SAIR

Spatial AI & Robotics Lab

CSE 473/573-A

W9: CLASSIFICATION & RECOGNITION

Chen Wang

Spatial AI & Robotics Lab

Department of Computer Science and Engineering

University at Buffalo The State University of New York

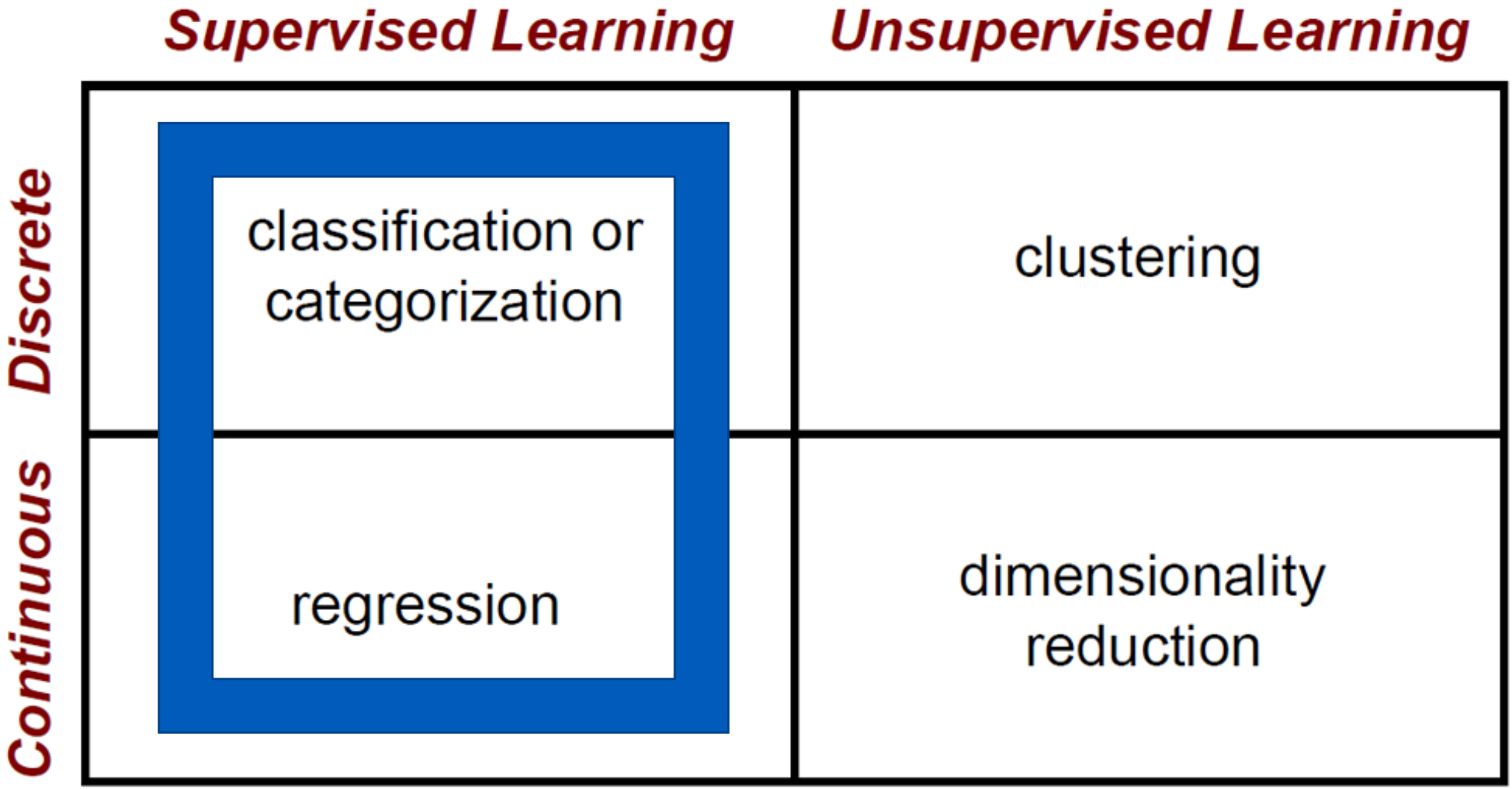
SAIR

Spatial AI & Robotics Lab

COMPUTER VISION

Classification

Machine Learning



The machine learning framework

- Apply a prediction function to a feature representation of the image to get the desired output:

f() = “apple”

f() = “tomato”

f() = “cow”

Slide credit: L. Lazebnik

The machine learning framework

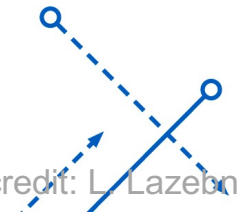
$$y = f(\mathbf{x})$$

output

prediction function

Image feature

- **Training:** given a *training set* of labeled examples $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\}$, estimate the prediction function f by minimizing the prediction error on the training set
- **Testing:** apply f to a never before seen *test example* $\mathbf{x}$ and output the predicted value $y = f(\mathbf{x})$



Slide credit: L. Lazebnik

Image Classification

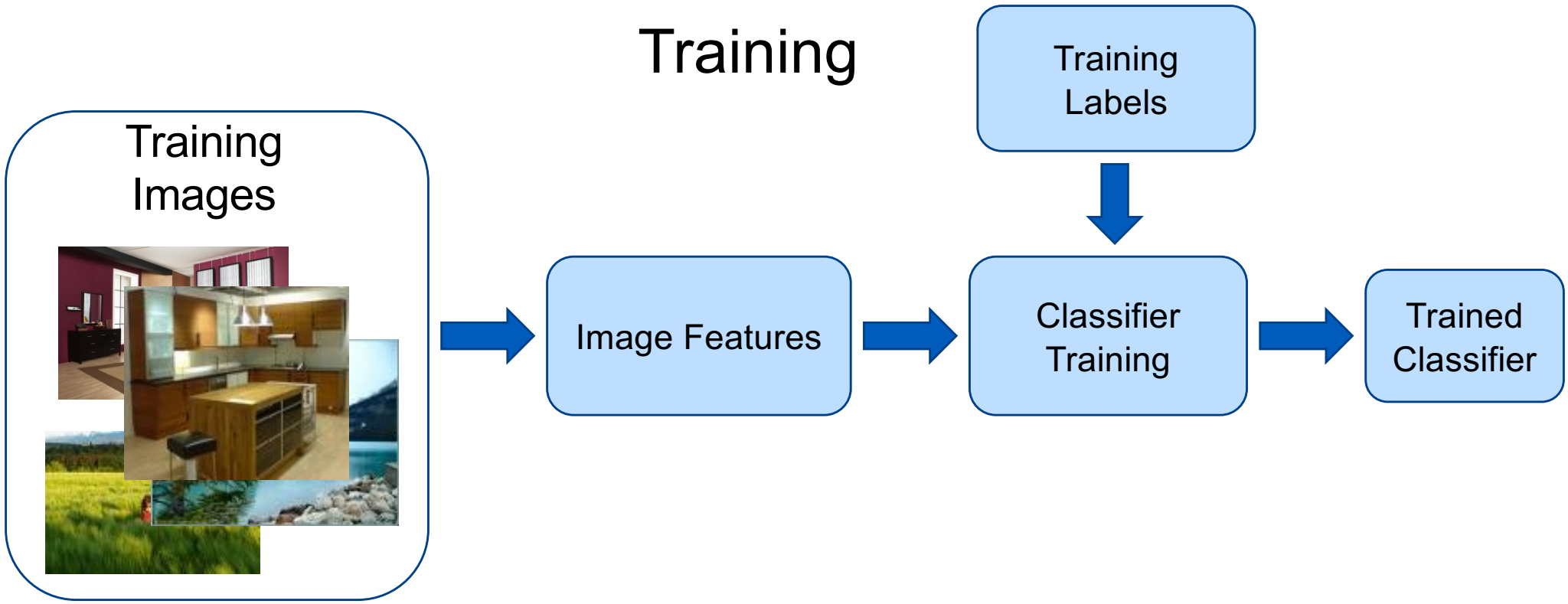
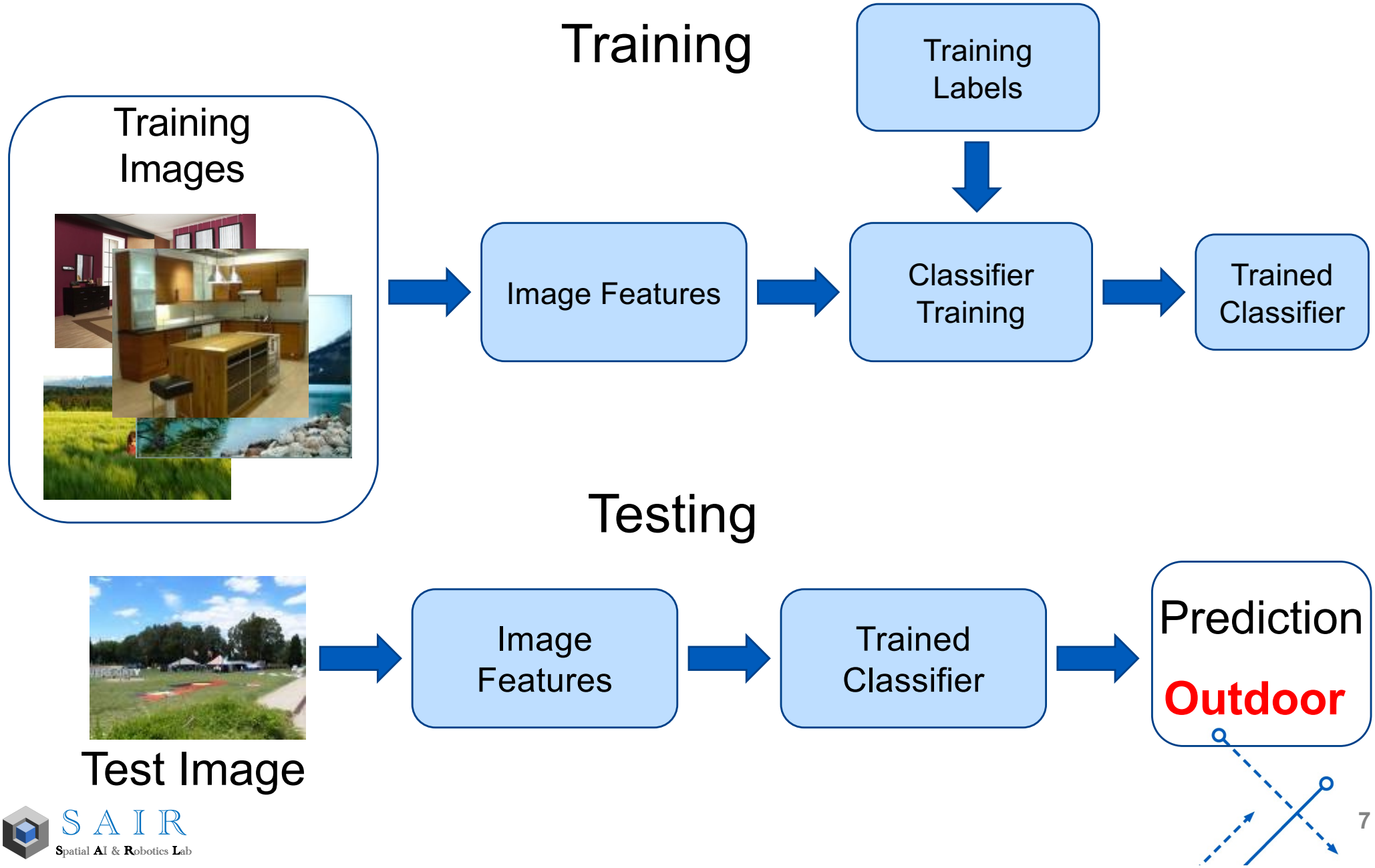


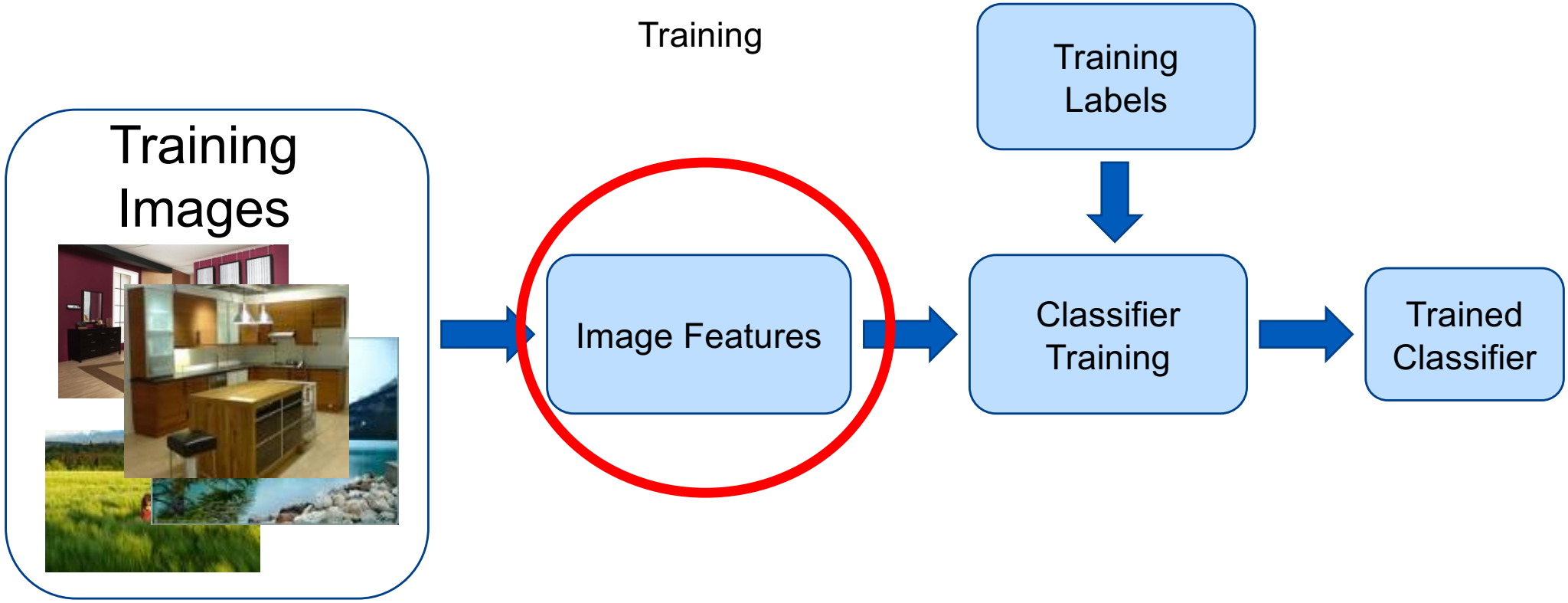
Image Categorization



Example: Scene Classification

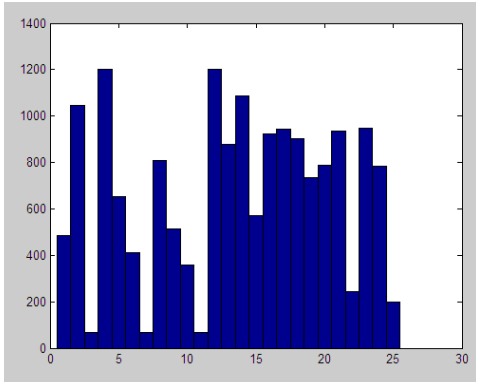
- Is this a kitchen?

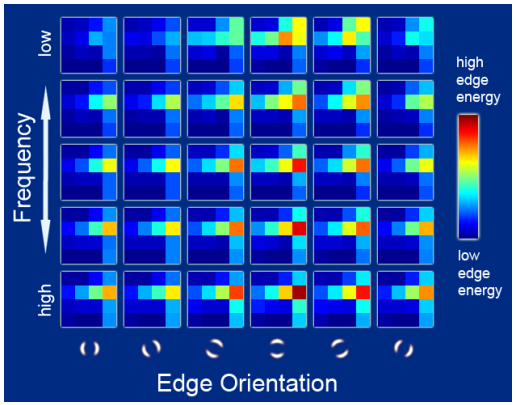
Image features



Types of Features

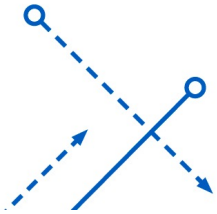
- Raw pixels
- Histograms
- SIFT descriptors
- CNN features
- …



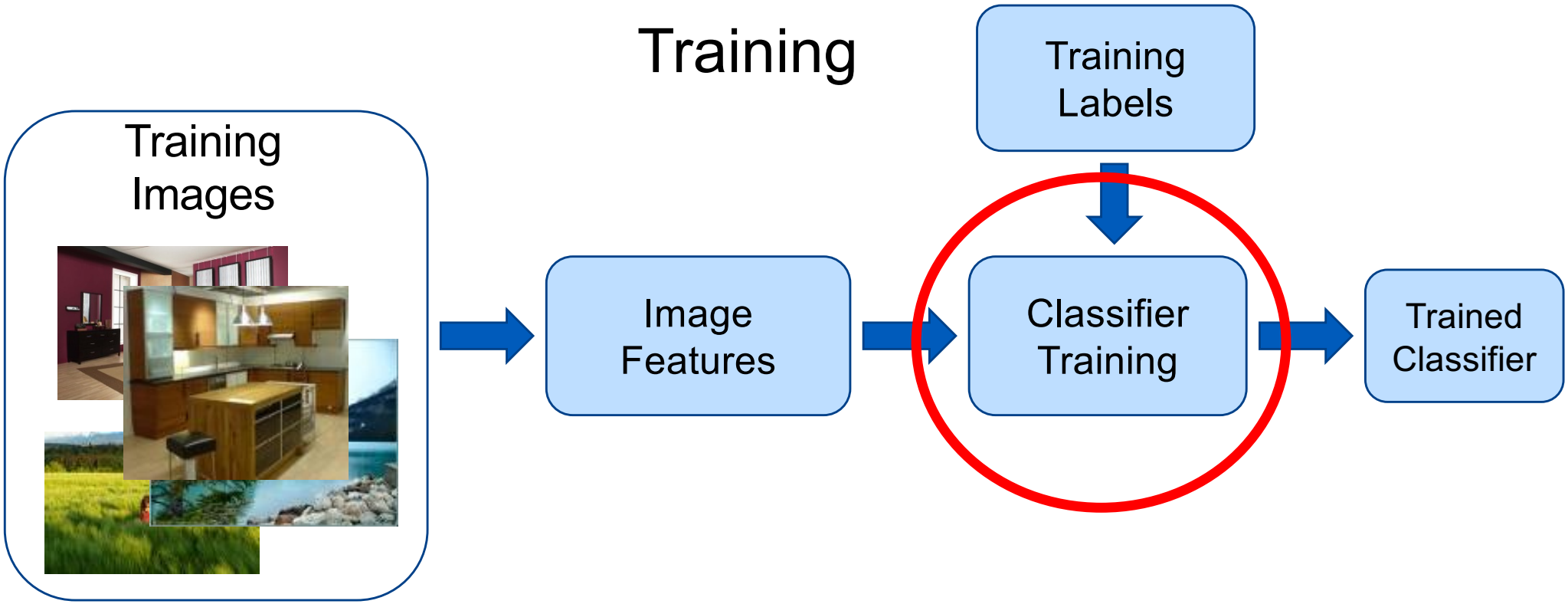


Desirable Properties of Features

- Coverage
  - Ensure that all relevant info is captured
- Concision
  - Minimize number of features without sacrificing coverage
- Directness/Uniqueness
  - Ideal features are independently useful for prediction

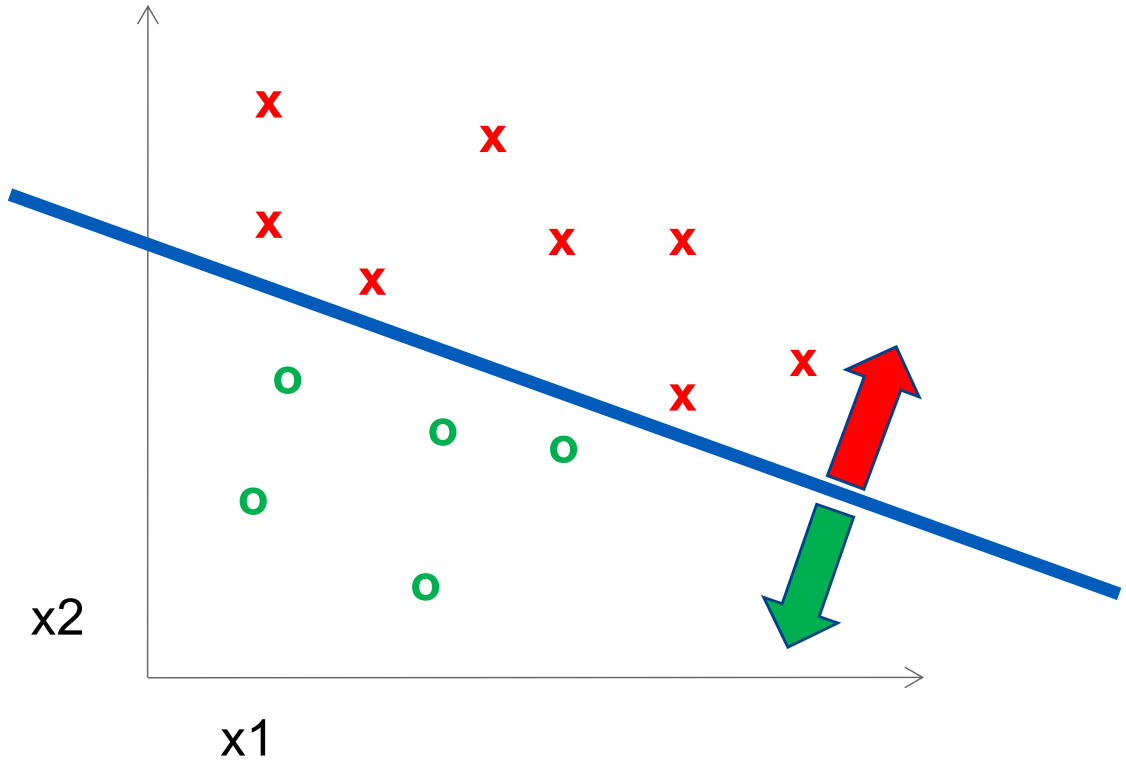


Classifiers



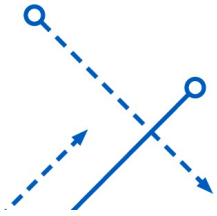
Learning a classifier

Given subset of features with corresponding labels, learn a function to predict the unseen labels.



