

Fall 2022

INTRODUCTION TO COMPUTER VISION

Atlas Wang

Assistant Professor, The University of Texas at Austin

Visual Informatics Group@UT Austin
<https://vita-group.github.io/>

Many slides here were adapted from Brown CSCI 1430

Recognition so far

Category:

- Is this a bedroom?
- What class of scene is this?
- Holistic features/quantization

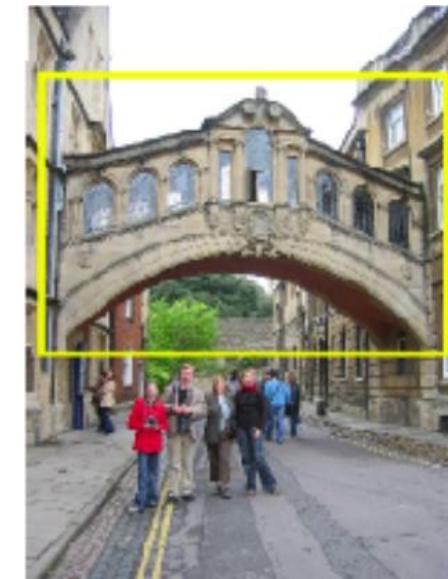


Instance:

- Find this specific famous building.
- Find this person.
- Local features/precise correspondence
- Often within a database of images



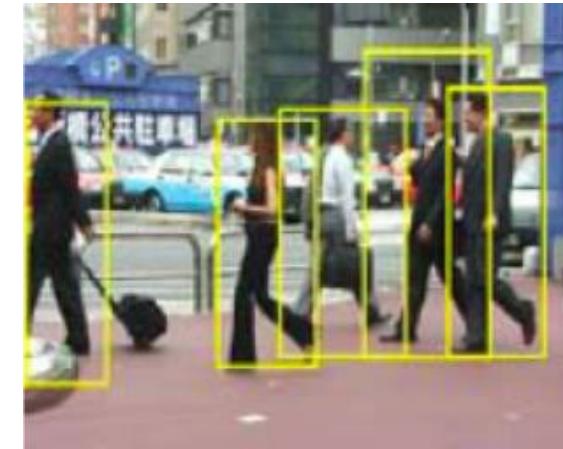
“Image classification is not real computer vision... so don’t be too obsessed with that”



Recognition so far

Object (category) detection:

- Find all the people
- Find all the faces
- Often within a single image
- Often ‘sliding window’



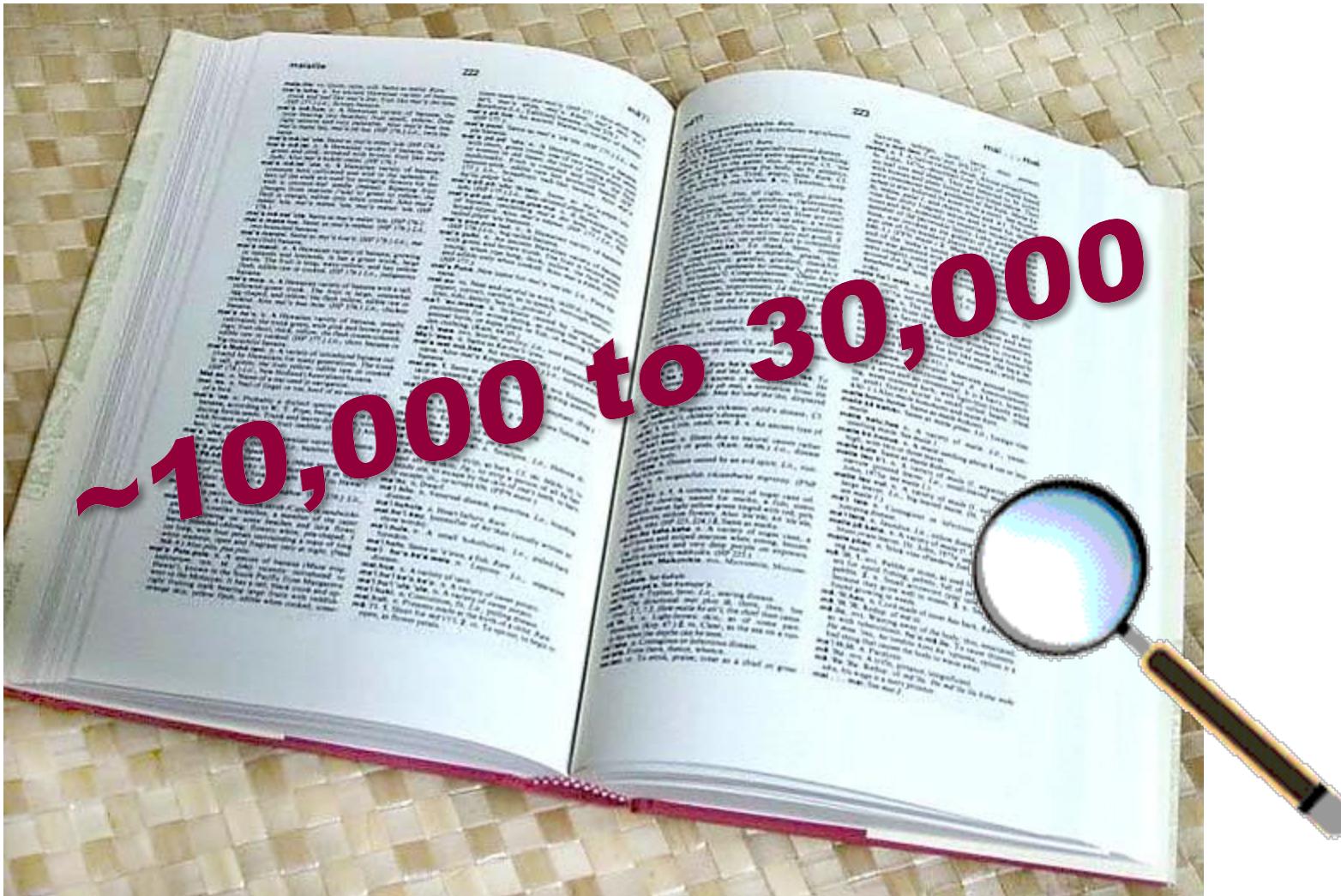
Scenes have “stuff” – distribution of materials and surfaces with arbitrary shape.

- Bag of Words ok!

Objects are “things” with shape, boundaries.

- **Bag of Words less ok as spatial layout is lost!**

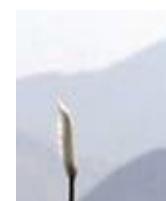
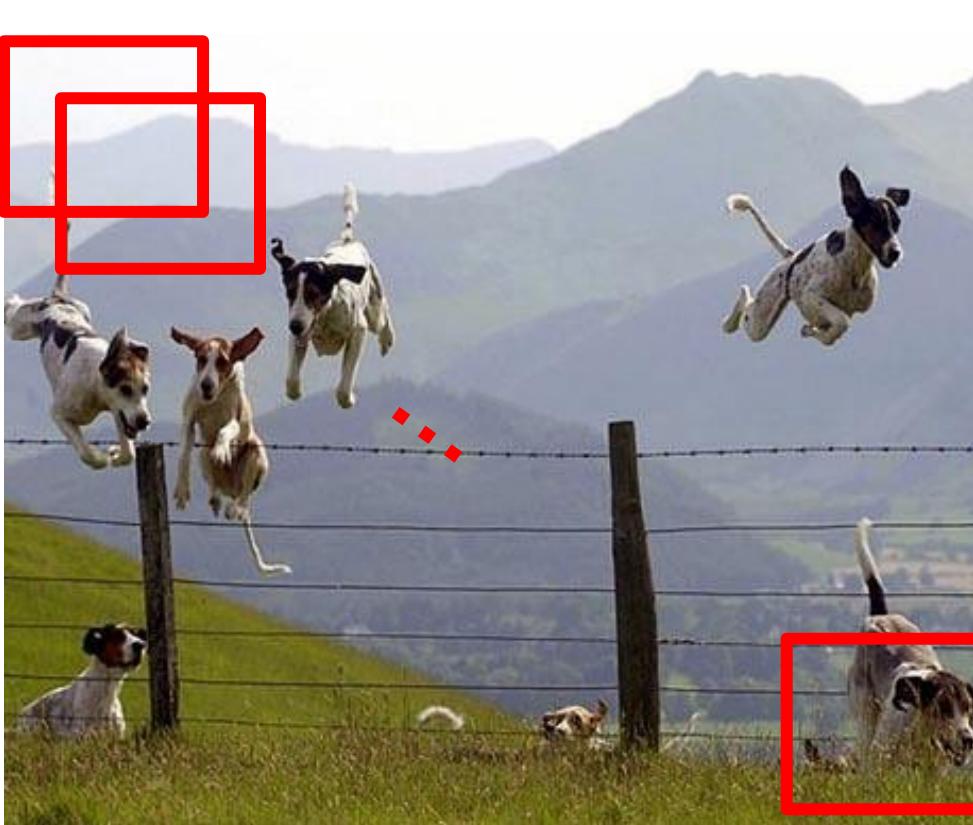
How many object categories are there?



Biederman 1987

Object Category Detection

- Focus on object search: “Where is it?”
- Build templates that quickly differentiate object patch from background patch



**Object or
Non-Object?**

Challenges in modeling the object class



Illumination



Object pose



'Clutter'



Occlusions



Intra-class
appearance

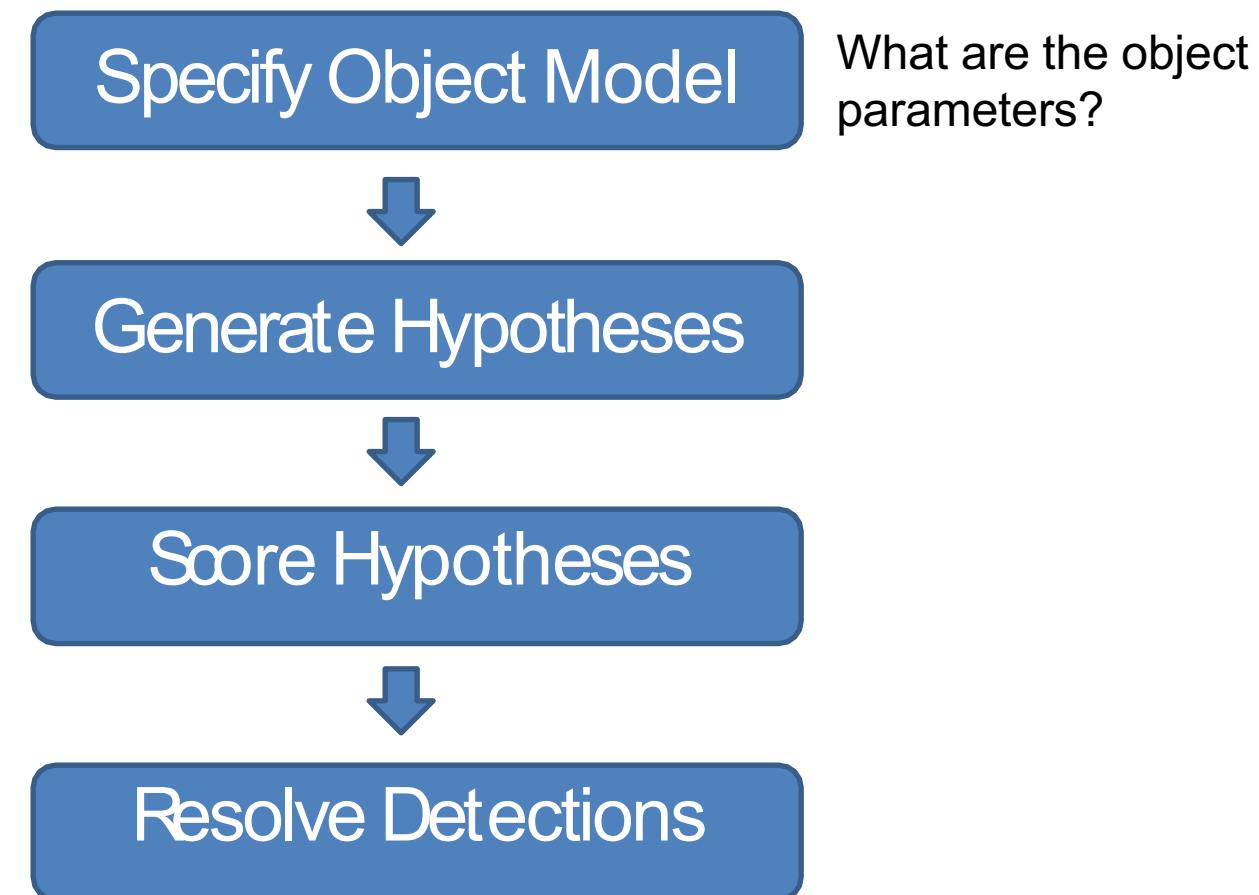


Viewpoint

Object Detection Design challenges

- How to efficiently search for likely objects
 - Even simple models require searching hundreds of thousands of positions and scales.
- Feature design and scoring
 - How should appearance be modeled?
 - What features correspond to the object?
- How to deal with different viewpoints?
 - Often train different models for a few different viewpoints

General Process of Object Detection



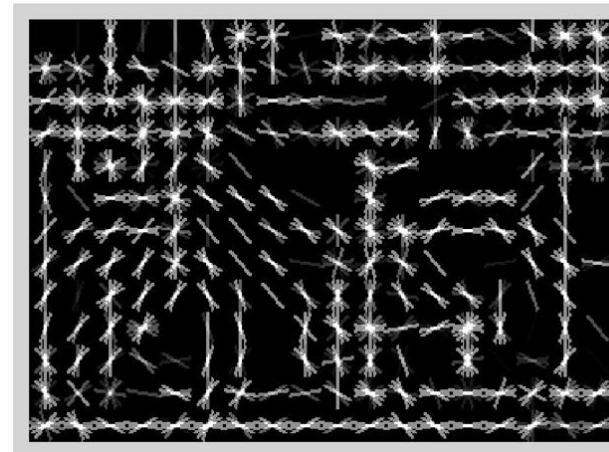
Specifying an object model

1. Statistical Template in Bounding Box

- Object is some (x,y,w,h) in image
- Features defined wrt bounding box coordinates



Image

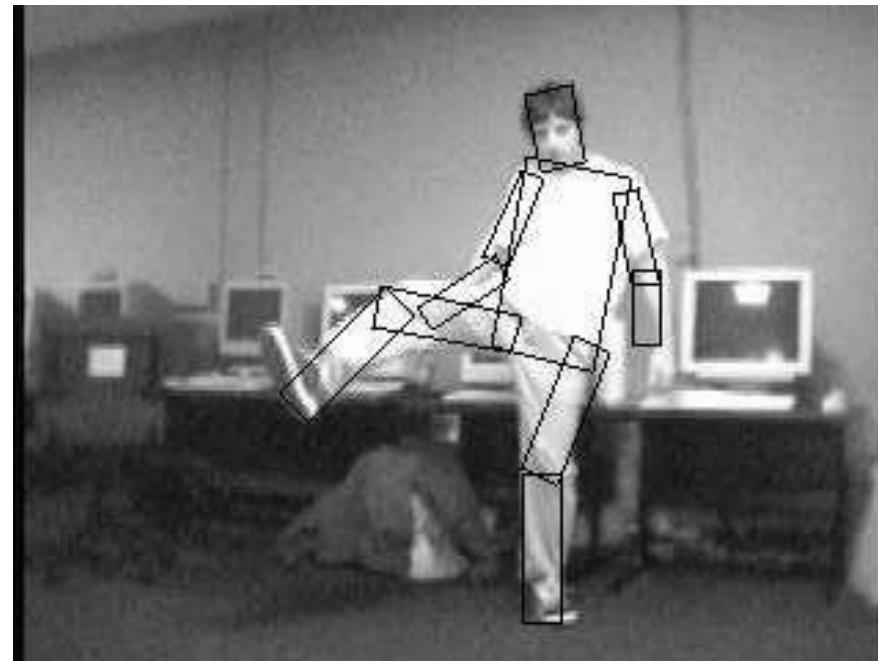
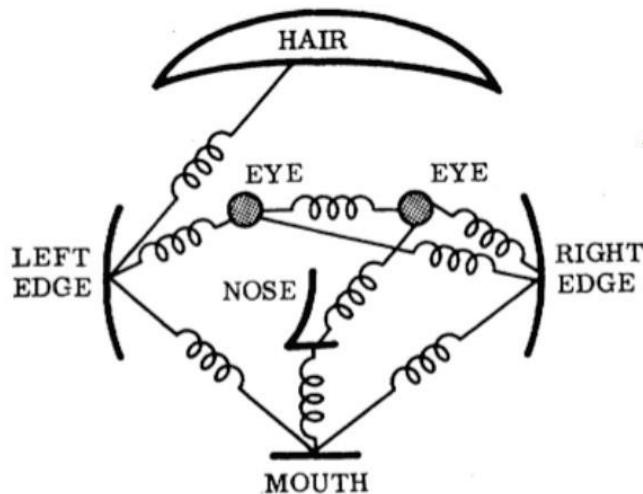


Template Visualization

Specifying an object model

2. Articulated parts model

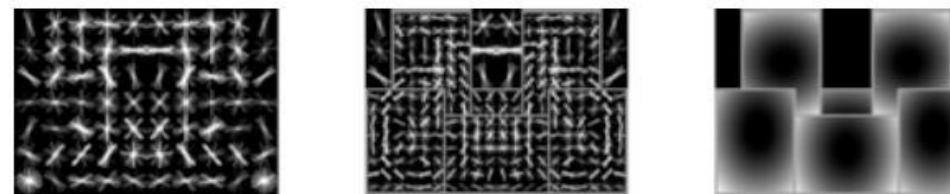
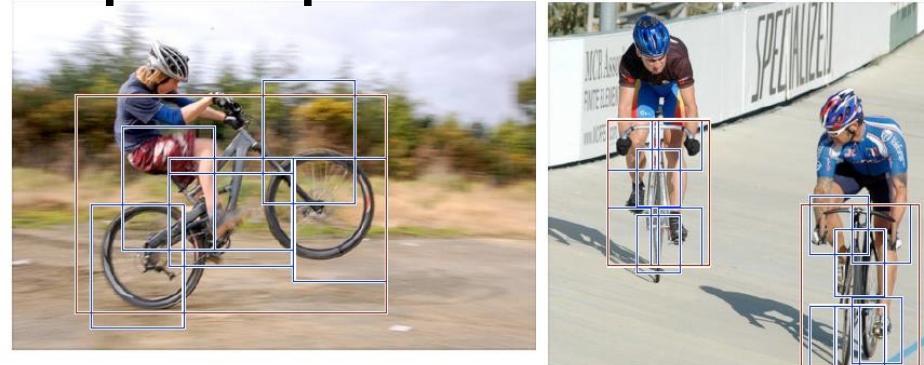
- Object is configuration of parts
- Each part is detectable



