0, 1 - y\mathbf{w}^{\mathrm{T}}\phi(\mathbf{I};\mathbf{x})\right)$$

Detection: $$s(\mathbf{I};\mathbf{x}) = \mathbf{w}^{*\mathrm{T}}\phi(\mathbf{I};\mathbf{x})$$

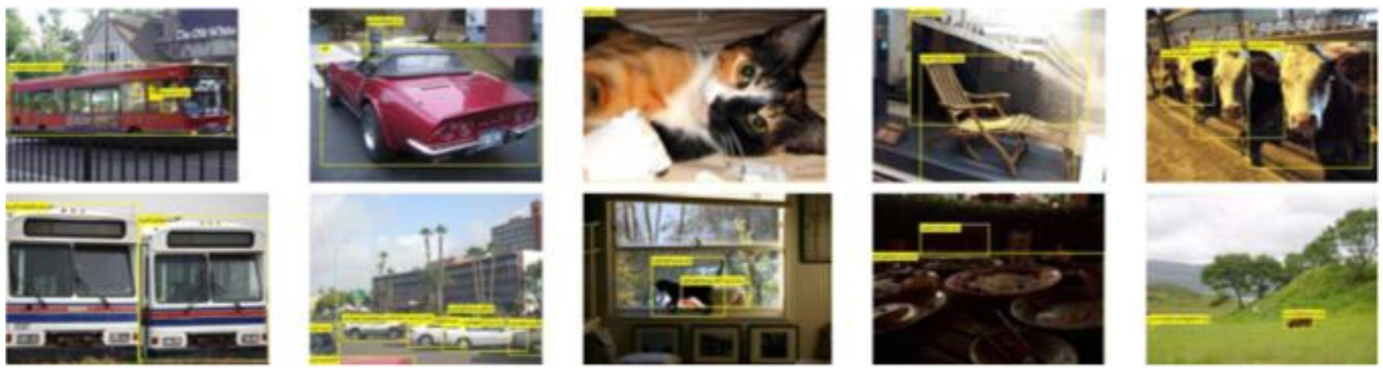


# Dalal-Triggs detector

Train a linear SVM on annotated images to predict object presence:

Training: $$\mathbf{w}^* = \operatorname*{argmin}_{\mathbf{w}} \max\left(0, 1 - y\mathbf{w}^{\mathrm{T}}\phi(\mathbf{I};\mathbf{x})\right)$$

Detection: $$s(\mathbf{I};\mathbf{x}) = \mathbf{w}^{*\mathrm{T}}\phi(\mathbf{I};\mathbf{x})$$



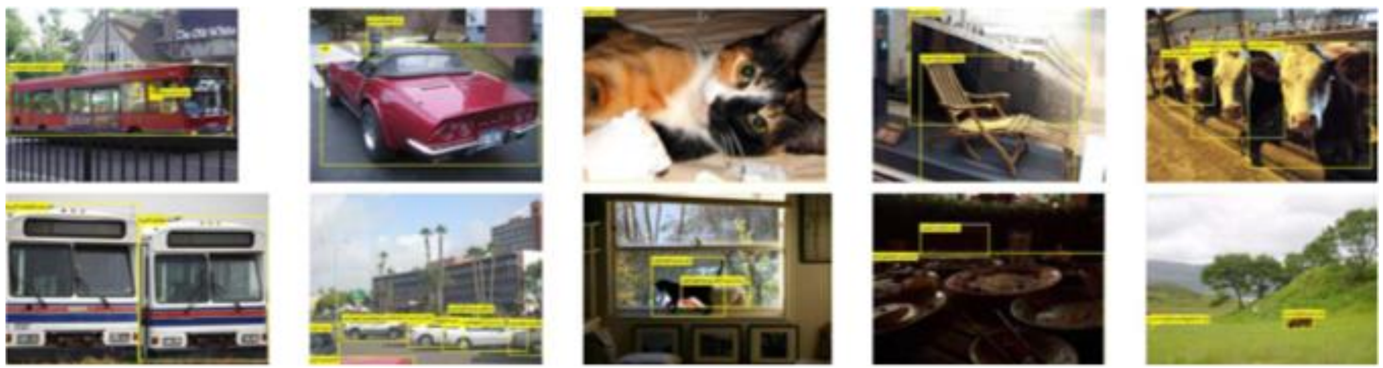
How do we get the *negative examples* to train the SVM?

# Dalal-Triggs detector

Train a linear SVM on annotated images to predict object presence:

Training: $\mathbf{w}^* = \underset{\mathbf{w}}{\operatorname{argmin}} \max\left(0, 1 - y\mathbf{w}^{\mathrm{T}}\phi(\mathbf{I};\mathbf{x})\right)$

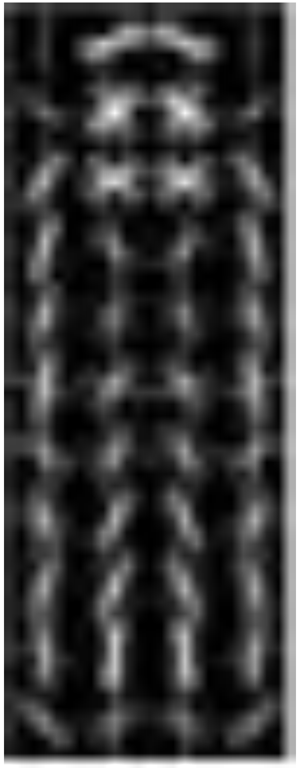
Detection: $s(\mathbf{I};\mathbf{x}) = \mathbf{w}^{*\mathrm{T}}\phi(\mathbf{I};\mathbf{x})$



How do we get the *negative examples* to train the SVM? Random patches!

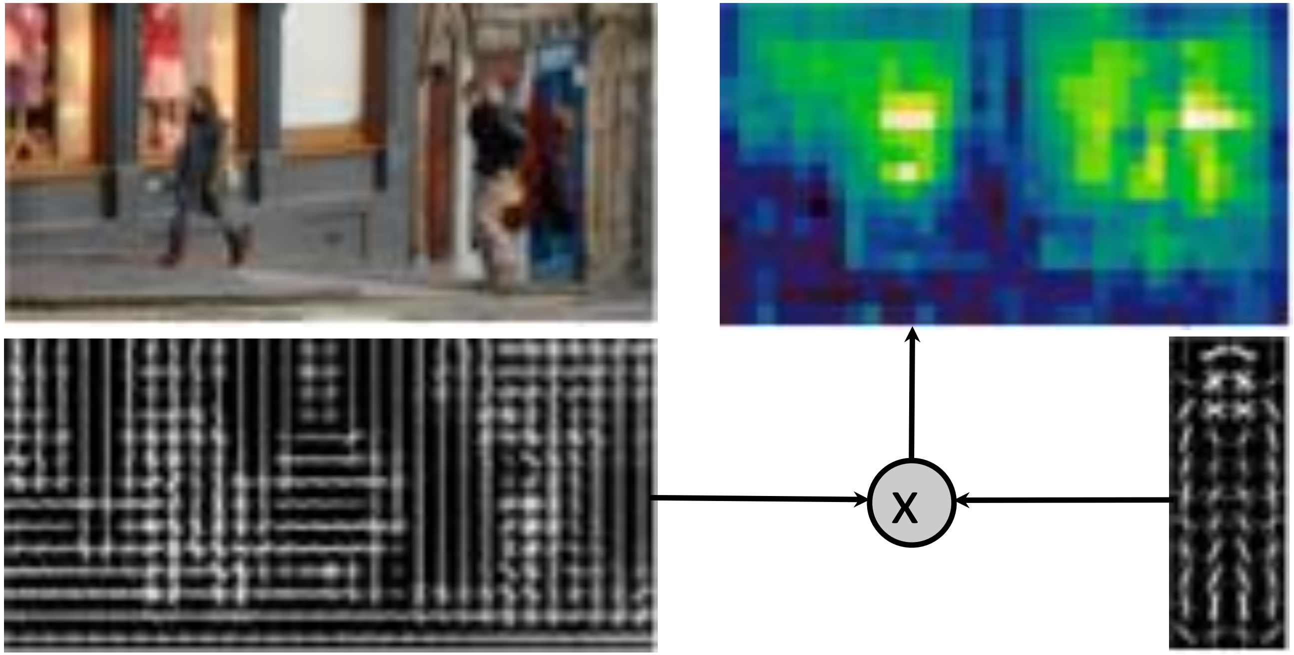
# Dalal-Triggs detector

HOG visualization of the SVM weights for a pedestrian detector:



# Dalal-Triggs detector

Applying the detector at each location leads to a *confidence map*:




*Non-maxima suppression* can be used to obtain the final detections

# Dalal-Triggs detector

Example of pedestrian detections using Dalal-Triggs detector:

# Pictorial structures

What can we do when a part of the object to be detected is occluded?

# Pictorial structures

What can we do when a part of the object to be detected is occluded?