Desirable properties of classifiers

- Can you list desirable properties of a classifier.
  - Minimize Prediction Error
  - Minimize Training Time
  - Minimize Training Data
  - Nonlinear Separability
  - Makes Hard and Soft Decisions
  - Generalizable
  - Can be Tuned
  - On-line adaptation
  - Non-Parametric



Generalization

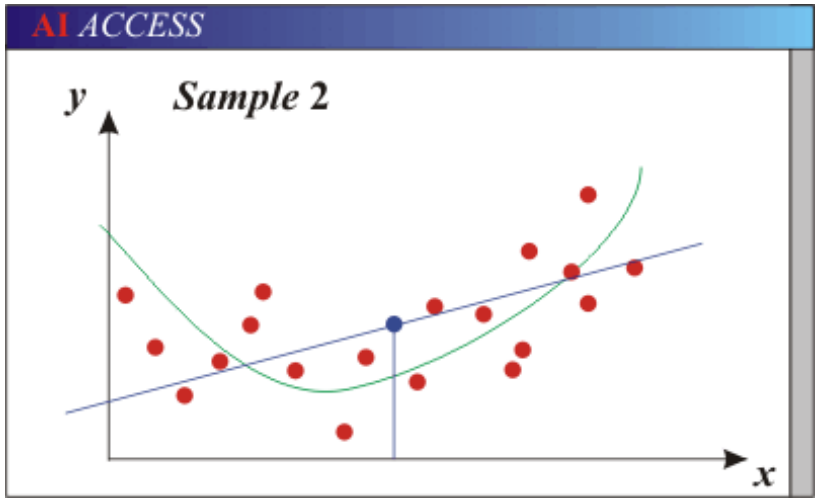
- How well does a learned model generalize from the data it was trained on to a new test set?

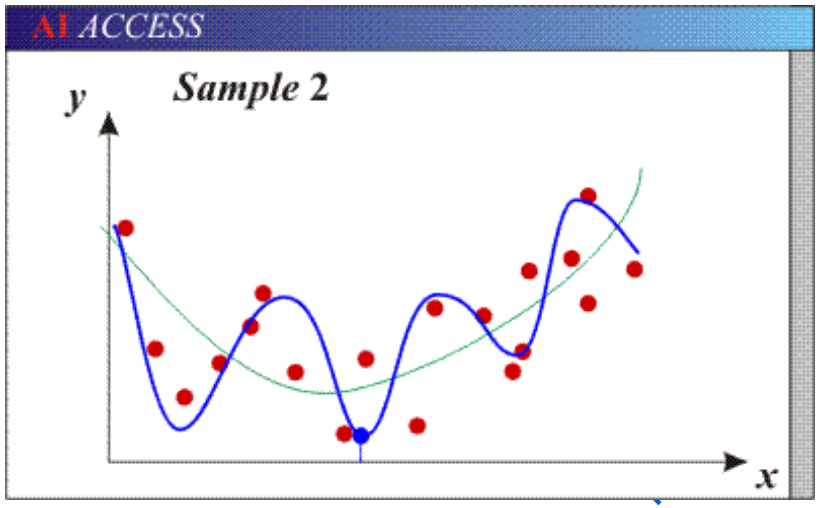
Training set (labels known)

Test set
(labels unknown)

Bias-Variance Trade-off

- No Free Lunch Theorem
- Models with too few parameters are inaccurate because of a large bias (not enough flexibility).
- Models with too many parameters are inaccurate because of a large variance (too much sensitivity to the sample).





Bias-Variance Trade-off

$$E(MSE) = Var\{noise\} + bias^2 + Var\{label\}$$

Inherent / Unavoidable error

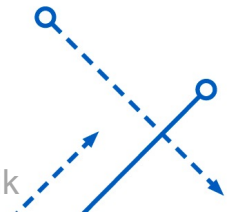
Error due to incorrect assumptions

Error due to variance of training samples

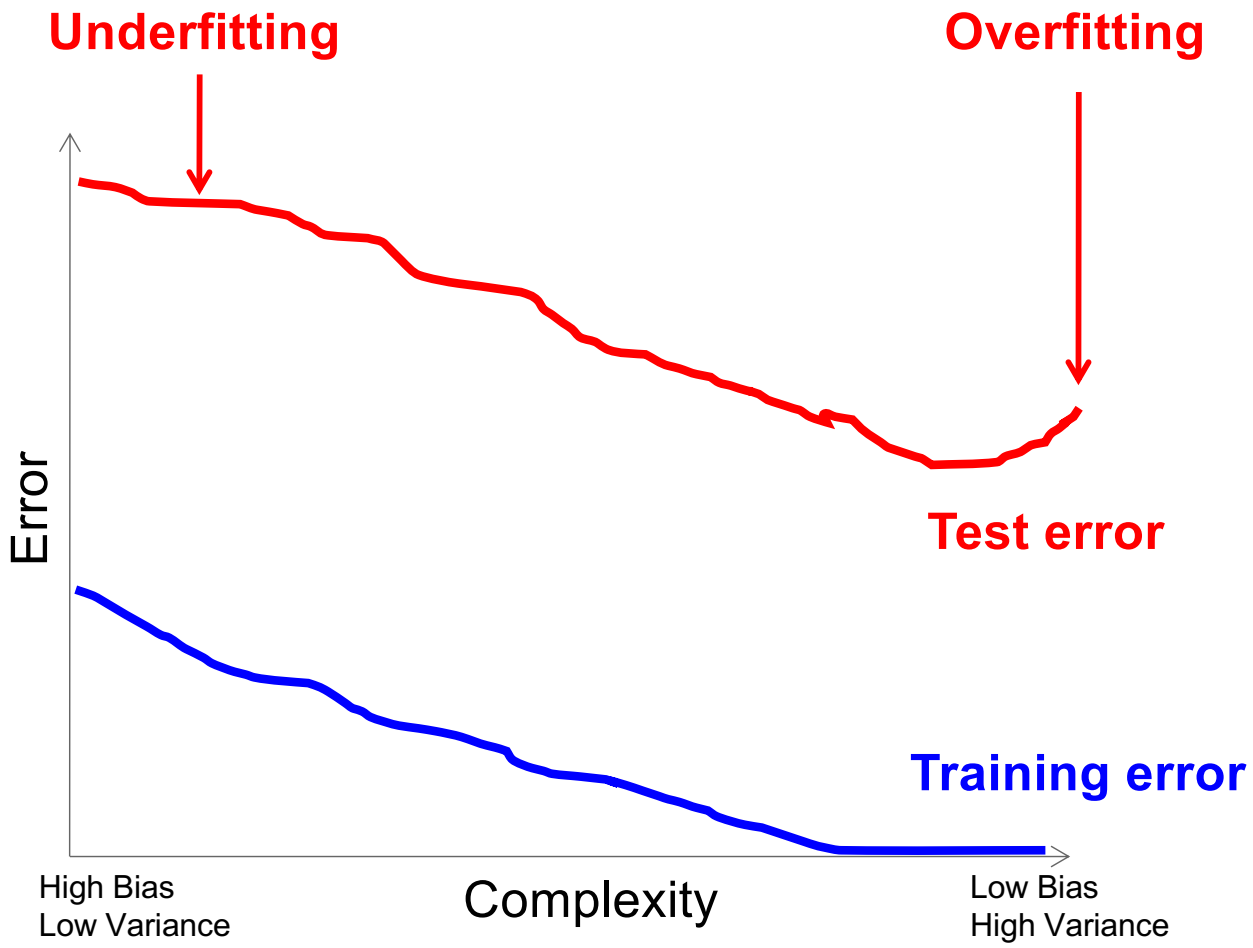
See explanations of bias-variance and Bishop's "Neural Networks" book

Generalization

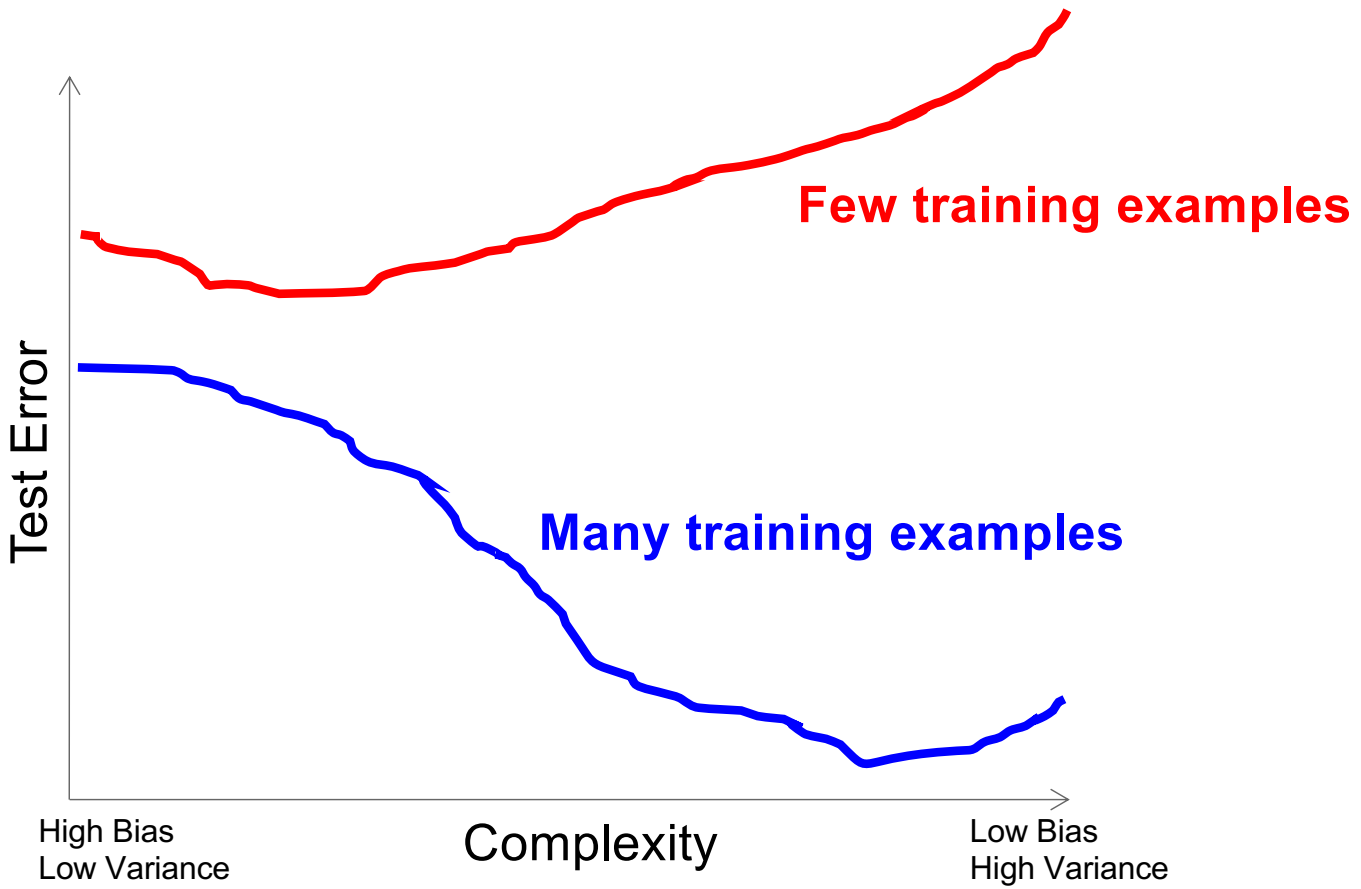
- Components of generalization error
  - **Bias**: how much the average model differs from the true model, over all training sets?
    - Error due to inaccurate assumptions made by the model.
  - **Variance**: how much models estimated from different training sets differ from each other.
  - **Underfitting**: model is too “simple” to represent all the relevant class characteristics
    - High bias (few degrees of freedom) and low variance
    - High training error and high test error
  - **Overfitting**: model is too “complex” and fits irrelevant characteristics (noise) in the data
    - Low bias (many degrees of freedom) and high variance
    - Low training error and high test error



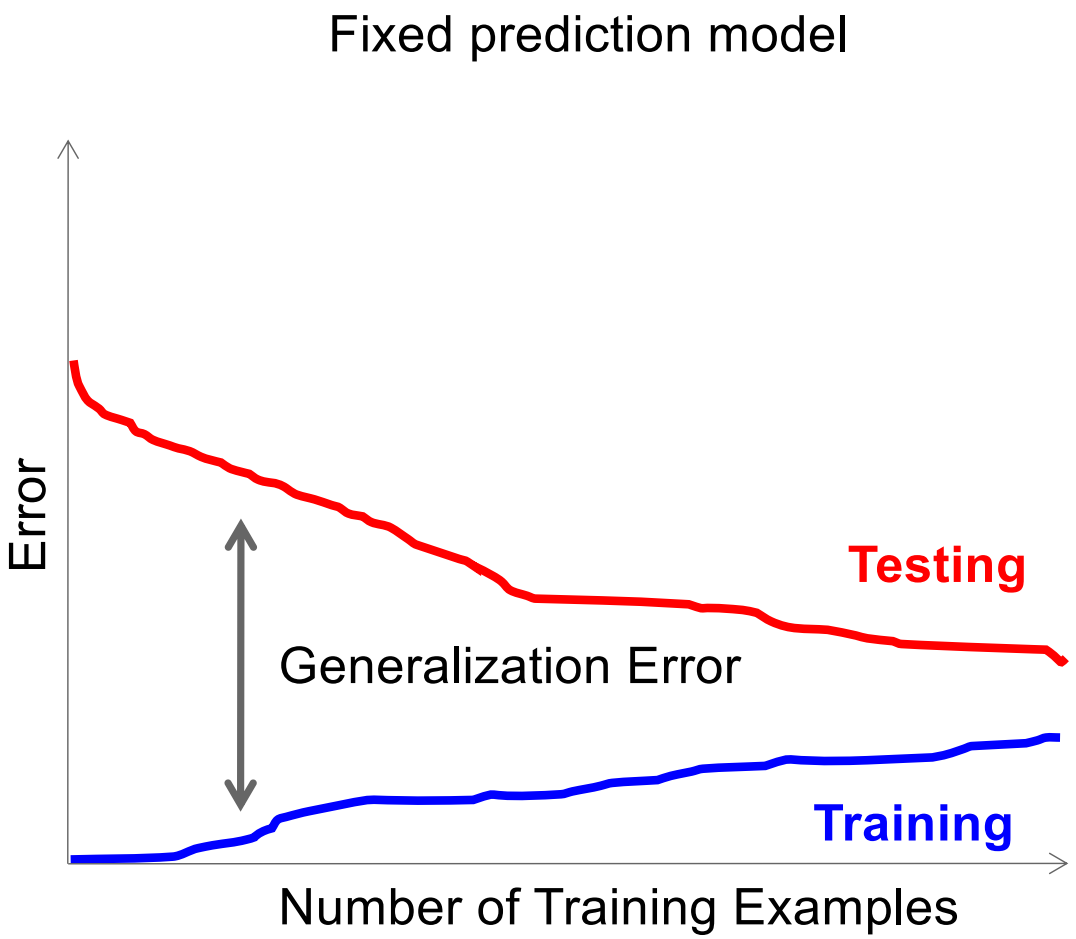
Bias-variance tradeoff



Bias-variance tradeoff

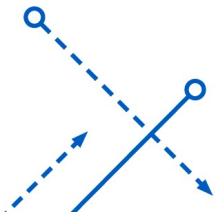


Effect of Training Size



The perfect classification algorithm

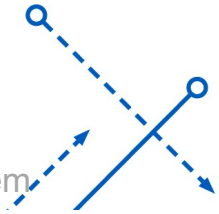
- Objective function: encodes the right loss for the problem
- Parameterization: makes assumptions that fit the problem
- Regularization: right level of regularization for amount of training data
- Training algorithm: can find parameters that maximize objective on training set
- Inference algorithm: can solve for objective function in evaluation



Things to Remember

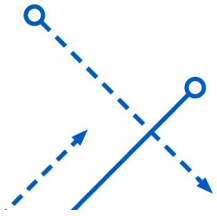
- No classifier is inherently better than any other
  - You need to make assumptions to generalize

- Three kinds of error
  - Inherent: unavoidable
  - Bias: due to inaccurate model
  - Variance: due to inability to perfectly estimate parameters from limited data



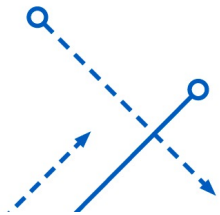
Some classifiers

- **K-nearest neighbor**
- **Support Vector Machine (SVM)**
- Boosted Decision Trees
- Neural networks
- Naïve Bayes
- Bayesian network
- Logistic regression
- Randomized Forests
- …



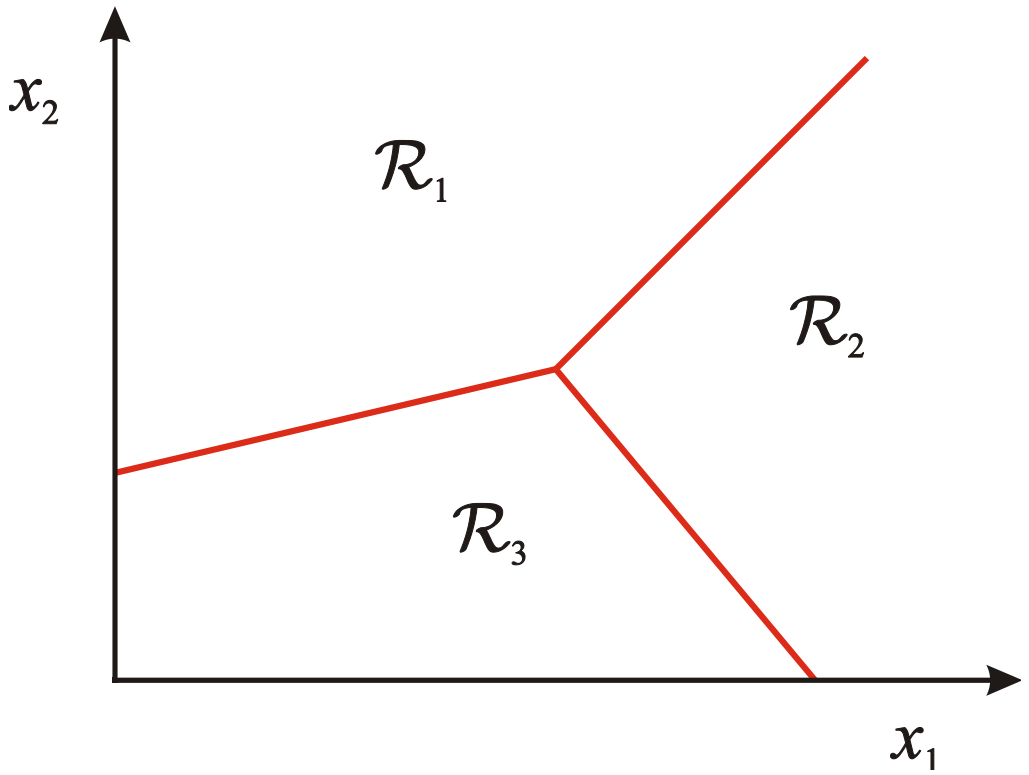
Generative vs. Discriminative Classifiers

- Generative Models
  - Represent both the data and the labels
  - Often, makes use of conditional independence and priors
  - Examples
    - Naïve Bayes classifier
    - Bayesian network
  - Models of data may apply to future prediction problems
- Discriminative Models
  - Learn to directly predict the labels from the data
  - Often, assume a simple boundary (e.g., linear)
  - Examples
    - Logistic regression
    - SVM
    - Boosted decision trees
  - Often easier to predict a label from the data than to model the data



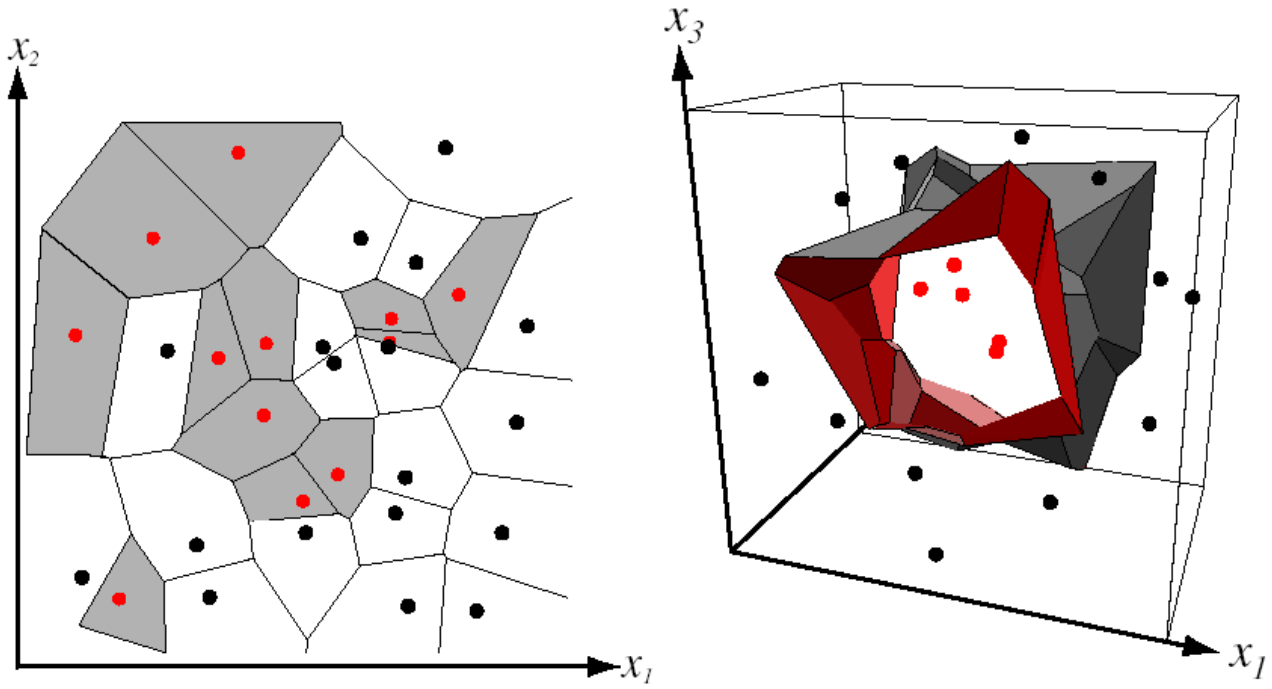
Classification

- Assign input vectors to one of two (more) classes
- Any **decision rule** divides input space into ***decision regions*** separated by ***decision boundaries***