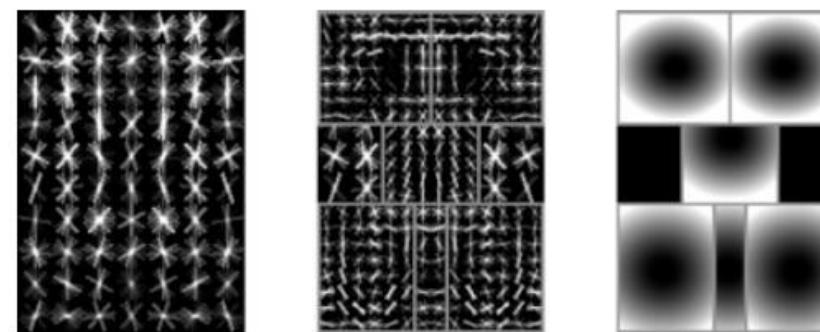
Specifying an object model

3. Hybrid template/parts model

Detections



Template Visualization



root filters
coarse resolution

part filters
finer resolution

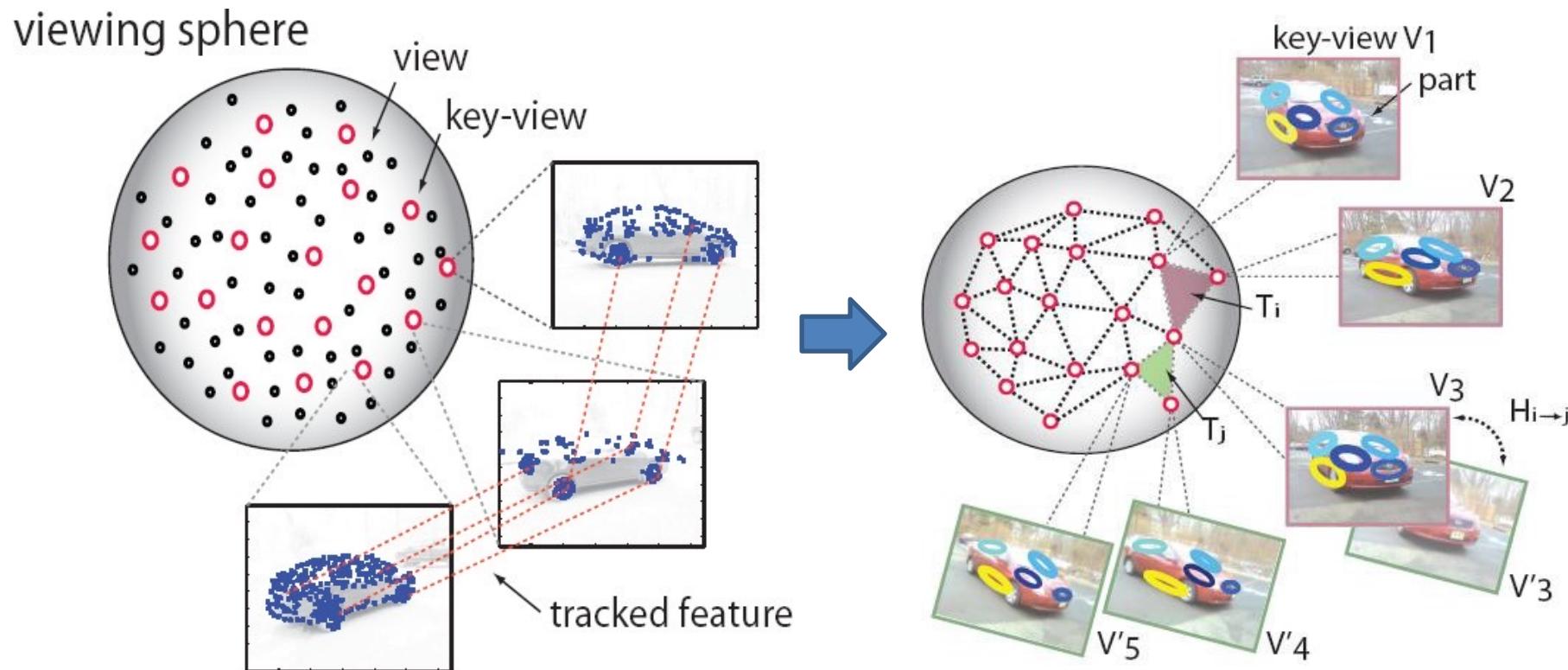
deformation
models

Felzenszwalb et al. 2008

Specifying an object model

4. 3D-ish model

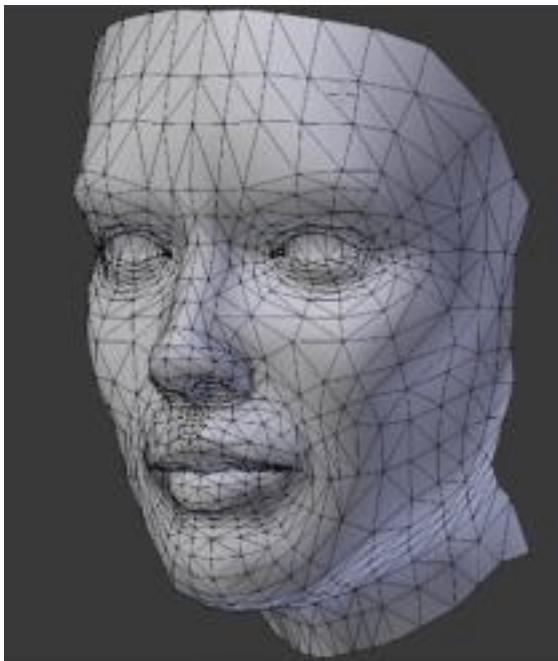
- Object is collection of 3D planar patches under affine transformation



Specifying an object model

5. Deformable 3D model

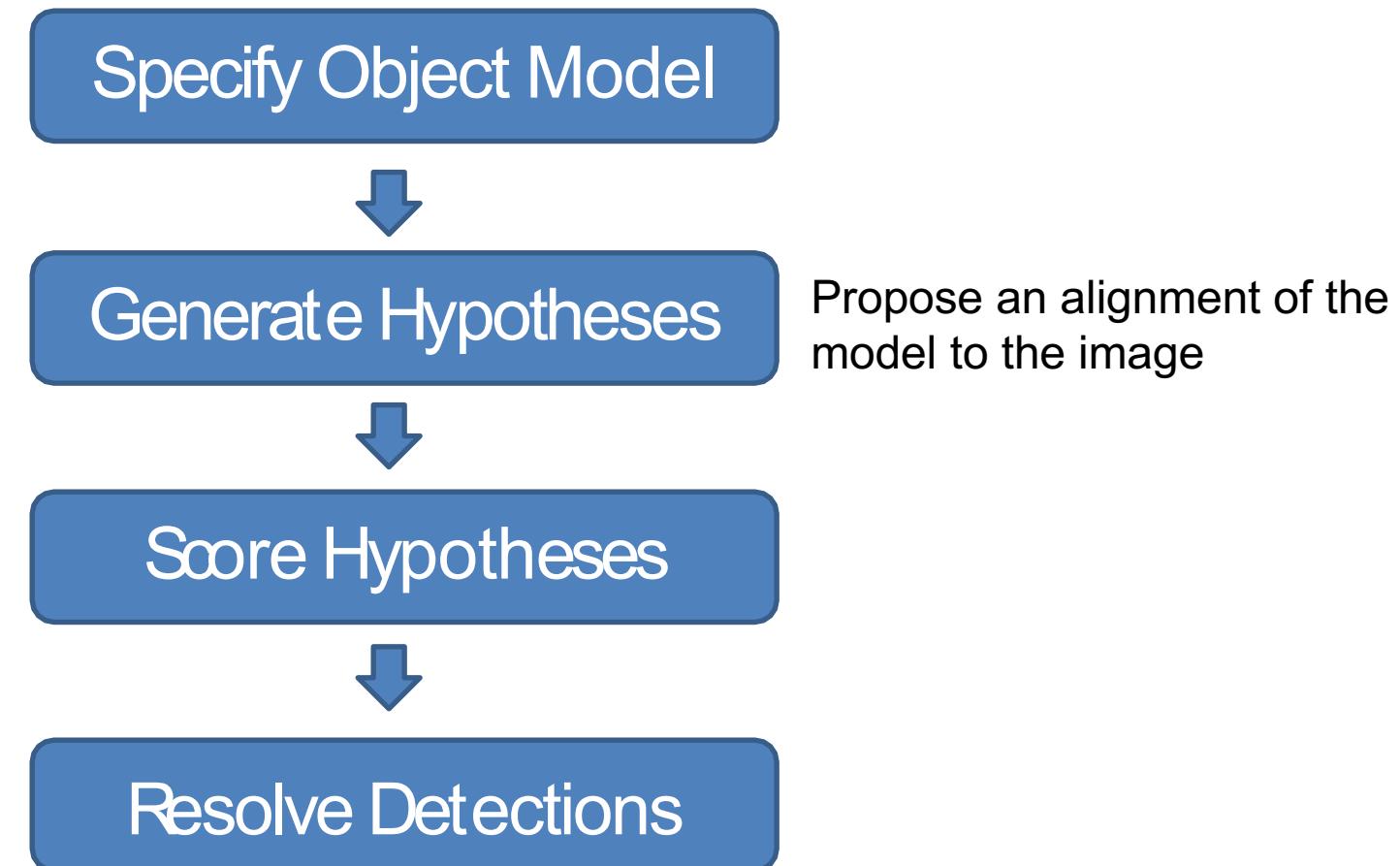
- Object is a parameterized space of shape/pose/deformation of class of 3D object



Why not just pick the most complex model?

- Inference is harder
 - More parameters
 - Harder to ‘fit’ (infer / optimize fit)
 - Longer computation
- “Bounding Box” is still practically the most popular

General Process of Object Detection



Generating hypotheses

1. 2D template model / sliding window
 - Test patch at each location and scale



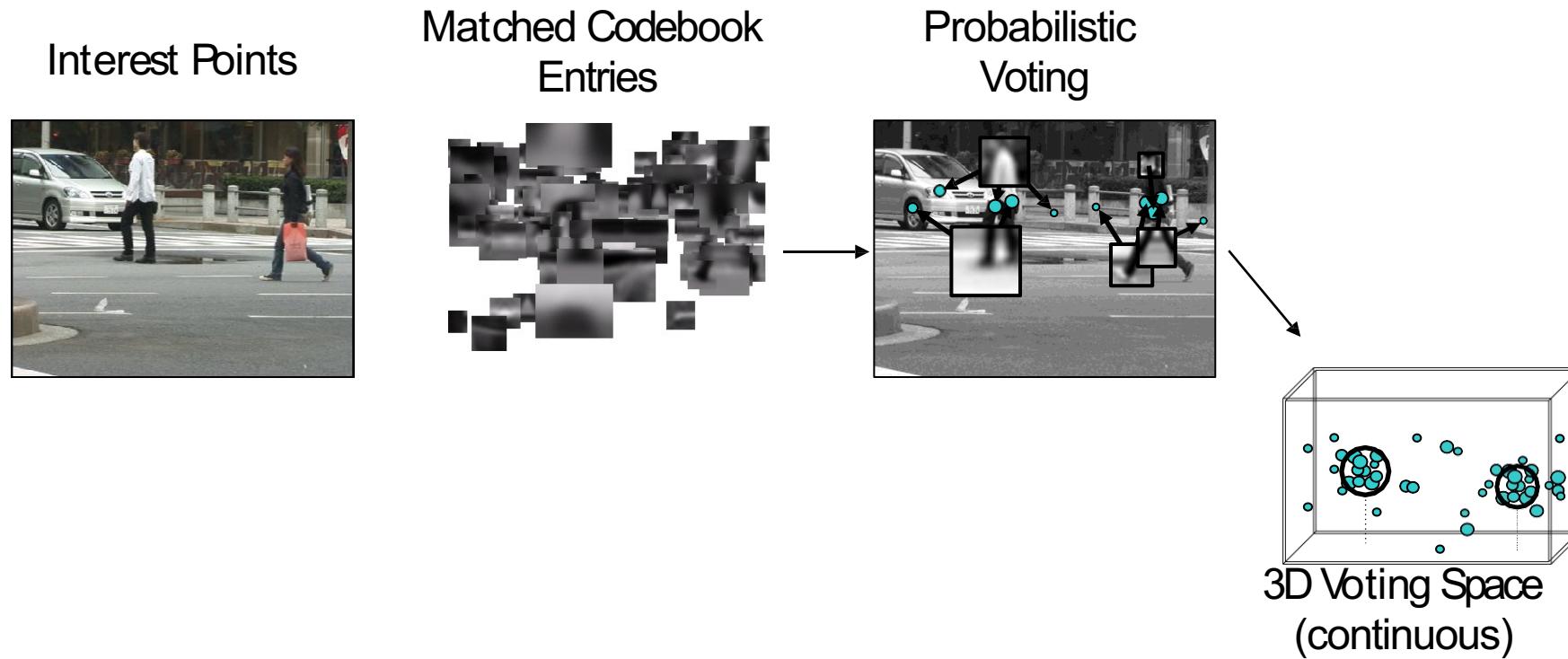
Note – Template did not change size

Each window is separately classified



Generating hypotheses

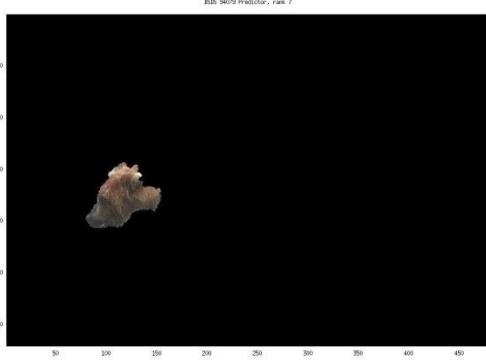
2. Voting from patches/keypoints



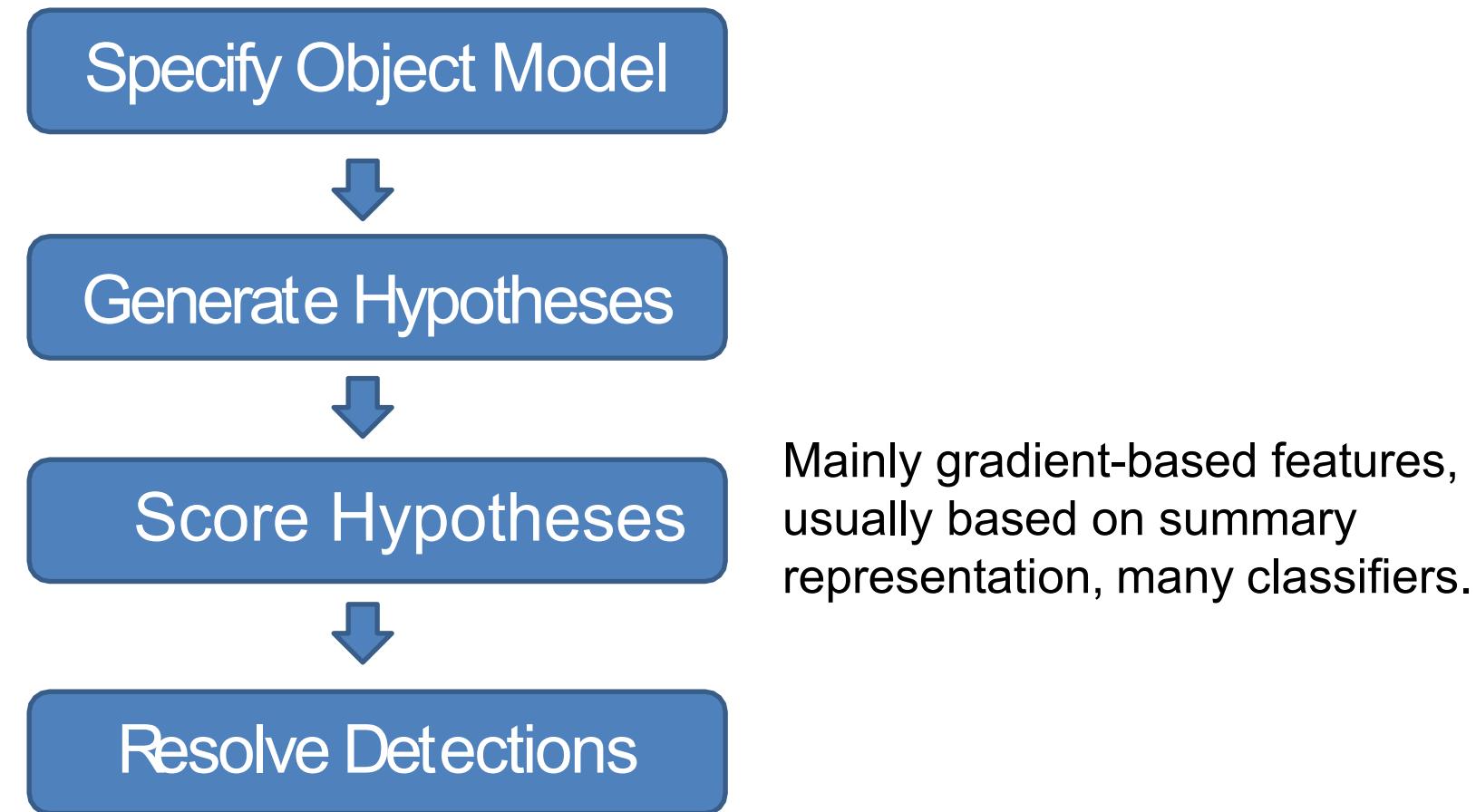
Generating hypotheses

3. Region-based proposal

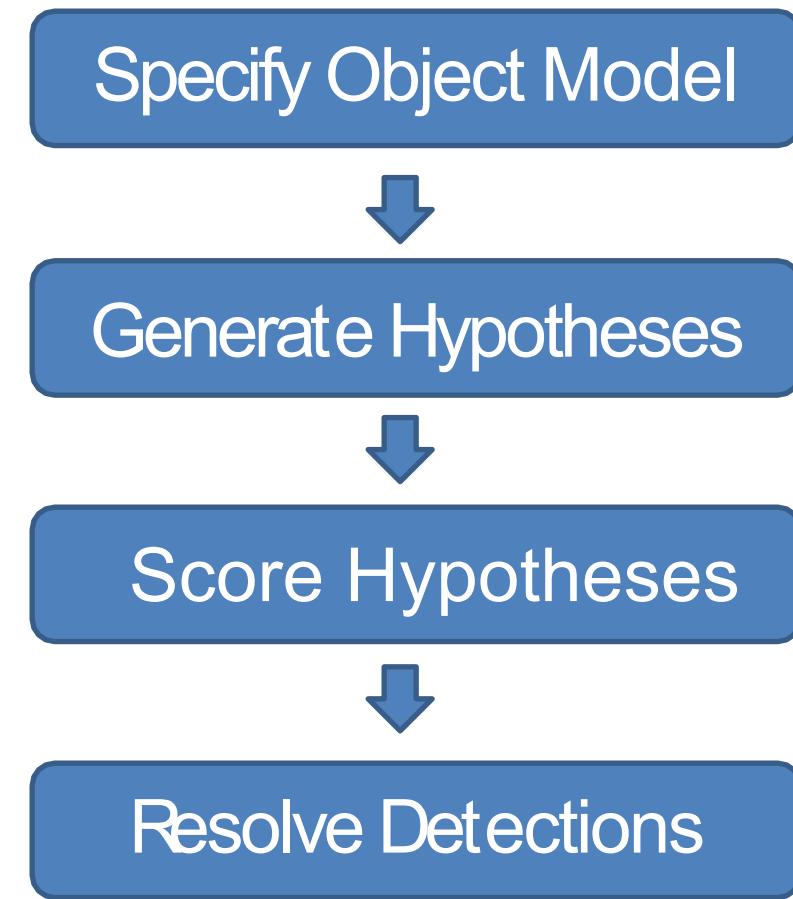
- Arbitrary bounding box + image ‘cut’ segmentation



General Process of Object Detection



General Process of Object Detection



“Globally” rescore each proposed object based on whole set, to resolve conflicts (non-max suppression, context-reasoning...)