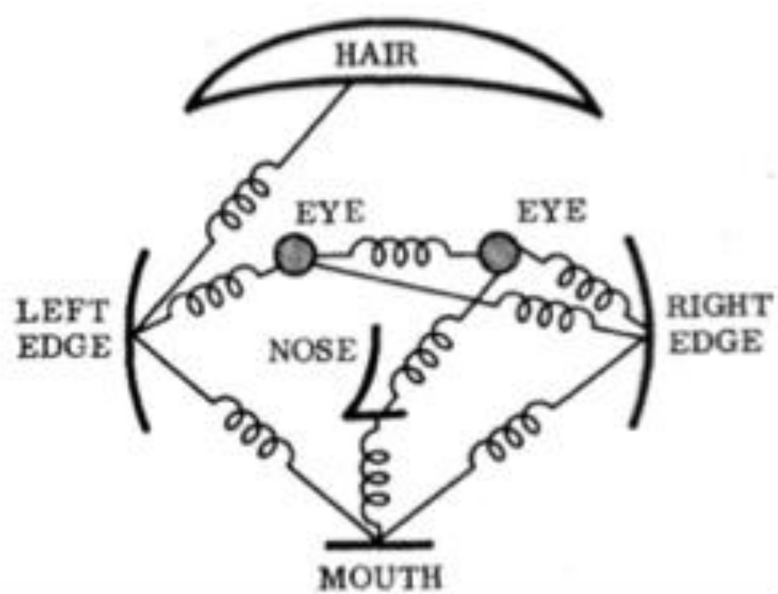
Exploit the fact that other parts of the object are still visible!

# Pictorial structures

What can we do when a part of the object to be detected is occluded?

Exploit the fact that other parts of the object are still visible!

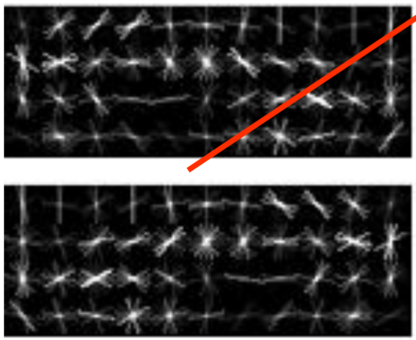
*Pictorial structures* does this by modeling objects as a constellation of parts:



# Deformable template models

Defines a *score function* that involves *parts* and *part deformations*:

$$s(\mathbf{I}; x_0, y_0, \ldots, x_{|V|}, y_{|V|}) = \mathbf{w}_0^{\mathrm{T}} \phi(\mathbf{I}; x_0, y_0)$$

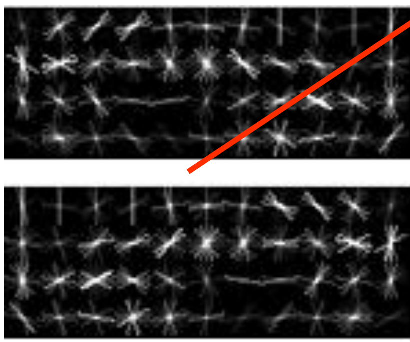


Global object model

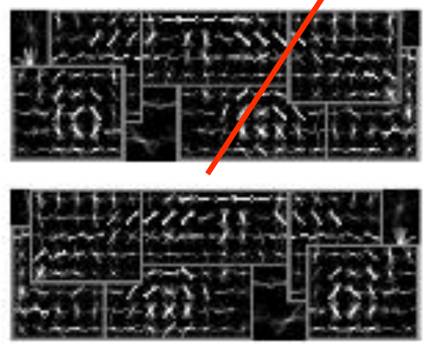
# Deformable template models

Defines a *score function* that involves *parts* and *part deformations*:

$$s(\mathbf{I}; x_0, y_0, \ldots, x_{|V|}, y_{|V|}) = \mathbf{w}_0^{\mathrm{T}} \phi(\mathbf{I}; x_0, y_0) + \sum_{i \in V} \mathbf{w}_i^{\mathrm{T}} \phi(\mathbf{I}; x_i, y_i)$$



Global object model

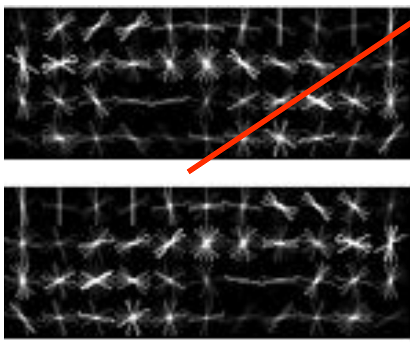


Object part models

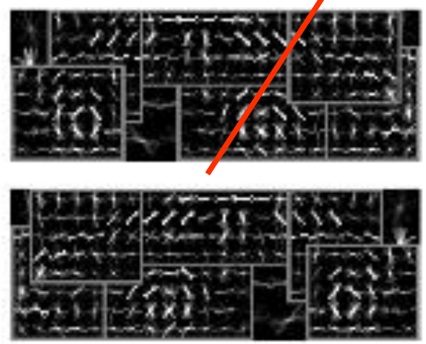
# Deformable template models

Defines a *score function* that involves *parts* and *part deformations*:

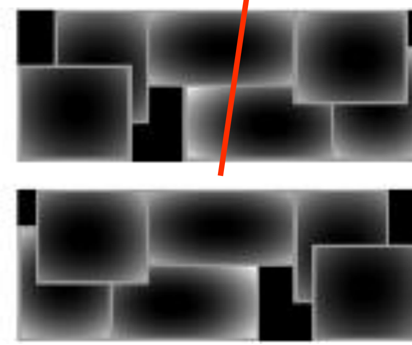
$$s(\mathbf{I}; x_0, y_0, \ldots, x_{|V|}, y_{|V|}) = \mathbf{w}_0^{\mathrm{T}} \phi(\mathbf{I}; x_0, y_0) + \sum_{i \in V} \mathbf{w}_i^{\mathrm{T}} \phi(\mathbf{I}; x_i, y_i) + \sum_{(i,j) \in E} d_{ij} \phi_d(x_i - x_j, y_i - y_j)$$



Global object model



Object part models

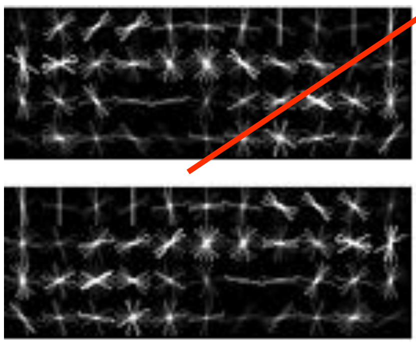


Deformation model

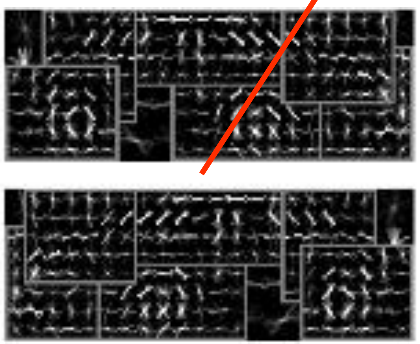
# Deformable template models

Defines a *score function* that involves *parts* and *part deformations*:

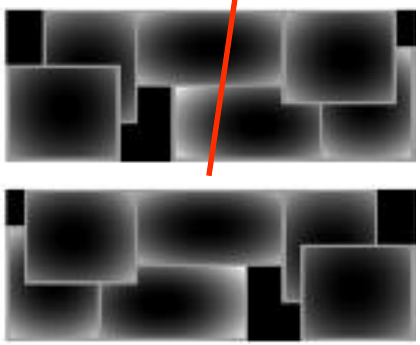
$$s(\mathbf{I}; x_0, y_0, \ldots, x_{|V|}, y_{|V|}) = \mathbf{w}_0^{\mathrm{T}} \phi(\mathbf{I}; x_0, y_0) + \sum_{i \in V} \mathbf{w}_i^{\mathrm{T}} \phi(\mathbf{I}; x_i, y_i) + \sum_{(i,j) \in E} d_{ij} \phi_d(x_i - x_j, y_i - y_j)$$



Global object model



Object part models

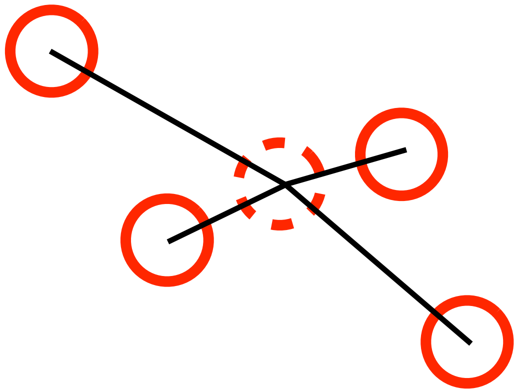


Deformation model

Deformable template models are much more robust against *partial occlusions* and *deformations* of non-rigid objects

# Graph structure

The graph structure for this model is a star-shaped tree:



# Pictorial structures

Find configuration of pict. structures model by maximizing over part locations:

$$\max_{x_0, y_0, \ldots, x_{|V|}, y_{|V|}} s(\mathbf{I}; x_0, y_0, \ldots, x_{|V|}, y_{|V|})$$

# Pictorial structures

Find configuration of pict. structures model by maximizing over part locations:

$$\max_{x_0, y_0, \ldots, x_{|V|}, y_{|V|}} s(\mathbf{I}; x_0, y_0, \ldots, x_{|V|}, y_{|V|})$$

For *squared-error* deformation models, this can be done very efficiently:

$$g(x_i) = \min_{x_j}(f(x_j) + (x_i - x_j)^2)$$

# Pictorial structures

Find configuration of pict. structures model by maximizing over part locations:

$$\max_{x_0, y_0, \ldots, x_{|V|}, y_{|V|}} s(\mathbf{I}; x_0, y_0, \ldots, x_{|V|}, y_{|V|})$$

For *squared-error* deformation models, this can be done very efficiently:

$$g(x_i) = \min_{x_j}(f(x_j) + (x_i - x_j)^2)$$

final score
with deformations

Hence, we have a parabola for every pixel $x_j$ rooted at $(x_j, f(x_j))$

# Pictorial structures

Find configuration of pict. structures model by maximizing over part locations:

$$\max_{x_0, y_0, \ldots, x_{|V|}, y_{|V|}} s(\mathbf{I}; x_0, y_0, \ldots, x_{|V|}, y_{|V|})$$

For *squared-error* deformation models, this can be done very efficiently:

$$g(x_i) = \min_{x_j}(f(x_j) + (x_i - x_j)^2)$$

final score
with deformations

negative part
model score

Hence, we have a parabola for every pixel $x_j$ rooted at $(x_j, f(x_j))$

# Pictorial structures

Find configuration of pict. structures model by maximizing over part locations:

$$\max_{x_0, y_0, \ldots, x_{|V|}, y_{|V|}} s(\mathbf{I}; x_0, y_0, \ldots, x_{|V|}, y_{|V|})$$

For *squared-error* deformation models, this can be done very efficiently:

$$g(x_i) = \min_{x_j}(f(x_j) + (x_i - x_j)^2)$$

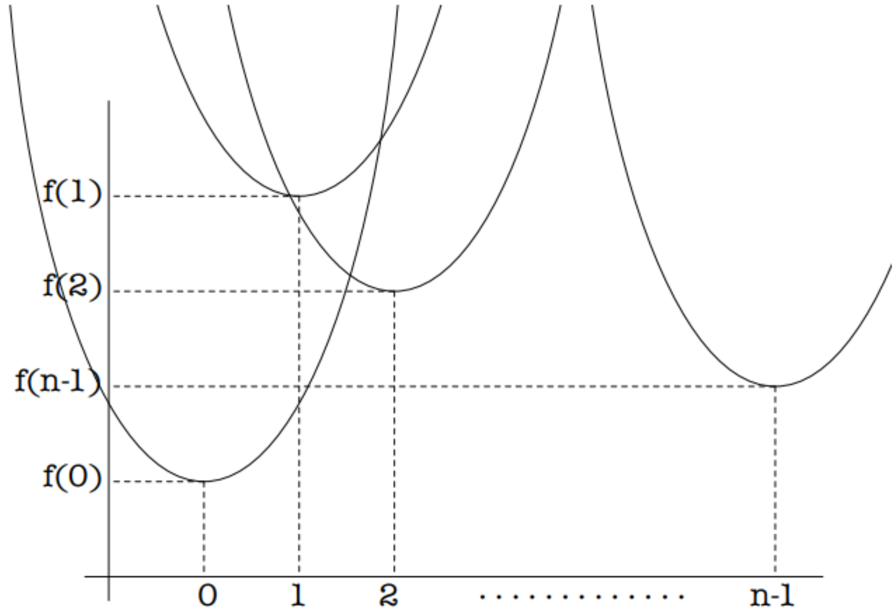
final score
with deformations

negative part
model score

deformation penalty

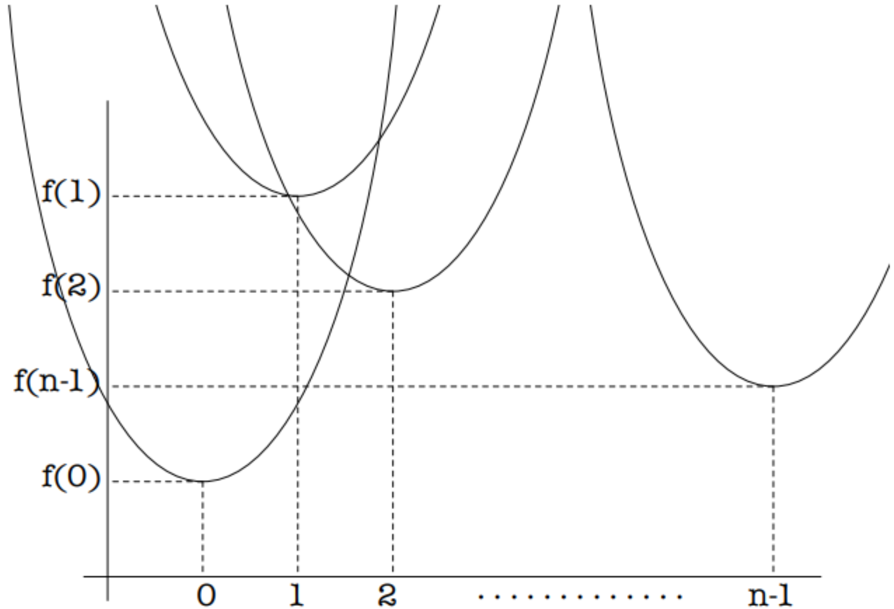
Hence, we have a parabola for every pixel $x_j$ rooted at $(x_j, f(x_j))$

# Pictorial structures




* Felzenszwalb & Huttenlocher, 2004

# Pictorial structures



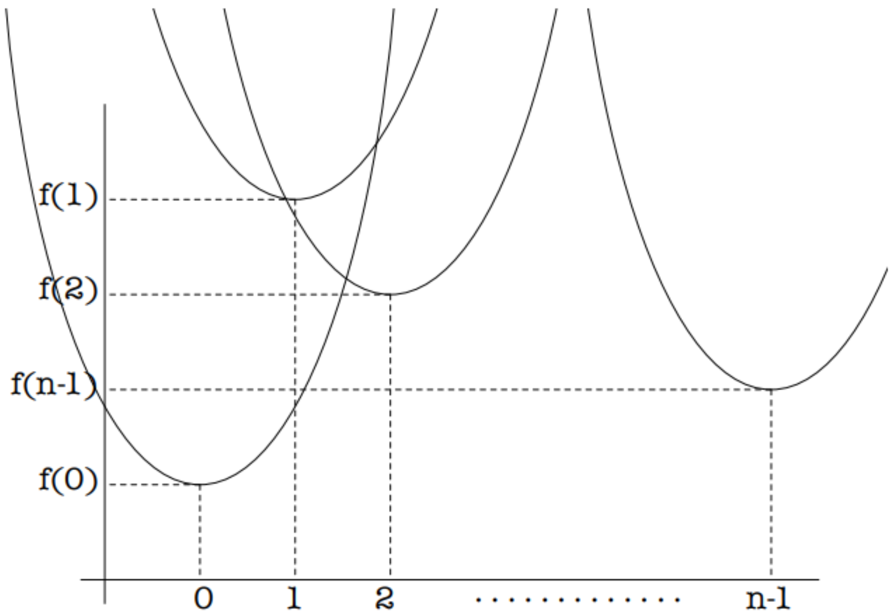

It is straightforward to compute the *intersection* between two parabolas:

$$i = \frac{(f(x_i) + x_i^2) - (f(x_j) + x_j^2)}{2x_i - 2x_j}$$

* Felzenszwalb & Huttenlocher, 2004

# Pictorial structures

If $x_j < x_i$: parabola corresponding to $x_j$ is *below* that of $x_i$ *left* of the intersection, and above it *right* of the intersection