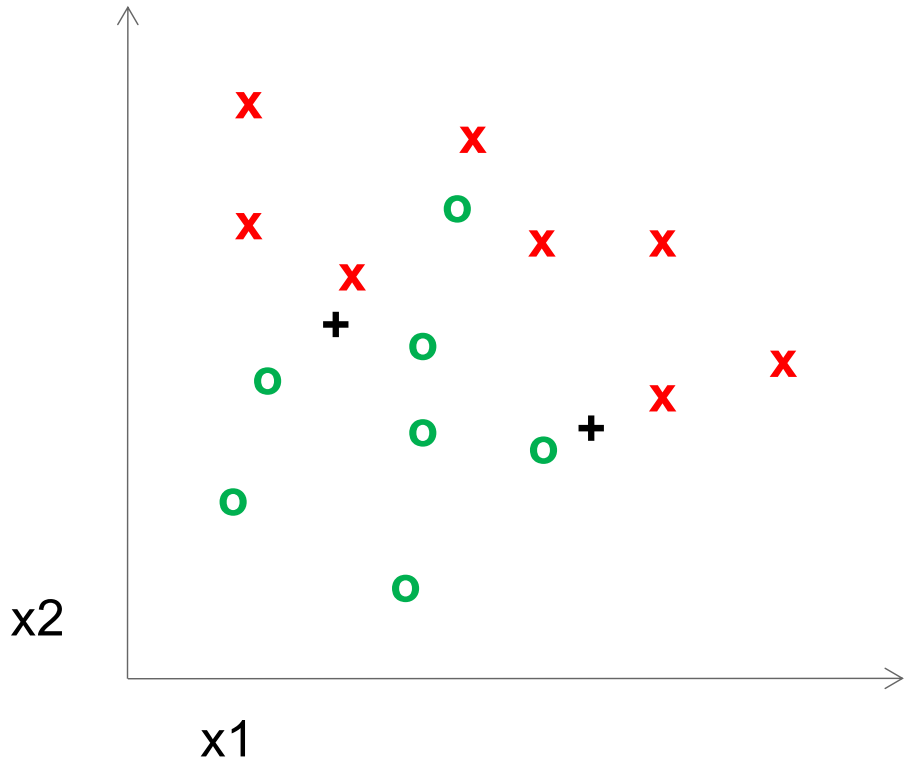
Nearest Neighbor Classifier

- Assign label of nearest training data point to each test data point

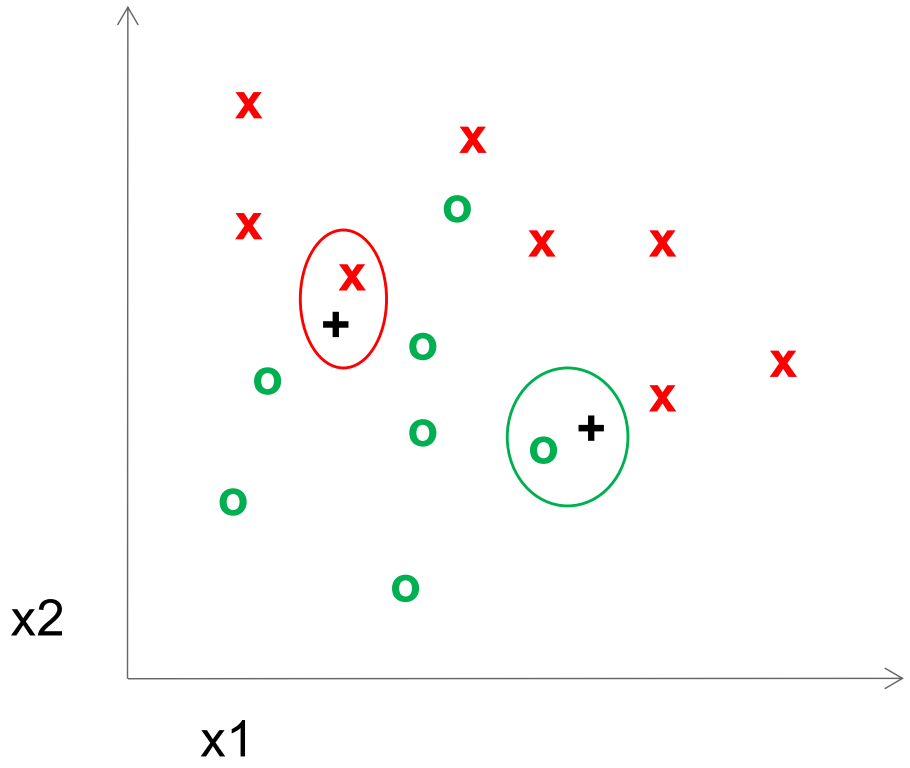


Voronoi partitioning of feature space
for two-category 2D and 3D data

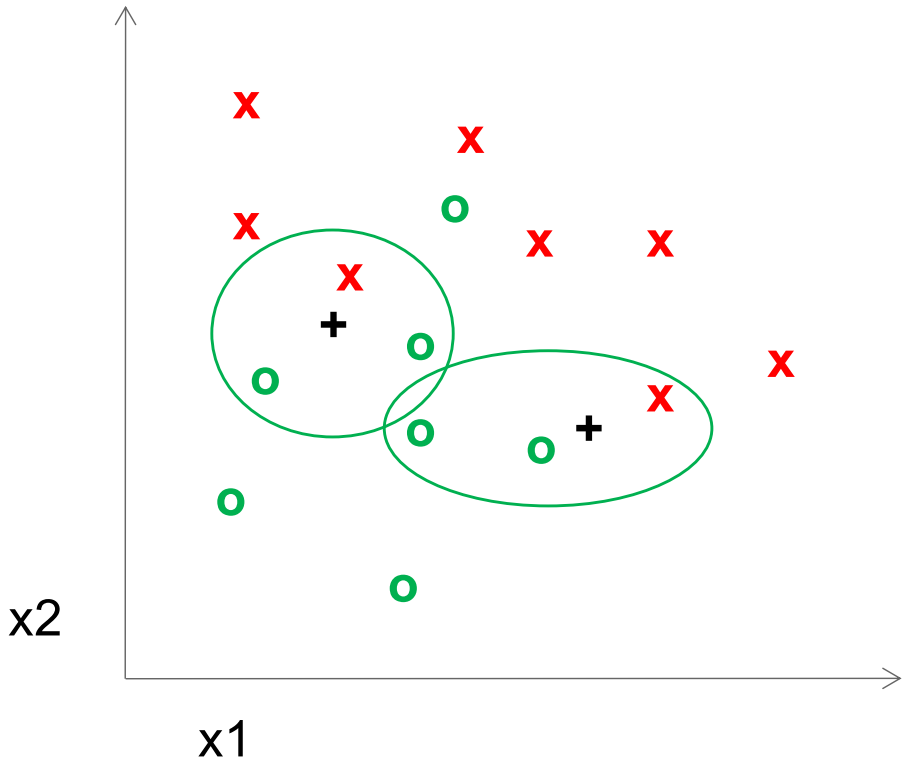
K-nearest neighbor



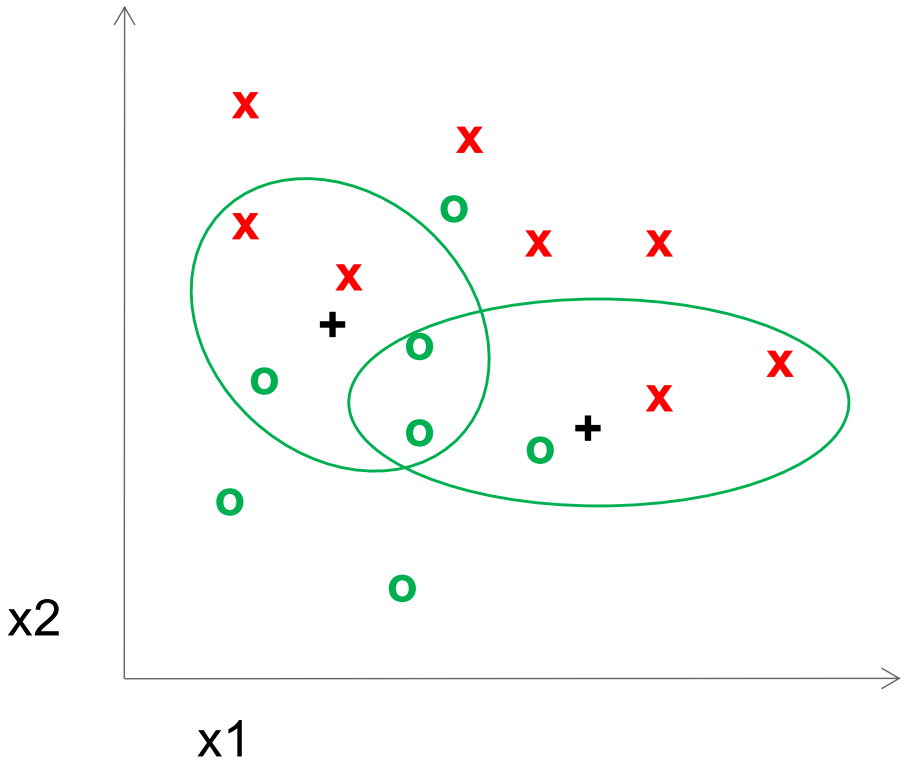
1-nearest neighbor



3-nearest neighbor



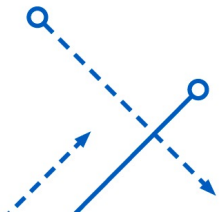
5-nearest neighbor



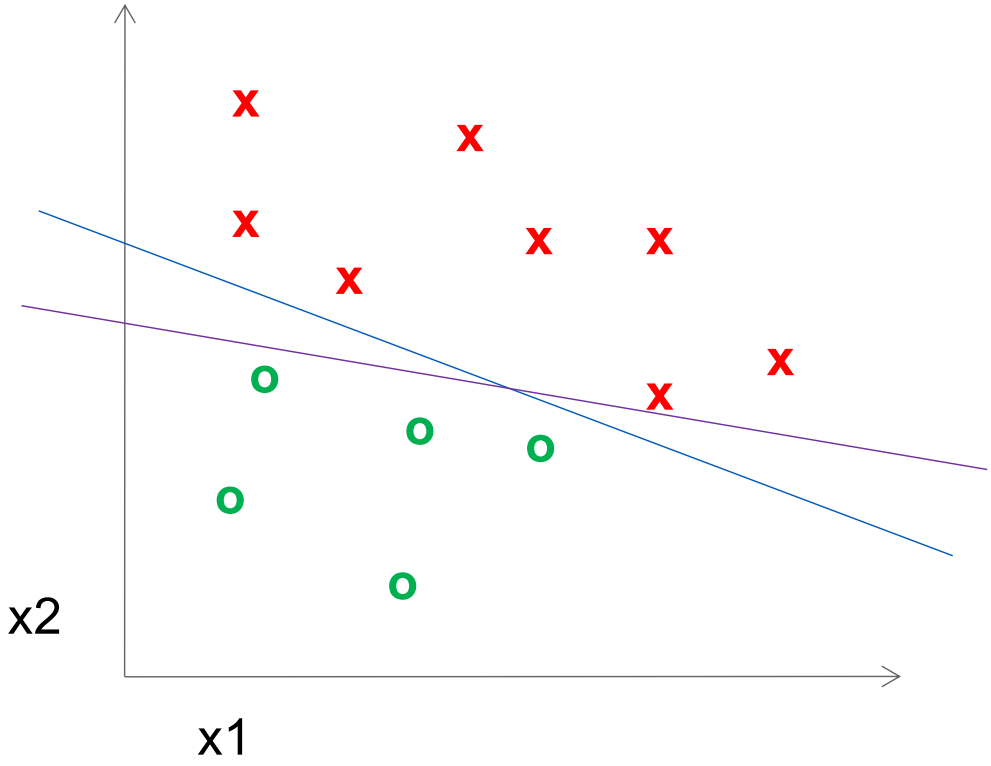
Using K-NN

- Simple, a good one to try first

- With infinite examples, 1-NN provably has error that is at most twice Bayes optimal error

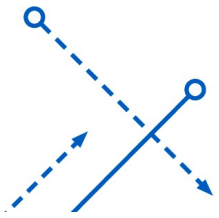


Linear Support Vector Machine (SVM)

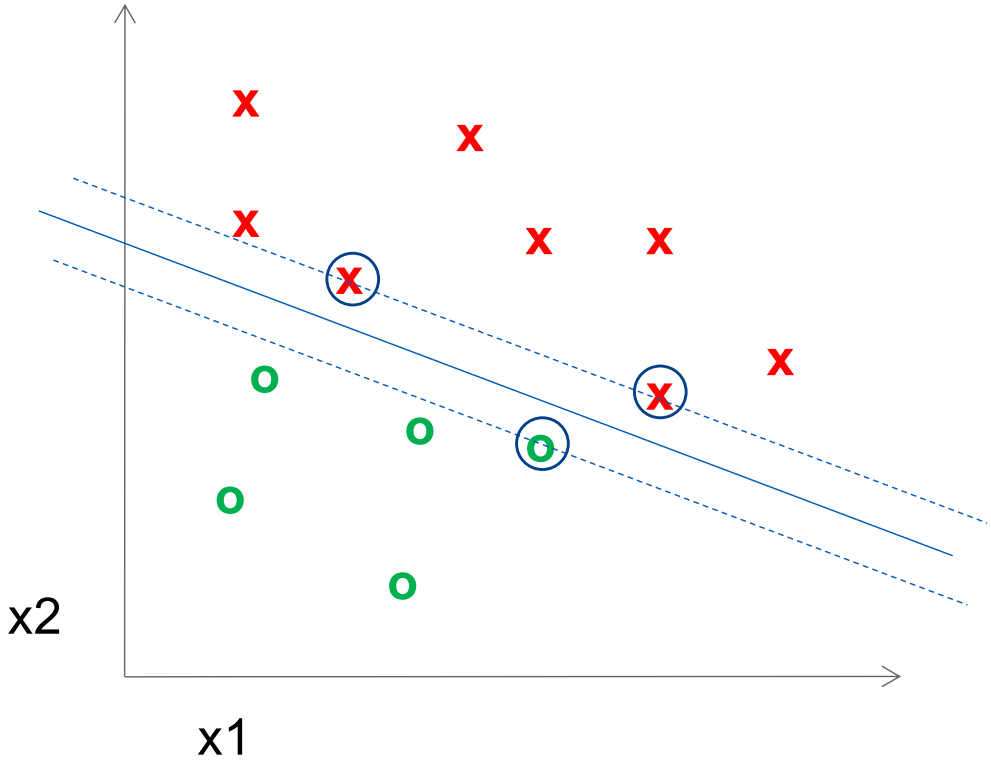


- Find a *linear function* to separate the classes:

$$f(\mathbf{x}) = \text{sgn}(\mathbf{w} \cdot \mathbf{x} + b)$$

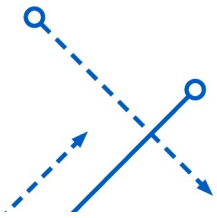


Linear Support Vector Machine (SVM)

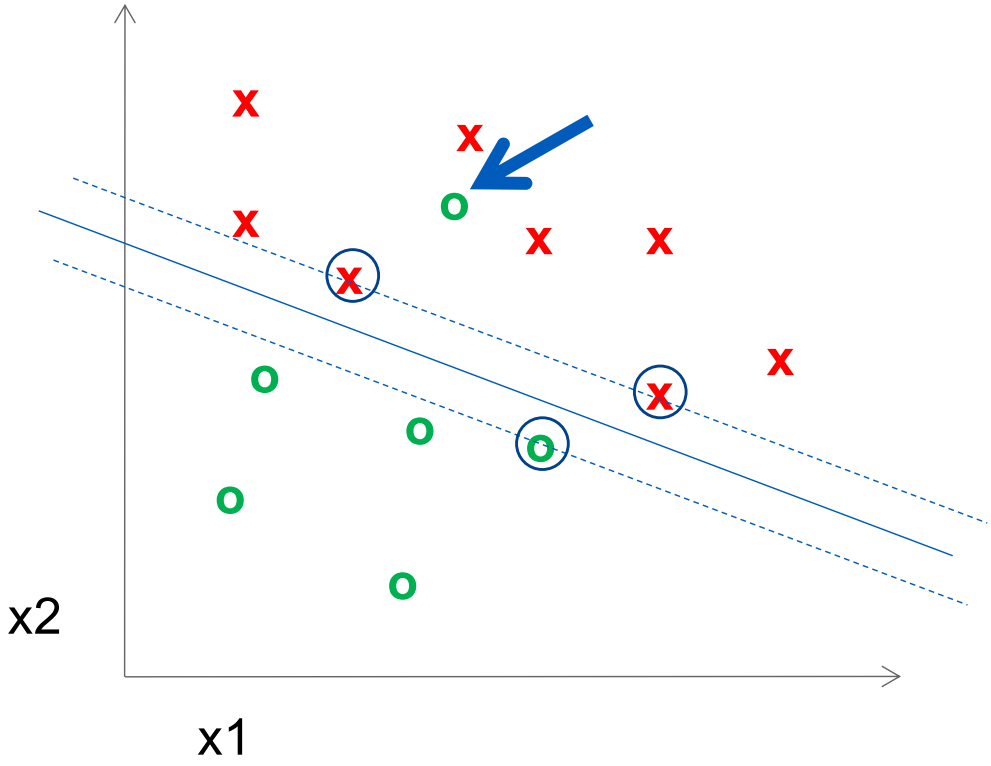


- Find a *linear function* to separate the classes:

$$f(\mathbf{x}) = \text{sgn}(\mathbf{w} \cdot \mathbf{x} + b)$$



Linear Support Vector Machine (SVM)

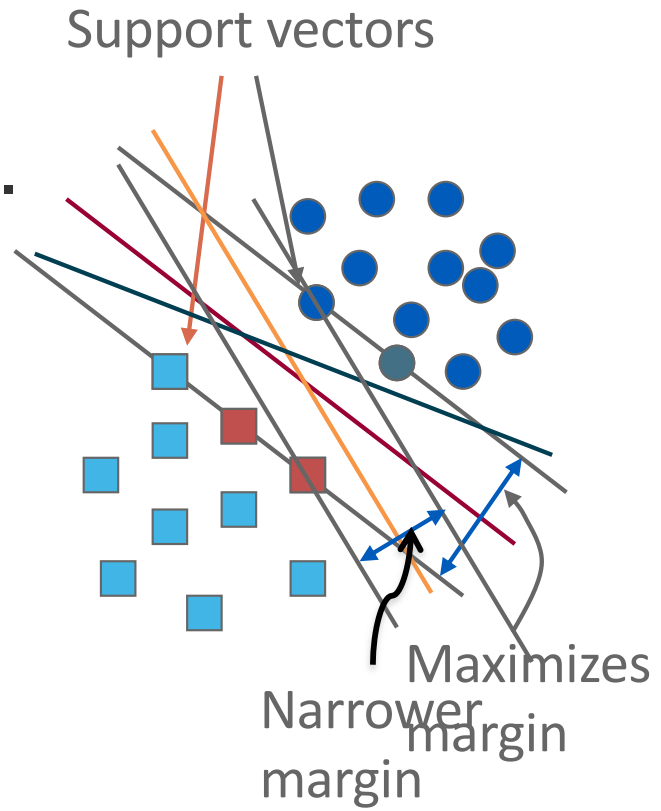


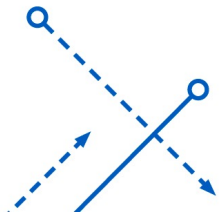
- Find a *linear function* to separate the classes:

$$f(\mathbf{x}) = \text{sgn}(\mathbf{w} \cdot \mathbf{x} + b)$$

Support Vector Machine (SVM)

- SVMs maximize the margin around the separating hyperplane.
  - A.k.a, large margin classifiers
- The decision function is fully specified by a subset of training samples, the support vectors.





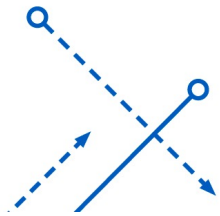
Soft SVM

- Cost Function (Hinge loss)

$$c(x, y, f(x)) = \begin{cases} 0, & \text{if } y * f(x) \geq 1 \\ 1 - y * f(x), & \text{else} \end{cases}$$

$$= \max\{0, 1 - y * f(x)\}$$

$$\stackrel{\text{def}}{=} \left(1 - y * f(x)\right)_+$$

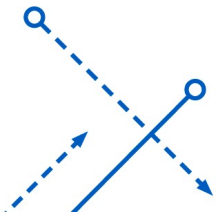


Linear Soft SVM

- Assume sample x is in a higher dimension.
  - The linear model is $f(x) = \langle x, w \rangle$.
- The loss function with a regularization term.

$$min_w \lambda \parallel w \parallel^2 + \sum_{i=1}^{n} (1 - y_i \langle x_i, w \rangle)_+$$

- Can be solved via gradient descent.