Influential Works in Object Detection

- Sung-Poggio (1994, 1998) : ~2000 citations
 - Basic idea of statistical template detection, bootstrapping to get “face-like” negative examples, multiple whole-face prototypes (in 1994)
- Rowley-Baluja-Kanade (1996-1998) : ~3600
 - “Parts” at fixed position, non-maxima suppression, simple cascade, rotation, pretty good accuracy, fast
- Schneiderman-Kanade (1998-2000,2004) : ~1700
 - Careful feature engineering, excellent results, cascade
- Viola-Jones (2001, 2004) : ~13,000
 - Haar-like features, Adaboost as feature selection, hyper-cascade, very fast
- Dalal-Triggs (2005) : ~16,000 citations
 - Careful feature engineering, excellent results, HOG feature, online code
- Felzenszwalb-McAllester-Ramanan (2008): ~4,600 citations
 - Template/parts-based blend
- Girshick et al. (2013): ~2000 citations
 - R-CNN / Fast R-CNN / Faster R-CNN. Deep learned models on object proposals.

Dalal-Triggs Object Detector

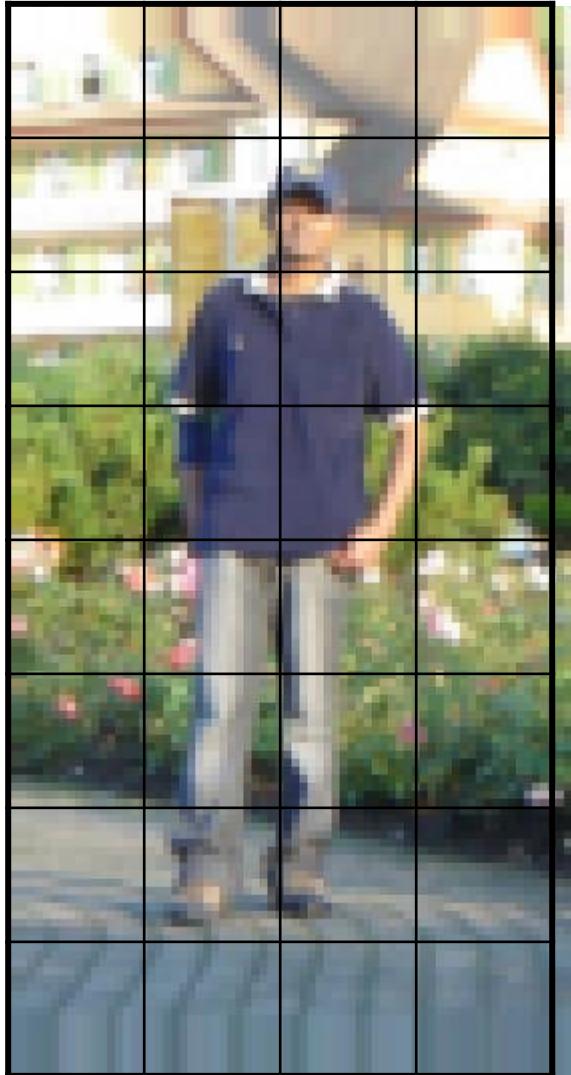


- Histograms of Oriented Gradients for Human Detection, [Navneet Dalal](#), [Bill Triggs](#), International Conference on Computer Vision & Pattern Recognition - June 2005
- <http://lear.inrialpes.fr/pubs/2005/DT05/>

Example: Dalal-Triggs pedestrian detection

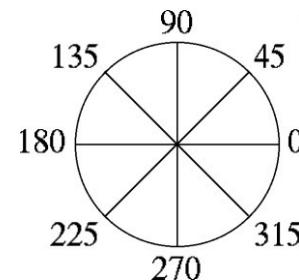


1. Extract fixed-sized (64x128 pixel) **window** at each position and scale
2. Compute **HOG** (histogram of oriented gradient) features within each window
3. Score the window with a **linear SVM classifier**
4. Perform **non-maxima suppression** to remove overlapping detections with lower scores

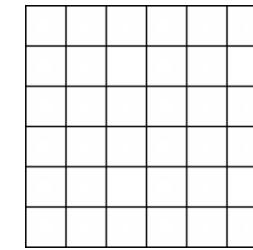


Histogram of Oriented Gradients

Orientation by bins



Histograms over
 $k \times k$ pixel cells



– Votes weighted by magnitude

Dalal-Triggs uses a template with a **rigid form**

- Human bodies are boxed shaped
- That's why Dalal-Triggs is best known for pedestrian detection

But...is there a way to learn the spatial layout more **fluidly**?

- Might help us capture more appearance variation...
- What about faster, too? Since many positions might be “filtered”

Face detection and recognition



Detection



Recognition

“Sally”

Challenges of Face Detection

Sliding window = tens of thousands of location/scale evaluations, especially since faces are small

- One megapixel image has $\sim 10^6$ pixels
- ...and a comparable number of candidate face locations

Faces are also rare: 0–10 per image

- For computational efficiency, spend as little time as possible on non-face windows.
- For 1M pix, to avoid having a false positive in every image, our false positive rate must be less than 10^{-6}

The Viola/Jones Face Detector

A seminal approach to real-time object detection. Training is slow, but detection is very fast

Key ideas:

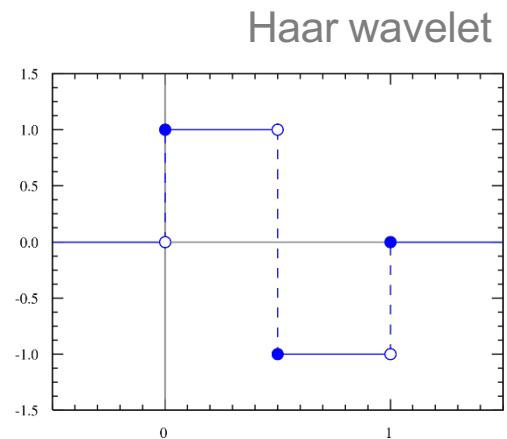
1. *Integral images* for fast feature evaluation
2. *Boosting* for feature selection
3. *Attentional cascade* for fast non-face window rejection

[P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. CVPR 2001.](#)

P. Viola and M. Jones. [Robust real-time face detection.](#) IJCV 57(2), 2004.

“Haar-like features”

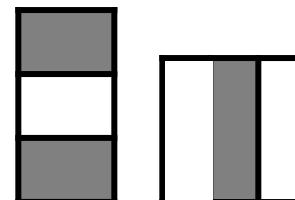
- Differences of sums of intensity
- Computed at different positions and scales within sliding window
- Very fast to compute (thanks to “integral image”)



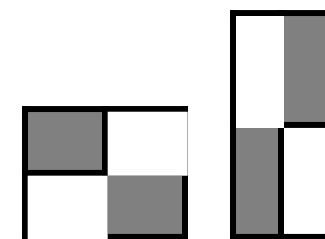
-1 +1



Two-rectangle features



Three-rectangle features



Etc.

But these features are rubbish...!

Yes, individually they are ‘weak classifiers’

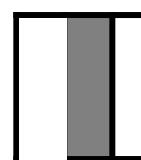
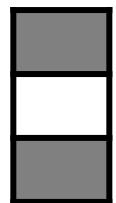
*Jargon: ‘feature’ and ‘classifier’ are used interchangeably here.
Also with ‘learner’, ‘filter’.*

But, what if we combine *thousands* of them...

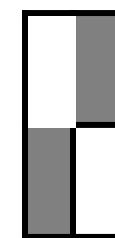
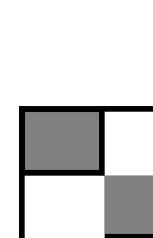
-1 +1



Two-rectangle features



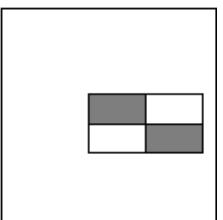
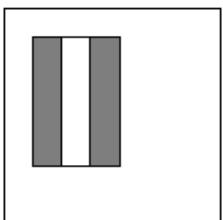
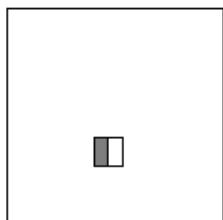
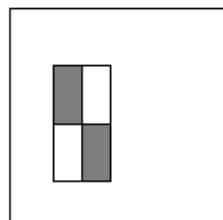
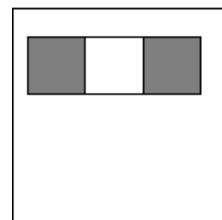
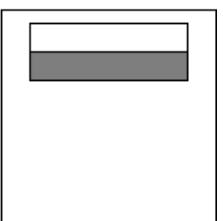
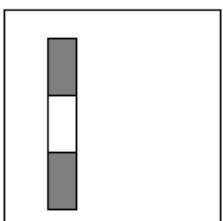
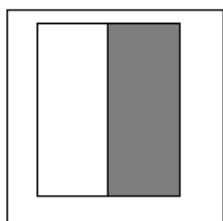
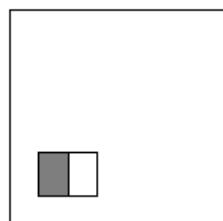
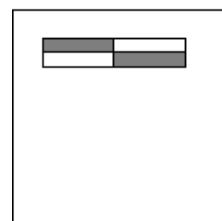
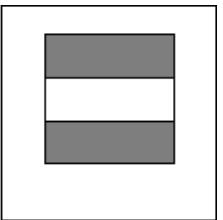
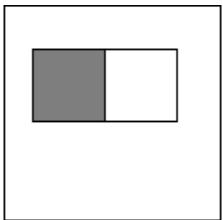
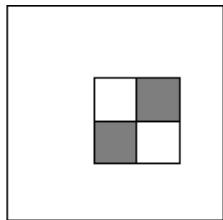
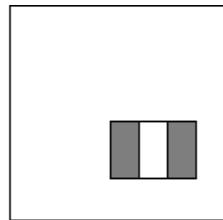
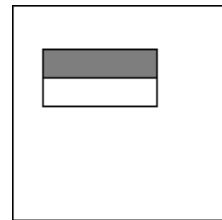
Three-rectangle features



Etc.

How many features are there?

For a 24x24 detection region, the number of possible rectangle features is ~160,000!



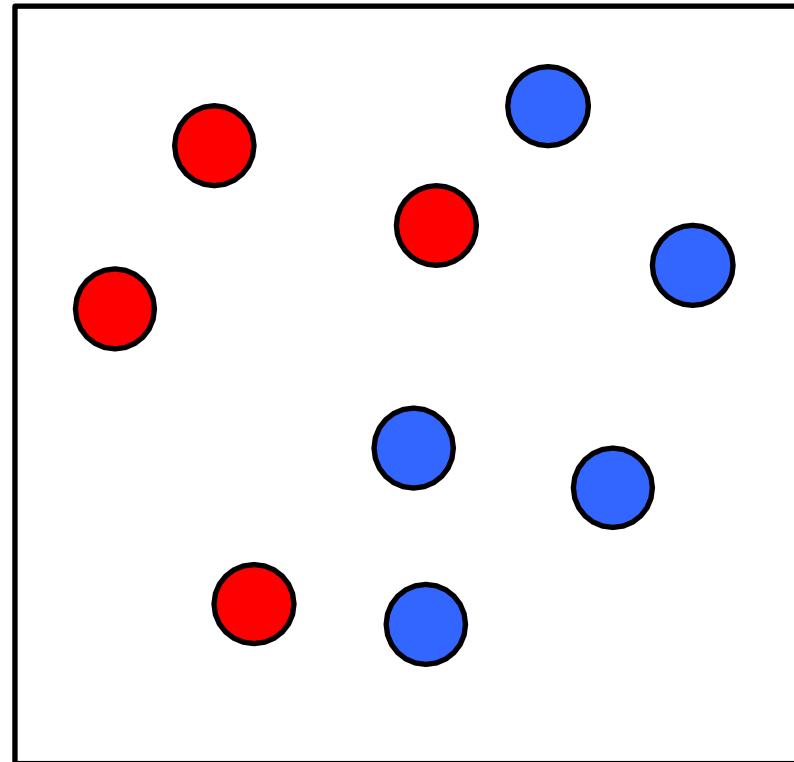
How many features are there?

- For a 24x24 detection region, the number of possible rectangle features is \sim 160,000!
- At test time, it is impractical to evaluate the entire feature set.
- Can we learn a ‘strong classifier’ using just a small subset of all possible features?

Boosting for feature selection

Initially, weight each training example equally.

Weight = size of point



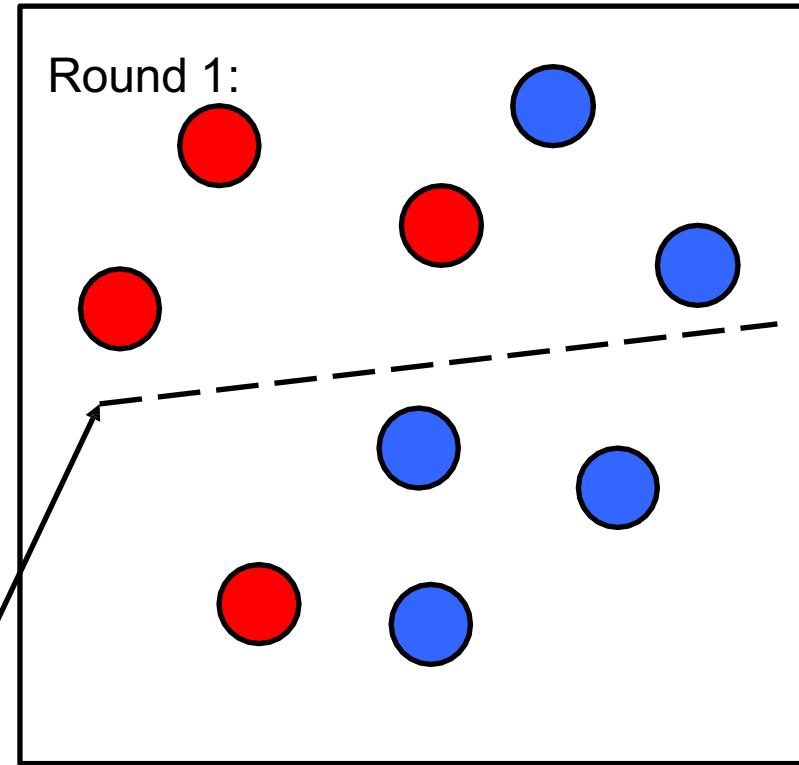
Boosting for feature selection

In each boosting round:

Find the weak classifier that achieves the lowest *weighted* training error.

Raise the weights of training examples misclassified by current weak classifier.

**Weak
Classifier 1**



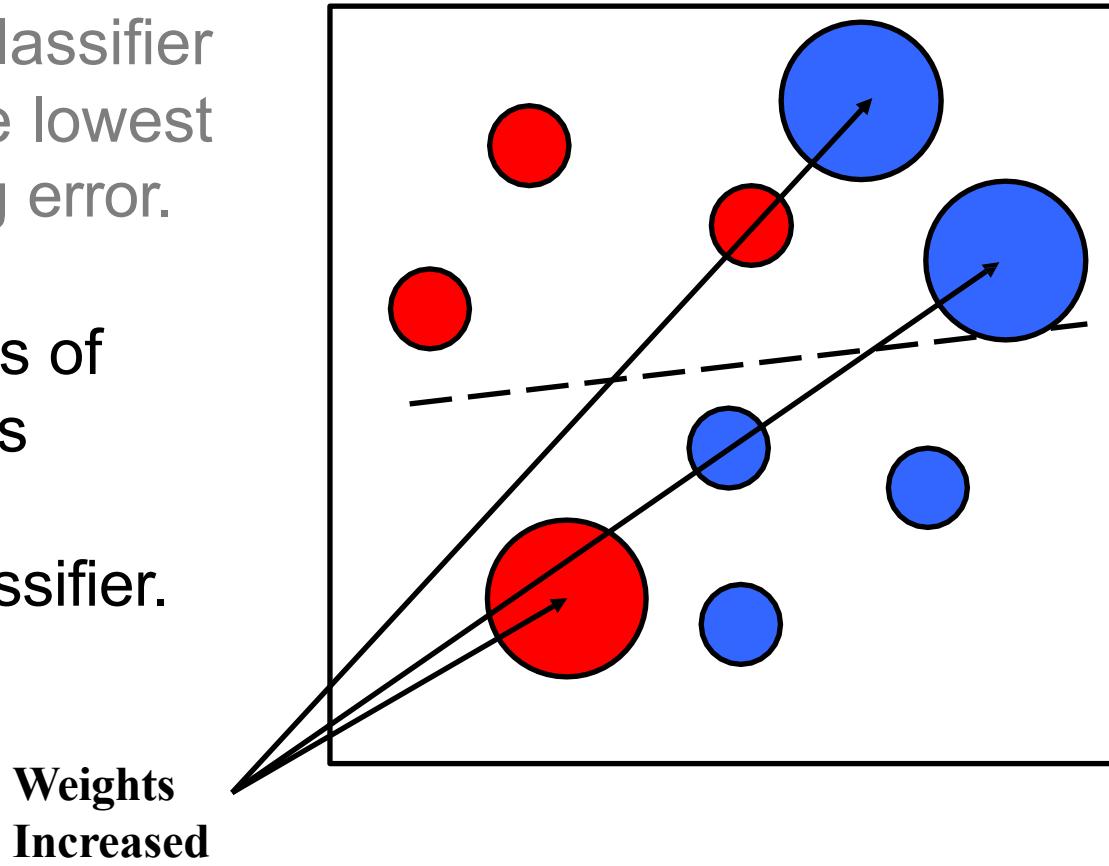
Boosting illustration

In each boosting round:

Find the weak classifier
that achieves the lowest
weighted training error.

Raise the weights of
training examples
misclassified by
current weak classifier.