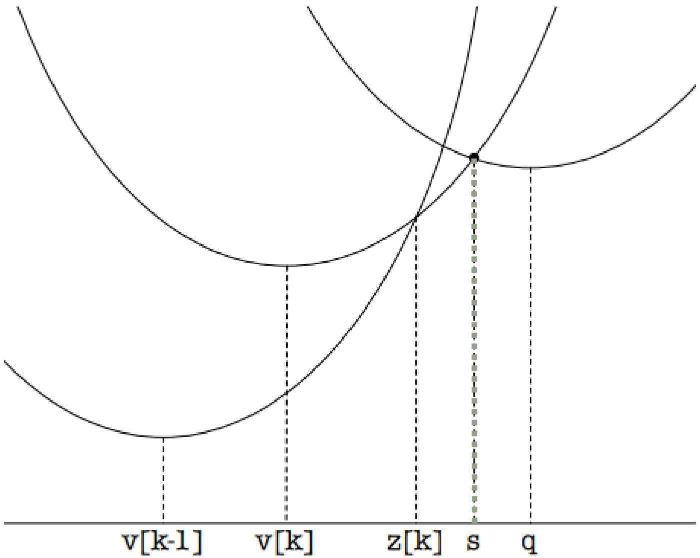



* Felzenszwalb & Huttenlocher, 2004

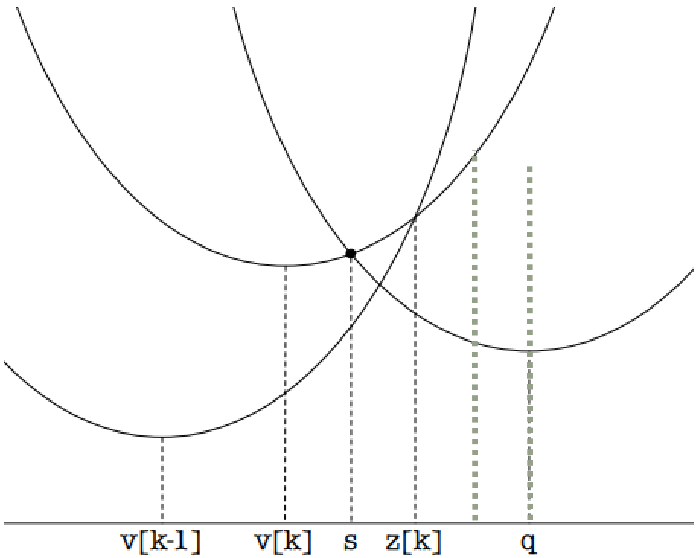
# Pictorial structures

Maintain the *lower envelope* of the parabolas (parabolas and intersections)

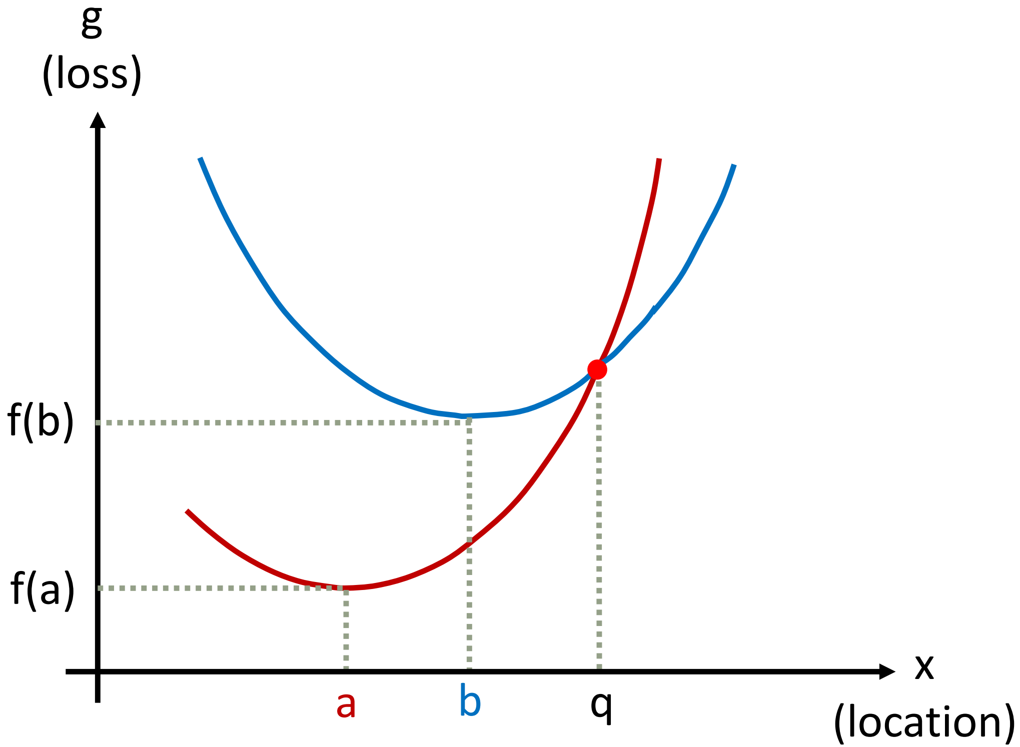
When adding a new parabola, there are two possibilities:

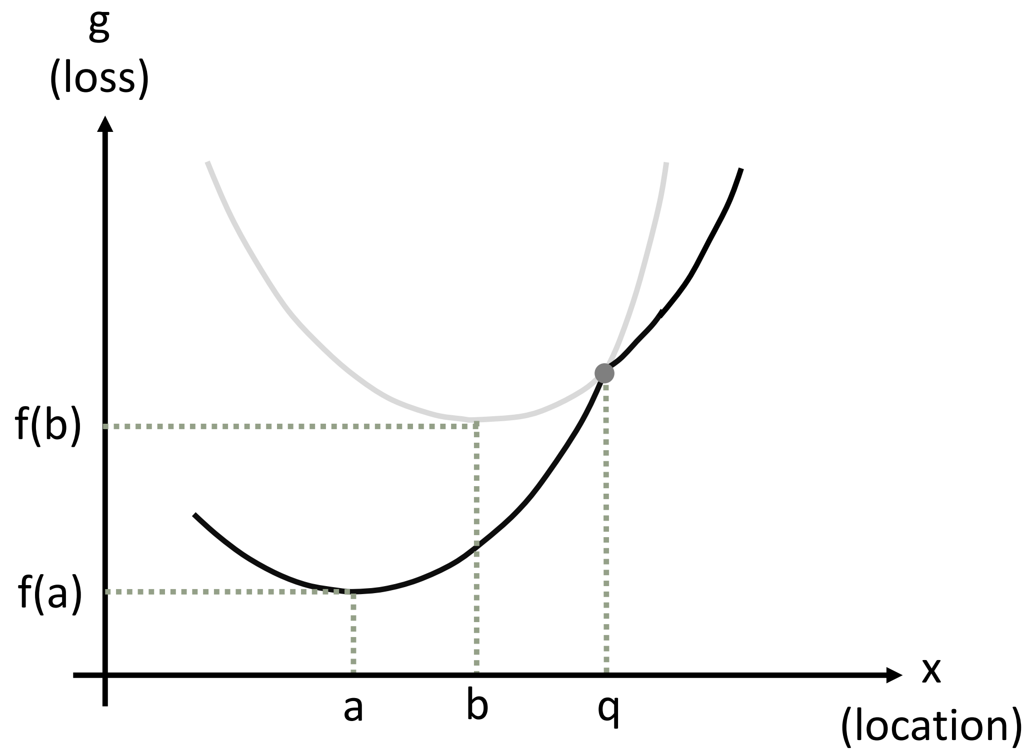


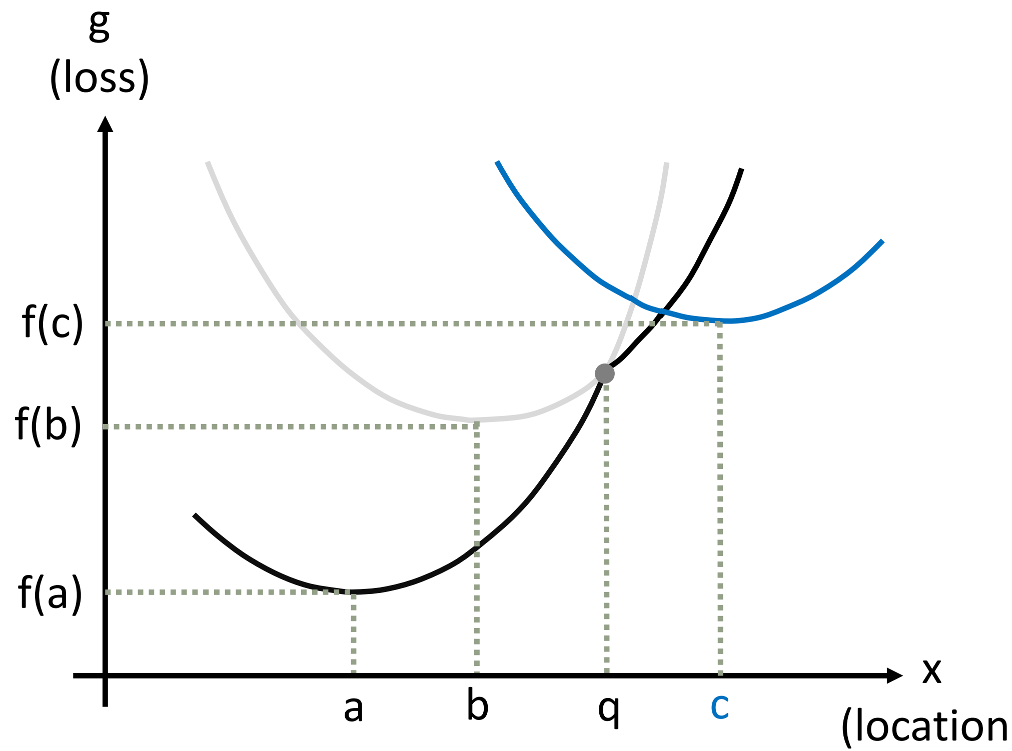

new intersection right of previous intersection: maintain previous parabola in the envelope