Stochastic Gradient Descent (SGD)

- Given loss function, we calculate the gradient (partial derivative) with respective the model weights.

$$\frac{\delta}{\delta w_k}\lambda \parallel w \parallel^2 = 2\lambda w_k$$

$$\frac{\delta}{\delta w_k}\left(1 - y_i\langle x_i, w\rangle\right)_+ = \begin{cases} 0, & \text{if } y_i\langle x_i, w\rangle \geq 1 \\ -y_i x_{ik}, & \text{else} \end{cases}$$

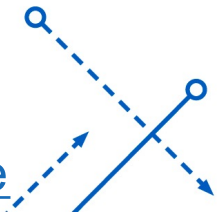
- We update the weights by gradient descent.

$$w = w - \alpha \cdot (2\lambda w)$$

Gradient Update — No misclassification

$$w = w + \alpha \cdot (y_i \cdot x_i - 2\lambda w)$$

Gradient Update — Misclassification



source

SGD for solving linear SVM

- Assume there is no regularization, during each iteration
  - Set the batch size T and do:

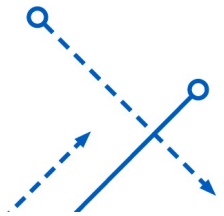
parameter: T;
initialization: $\theta^{(1)} = \mathbf{0}$;
for $t = 1, \cdots, T$ **do**
  $\mathbf{w}^{(t)} = \frac{1}{\lambda t}\theta^{(t)}$;
  Choose i randomly from $[N]$;
  if $y_i \left\langle \mathbf{w}^{(t)}, \mathbf{x}_i \right\rangle \geqslant 1$ **then**
    $\theta^{(t+1)} = \theta^{(t)}$;
  else
    $\theta^{(t+1)} = \theta^{(t)} + y_i \mathbf{x}_i$;
Result: $\bar{\mathbf{w}} = \frac{1}{T} \sum_{t=1}^{T} \mathbf{w}^{(t)}$

Number of sample in each iteration (batch)

Record weight gradient update from each sample

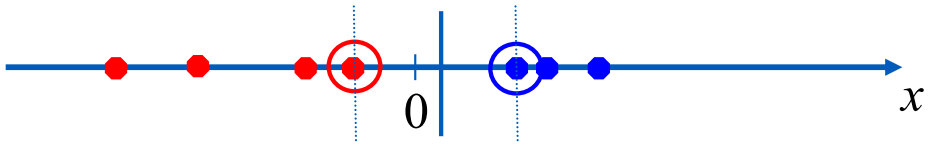
Averaged updated weights in the batch

- Update the solution $w = w + \bar{w}$

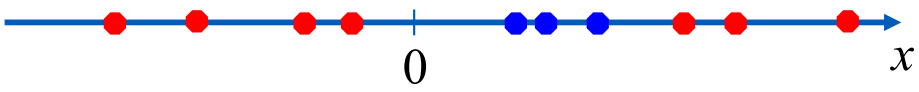


Nonlinear SVMs

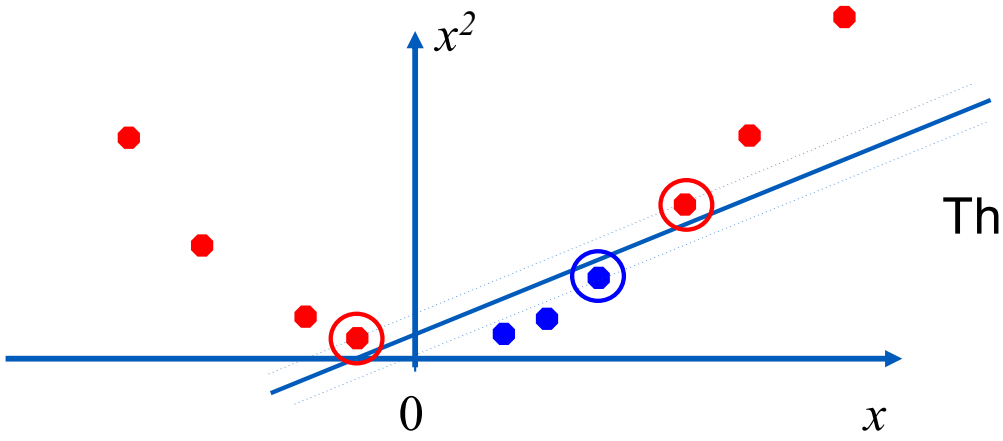
- Datasets that are linearly separable work out great:



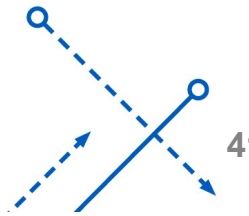
- But what if the dataset is NOT linearly separable?



- We can map it to a higher-dimensional space:



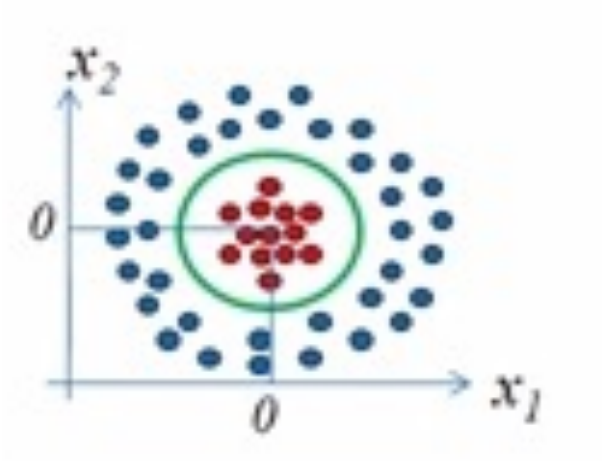
The new feature: $[x, x^2]$

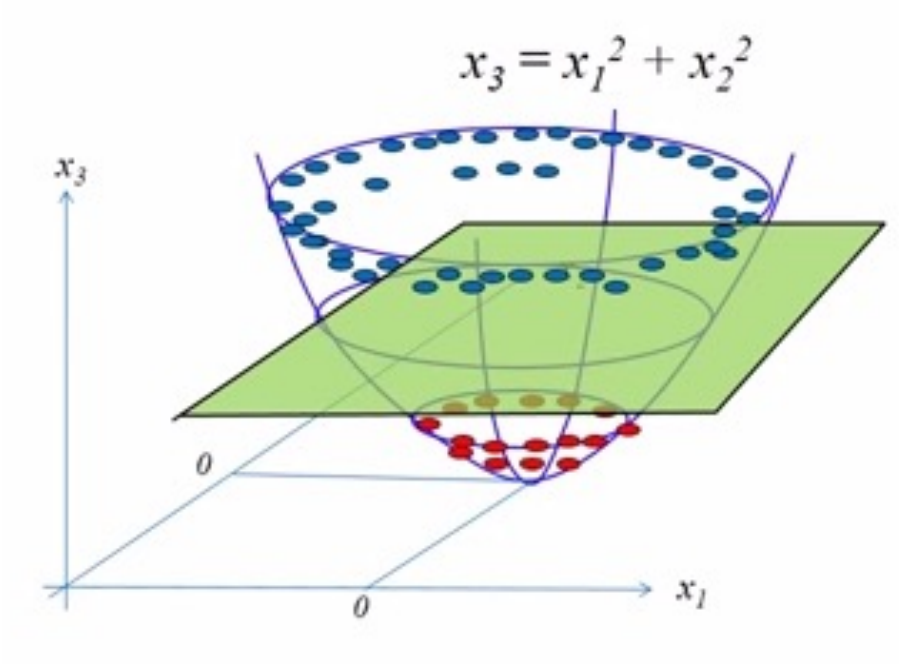


Andrew Moore

Nonlinear SVMs

- Design new features with any higher dimension.
- Nonlinear classification becomes a linear problem!

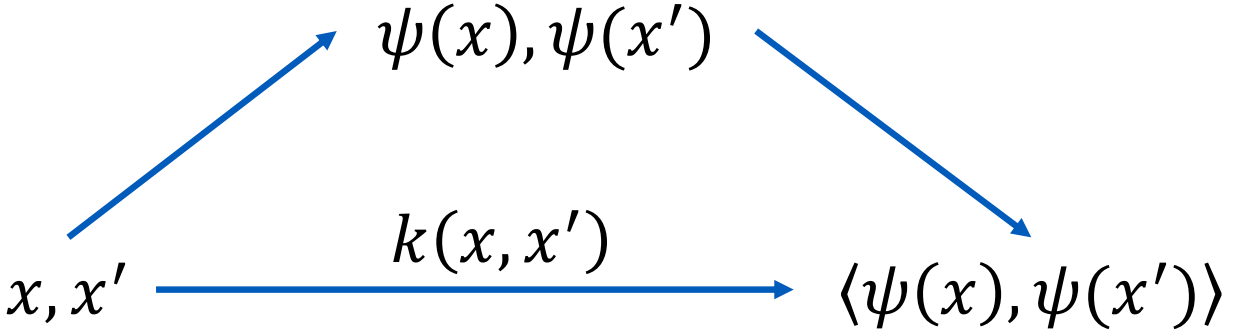




The new feature: $\boldsymbol{x} = [x_1,\ x_2,\ x_1^2 + x_2^2]$

Nonlinear SVMs with Kernel Trick

- Higher feature dimension means more computation.
- Kernel trick is to bypass the high dimension data.
- Recall that during training and testing we only want the inner product (scalar) of training samples, instead of explicit high dimension features.
- Assume nonlinear mapping $\psi(x)$: $\mathbb{R}^M \rightarrow \mathbb{R}^n$, $M \ll N$
- Can define a kernel function to directly calculate inner products



Kernel functions

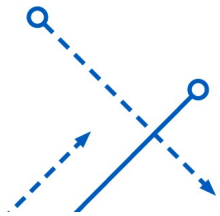
- Widely-used kernel functions

Polynomial: $k(x_m, x_n) = (x_m^T x_n + r)^d$

Linear: $k(x_m, x_n) = x_m^T x_n$

Gaussian: $k(x_m, x_n) = e^{-\frac{\|x_m - x_n\|^2}{2\sigma^2}}$

Laplace: $k(x_m, x_n) = e^{-\alpha\|x_m - x_n\|}$



Kernel Trick

- Gaussian Kernel

Let the original space $\mathcal{X} = \mathbb{R}$ and consider the mapping ψ given by

$$\psi(x) = \left\{ \frac{1}{\sqrt{m!}} e^{-\frac{x^2}{2}} x^m \right\}_{m=0}^{\infty}.$$

- Then we can prove

$$\begin{aligned}
\langle \psi(x), \psi(x') \rangle &= \sum_{m=0}^{\infty} \left(\frac{1}{\sqrt{m!}} e^{-\frac{x^2}{2}} x^m \right) \left(\frac{1}{\sqrt{m!}} e^{-\frac{(x')^2}{2}} (x')^m \right) \\
&= e^{-\frac{x^2+(x')^2}{2}} \sum_{m=0}^{\infty} \left(\frac{(xx')^m}{m!} \right) \\
&= e^{-\frac{(x-x')^2}{2}}.
\end{aligned}$$

- So we have $k(x, x') = e^{-\frac{(x-x')^2}{2}}$

Kernel Matrix

- For training and testing, we only need to calculate the kernel matrix instead of the explicit infinite high dimension features.
- Use the values when necessary, during training.

$$\boldsymbol{\phi}\boldsymbol{\phi}^T = K = \begin{bmatrix} \phi(x_1)^T\phi(x_1) & \phi(x_1)^T\phi(x_2) & . & \phi(x_1)^T\phi(x_n) \\ \phi(x_2)^T\phi(x_1) & \phi(x_2)^T\phi(x_2) & . & . \\ . & . & . & . \\ \phi(x_m)^T\phi(x_1) & \phi(x_m)^T\phi(x_2) & . & \phi(x_m)^T\phi(x_n) \end{bmatrix}$$

$$= \begin{bmatrix} k(x_1, x_1) & k(x_1, x_2) & . & k(x_1, x_n) \\ k(x_2, x_1) & k(x_2, x_2) & . & . \\ . & . & . & . \\ k(x_m, x_1) & k(x_m, x_2) & . & k(x_m, x_n) \end{bmatrix}$$

Dive into nonlinear kernel SVM

- Let the weights w being $\mathbf{w} = \sum_{i=0}^{N-1} a_i \phi(\mathbf{x}_i)$
- Then inner products $f(x)$ and $||w||^2$become

$$f(x) = \langle \mathbf{w}, \phi(\mathbf{x}) \rangle = \sum_{j=0}^{N-1} a_j \langle \phi(\mathbf{x}_j), \phi(\mathbf{x}) \rangle = \sum_{i=0}^{N-1} a_i \kappa(\mathbf{x}_i, \mathbf{x}).$$

$$\|\mathbf{w}\|^2 = \left\langle \sum_{j=0}^{N-1} a_j \phi(\mathbf{x}_j), \sum_{j=0}^{N-1} a_j \phi(\mathbf{x}_j) \right\rangle = \sum_{i,j=0}^{N-1} a_i a_j \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle .$$

- Then the loss function in page 38 becomes

$$\min_{\mathbf{a} \in \mathbb{R}^N} \left(\lambda \mathbf{a}^T \mathbf{K} \mathbf{a} + \frac{1}{N} \sum_{i=0}^{N-1} \max\{0, 1 - y_i (\mathbf{K}\mathbf{a})_i\} \right),$$

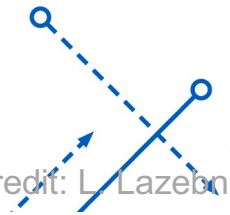
i-th element of the vector Ka.

- Then solution is like linear SVM!

Page 14-16 in C. Wang, Kernel learning for visual perception.

Summary: SVMs for image classification

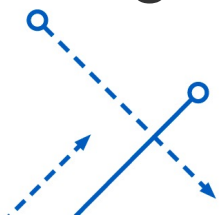
1. Pick an image representation
   - In our case, bag of features
2. Pick a kernel function for that representation
3. Compute the kernel matrix.
4. Feed the kernel matrix into the SVM solver, e.g., SGD, to obtain support vectors and weights.
5. At test time: compute kernel values for your test example and each support vector, and combine them with the learned weights to get the value of the decision function.



Slide credit: L. Lazebnik

What about multi-class SVMs?

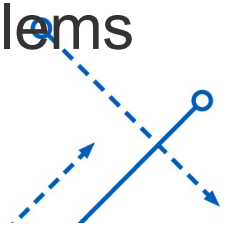
- Unfortunately, there is no “definitive” multi-class SVM formulation
- In practice, we have to obtain a multi-class SVM by combining multiple two-class SVMs
- One vs. others
  - Training: learn an SVM for each class vs. the others
  - Testing: apply each SVM to test example and assign to it the class of the SVM returning highest decision value
- One vs. one
  - Training: learn an SVM for each pair of classes
  - Testing: each learned SVM “votes” for a class to assign to the test example



SVMs: Pros and Cons

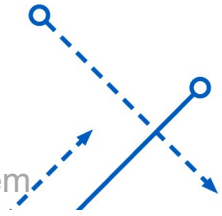
- Pros
  - Many publicly available SVM packages: http://www.kernel-machines.org/software
  - Kernel-based framework is very powerful, flexible
  - SVMs work very well in practice, even with very small training sample sizes

- Cons
  - No “direct” multi-class SVM, combine two-class SVMs
  - Computation, memory
    - During training time, must compute the big kernel matrix.
    - Learning can take a very long time for large-scale problems



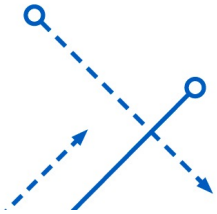
What to remember about classifiers

- No free lunch: machine learning algorithms are tools, not dogmas
- Try SIMPLE classifiers first
- Better to have smart features and simple classifiers than simple features and smart classifiers
- Use increasingly powerful classifiers with more training data (bias-variance tradeoff)



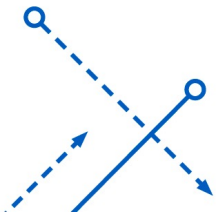
Machine Learning Considerations

- 3 important design decisions:
  - *** What data do I use?
  - ** How do I represent my data (what feature)?
  - * What classifier / regressor / models do I use?
- These are in decreasing order of importance
- Deep learning addresses 2 and 3 simultaneously (and blurs the boundary between them).
- You can take the features/representation from deep learning and use it with any classifier.



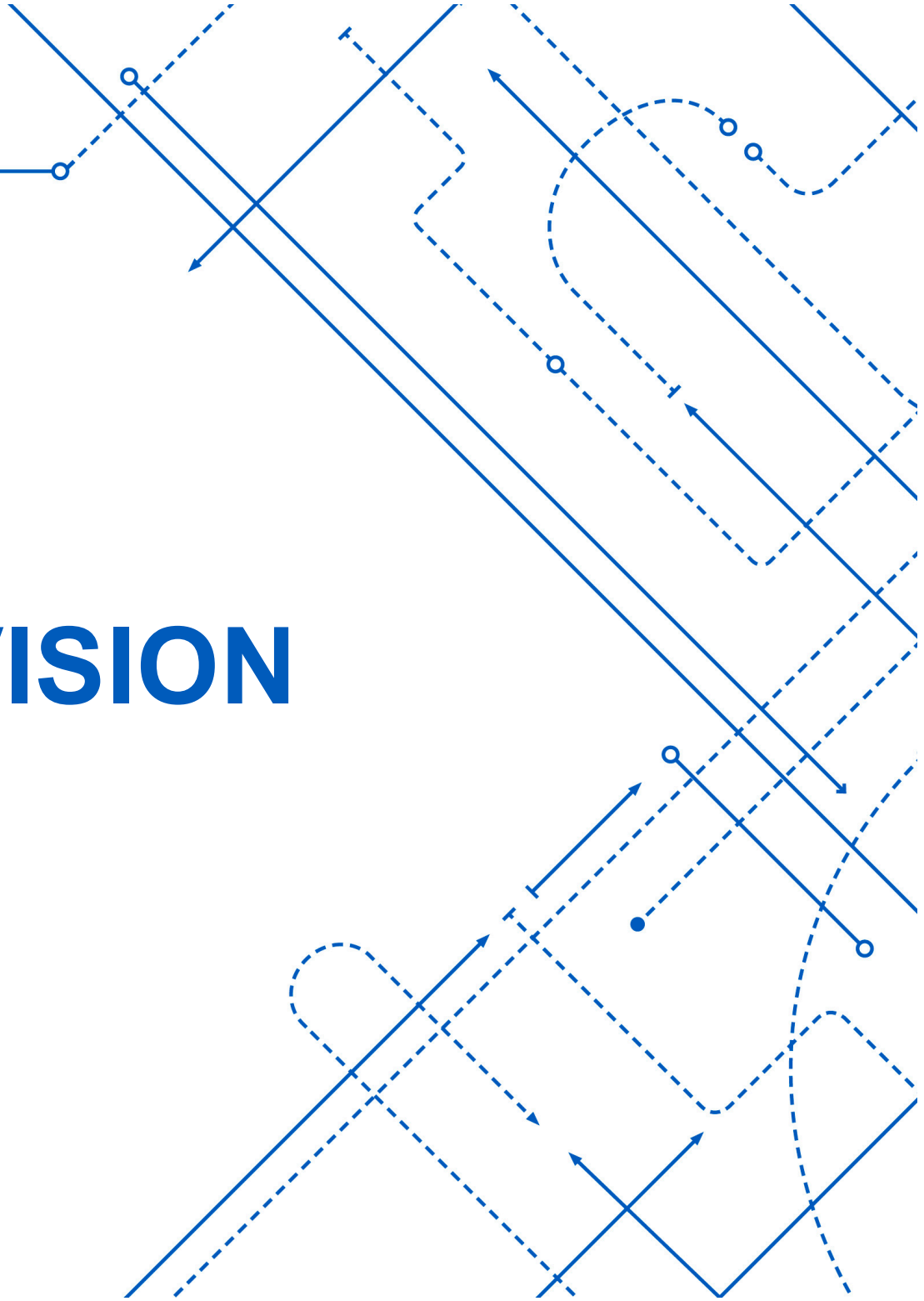
Machine Learning Considerations

- How to overcome overfitting?
  - Simpler Model
  - Data augmentation
  - Early stopping
  - Regularization
  - Ensembling
  - …