Round 1:



Boosting illustration

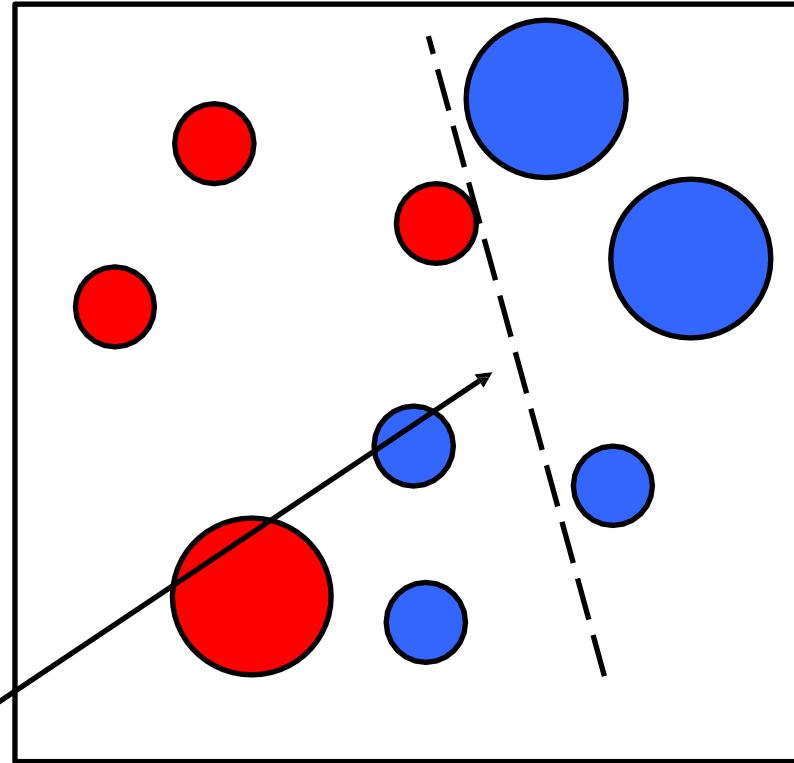
In each boosting round:

Find the weak classifier that achieves the lowest *weighted* training error.

Raise the weights of training examples misclassified by current weak classifier.

Weak
Classifier 2

Round 2:



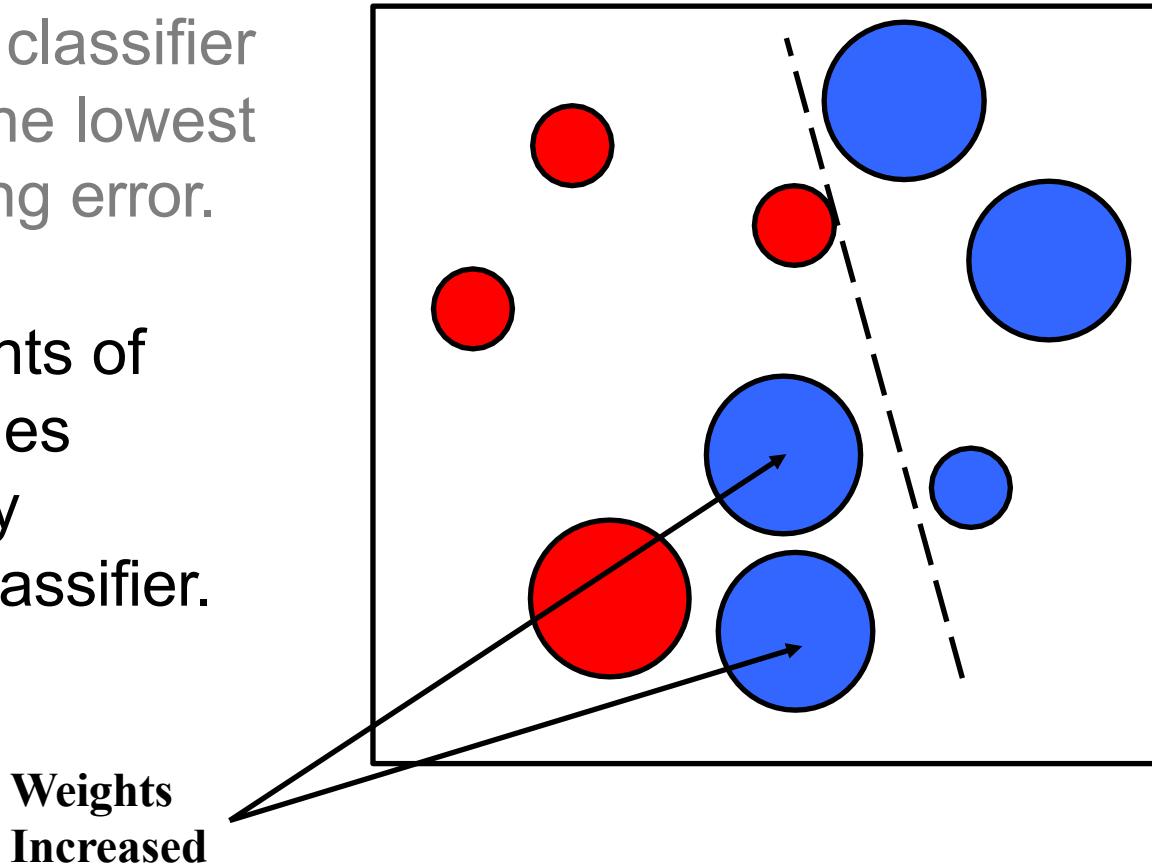
Boosting illustration

In each boosting round:

Find the weak classifier
that achieves the lowest
weighted training error.

Raise the weights of
training examples
misclassified by
current weak classifier.

Round 2:



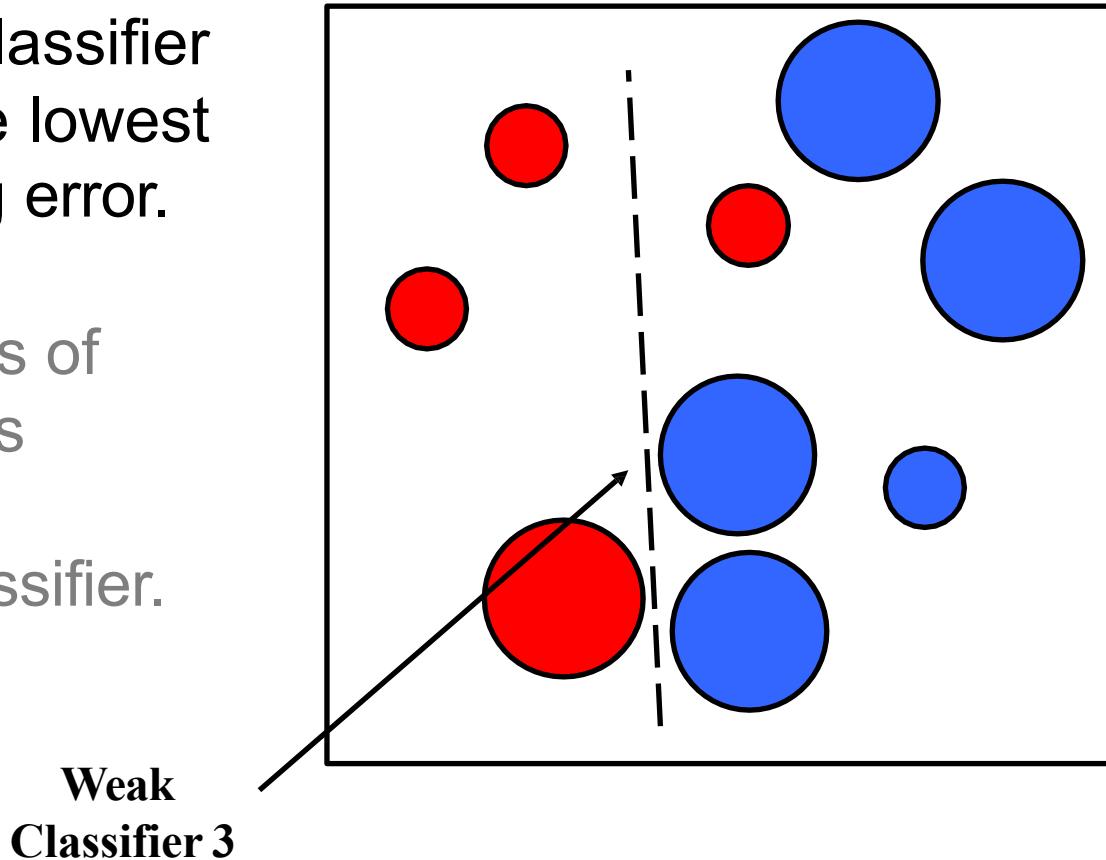
Boosting illustration

In each boosting round:

Find the weak classifier that achieves the lowest *weighted* training error.

Raise the weights of training examples misclassified by current weak classifier.

Round 3:

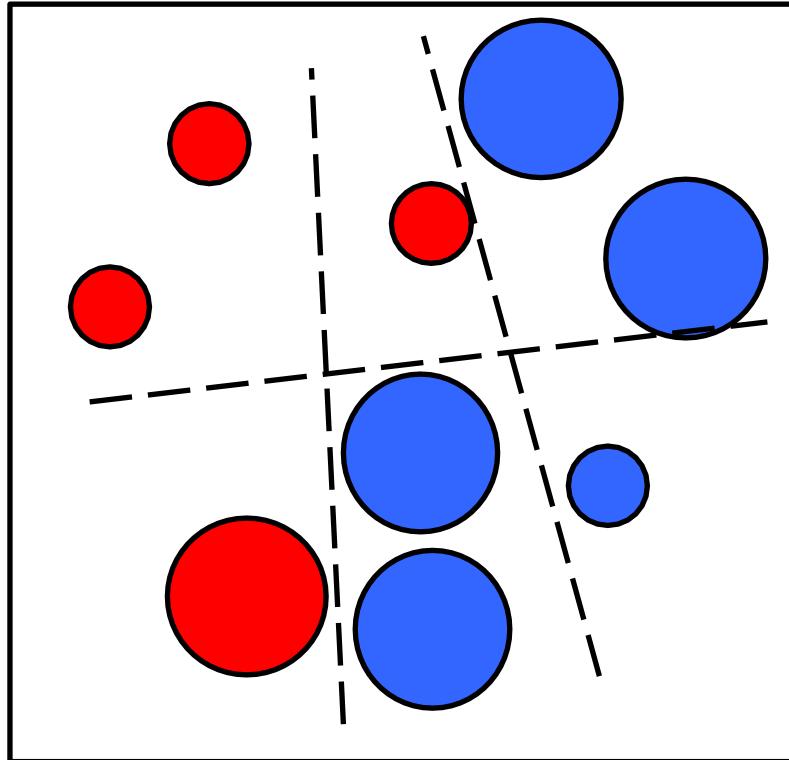


Boosting illustration

Compute final classifier as linear combination of all weak classifier.

Weight of each classifier is directly proportional to its accuracy.

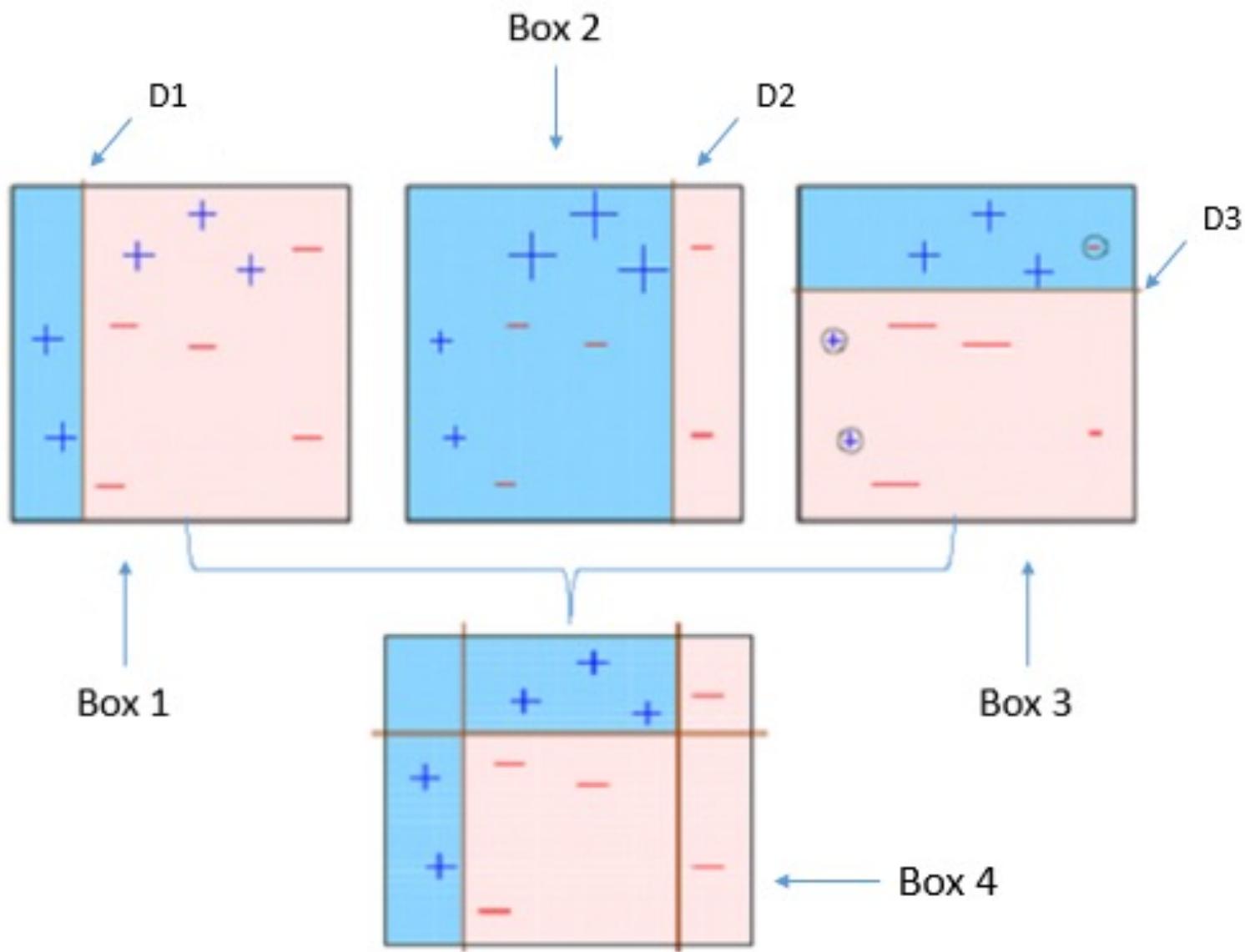
Round 3:



Exact formulas for re-weighting and combining weak learners depend on the boosting scheme (e.g., AdaBoost).

Y. Freund and R. Schapire, [A short introduction to boosting](#),
Journal of Japanese Society for Artificial Intelligence, 14(5):771-780, September, 1999.

Boosting illustration



Feature selection with boosting

- Create a large pool of features (160K)
- Select discriminative features that work well together

$$h(\mathbf{x}) = \text{sign} \left(\sum_{j=1}^M \alpha_j h_j(\mathbf{x}) \right)$$

Final strong learner
window
Weak learner
Learner weight

– “Weak learner” = feature + threshold + ‘polarity’

$$h_j(\mathbf{x}) = \begin{cases} -s_j & \text{if } f_j < \theta_j \\ s_j & \text{otherwise} \end{cases}$$

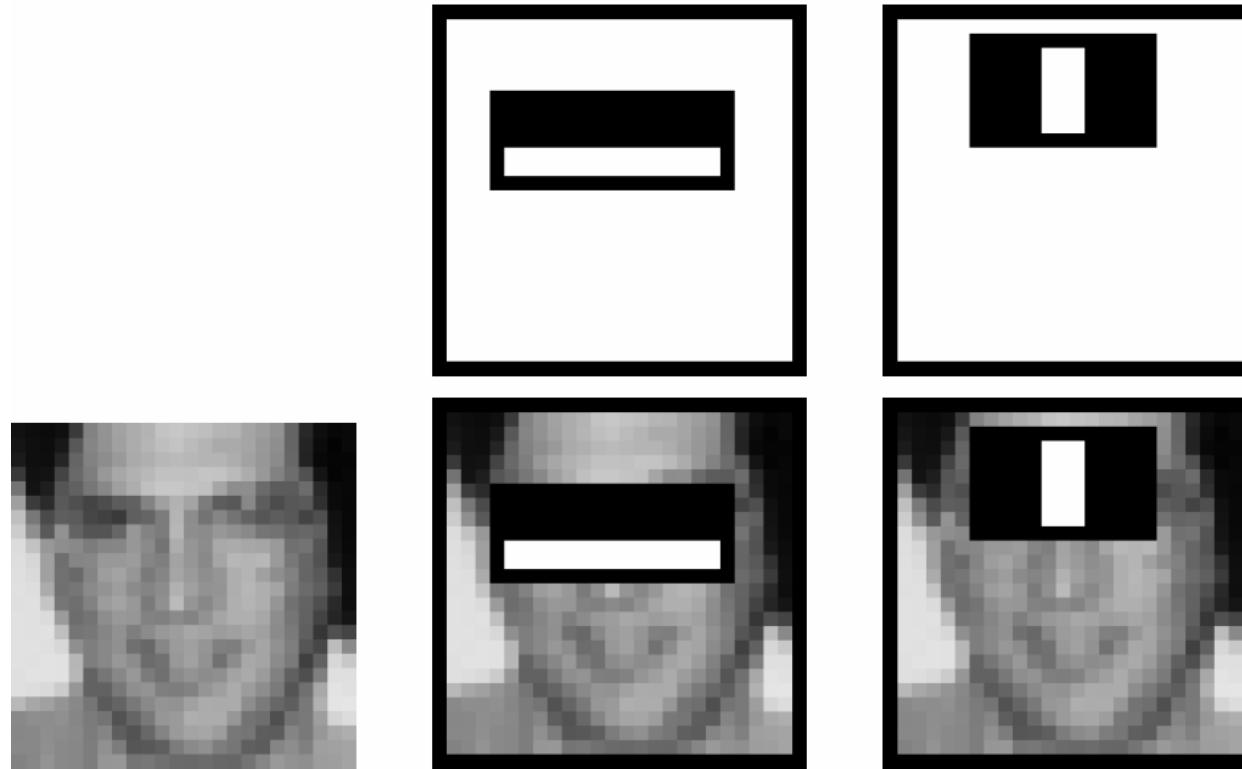
value of rectangle feature
threshold

‘polarity’ = black or white region flip $\rightarrow s_j \in \pm 1$

– Train & choose weak learner that minimizes error on the weighted training set, then reweight