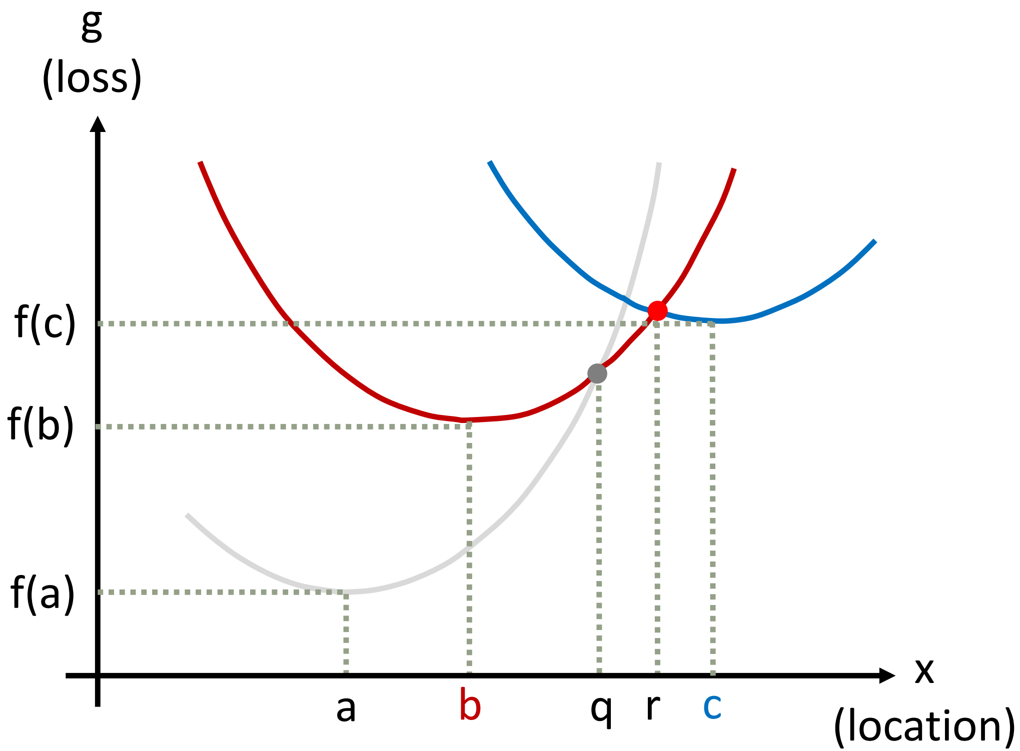



new intersection left of previous intersection: remove previous parabola from the envelope