SAIR

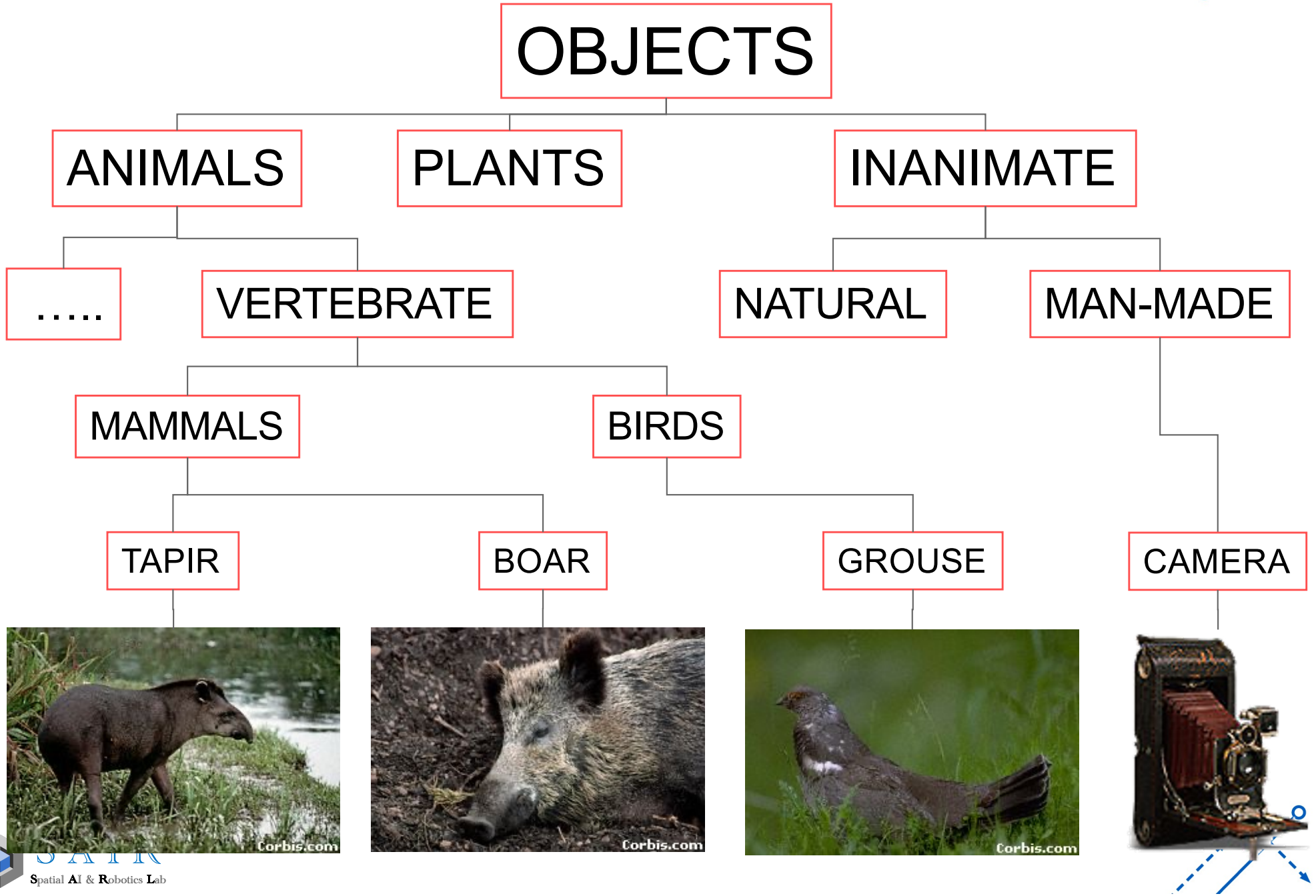
Spatial AI & Robotics Lab



COMPUTER VISION

Recognition

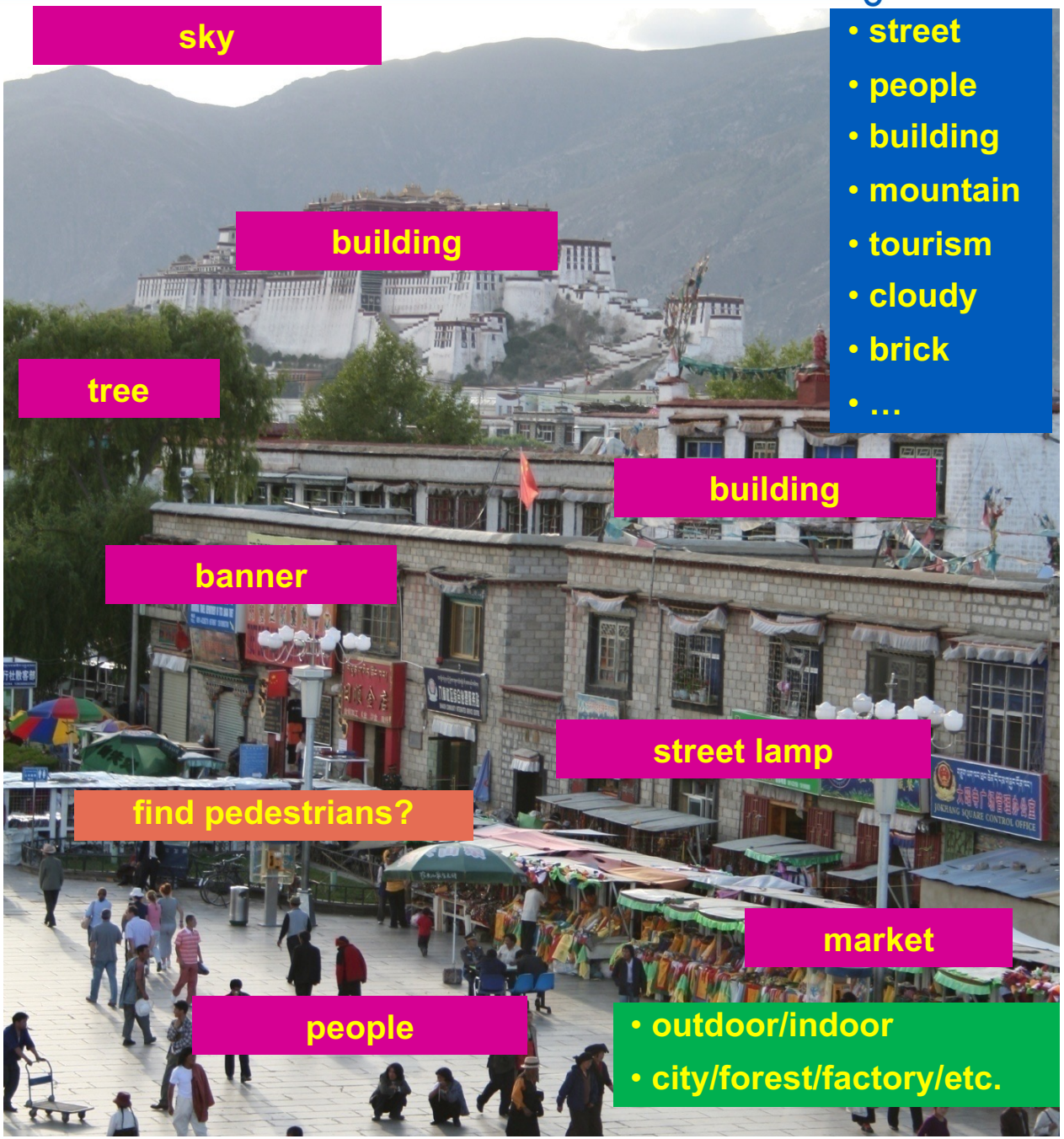
Object Category



Computer Vision.

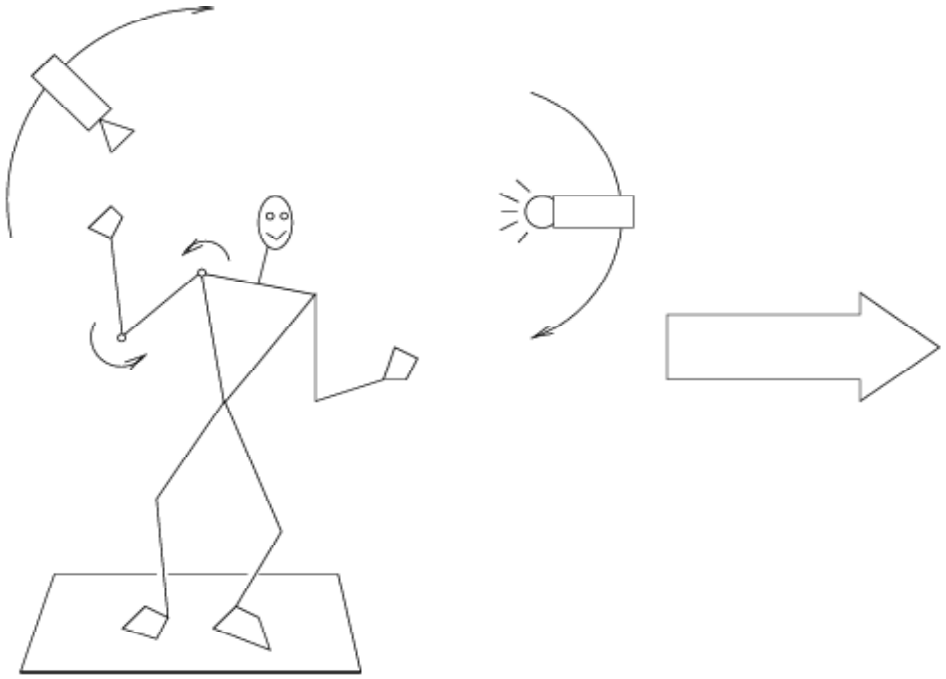
- Object Detection
- Scene Categorization
- Scene Tagging
- Image Parsing & Segmentation

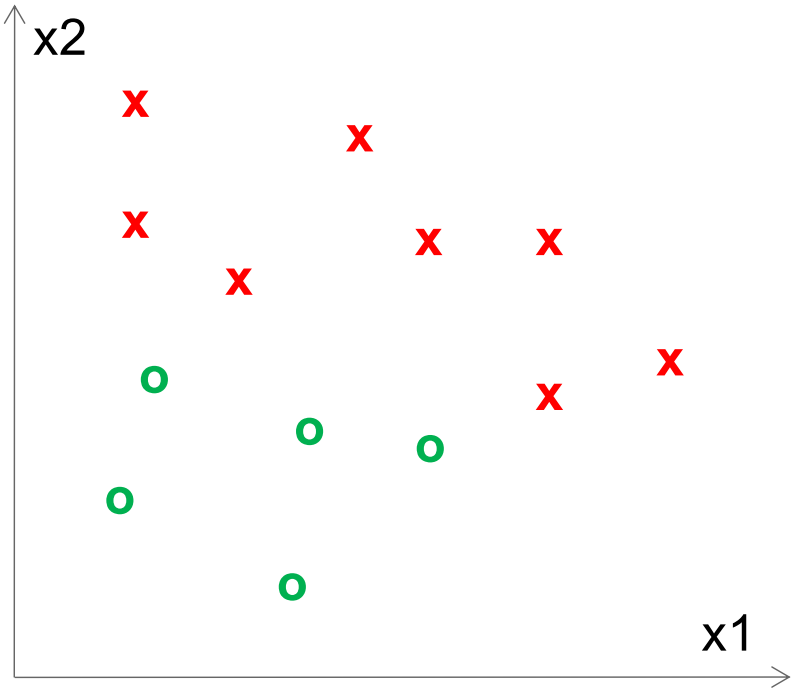
- Scene Understanding?



Recognition is all about modeling variability

- Invariance is the key challenge to computer vision.

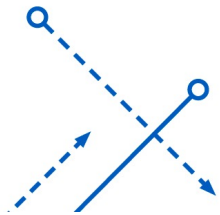




Variability:

Camera position
Illumination
Shape parameters

Within-class variations?



Within-class variations

History of ideas in recognition

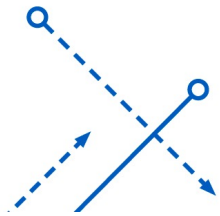
- 1960s – early 1990s: the geometric era

Variability: Camera position
Illumination

Shape: Assumed known

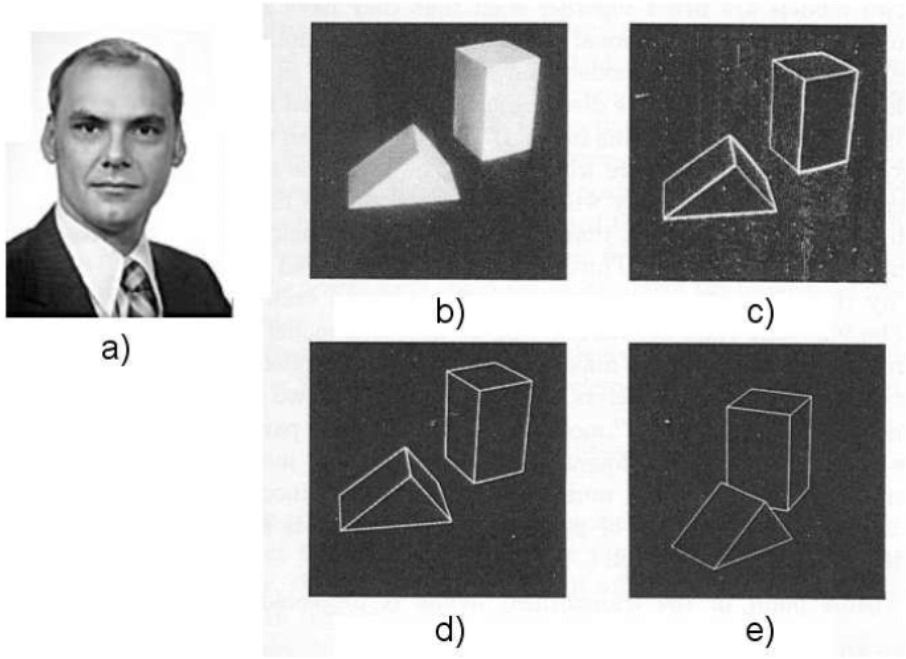
Focus: **Alignment**

Roberts (1965); Lowe (1987); Faugeras & Hebert (1986);
Grimson & Lozano-Perez (1986); Huttenlocher & Ullman (1987)



History of ideas in recognition

- Recognition as an alignment problem: Block world



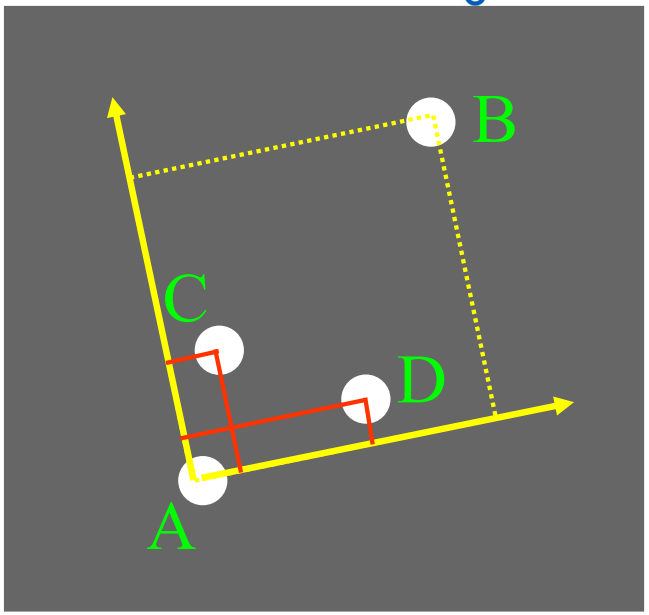
L. G. Roberts, *Machine Perception of Three Dimensional Solids*, Ph.D. thesis, MIT Department of Electrical Engineering, 1963.

Fig. 1. A system for recognizing 3-d polyhedral scenes. a) L.G. Roberts. b)A blocks world scene. c)Detected edges using a 2x2 gradient operator. d) A 3-d polyhedral description of the scene, formed automatically from the single image. e) The 3-d scene displayed with a viewpoint different from the original image to demonstrate its accuracy and completeness. (b) - e) are taken from [64] with permission MIT Press.)

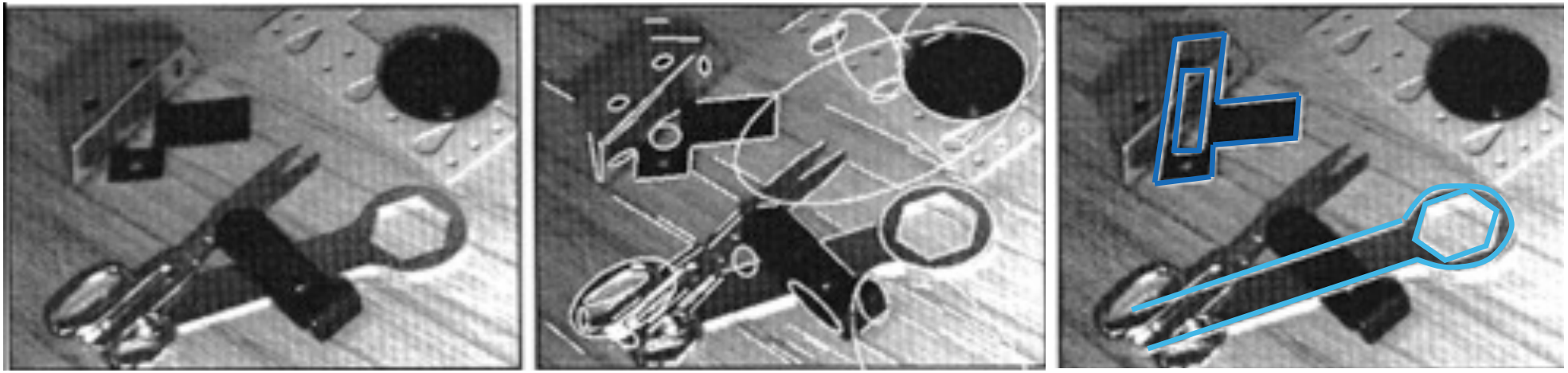
J. Mundy, Object Recognition in the Geometric Era: a Retrospective, 2006

History of ideas in recognition

Example:
invariant to similarity transformations
computed from four points



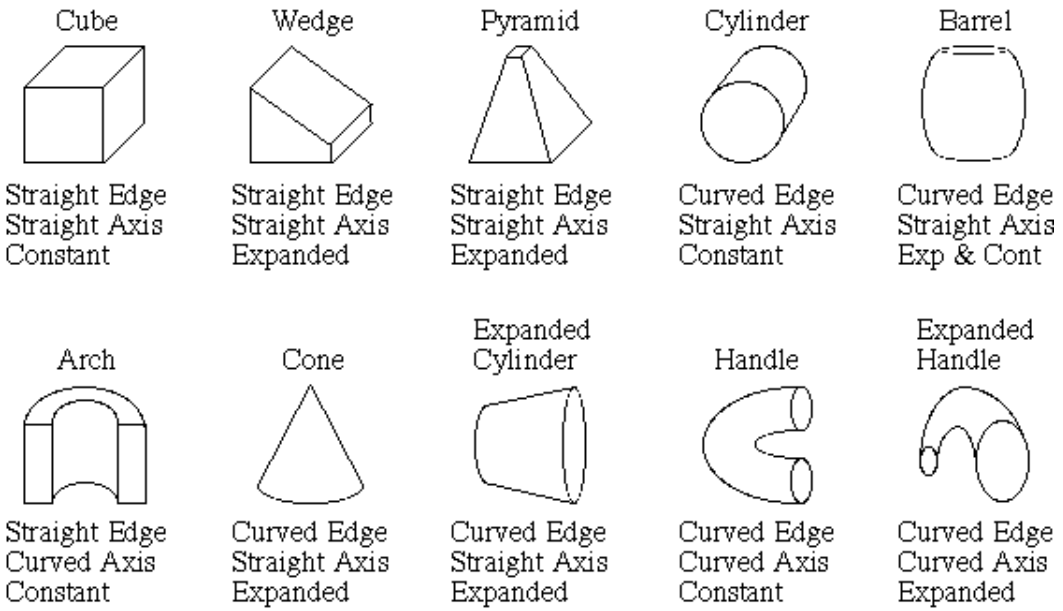
Projective invariants (Rothwell et al., 1992):



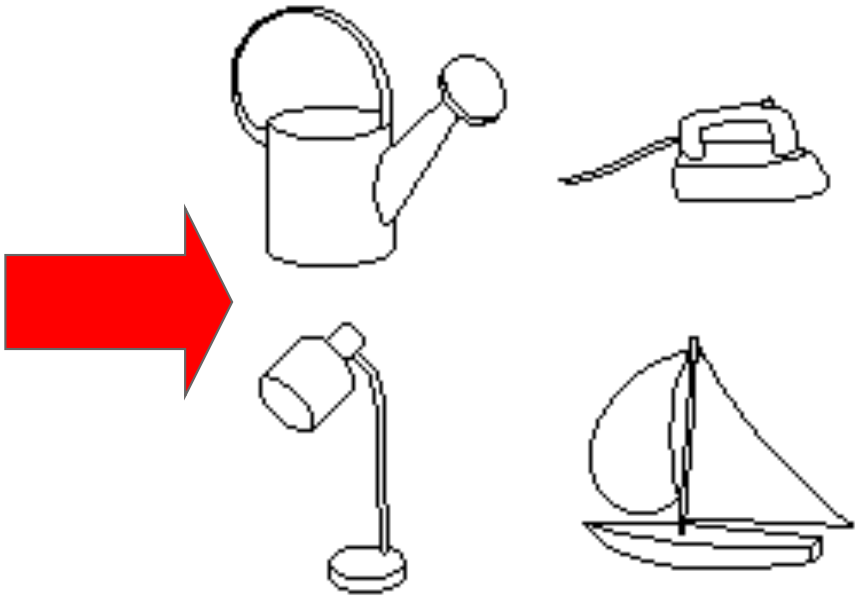
General 3D objects do not admit monocular viewpoint invariants (Burns et al., 1993)

Recognition by components

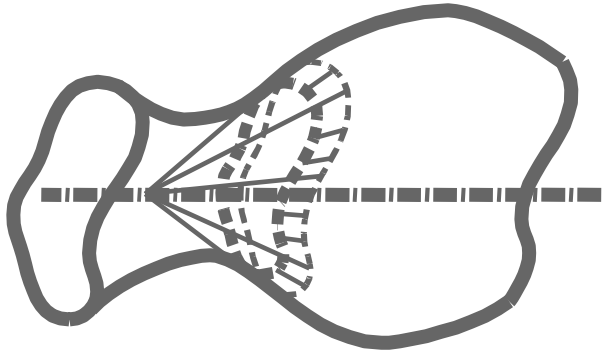
Primitives (geons) Biederman (1987)



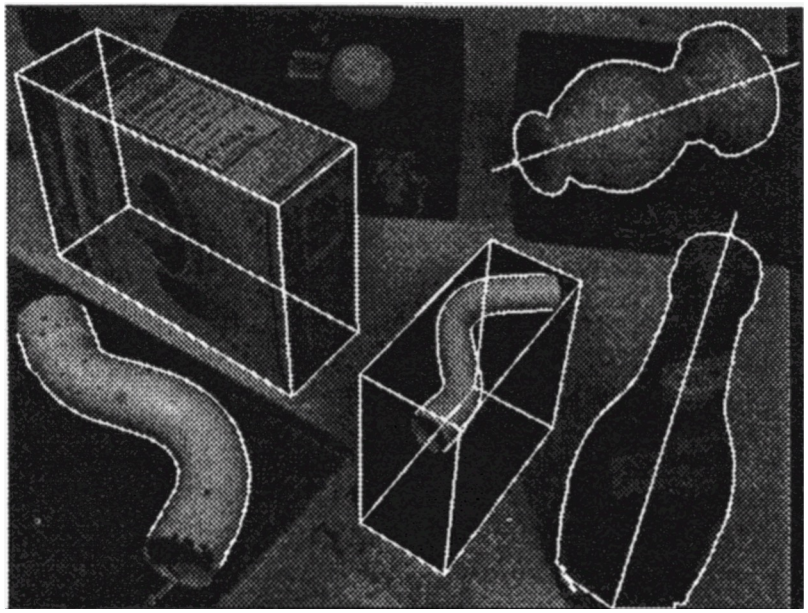
Objects



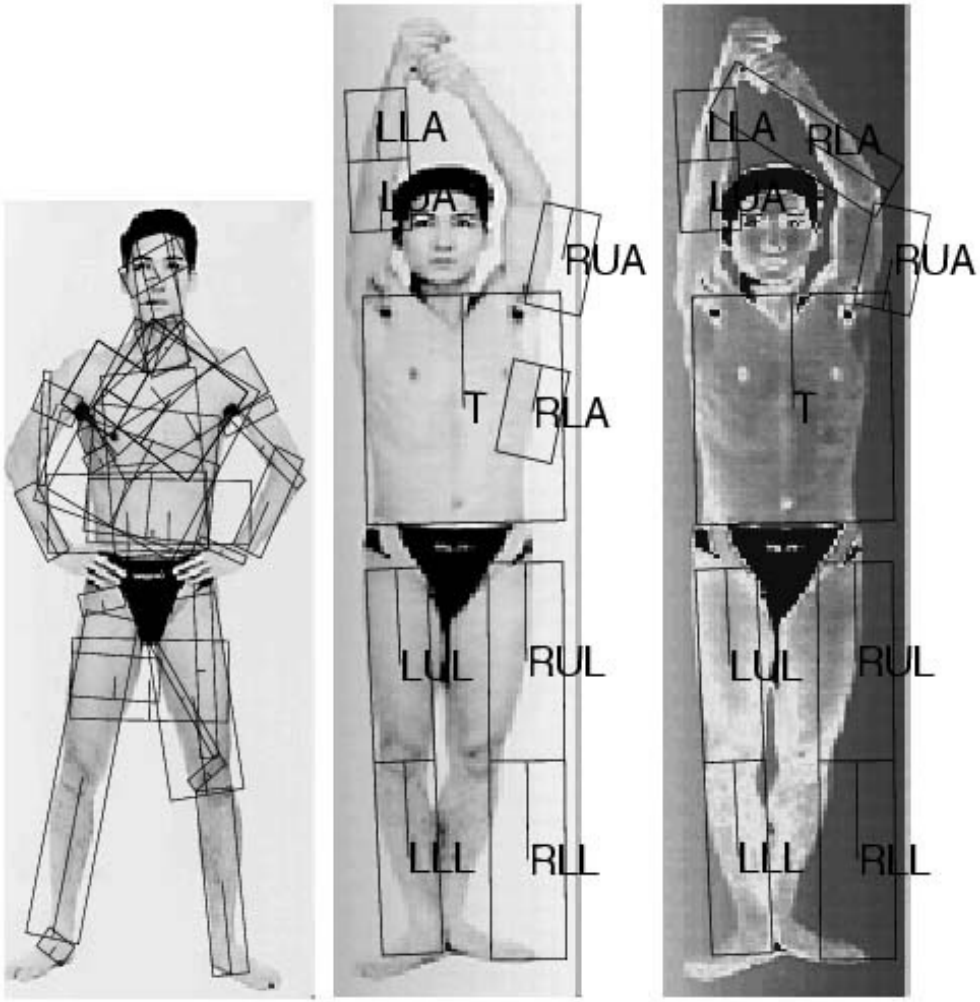
Recognition by components



Generalized cylinders
[Ponce et al. (1989)]