Boosting for face detection

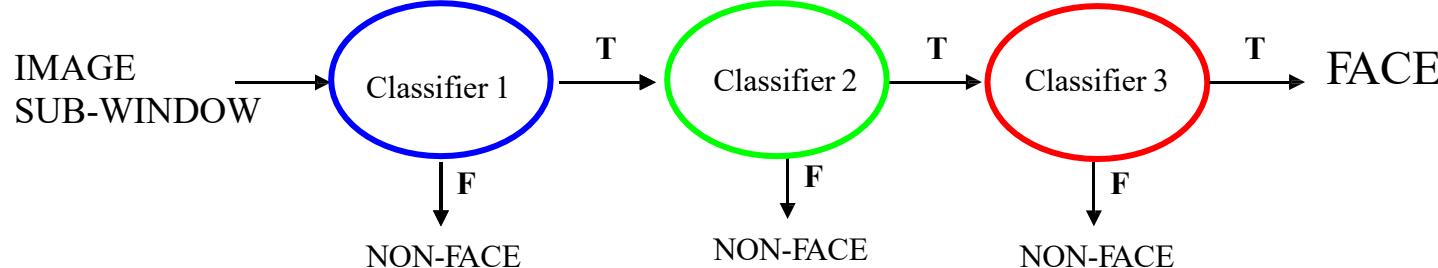
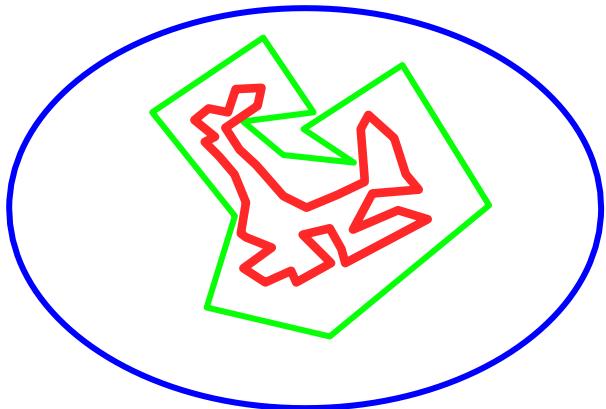
- First two features selected by boosting:



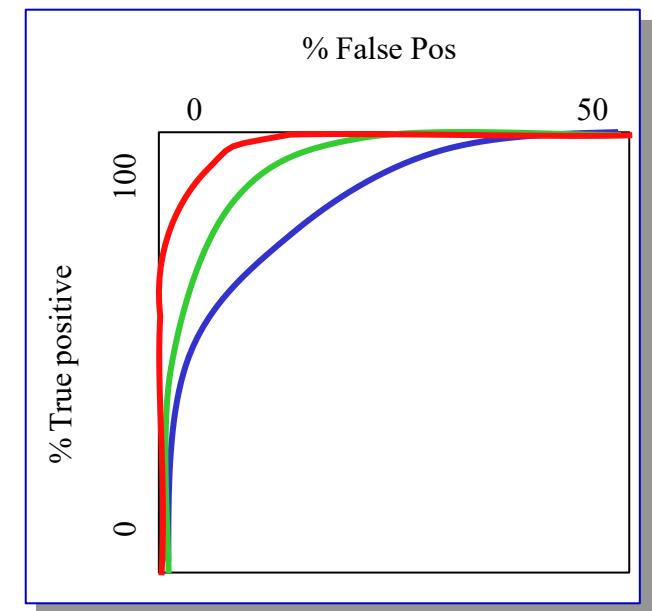
This feature combination can yield 100% recall and 50% false positive rate

3. Attentional cascade

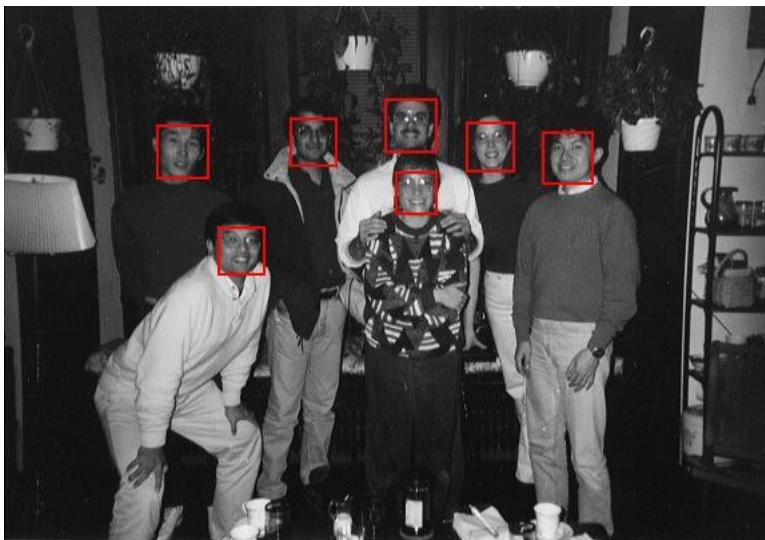
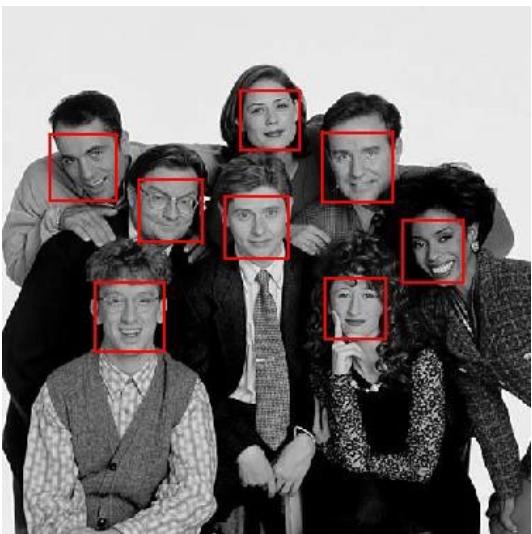
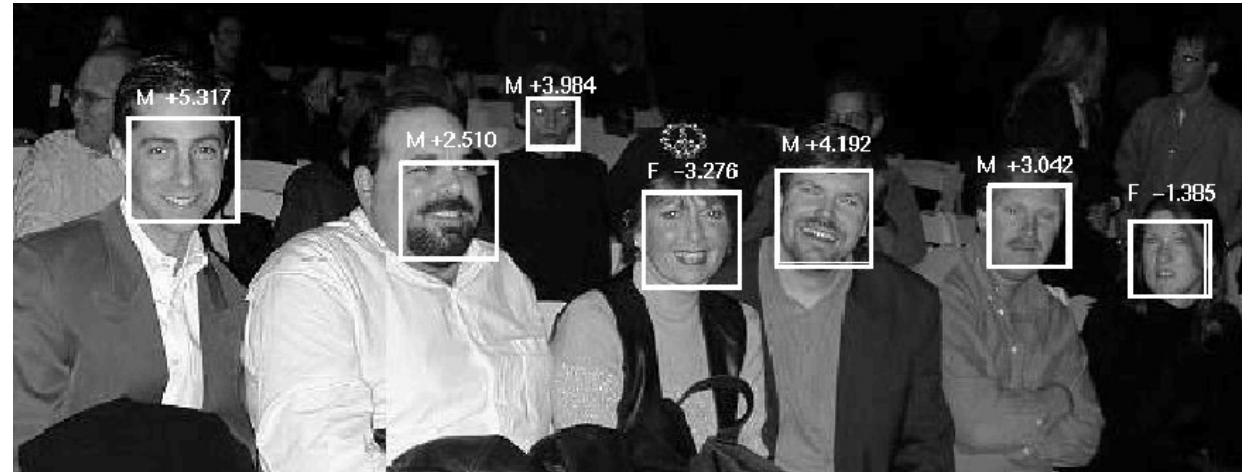
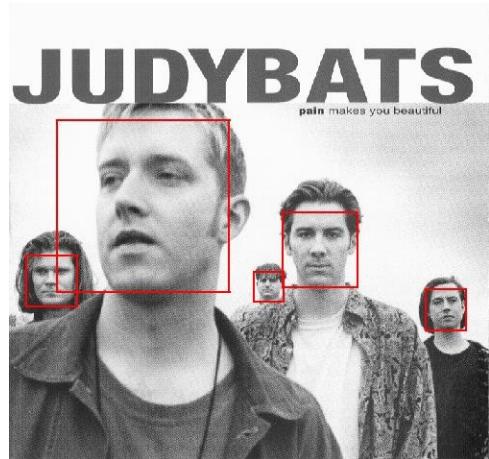
- Chain classifiers that are progressively more complex
- Minimize *false positive rates* at each stage, not absolute error



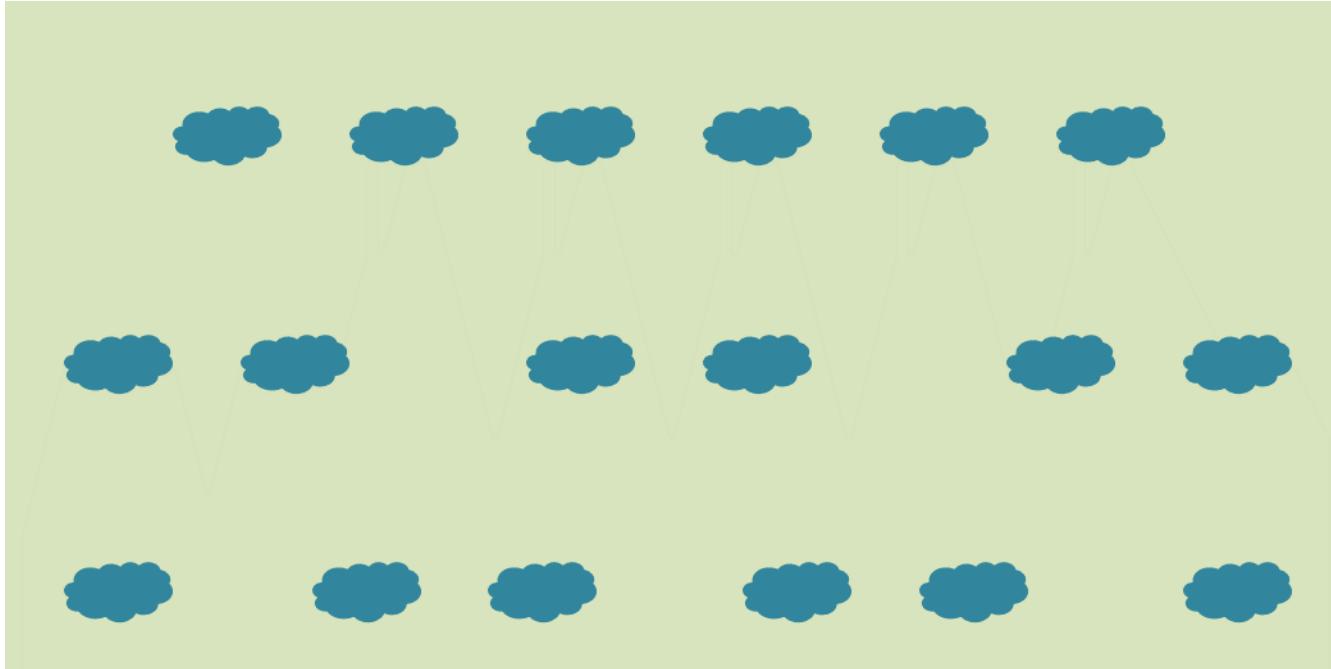
Receiver operating characteristic



Viola/Jones detector is very powerful

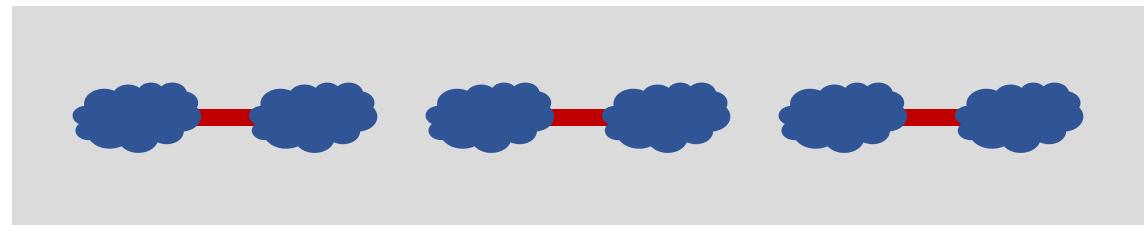
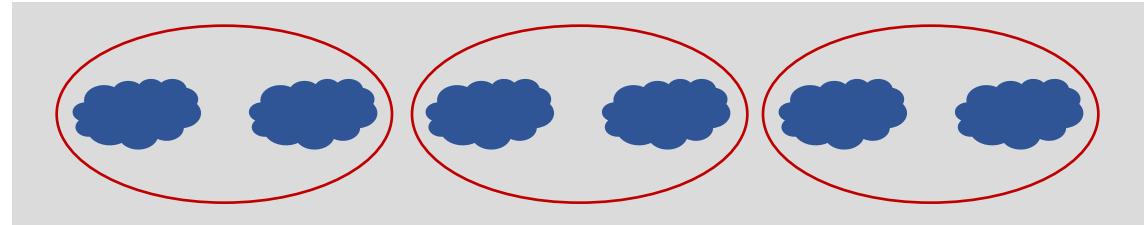


Question: what makes an object “segmentable”?



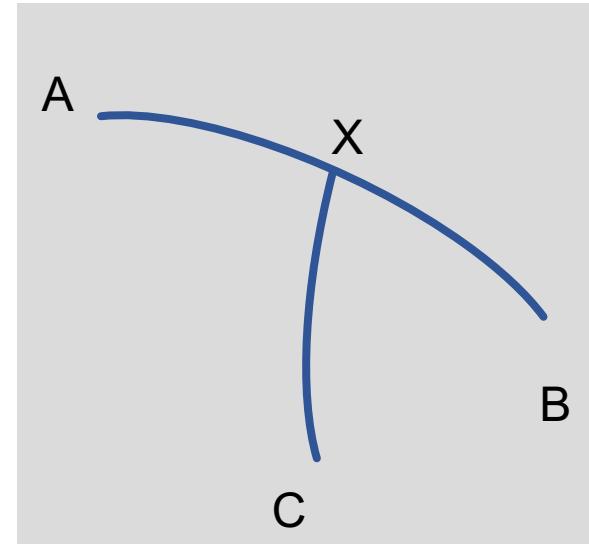
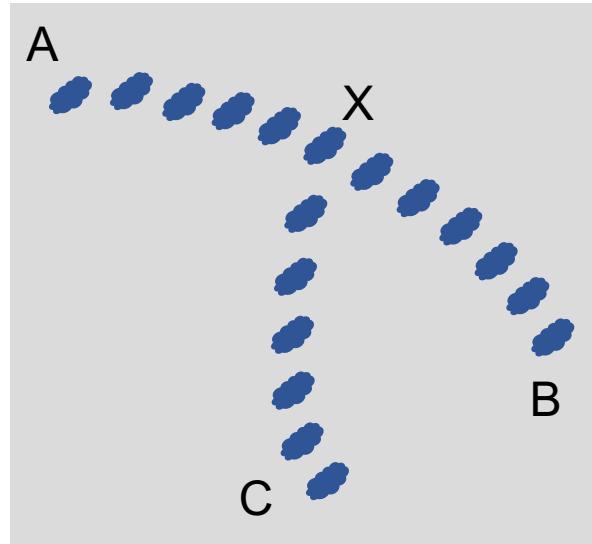
Objects with similar motion or change in appearance are grouped together

Common Region/Connectivity



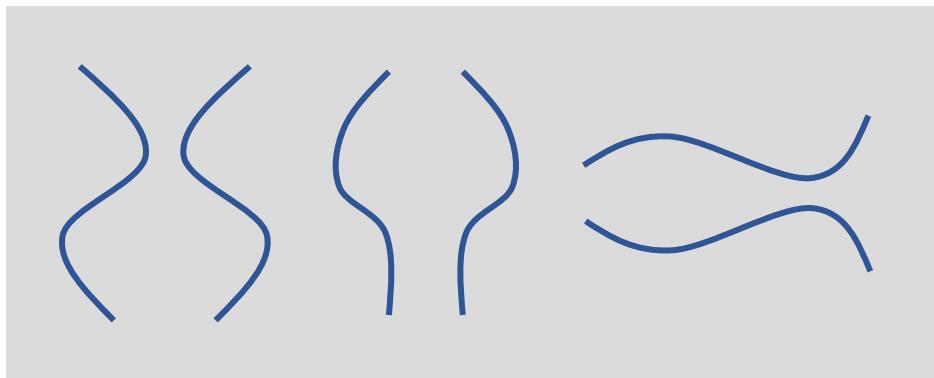
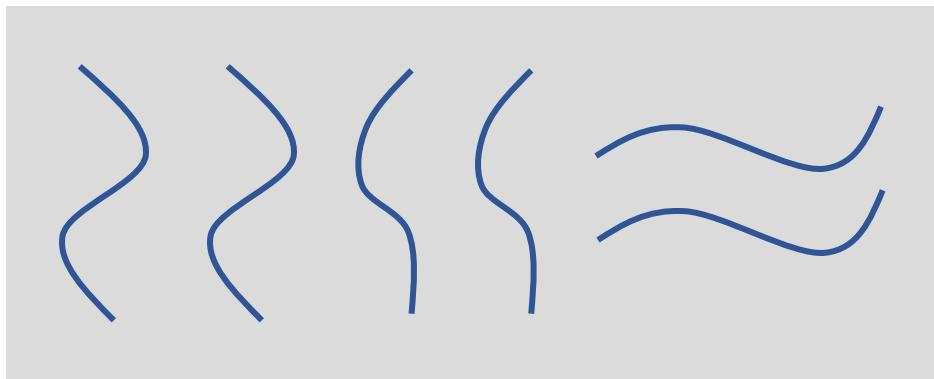
Connected objects are grouped together