g
(loss)
f(b)
f(a)
x
(location)
a
b
q

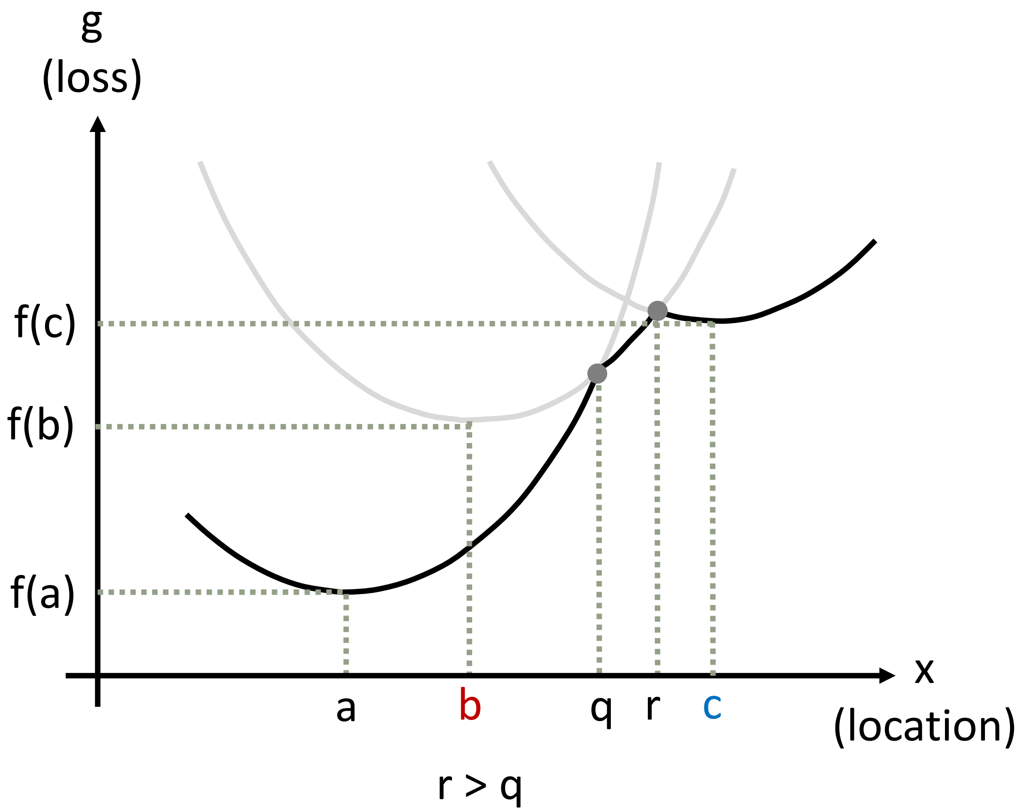

g
(loss)
f(b)
f(a)
x
(location)
a
b
q


g
(loss)
f(c)
f(b)
f(a)
x
(location)
a
b
q
c



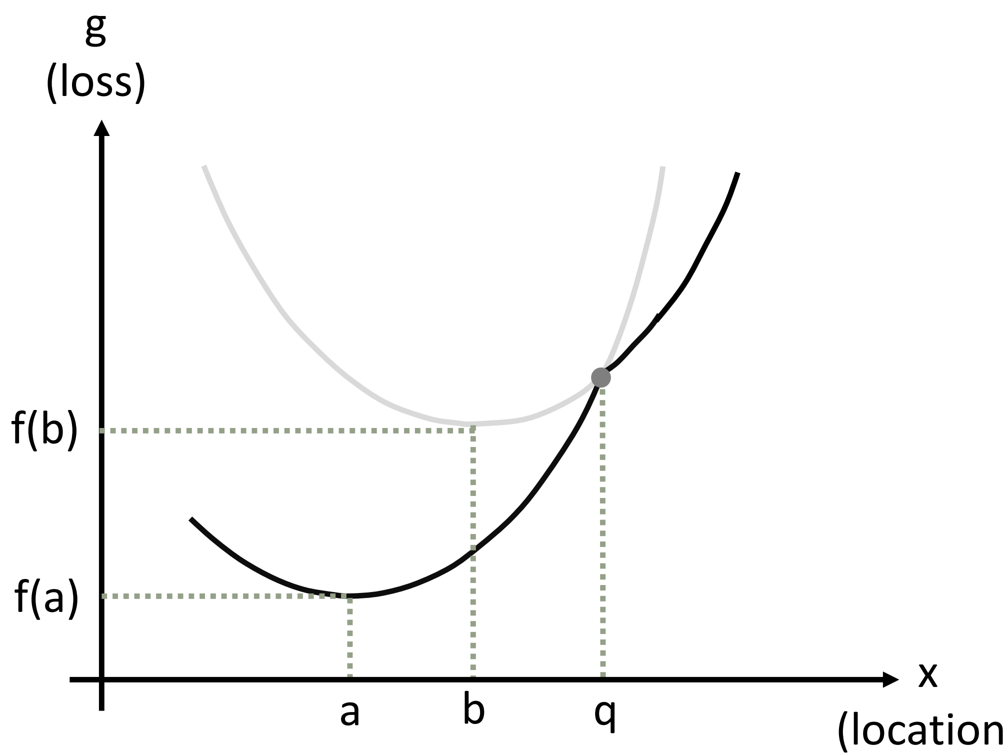

r > q

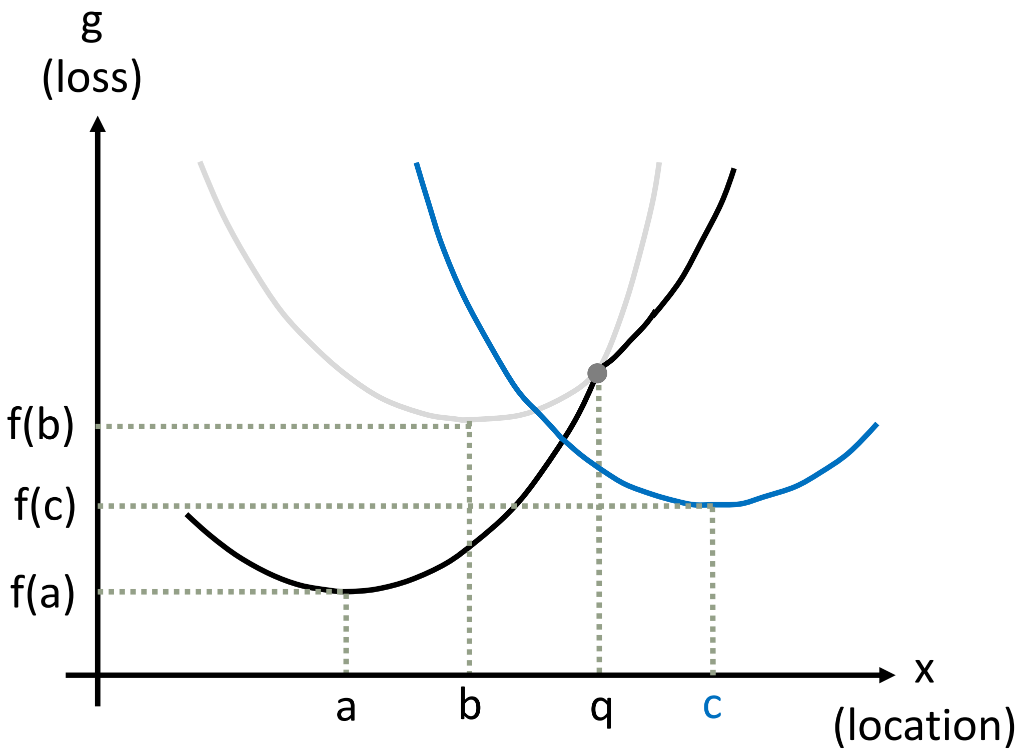
new intersection is at the right of previous intersection:
maintain previous parabola in the envelope



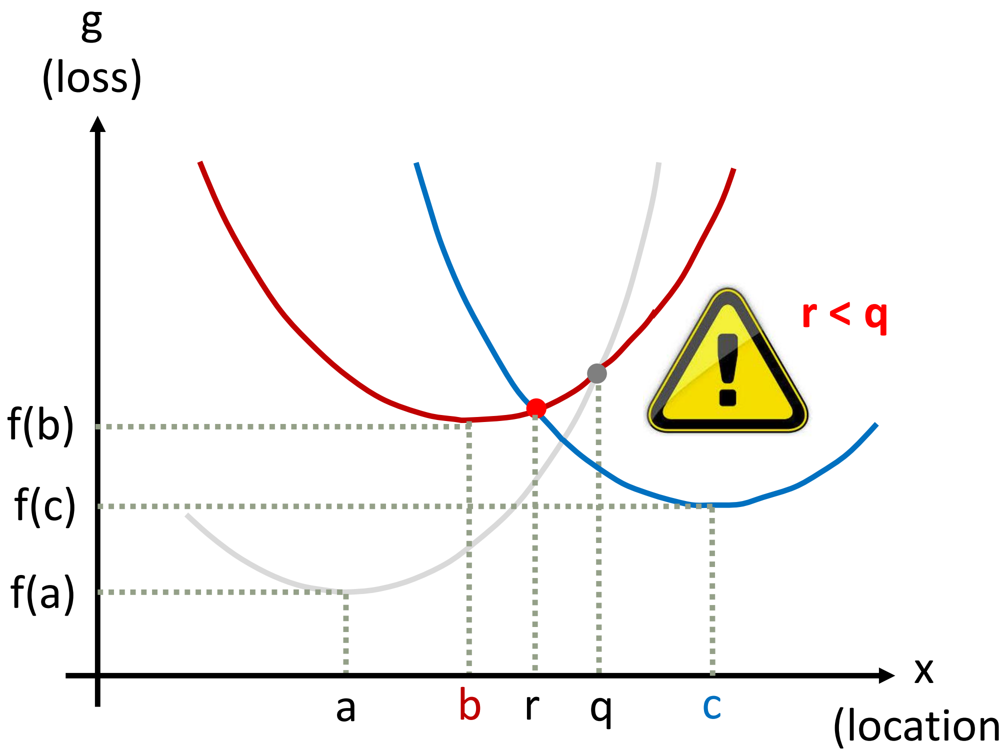

new intersection is at the right of previous intersection:
maintain previous parabola in the envelope

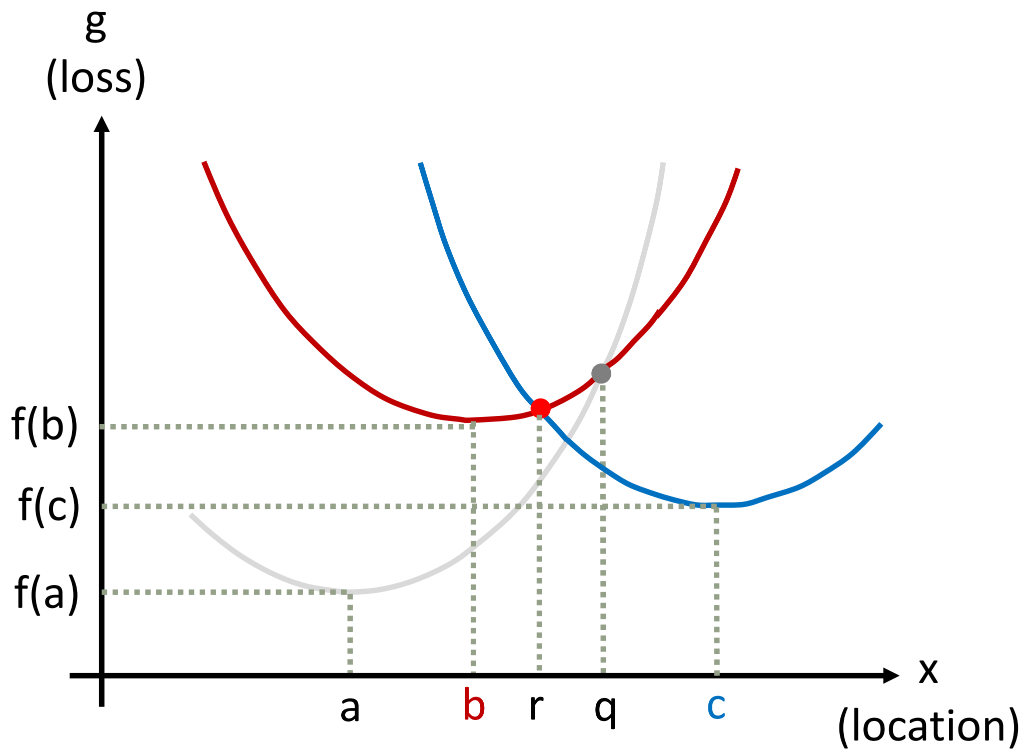
# Let’s rewind


g
(loss)
f(b)
f(a)
x
(location)
a
b
q



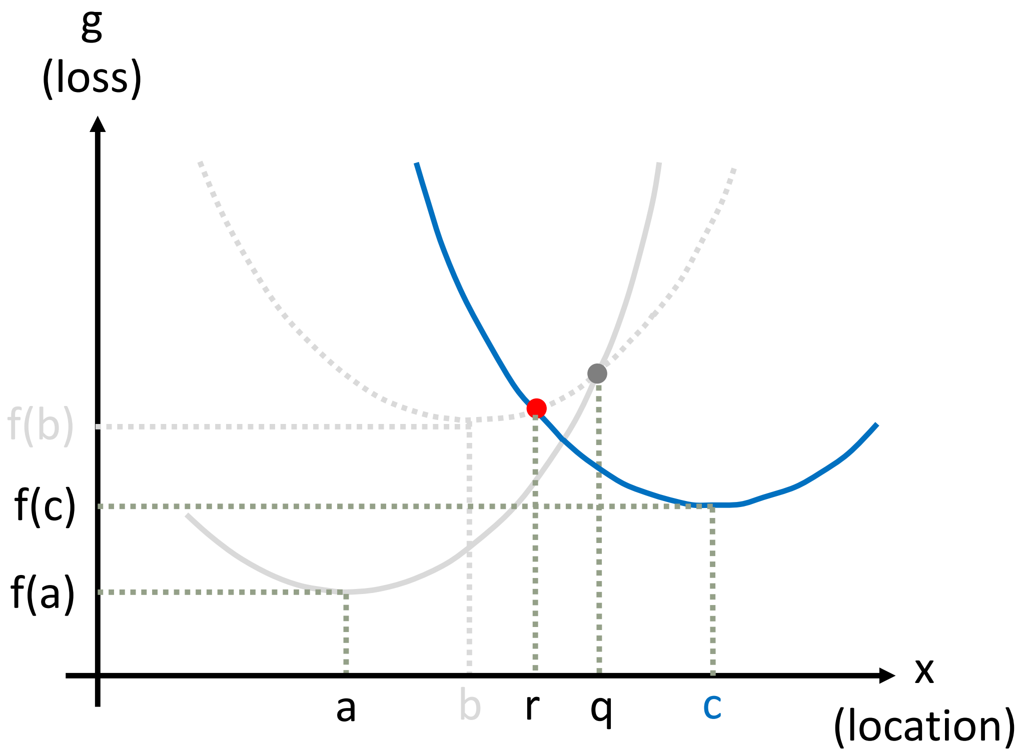
g
(loss)
f(b)
f(c)
f(a)
x
(location)
a
b
q
c