Zisserman et al. (1995)



Forsyth (2000)

History of ideas in recognition

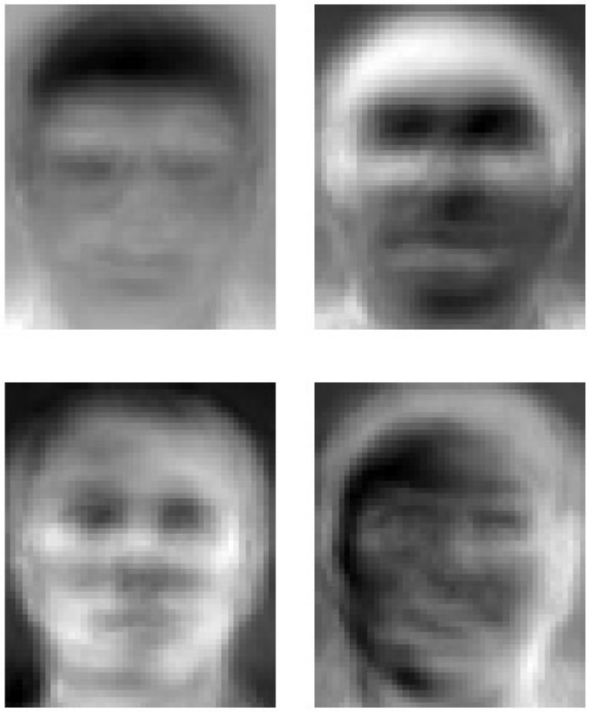
- 1960s – early 1990s: the geometric era
- 1990s: appearance-based models

Empirical models of image variability

Appearance-based techniques

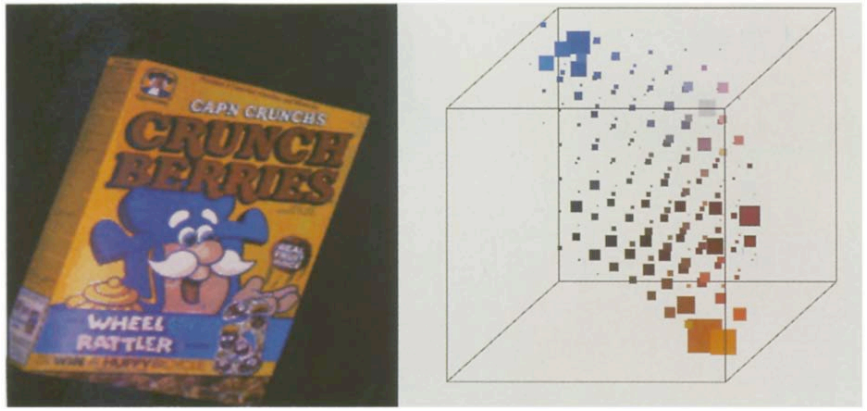
Turk & Pentland (1991); Murase & Nayar (1995); etc.

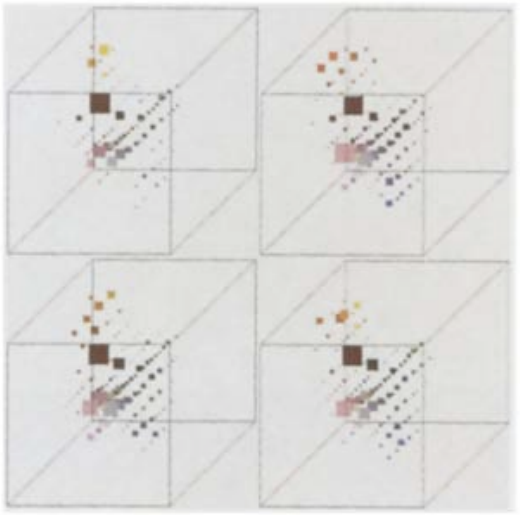
Eigenfaces (Turk & Pentland, 1991)



Experimental	Correct/Unknown Recognition Percentage		
Condition	Lighting	Orientation	Scale
Forced classification	96/0	85/0	64/0
Forced 100% accuracy	100/19	100/39	100/60
Forced 20% unknown rate	100/20	94/20	74/20

Color Histograms

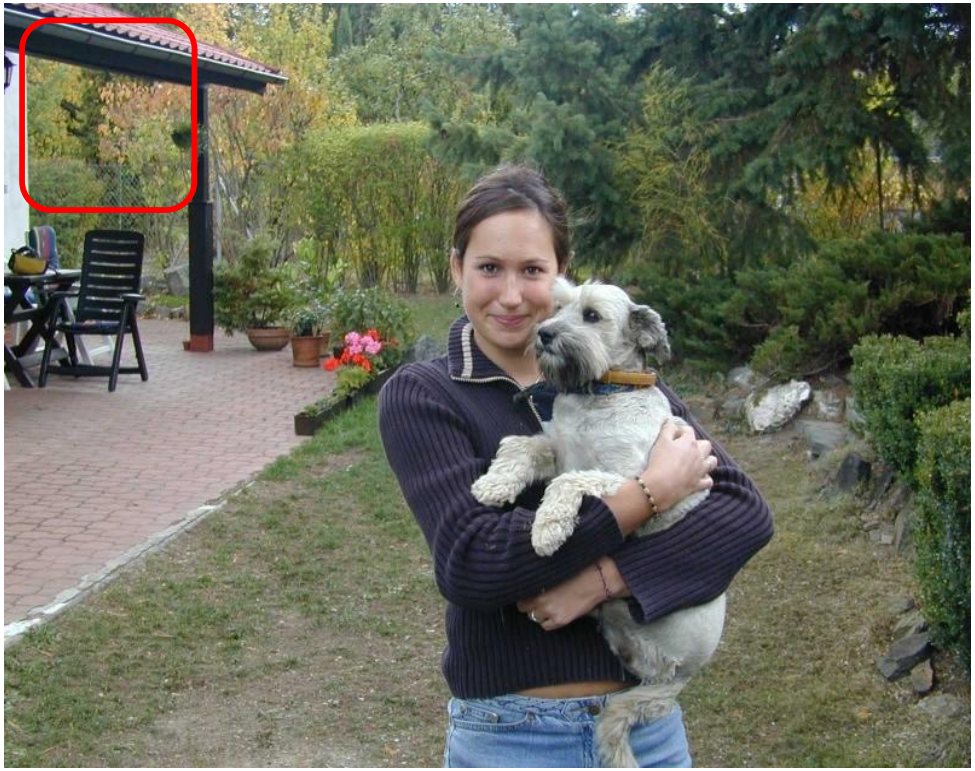




Swain and Ballard, Color Indexing, IJCV 1991.

History of ideas in recognition

- 1960s – early 1990s: the geometric era
- 1990s: appearance-based models
- 1990s – present: sliding window approaches



Sliding window approaches

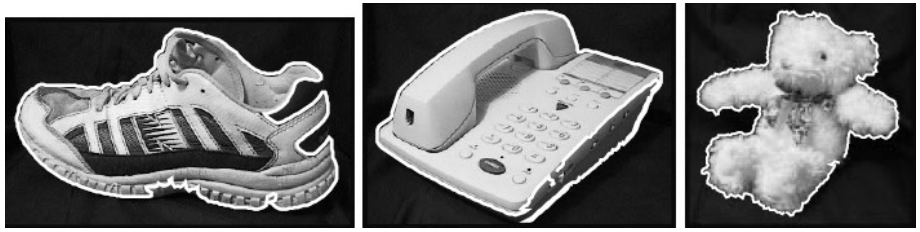
Turk and Pentland, 1991
Belhumeur, Hespanha, & Kriegman, 1997
Schneiderman & Kanade 2004
Viola and Jones, 2000

Schneiderman & Kanade, 2004
Argawal and Roth, 2002
Poggio et al. 1993

History of ideas in recognition

- 1960s – early 1990s: the geometric era
- 1990s: appearance-based models
- Mid-1990s: sliding window approaches
- Late 1990s: local features

Local features for instance recognition



D. Lowe (1999, 2004)

Large-scale image search

Combining local features, indexing, and spatial constraints

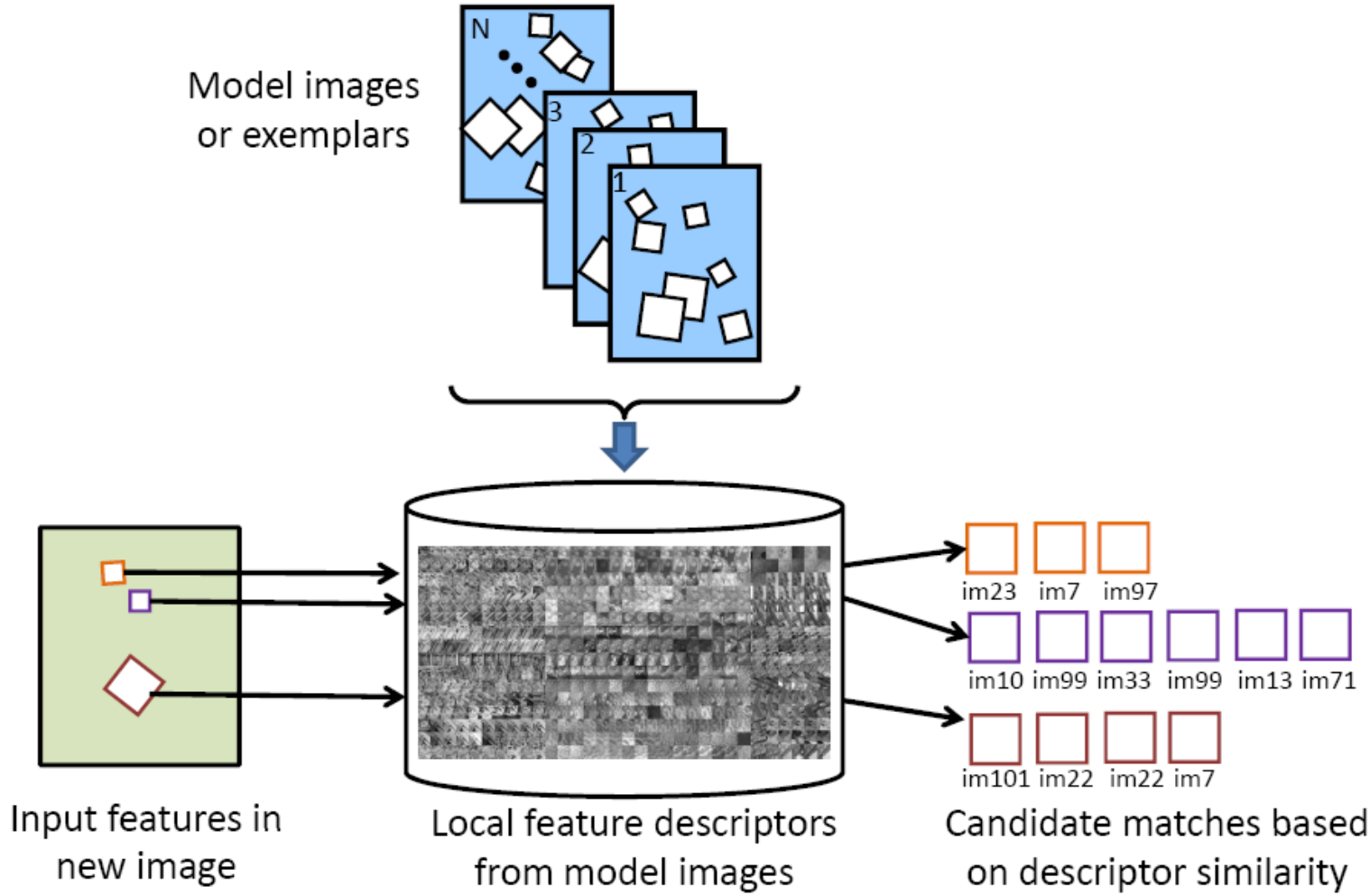
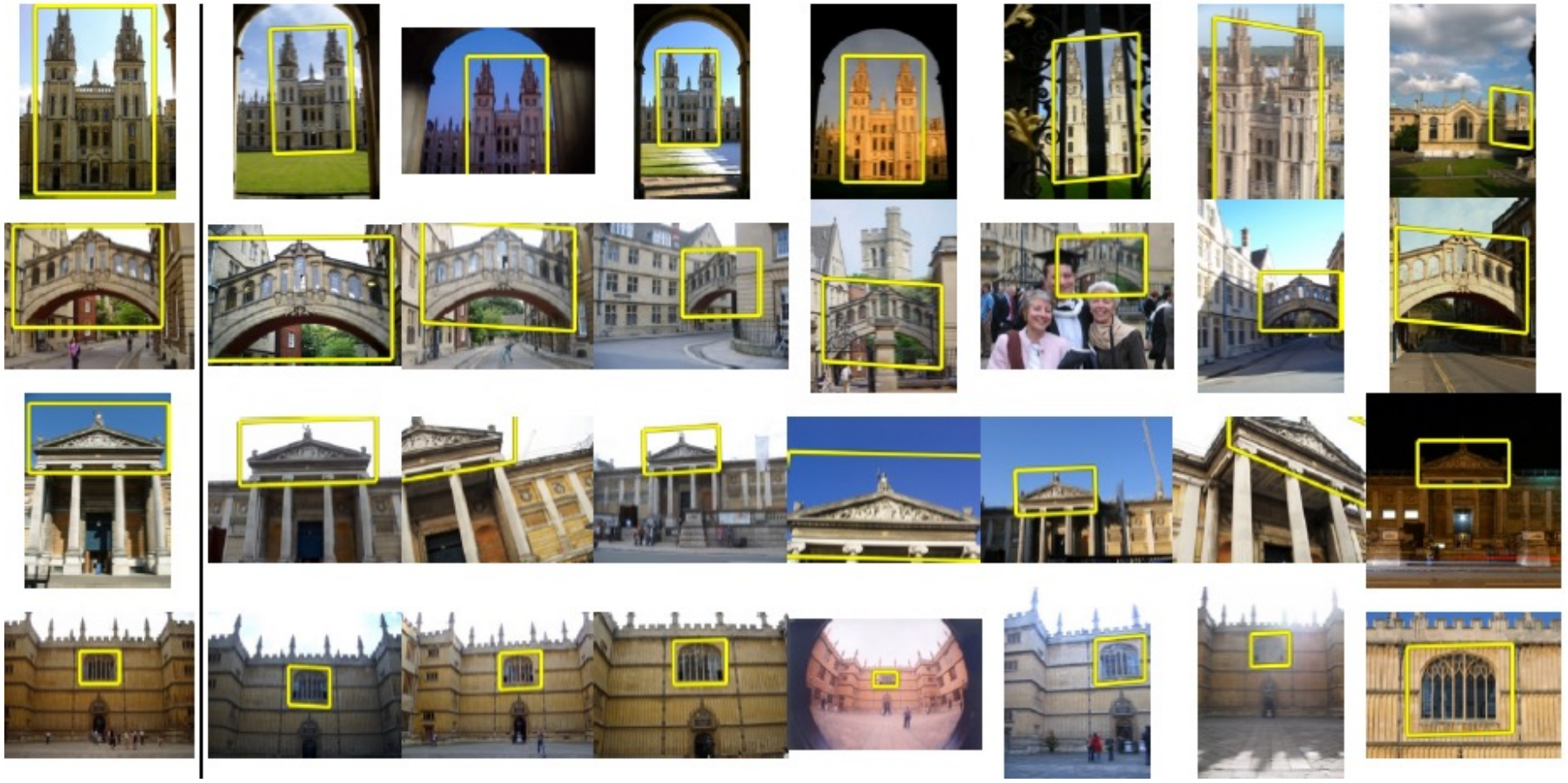


Image credit: K. Grauman and B. Leibe

Large-scale image search

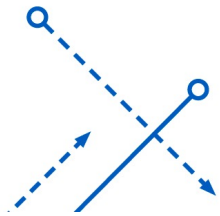
Combining local features, indexing, and spatial constraints



Philbin et al. ‘07

History of ideas in recognition

- 1960s – early 1990s: the geometric era
- 1990s: appearance-based models
- Mid-1990s: sliding window approaches
- Late 1990s: local features
- Early 2000s: parts-and-shape models



Parts-and-shape models

- Model:
  - Object as a set of parts
  - Relative locations between parts
  - Appearance of part

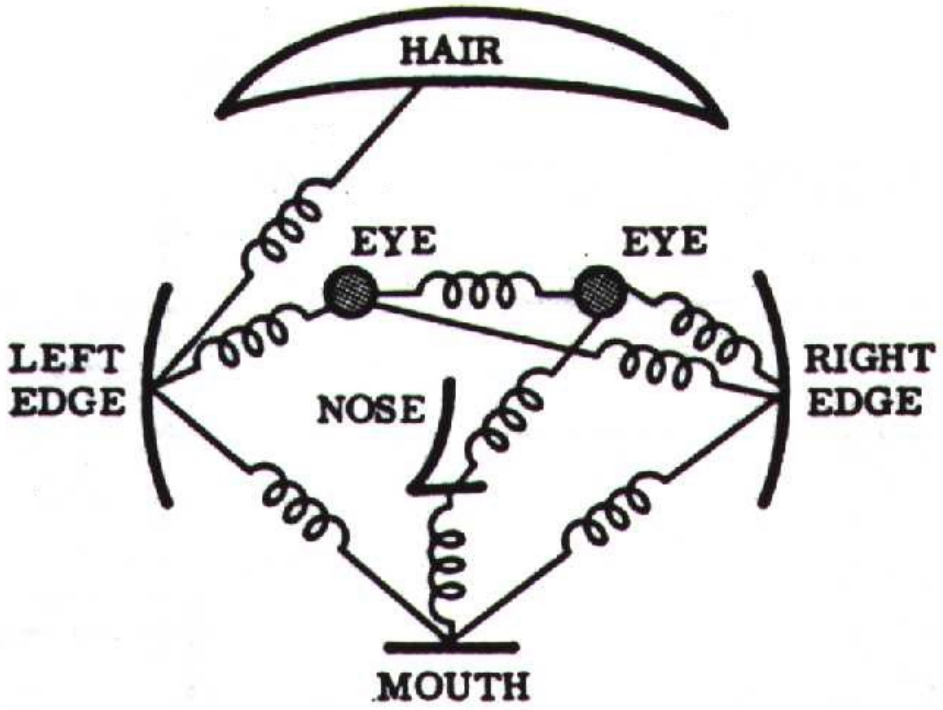
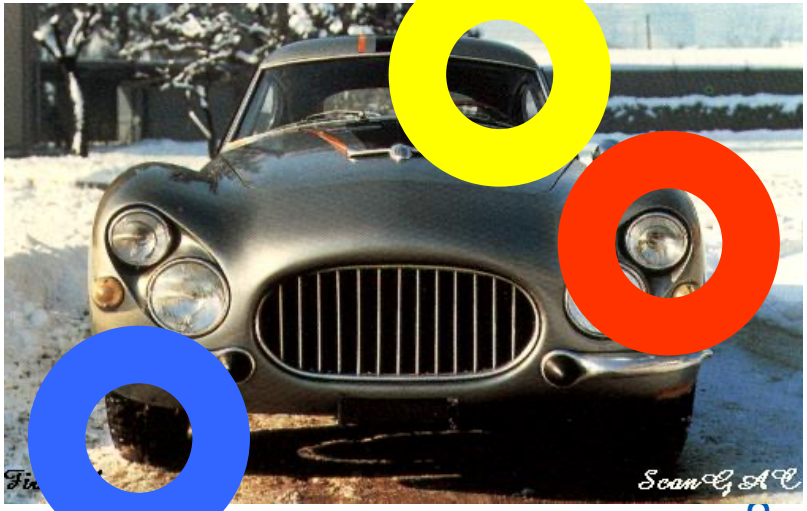


Figure from [Fischler & Elschlager 73]