Continuity Principle

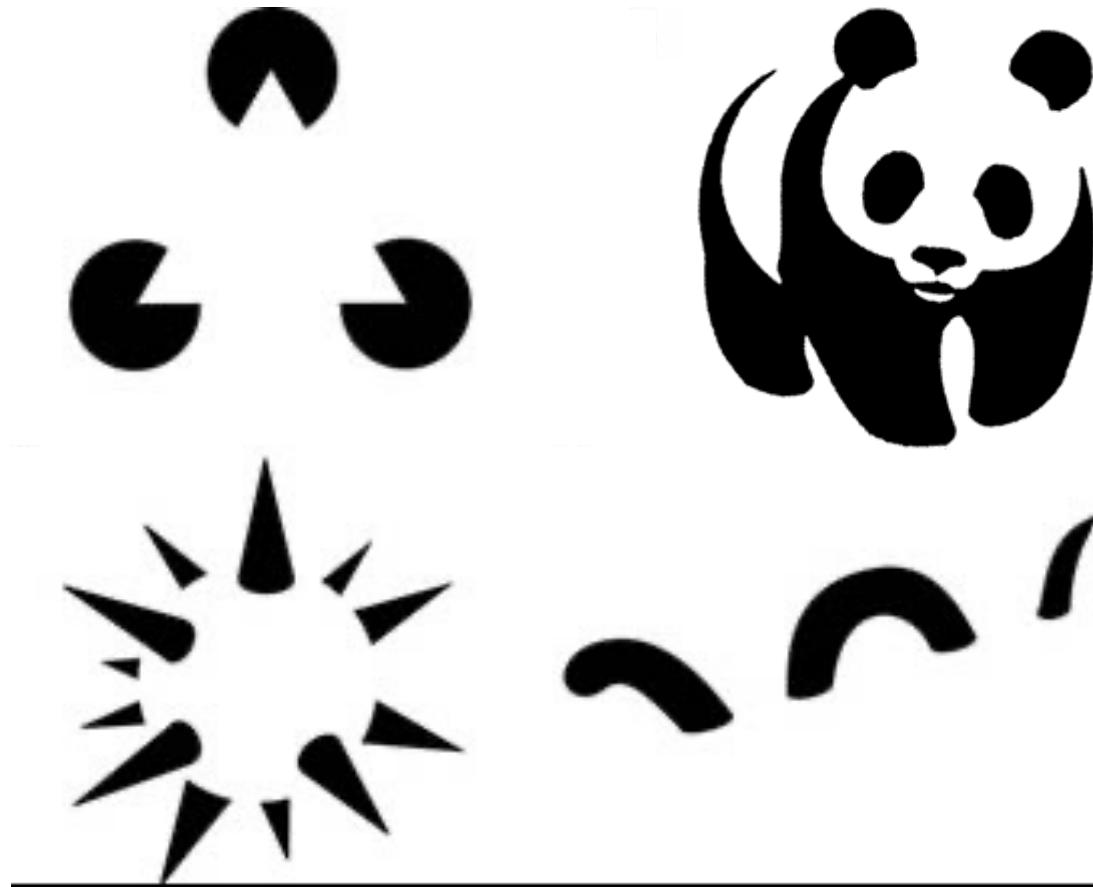


Features on a continuous curve are grouped together

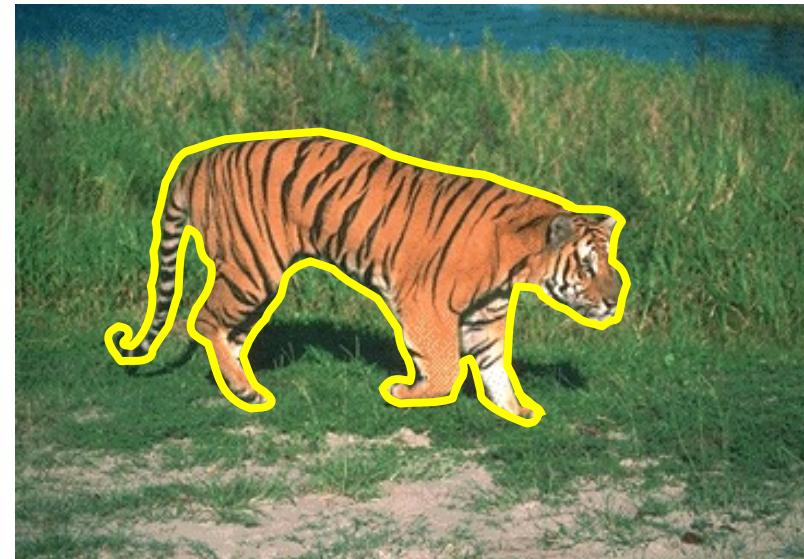
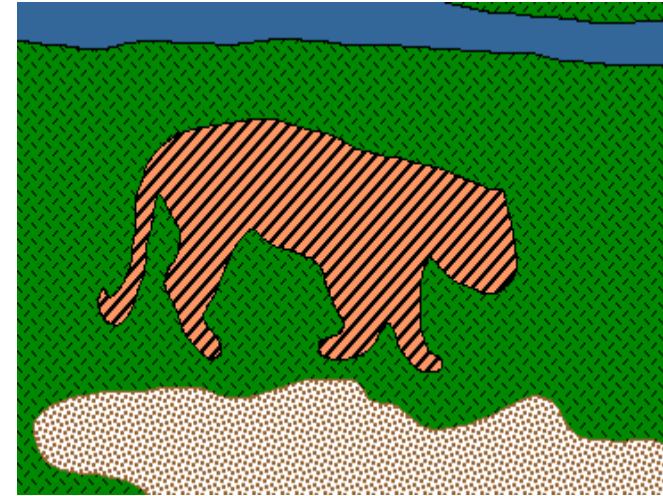
Symmetry Principle

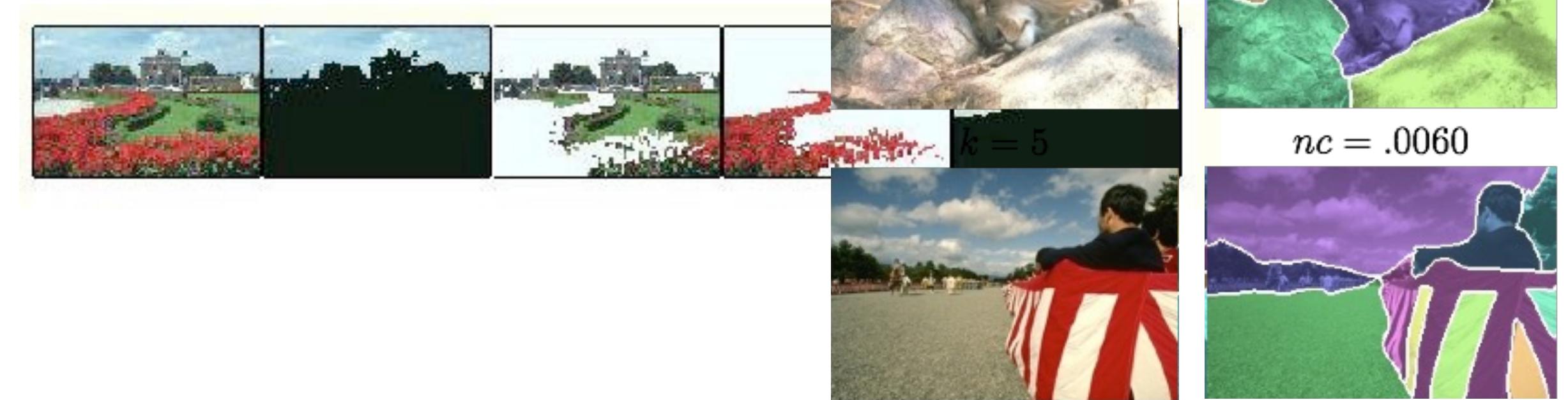


Completion



Illusory or subjective contours are perceived

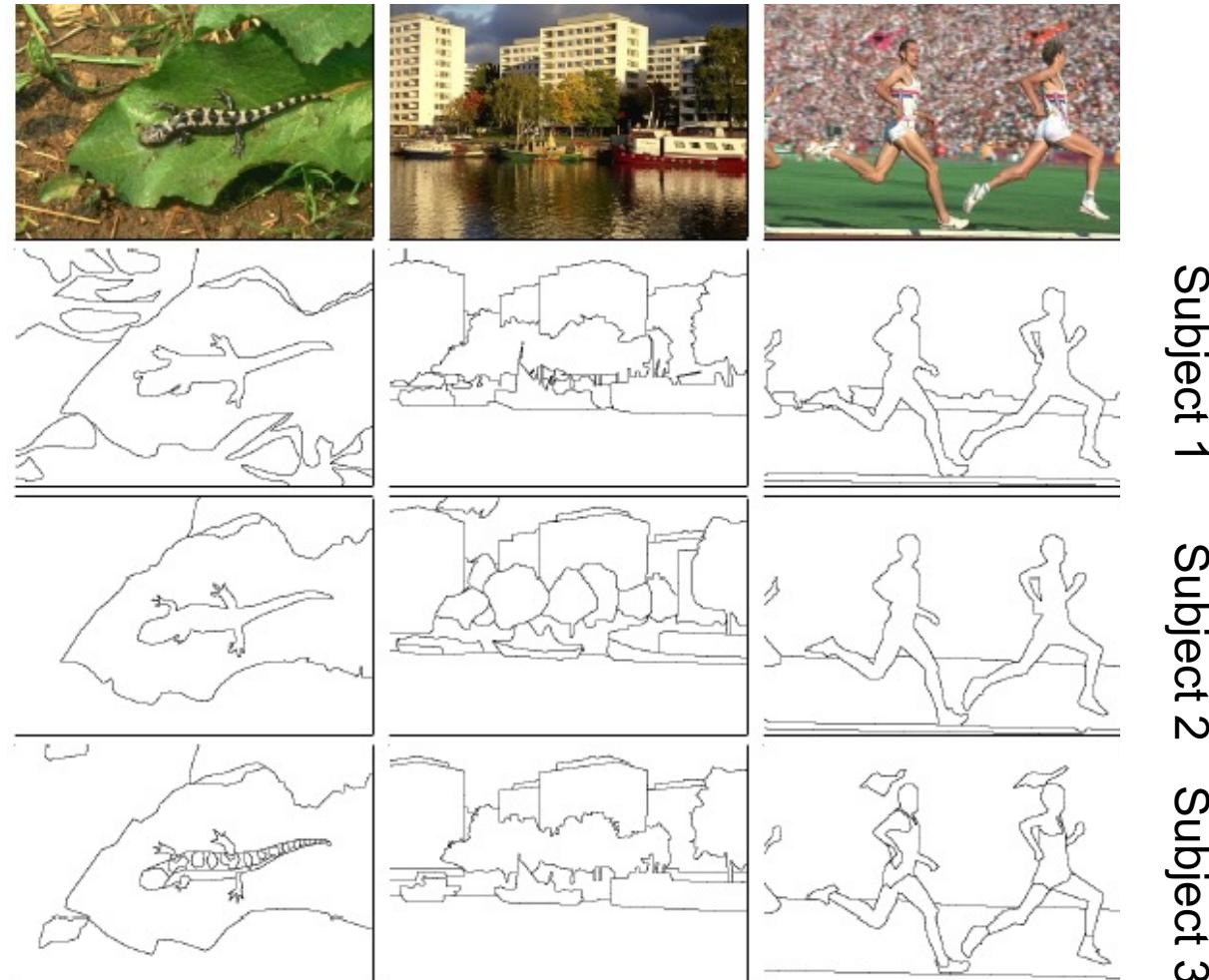




What is a “good”
segmentation??

First idea: Compare to human “ground truth”

No objective definition of segmentation!



- <http://www.eecs.berkeley.edu/Research/Projects/CS/vision/grouping/resources.html>

Evaluation: Intersection-over-Union (IoU) with ground truth



.825

$$OS(S, G) = \frac{|S \cap G|}{|S \cup G|}$$

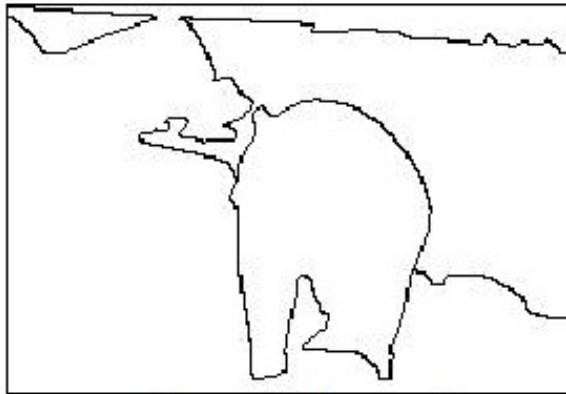


.892

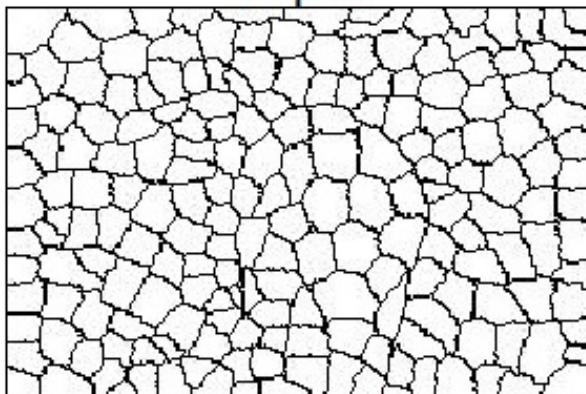
Second idea: Superpixels



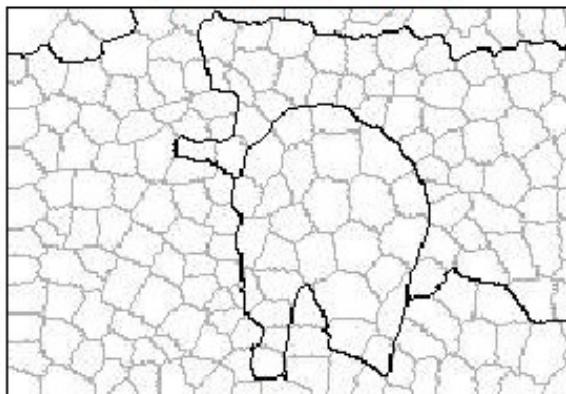
Input



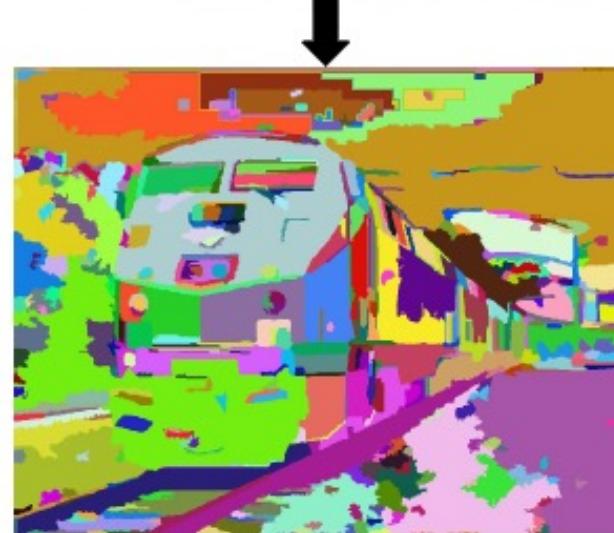
Ground truth



Superpixels



Overlay

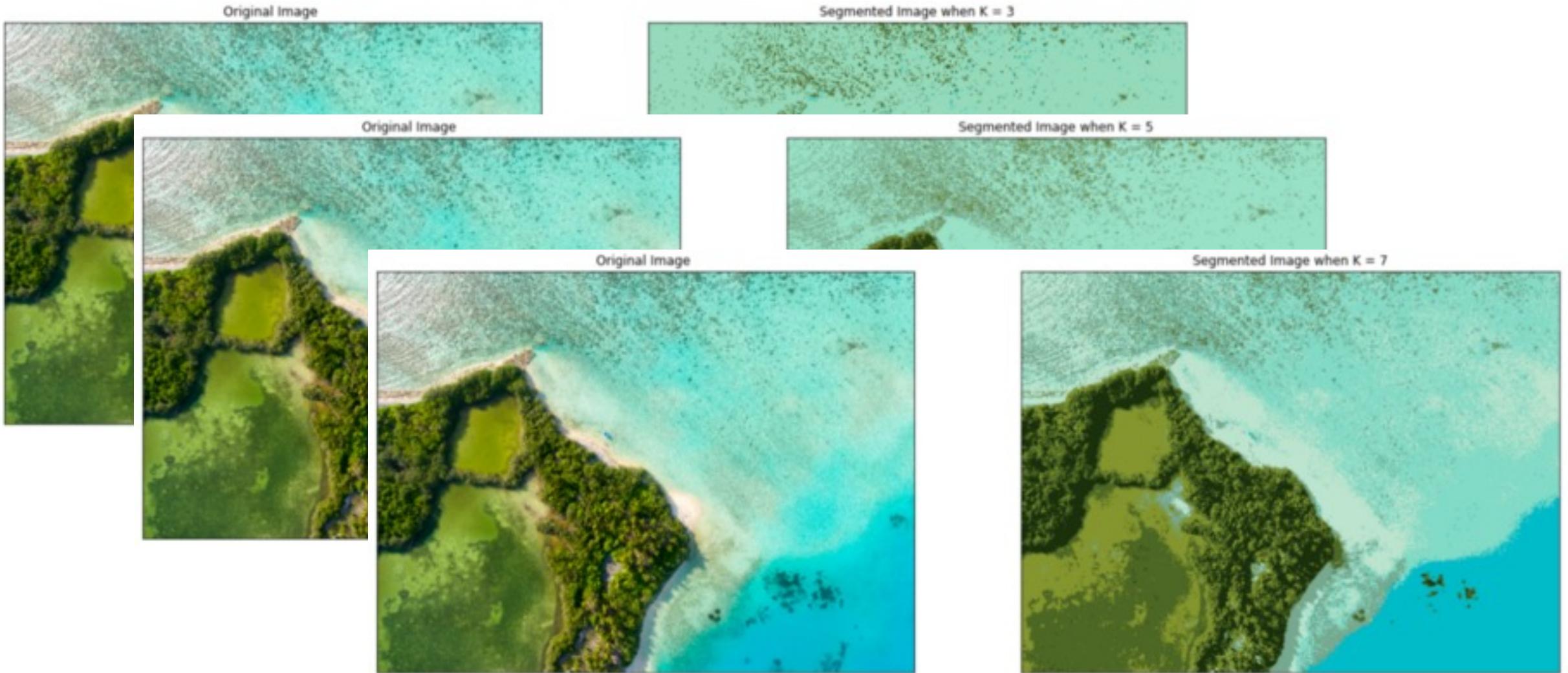


- Let's not even try to compute a “correct” segmentation
- Let's be content with an *oversegmentation* in which each region is very likely (formal guarantees are hard) to be uniform

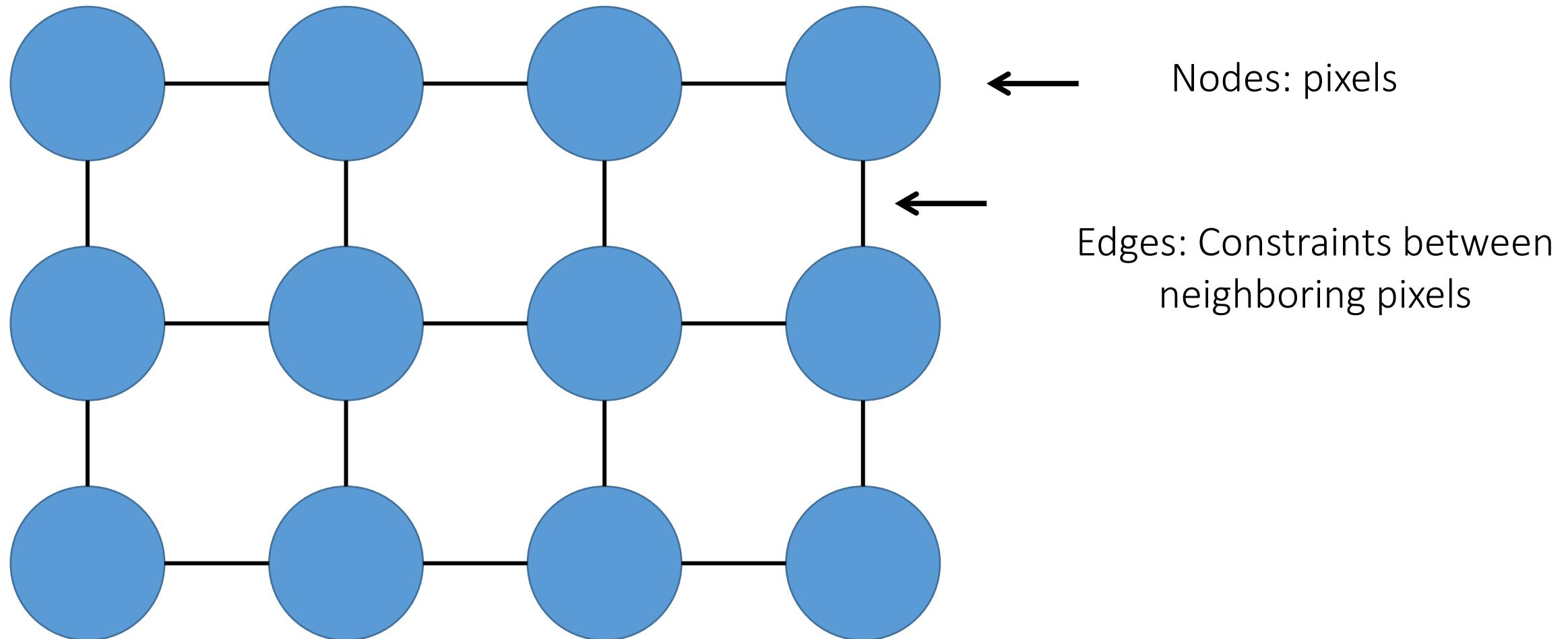
Main approaches

- Spectral techniques
- Segmentation as boundary detection
- **Graph-based techniques**
- Clustering (K-means and probabilistic)
- Mean shift