g
(loss)
f(b)
f(c)
f(a)
r < q
x
(location)
a
b
r
q
c



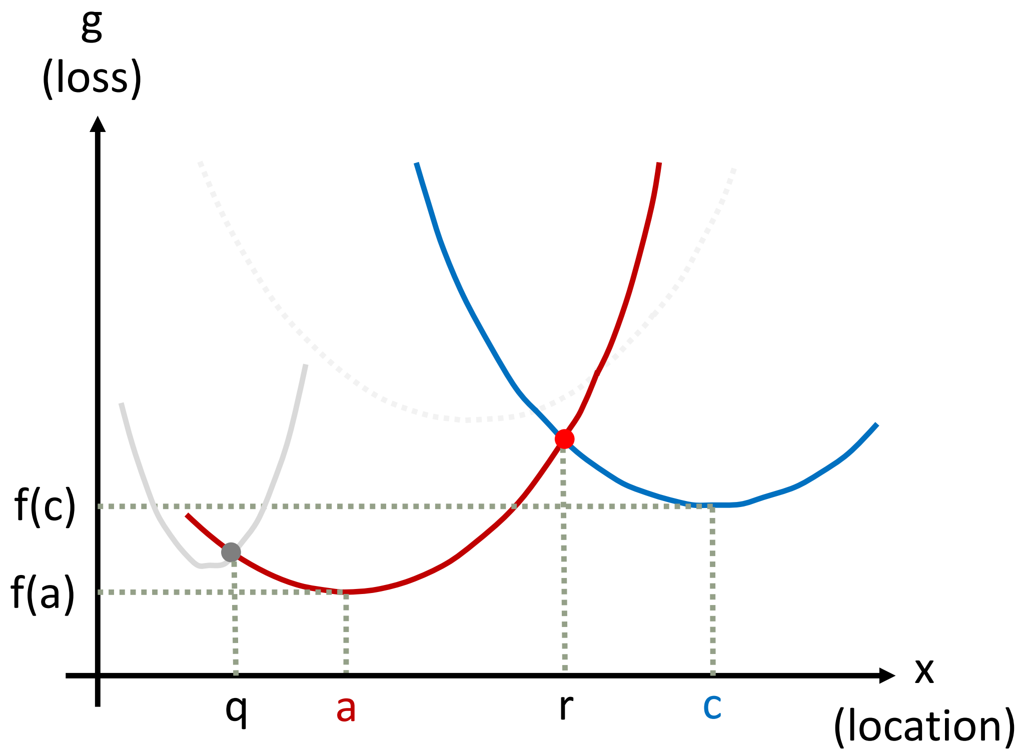

r < q

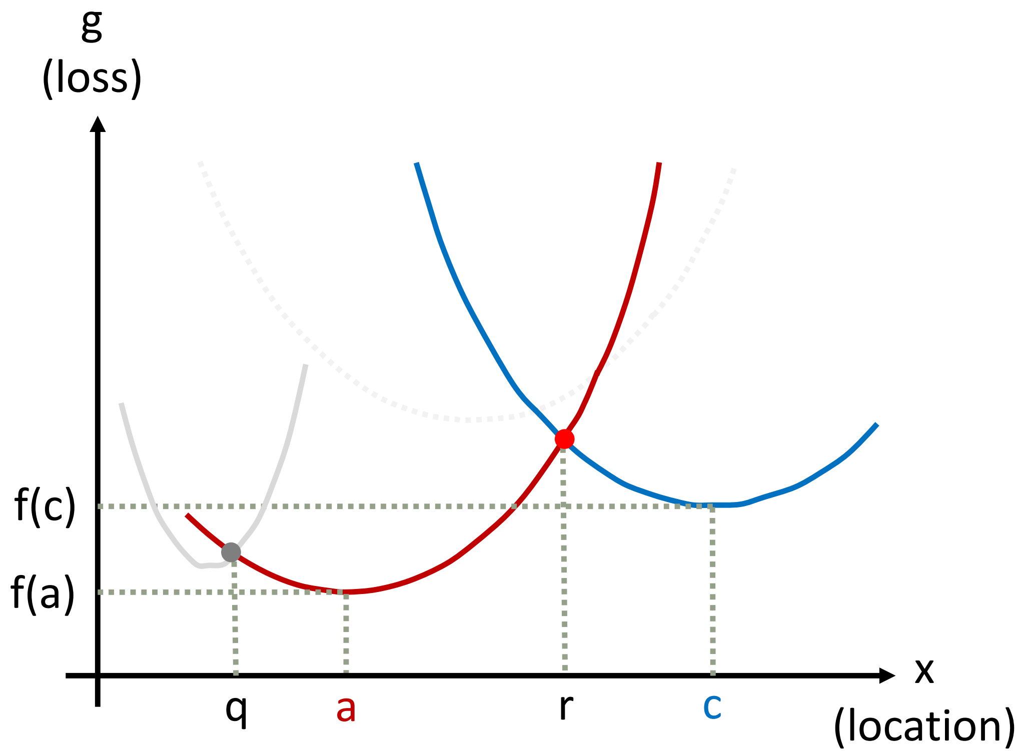
new intersection is at the left of previous intersection:
remove previous parabola from the envelope




r < q

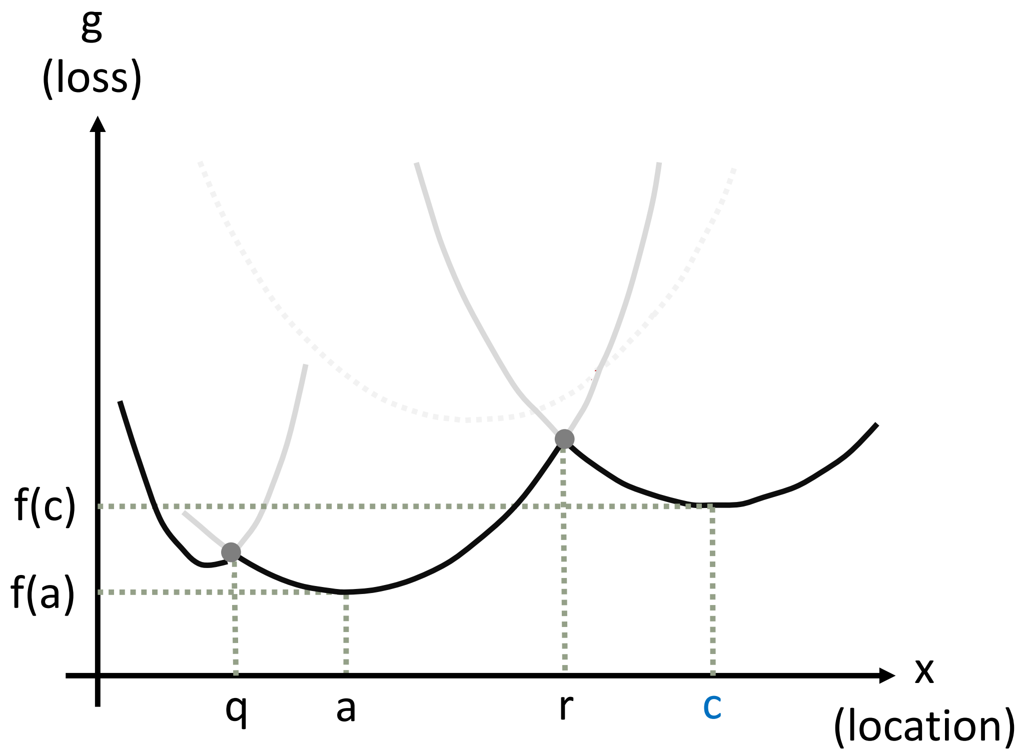
new intersection is at the left of previous intersection:
remove previous parabola from the envelope


g
(loss)
f(c)
f(a)
x
(location)
q
a
r
c




r > q

new intersection is at the right of previous intersection:
maintain previous parabola in the envelope




r > q

new intersection is at the right of previous intersection:
maintain previous parabola in the envelope