Constellation models

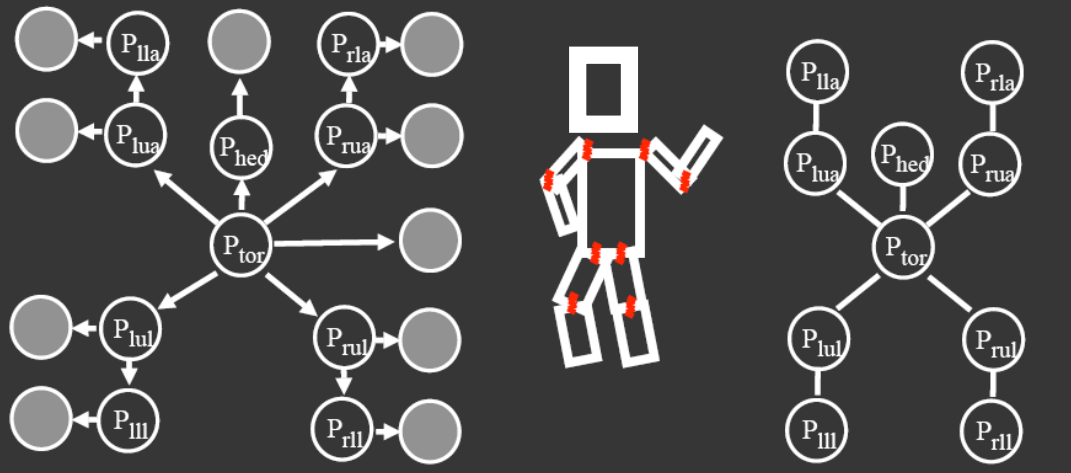


Weber, Welling & Perona (2000), Fergus, Perona & Zisserman (2003)

Representing people

Pictorial structure model

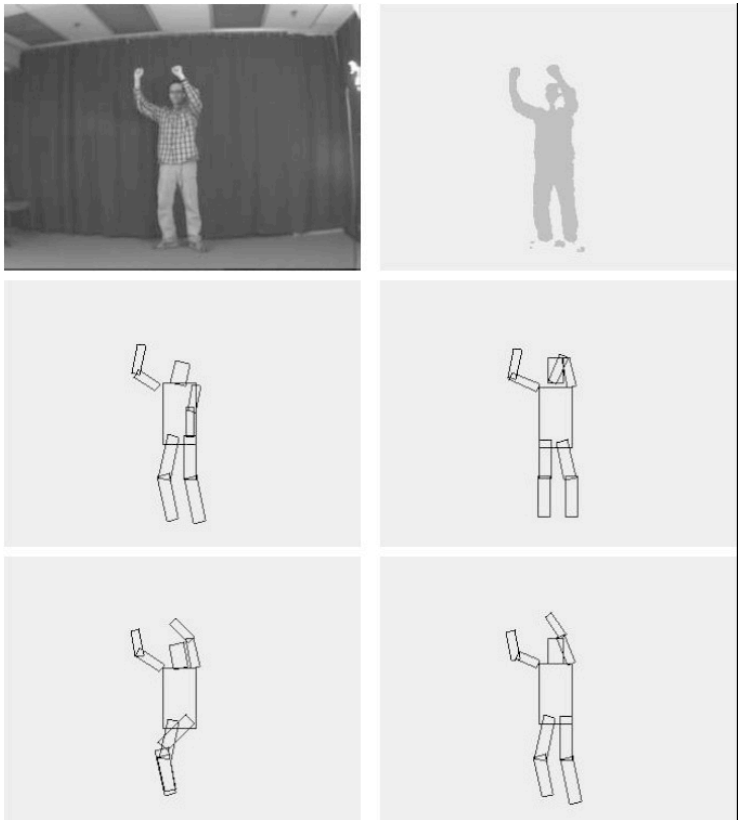
Fischler and Elschlager(73), Felzenszwalb and Huttenlocher(00)



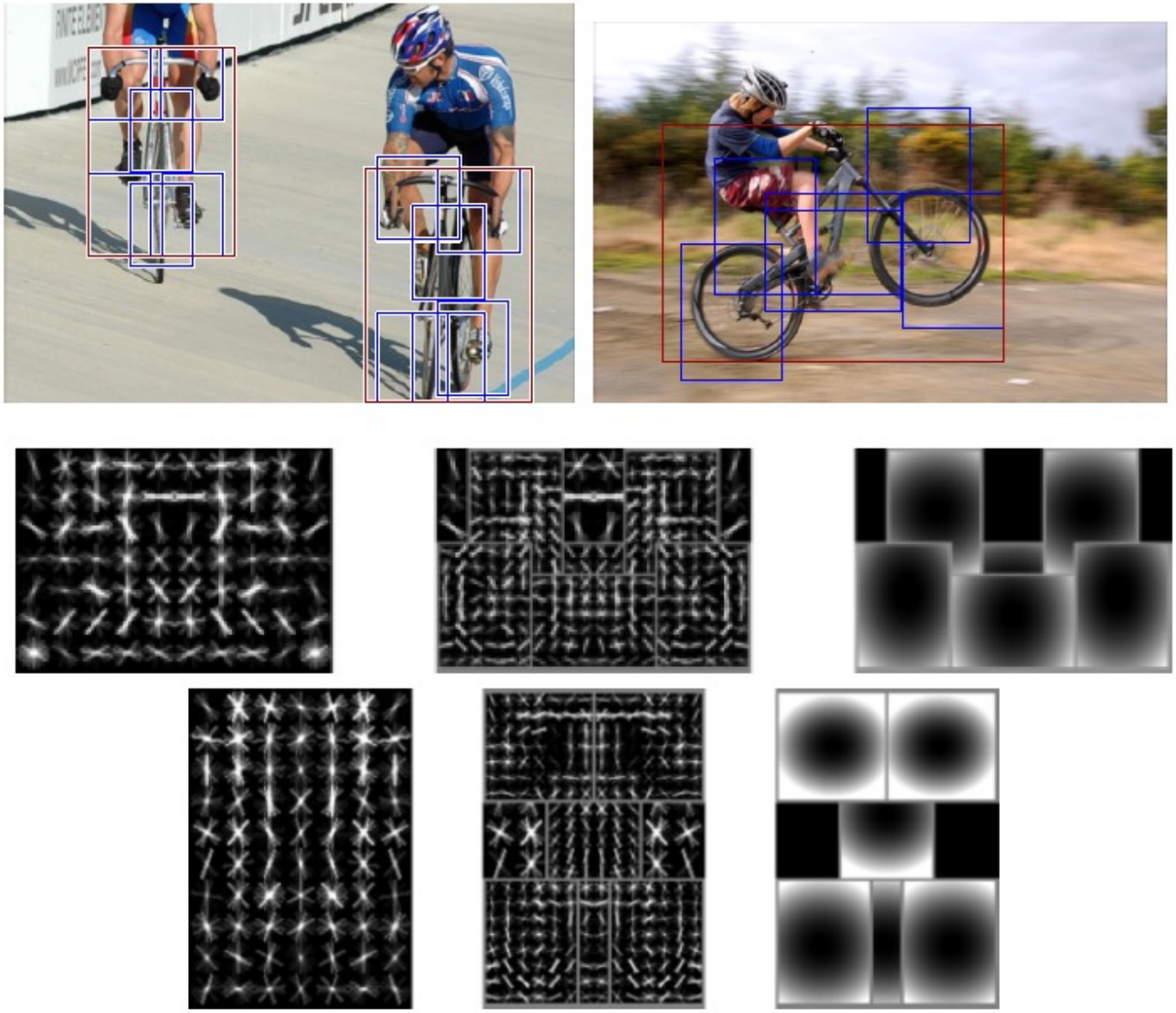
$$\Pr(P_{tor}, P_{arm}, \ldots | \mathrm{Im}) \propto \prod_{i,j} \Pr(P_i | P_j) \prod_i \Pr(\mathrm{Im}(P_i))$$

part geometry

part appearance



Discriminatively trained part-based models

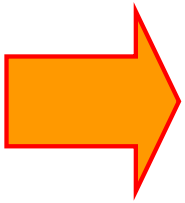


P. Felzenszwalb, R. Girshick, D. McAllester, D. Ramanan, **"Object Detection with Discriminatively Trained Part-Based Models,"** PAMI 2009

History of ideas in recognition

- 1960s – early 1990s: the geometric era
- 1990s: appearance-based models
- Mid-1990s: sliding window approaches
- Late 1990s: local features
- Early 2000s: parts-and-shape models
- Mid-2000s: bags of features

Bag-of-features models



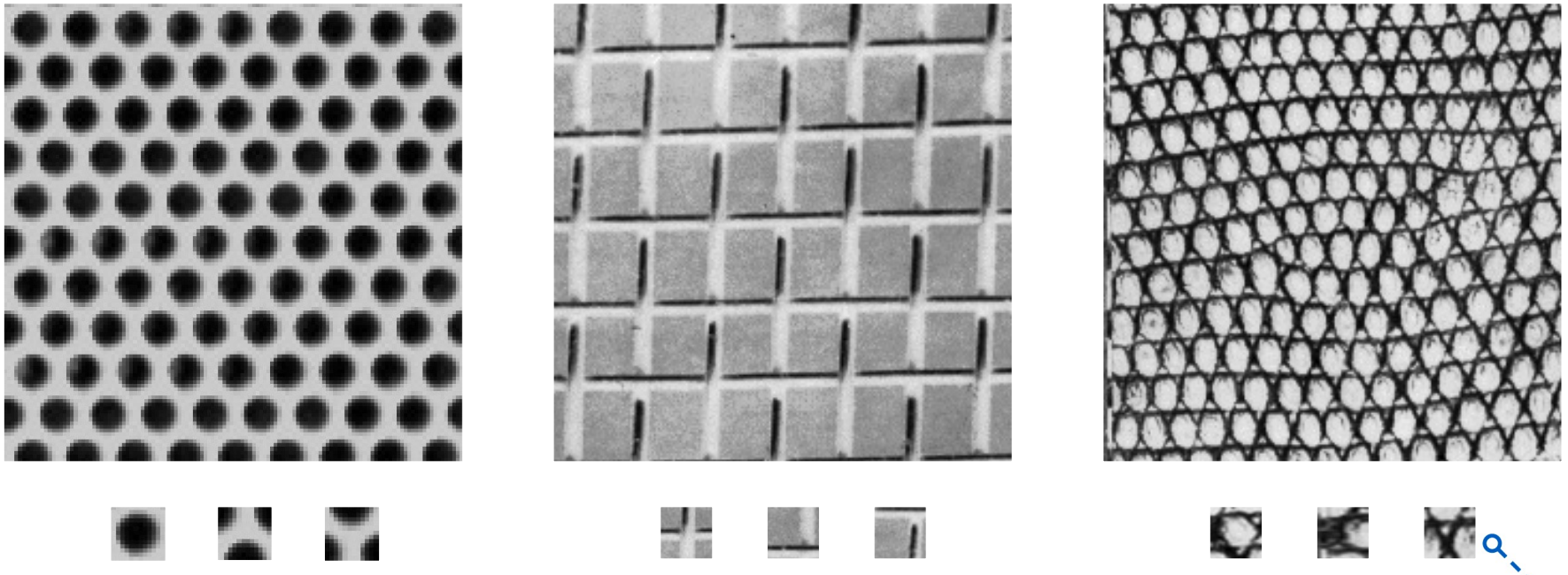
Bag-of-features models

Object

Bag of ‘words’

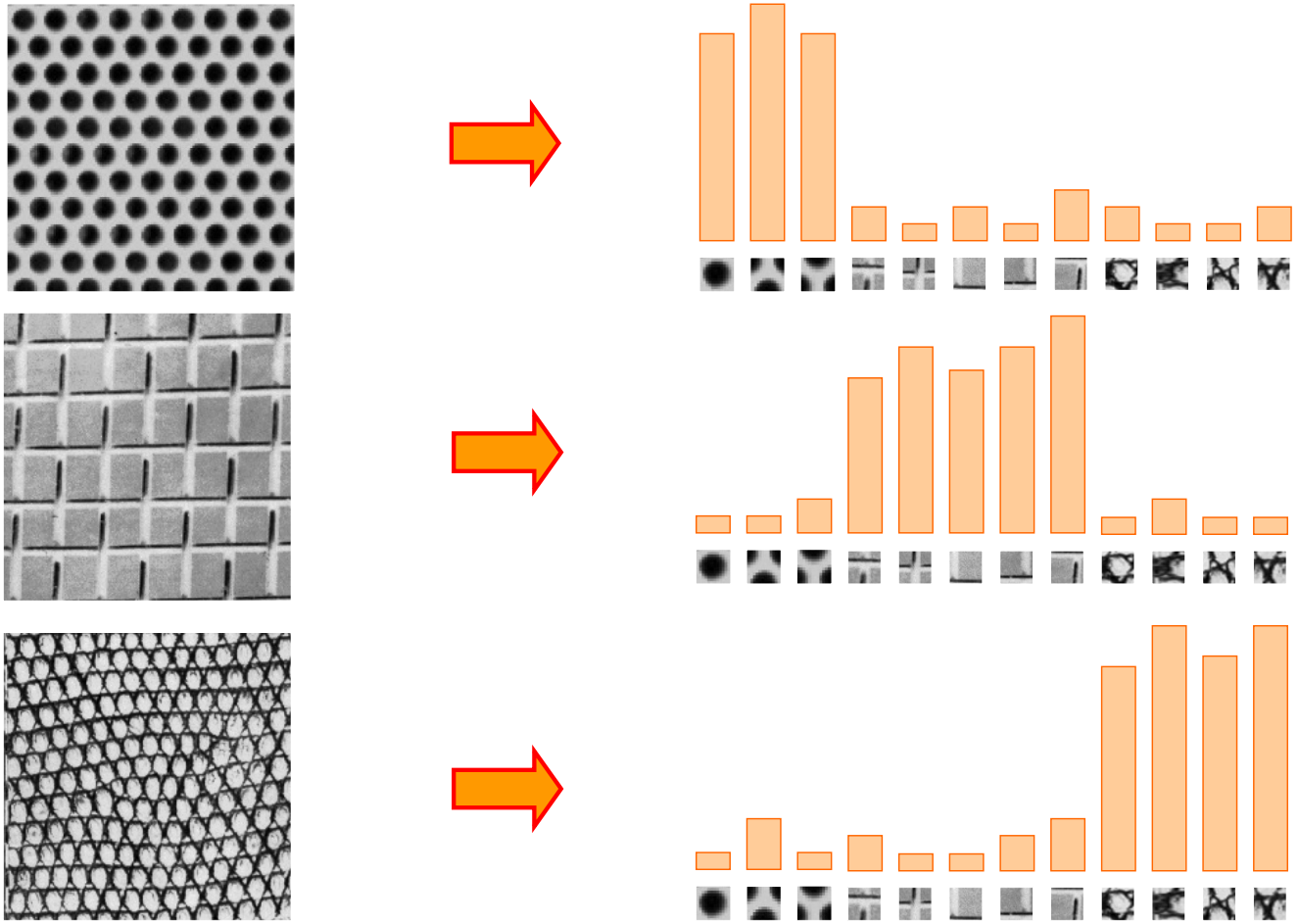
Origin 1: Texture recognition

- Texture is characterized by the **repetition** of **basic elements** or ***textons***
- For stochastic textures, the identity of the textons is more important than their spatial arrangement



Julesz, 1981; Cula & Dana, 2001; Leung & Malik 2001; Mori, Belongie & Malik, 2001; Schmid 2001; Varma & Zisserman, 2002, 2003; Lazebnik, Schmid & Ponce, 2003

Origin 1: Texture recognition



Julesz, 1981; Cula & Dana, 2001; Leung & Malik 2001; Mori, Belongie & Malik, 2001; Schmid 2001; Varma & Zisserman, 2002, 2003; Lazebnik, Schmid & Ponce, 2003

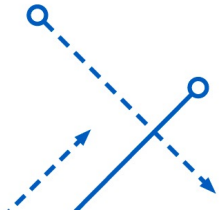
Origin 2: Bag-of-words models

- Orderless document representation: frequencies of words from a dictionary Salton & McGill (1983)

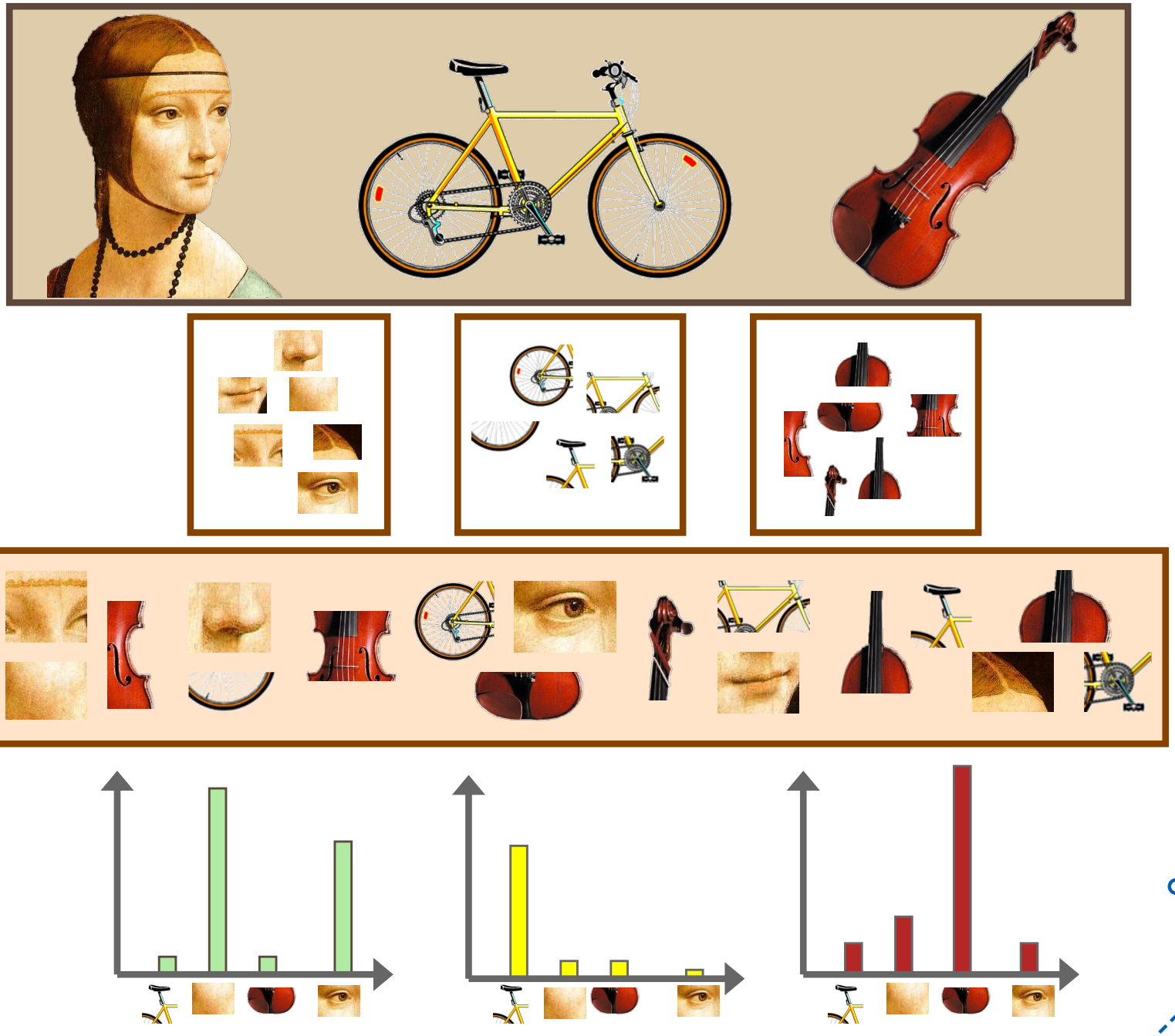
US Presidential Speeches Tag Cloud
http://chir.ag/phernalia/preztags/

Bag-of-features steps

1. Extract features
2. Learn “visual vocabulary”
3. Quantize features using visual vocabulary
4. Represent images by frequencies of “visual words”