K-means can be “okay” image segmentation

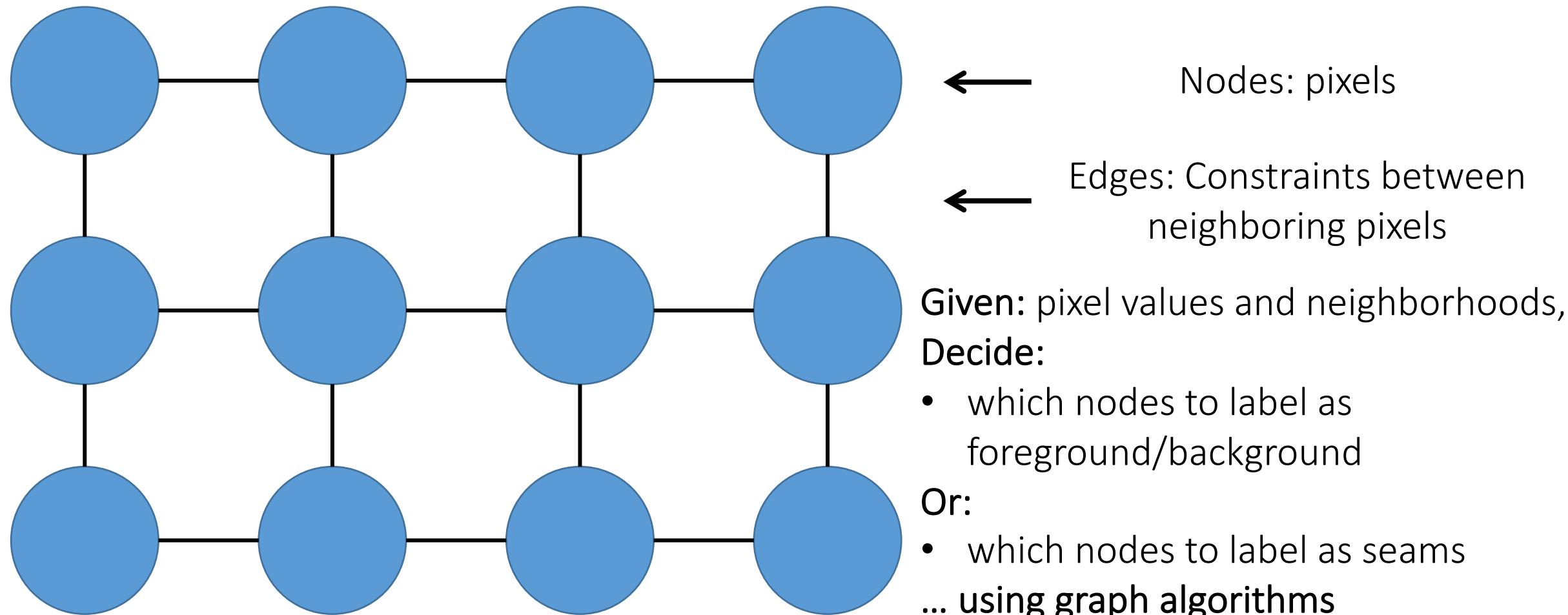


Images can be viewed as graphs



Graph-view of segmentation problem

Segmentation is node-labeling



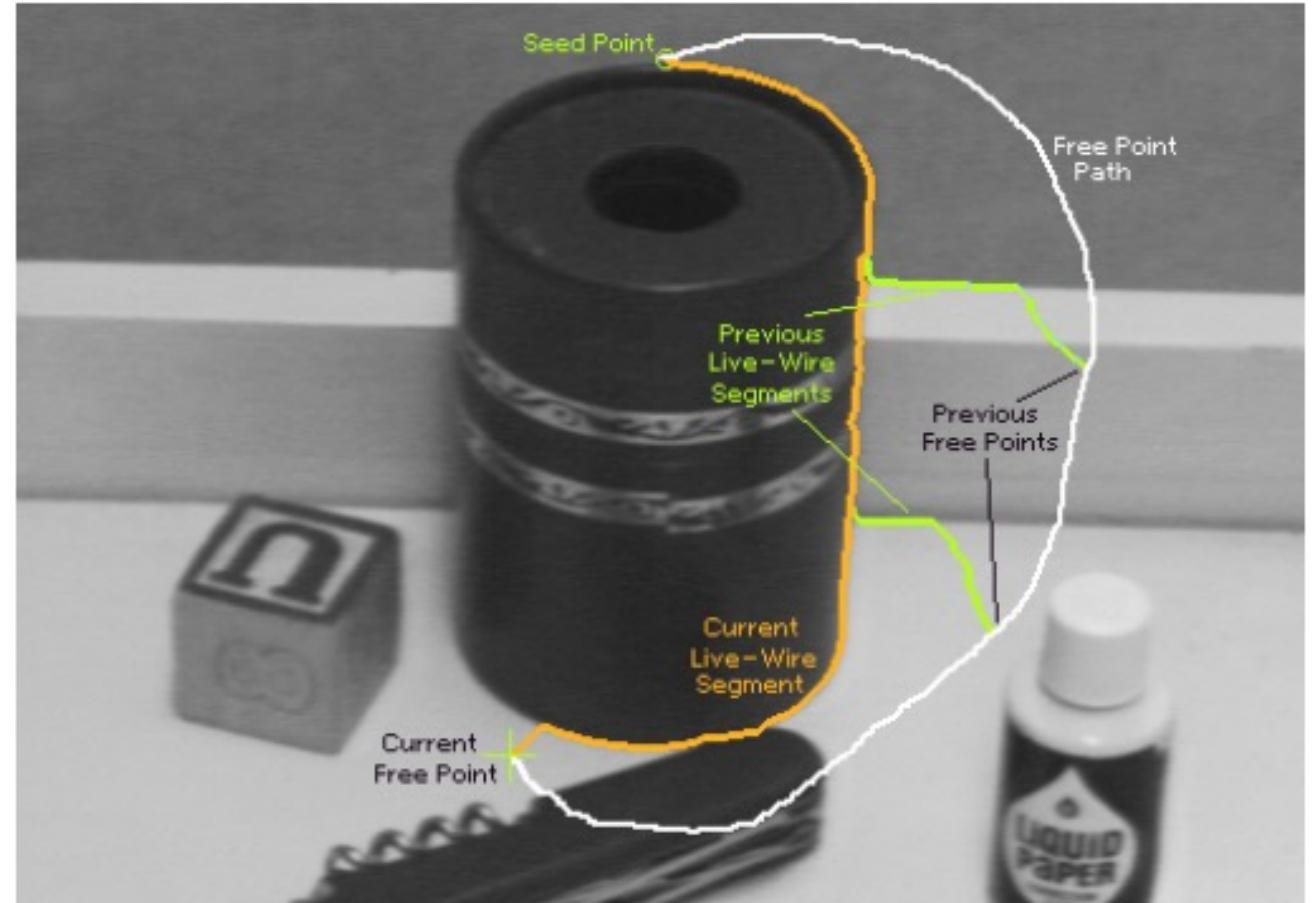
Intelligent scissors

Problem statement:

Given two seed points, find a good boundary connecting them

Challenges:

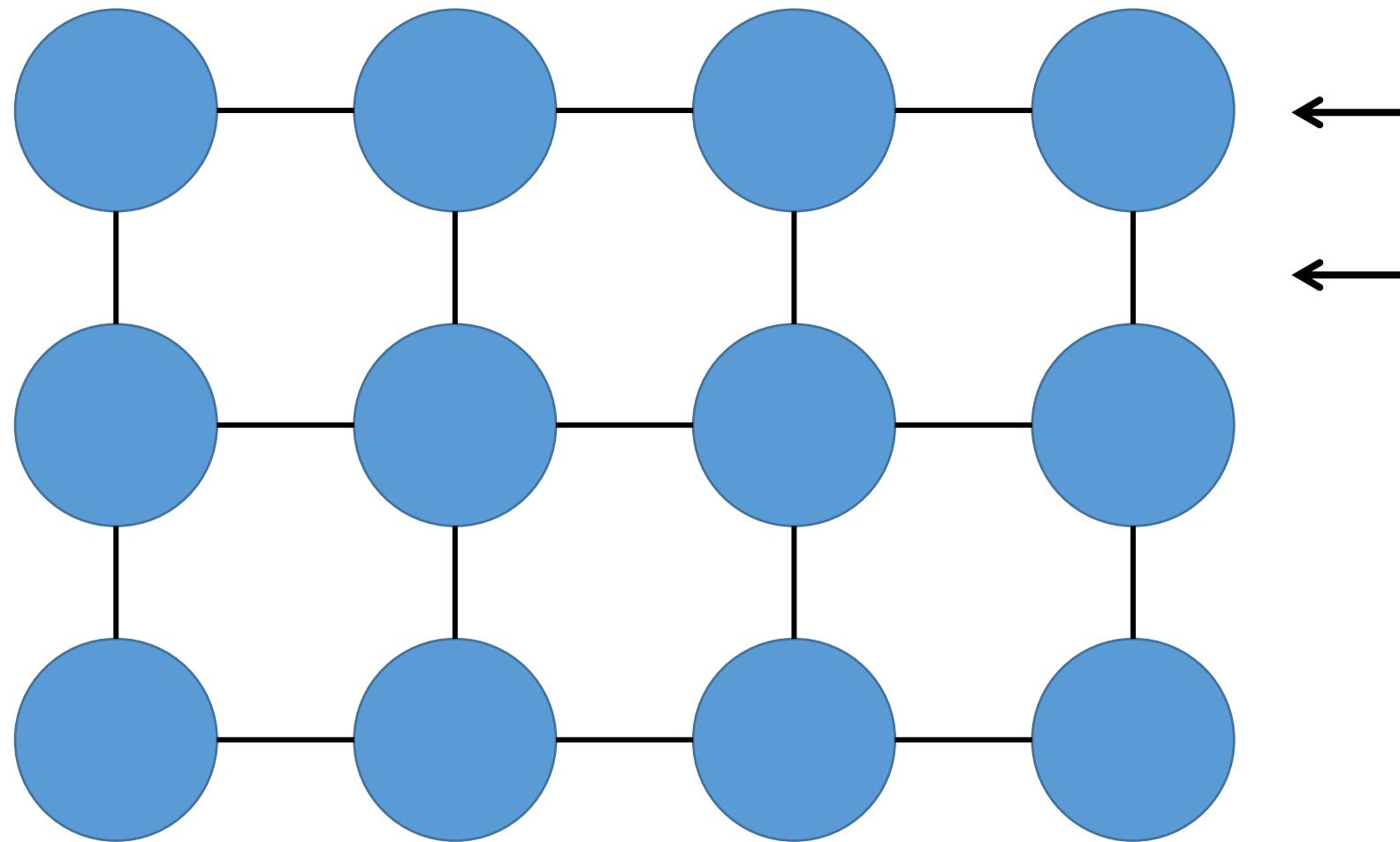
- Make this real-time for interaction
- Define what makes a good boundary



Mortenson and Barrett (SIGGRAPH 1995)
(you can tell it's old from the paper's low quality teaser figure)

Graph-view of this problem

Images can be viewed as graphs

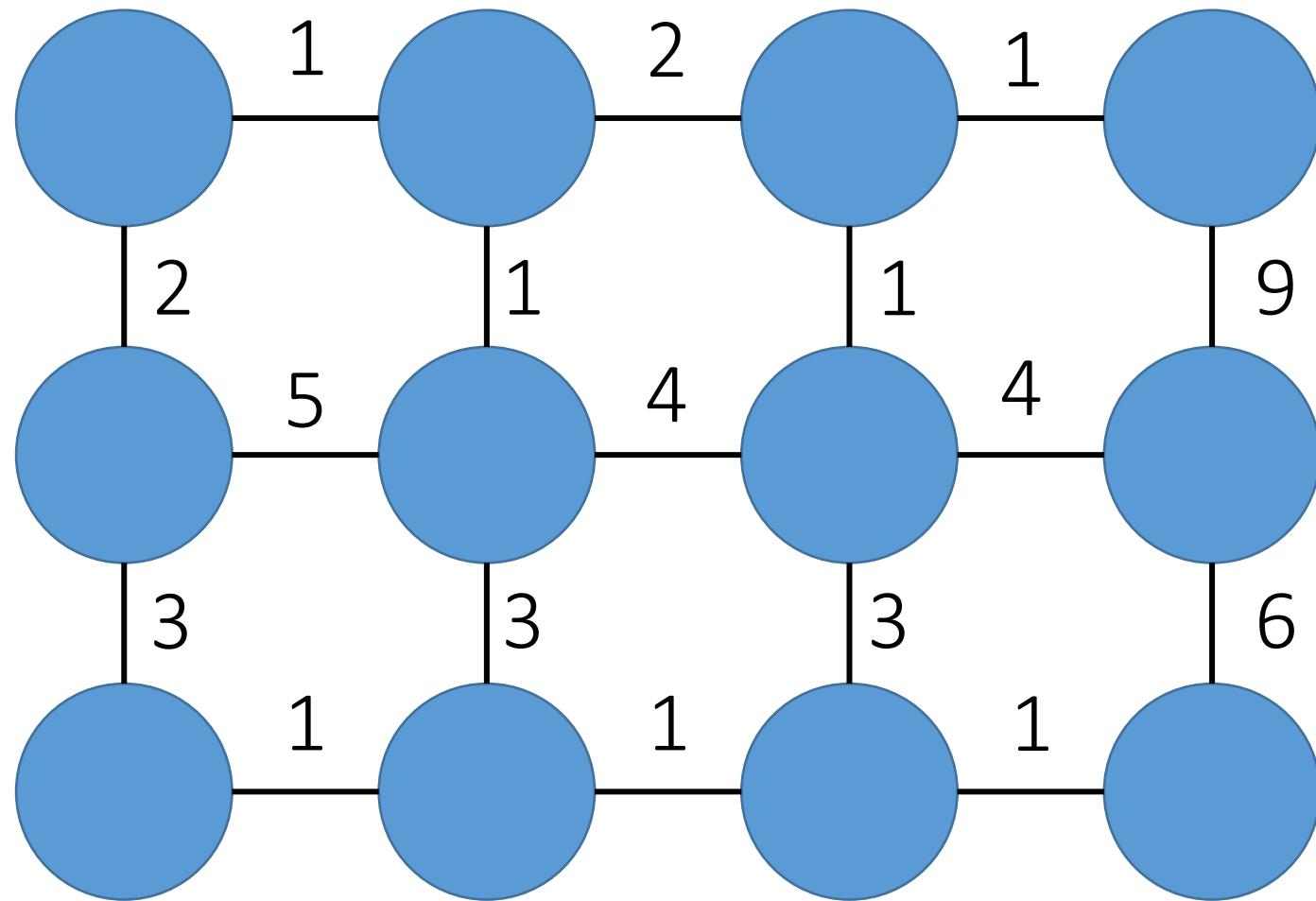


Nodes: pixels

Edges: Constraints between
neighboring pixels

Graph-view of this problem

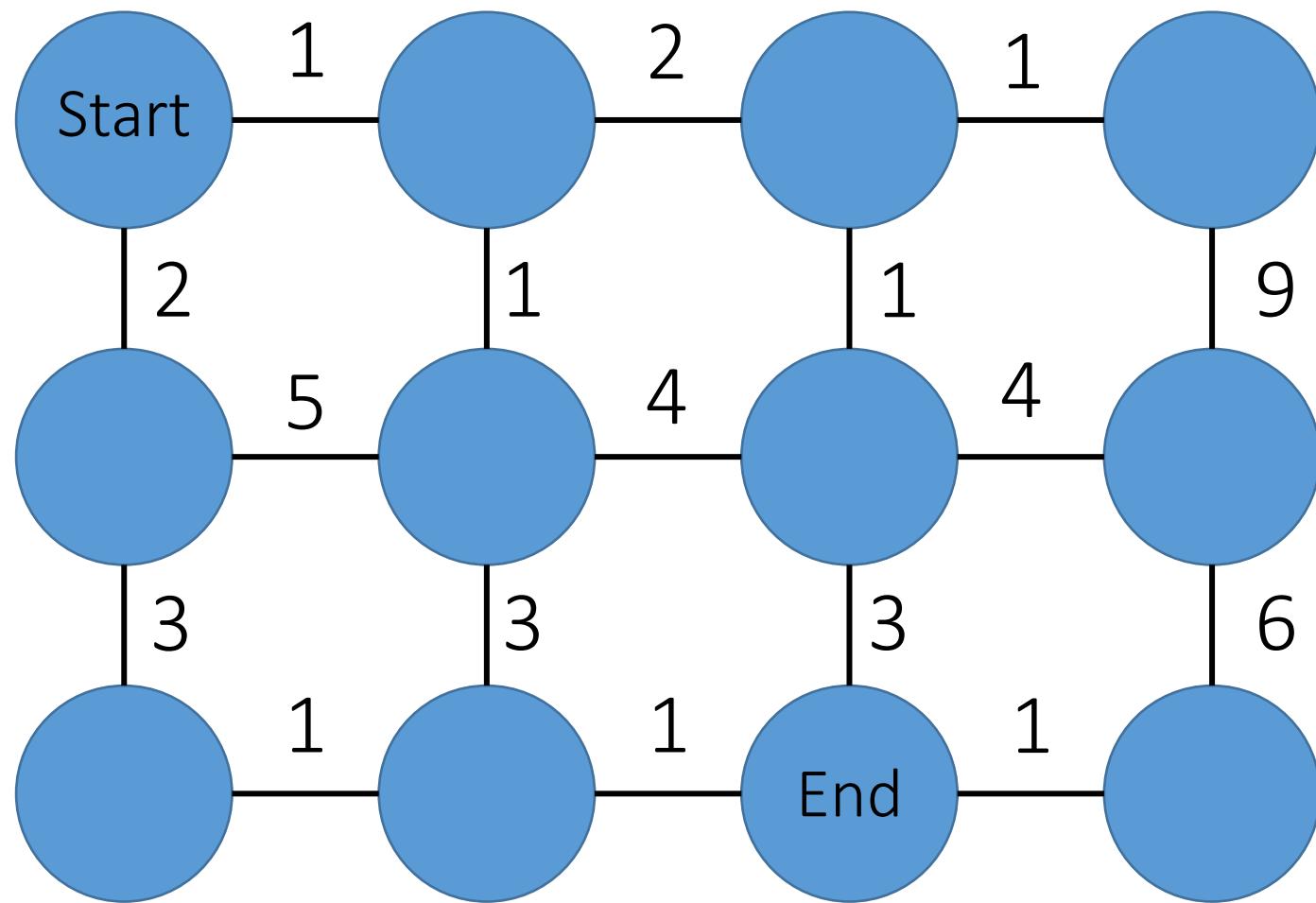
Graph-view of intelligent scissors:



1. Assign weights (costs) to edges

Graph-view of this problem

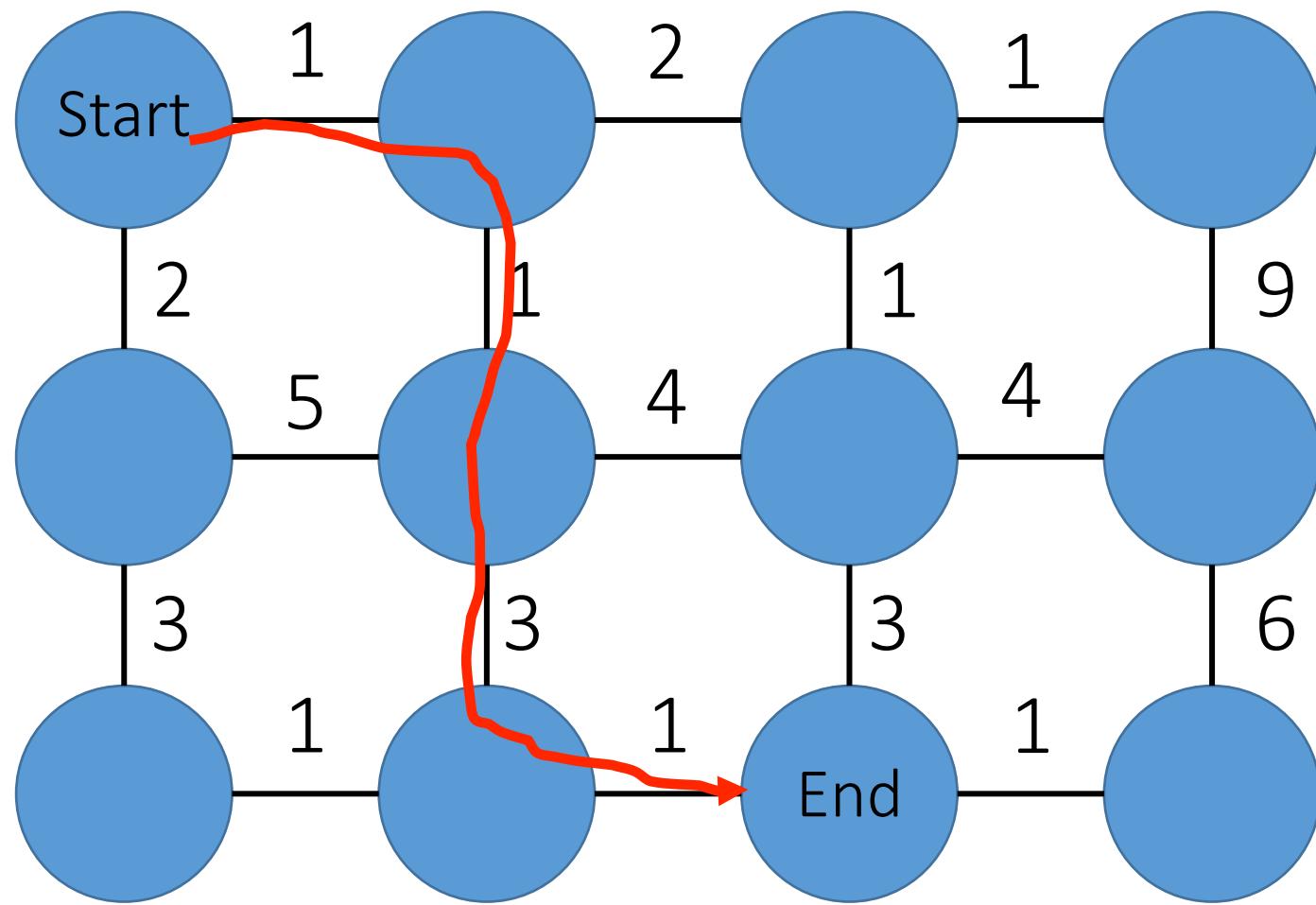
Graph-view of intelligent scissors:



1. Assign weights (costs) to edges
2. Select the seed nodes

Graph-view of this problem

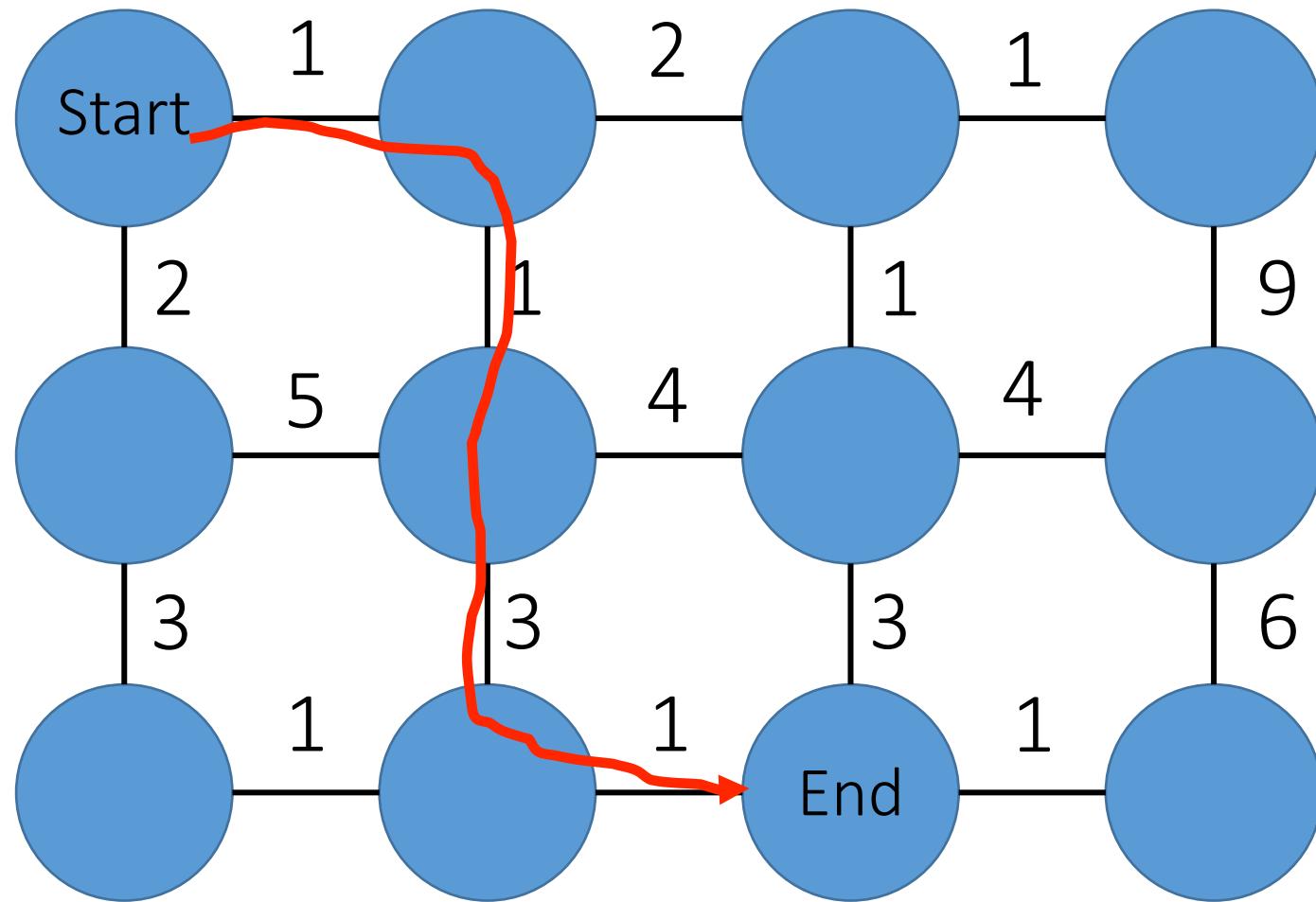
Graph-view of intelligent scissors:



1. Assign weights (costs) to edges
2. Select the seed nodes
3. Find shortest path between them

Graph-view of this problem

Graph-view of intelligent scissors:

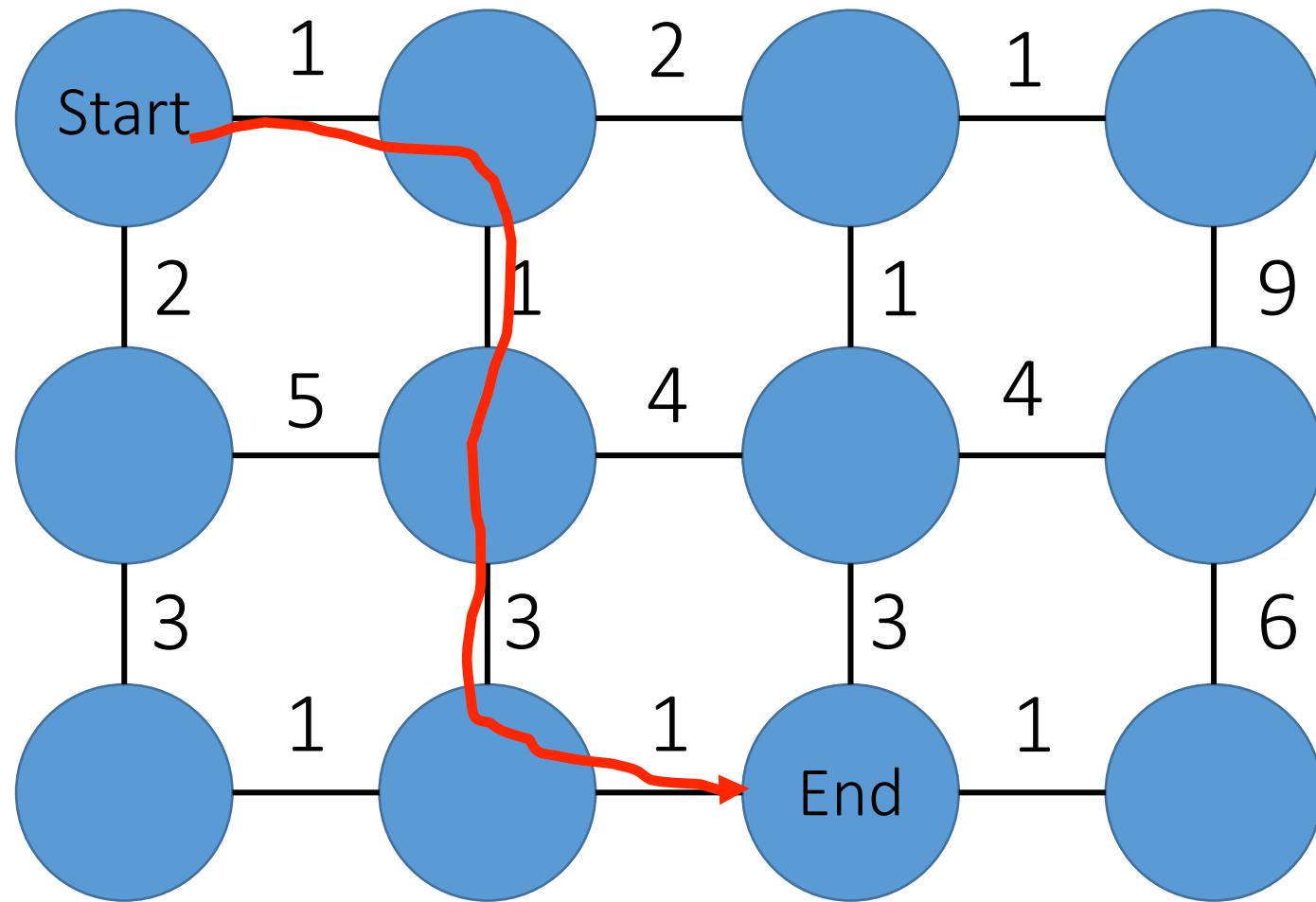


1. Assign weights (costs) to edges
2. Select the seed nodes
3. Find shortest path between them

What algorithm can we use to find the shortest path?

Graph-view of this problem

Graph-view of intelligent scissors:



1. Assign weights (costs) to edges
2. Select the seed nodes
3. Find shortest path between them

What algorithm can we use to find the shortest path?

- Dijkstra's algorithm (dynamic programming)

Dijkstra's shortest path algorithm

Initialize, given seed s (pixel ID) :

- $\text{cost}(s) = 0$ % total cost from seed to this point
- $\text{cost}(!s) = \text{big}$
- $\mathbf{A} = \{\text{all pixels}\}$ % set to be expanded
- $\mathbf{prev}(s) = \text{undefined}$ % pointer to pixel that leads to $q=s$

Precompute $\text{cost}_2(q, r)$ % cost between q to neighboring pixel r

Loop while \mathbf{A} is not empty

1. $q = \text{pixel in } \mathbf{A} \text{ with lowest cost}$

2. Remove q from \mathbf{A}

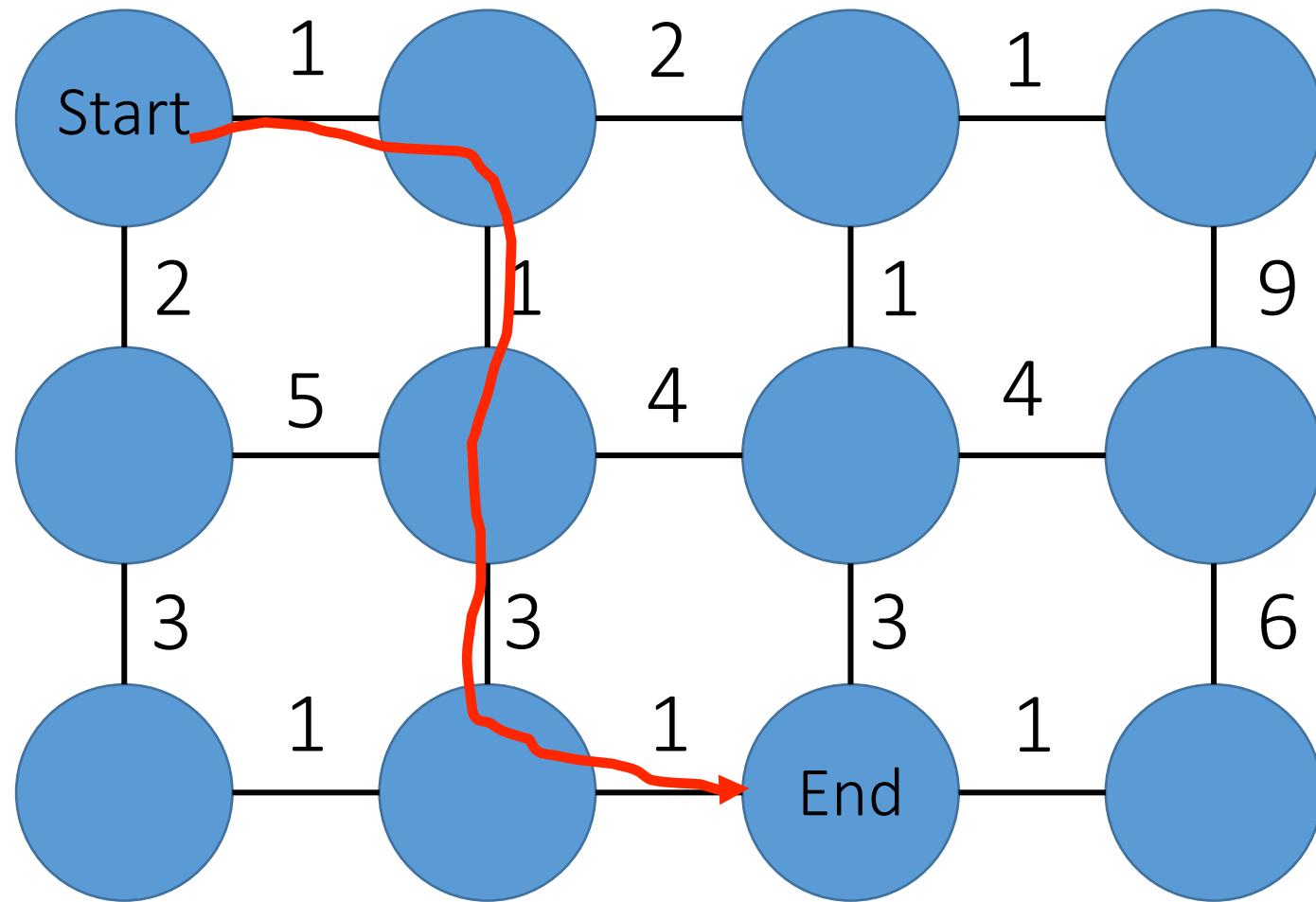
3. For each pixel r in neighborhood of q that is in \mathbf{A}

a) $\text{cost_tmp} = \text{cost}(q) + \text{cost}_2(q, r)$ %this updates the costs

b) if ($\text{cost_tmp} < \text{cost}(r)$)
i. $\text{cost}(r) = \text{cost_tmp}$
ii. $\mathbf{prev}(r) = q$

Graph-view of this problem

Graph-view of intelligent scissors:



1. Assign weights (costs) to edges
2. Select the seed nodes
3. Find shortest path between them

What algorithm can we use to find the shortest path?

- Dijkstra's algorithm (dynamic programming)

How should we select the edge weights to get good boundaries?

Selecting edge weights

Define boundary cost between neighboring pixels:

1. Lower if an image edge is present (e.g., as found by Sobel filtering).
2. Lower if the gradient magnitude at that point is strong.
3. Lower if gradient is similar in boundary direction.



Selecting edge weights

Gradient magnitude



Edge image



Pixel-wise cost

More Advanced Graph-based Segmentations...





The University of Texas at Austin
**Electrical and Computer
Engineering**
Cockrell School of Engineering