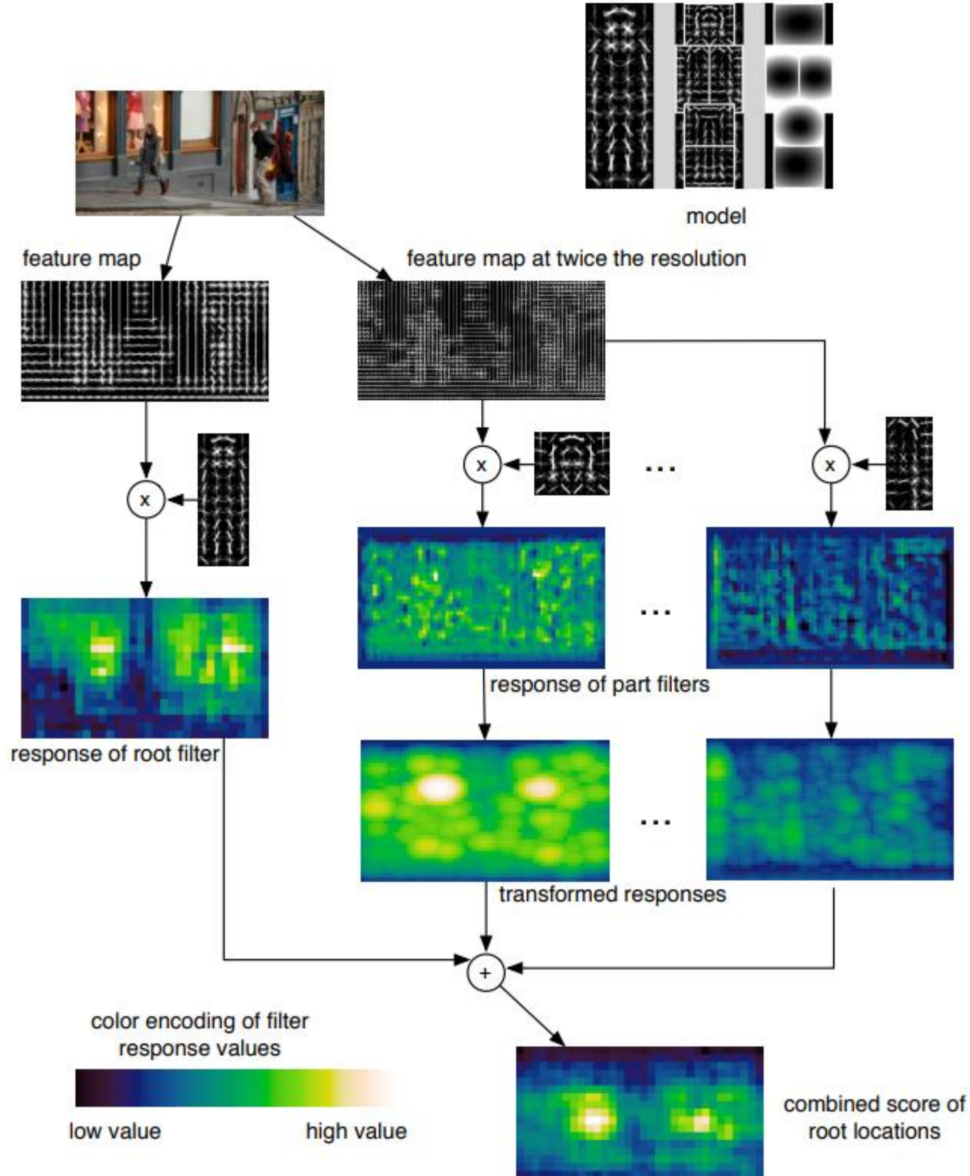
# Pictorial structures

This suggests a simple algorithm that is *linear* in the number of pixels:

- Maintain list with the *lower envelope* of the parabolas (indices and intersections)
- Move from *left to right* through all parabolas; and do for each parabola:
  - Find intersection of parabola with the previous parabola in lower envelope
  - If intersection is left of previous intersection in lower envelope: remove previous parabola from lower envelope, and go back one step
  - Add parabola to lower envelope, starting from intersection

* Felzenszwalb & Huttenlocher, 2004

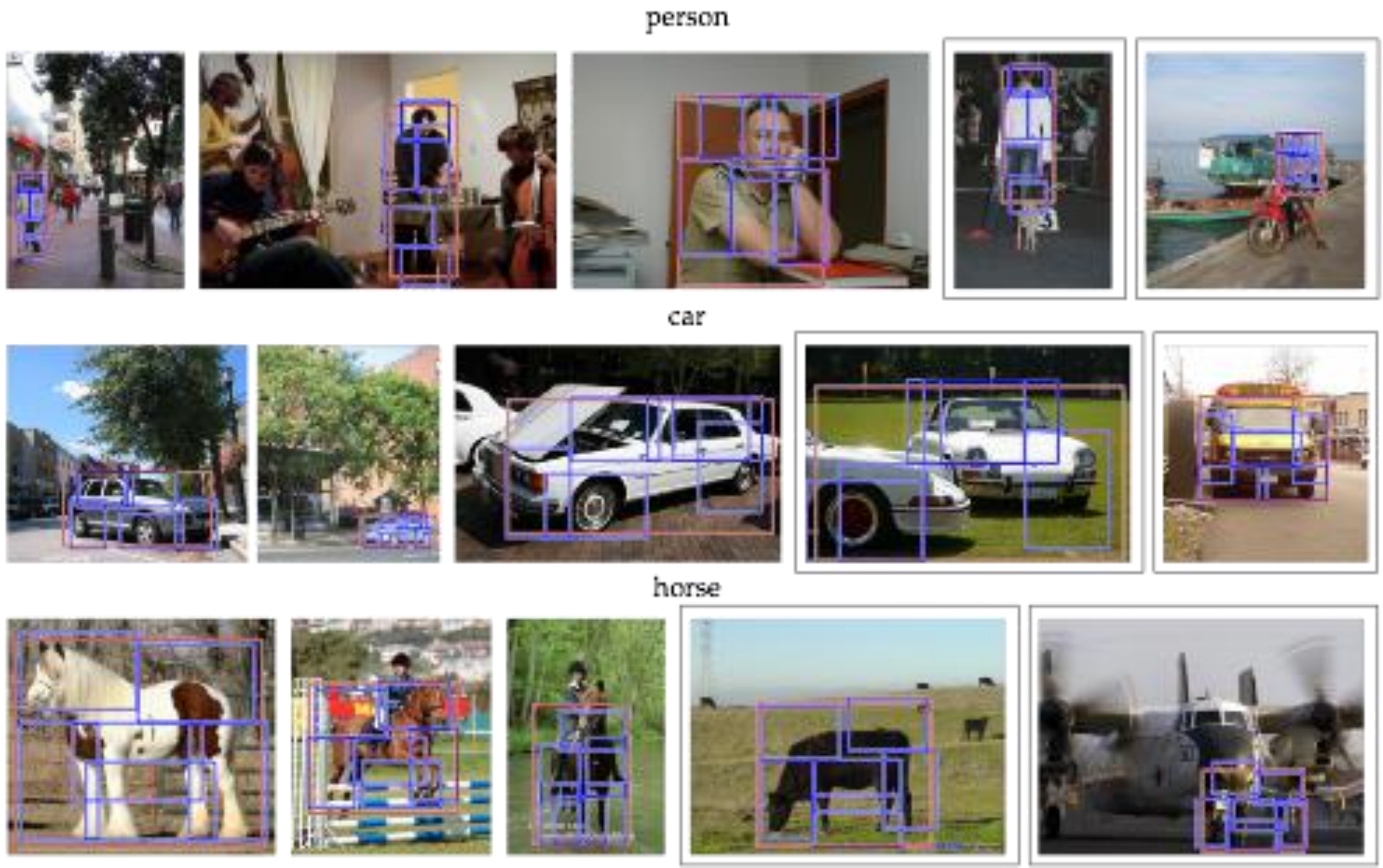
* Felzenszwalb & Huttenlocher, 2004



# Deformable template models

Examples of object detections by deformable template models:

# Example detections



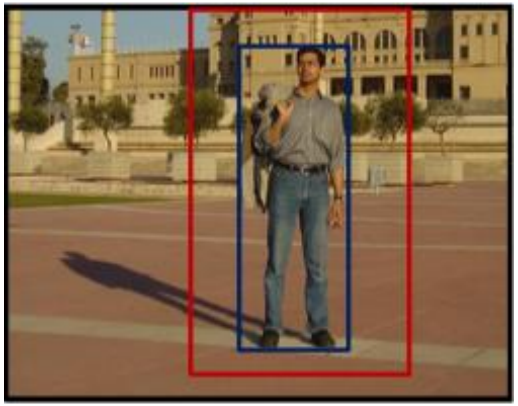
# The miserable life of a person detector...

# The miserable life of a person detector...

# Incorporating context

Incorporating context can help to partially solve recognition problems:

- Local pixel context (larger bounding box)
- Semantic context (scene category, other objects present)
- Geographic context (GPS location, landmarks)
- Temporal context (objects do not change rapidly)
- Etcetera...



**Reading material:**

- Section 14 of Szeliski
- S. Belongie, J. Malik, and J. Puzicha. "Shape context: A new descriptor for shape matching and object recognition." In *Advances in Neural Information Processing Systems*, pp. 831-837. 2001.

**Optional:**

- P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan. "Object detection with discriminatively trained part-based models." *IEEE Transactions on Pattern Analysis and Machine Intelligence* 32, no. 9 (2009): 1627-1645.