Bag-of-features steps

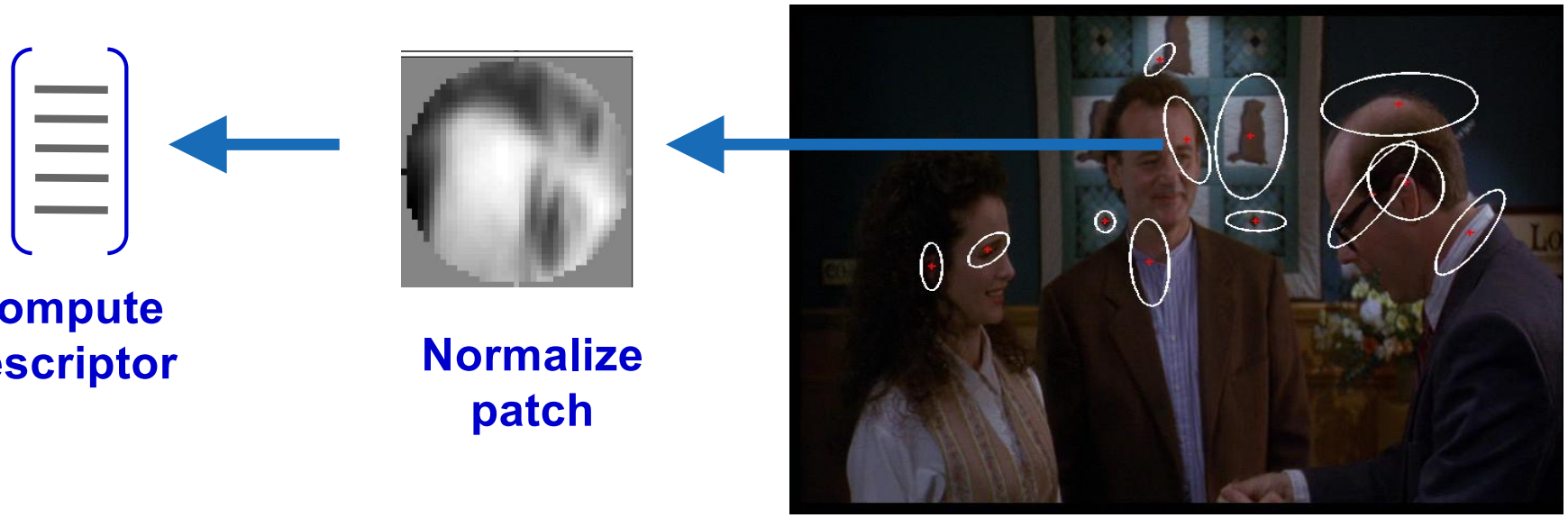


1. Feature extraction

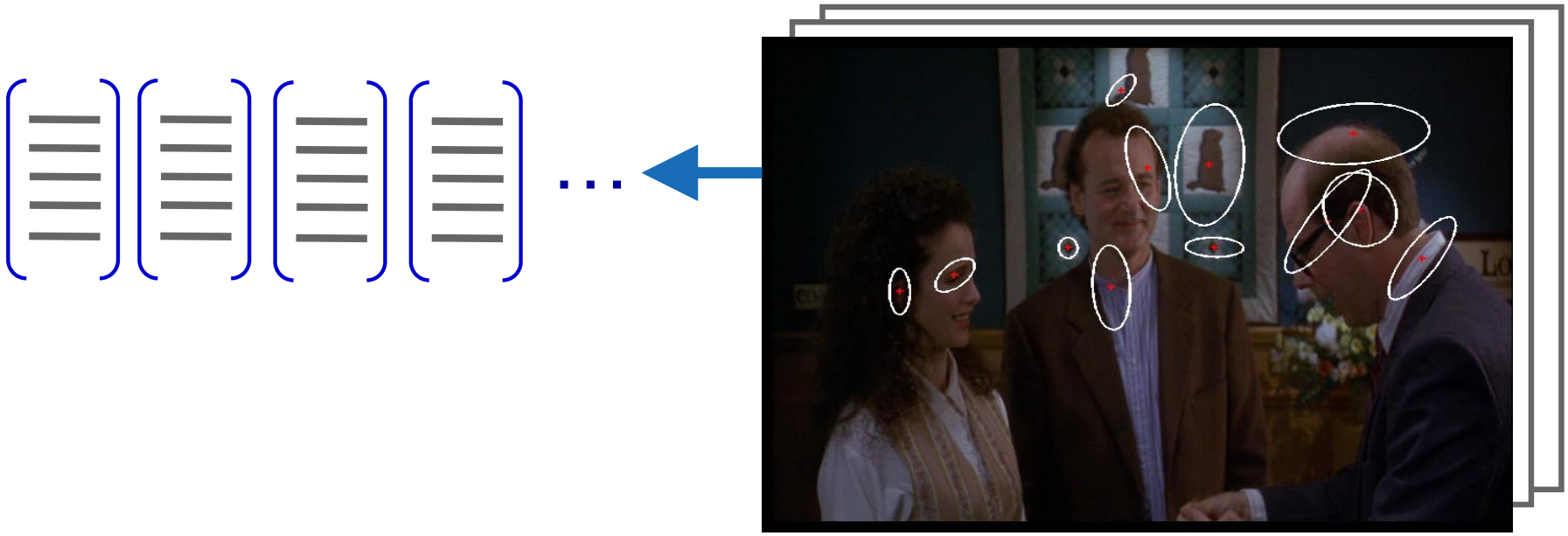
Regular grid

Interest regions

1. Feature extraction

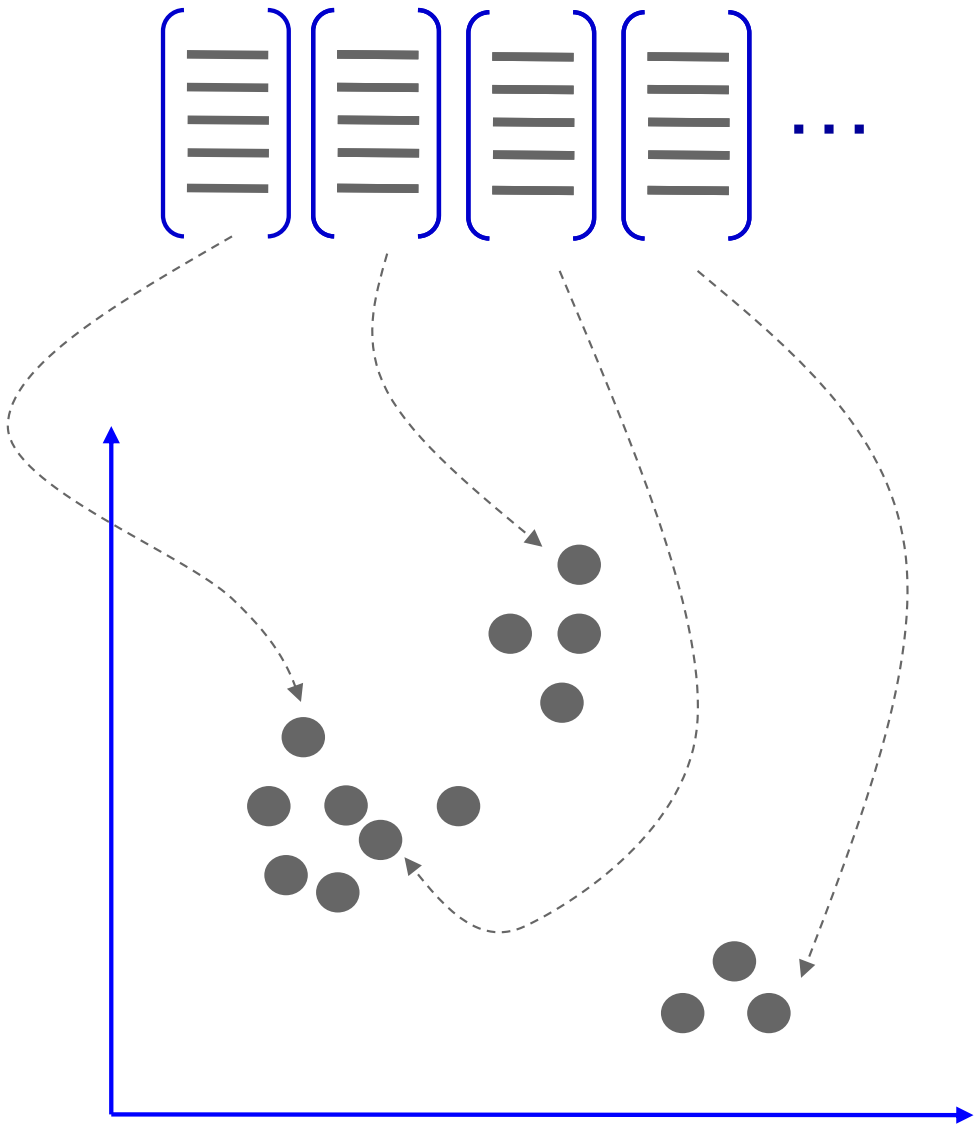


1. Feature extraction

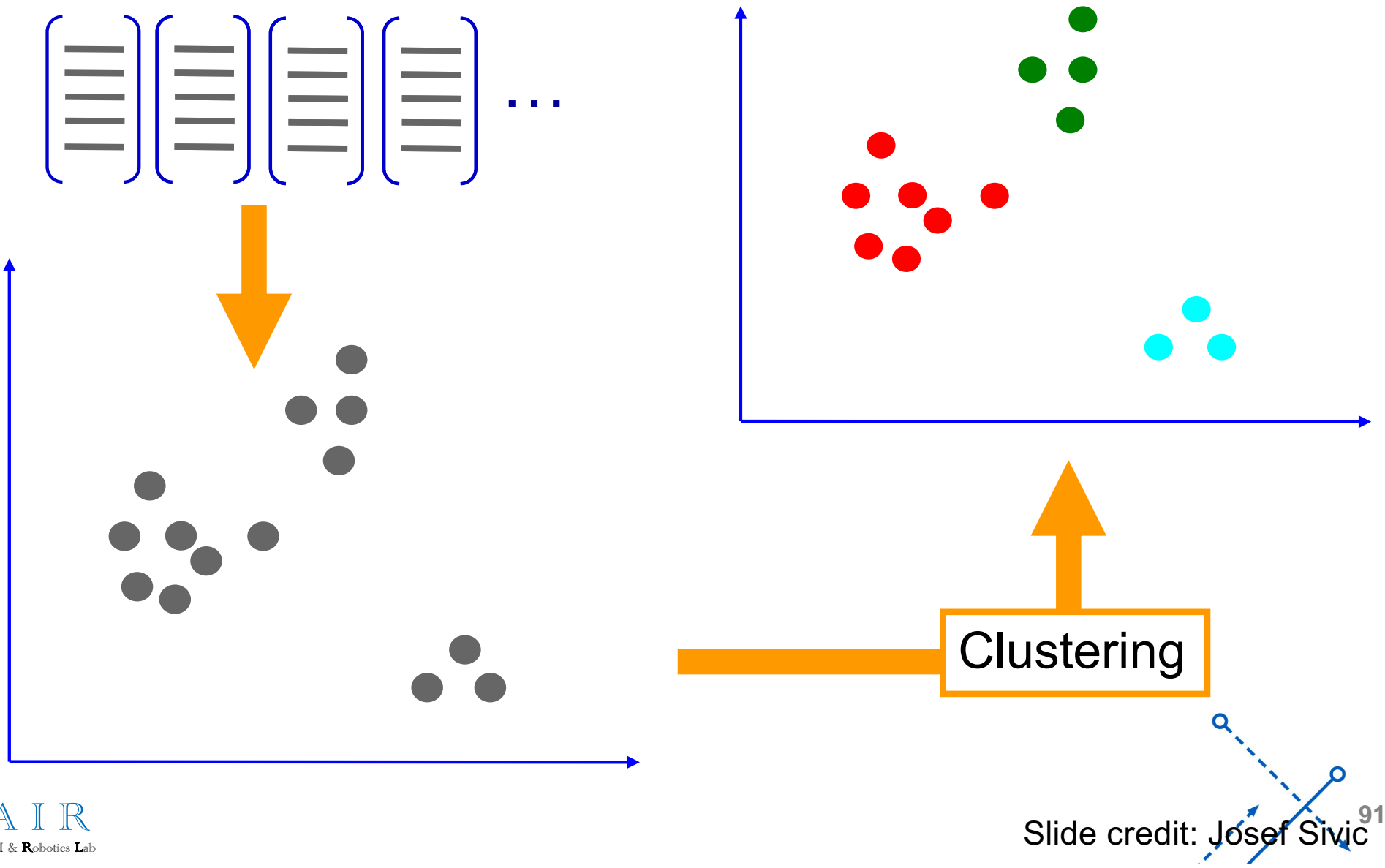


Slide credit: Josef Sivic

2. Learning the visual vocabulary

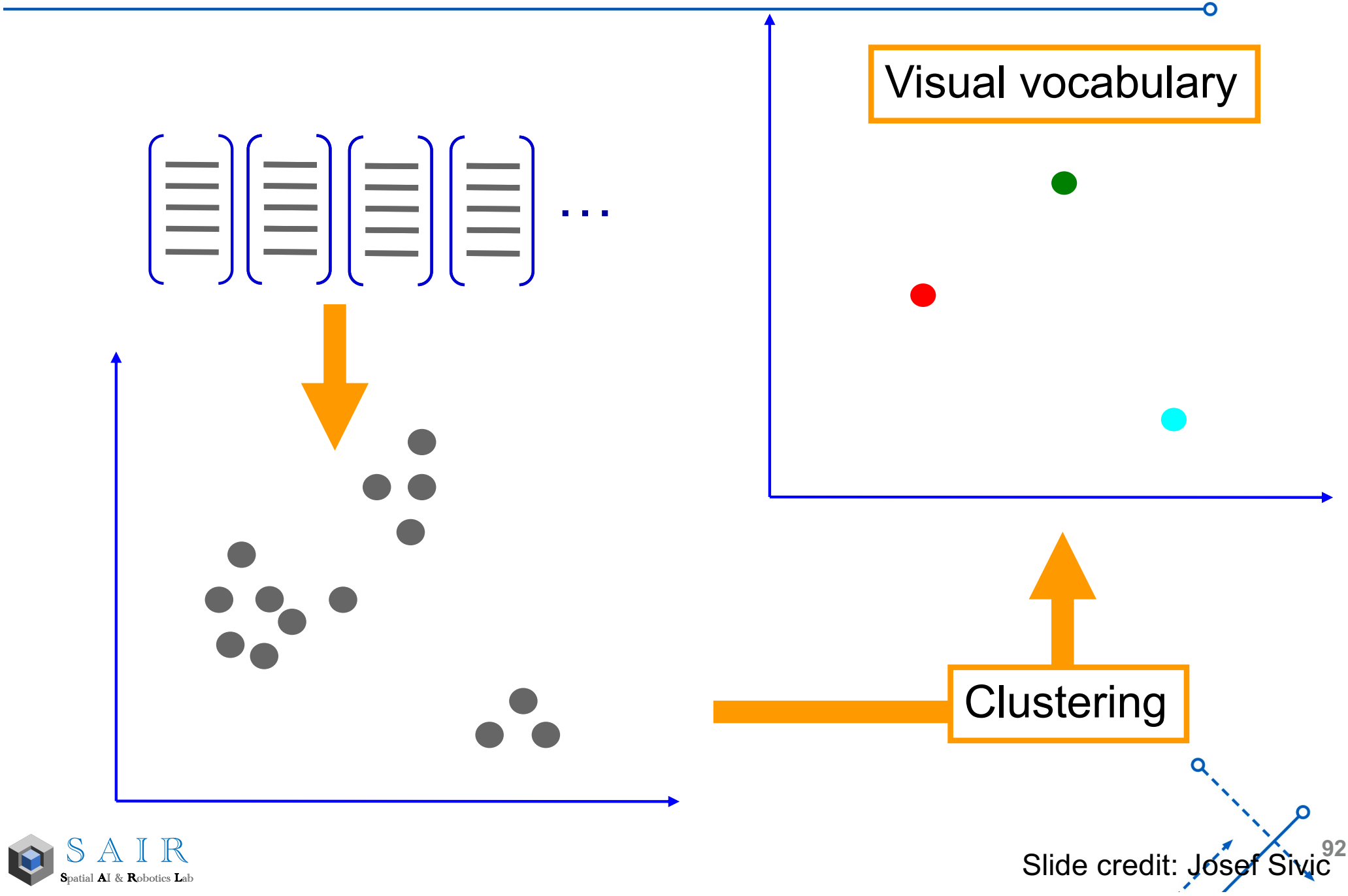


2. Learning the visual vocabulary



Slide credit: Josef Sivic

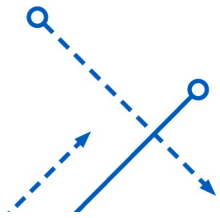
2. Learning the visual vocabulary



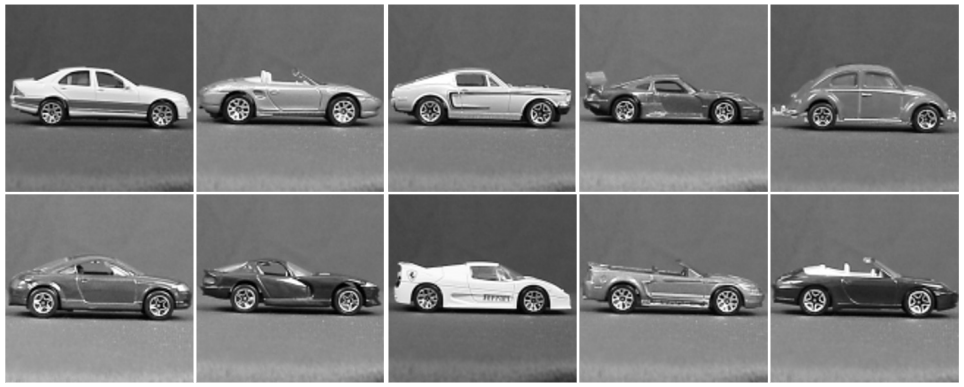
Slide credit: Josef Sivic

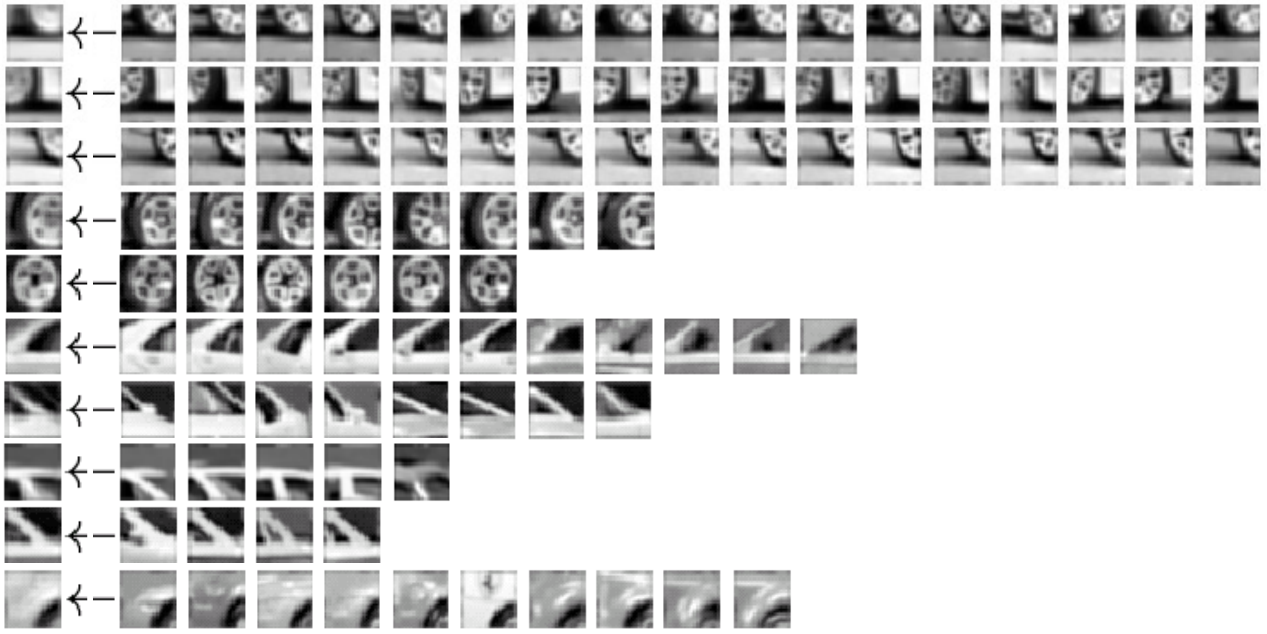
Clustering and vector quantization

- Clustering: learning a visual vocabulary or codebook
  - Unsupervised learning process
  - Each cluster center (e.g., k-means) becomes a codevector
  - Codebook can be learned on a separate training set
  - If the training set is sufficiently representative, the codebook will be “universal”
- The codebook is used for quantizing features
  - A *vector quantizer* takes a feature vector and maps it to the index of the nearest codevector in a codebook
  - Codebook = visual vocabulary
  - Codevector = visual word



Example Codebook

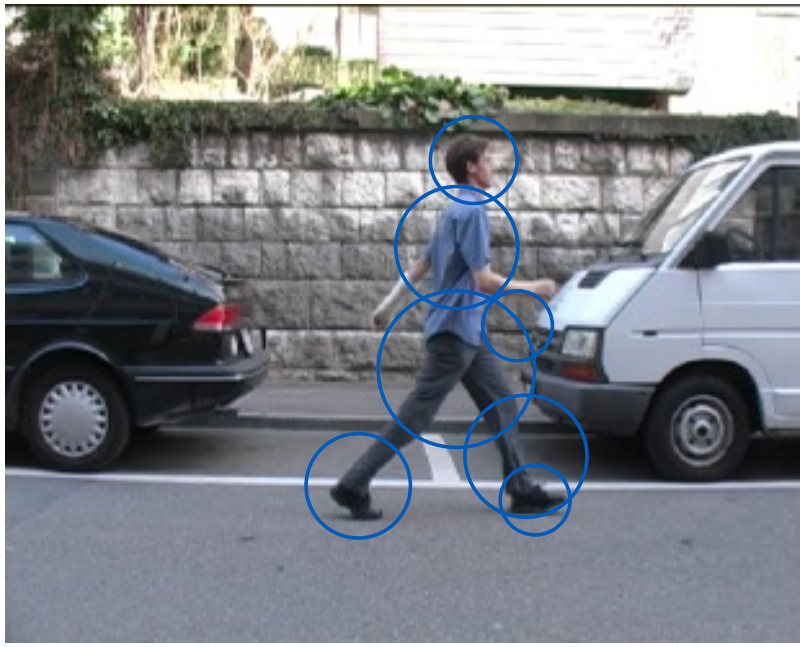


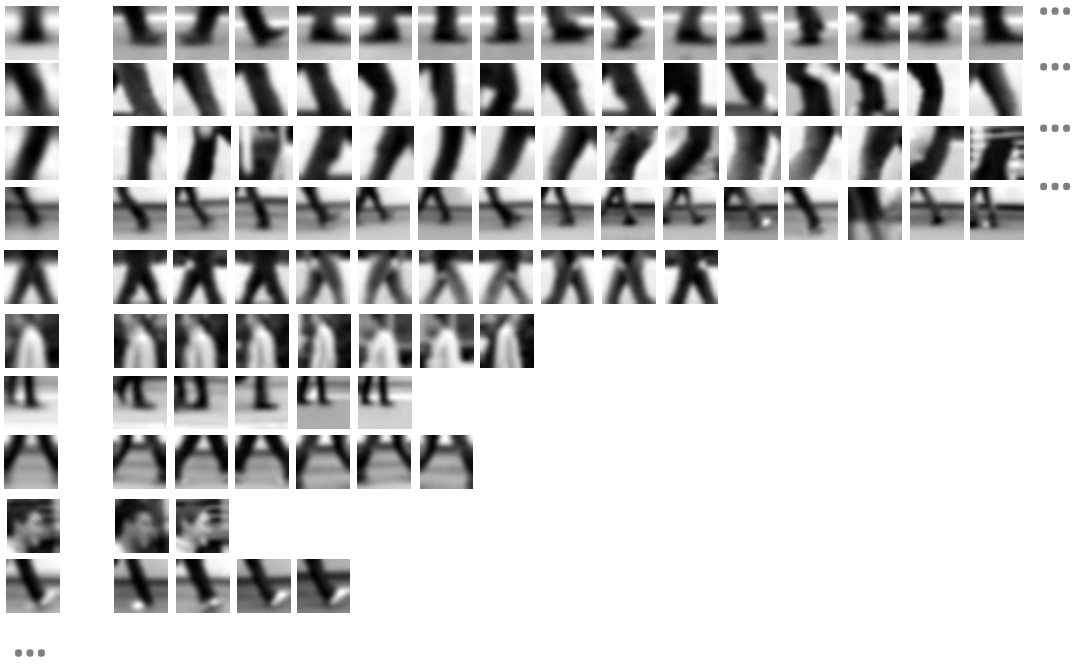


Appearance codebook

Source: B. Leibe

Another codebook





Appearance codebook

Source: B. Leibe

Visual vocabularies: Issues

- How to choose vocabulary size?
  - Too small: visual words not representative of all patches
  - Too large: quantization artifacts, overfitting
- Computational efficiency
  - Vocabulary trees (Nister & Stewenius, 2006)