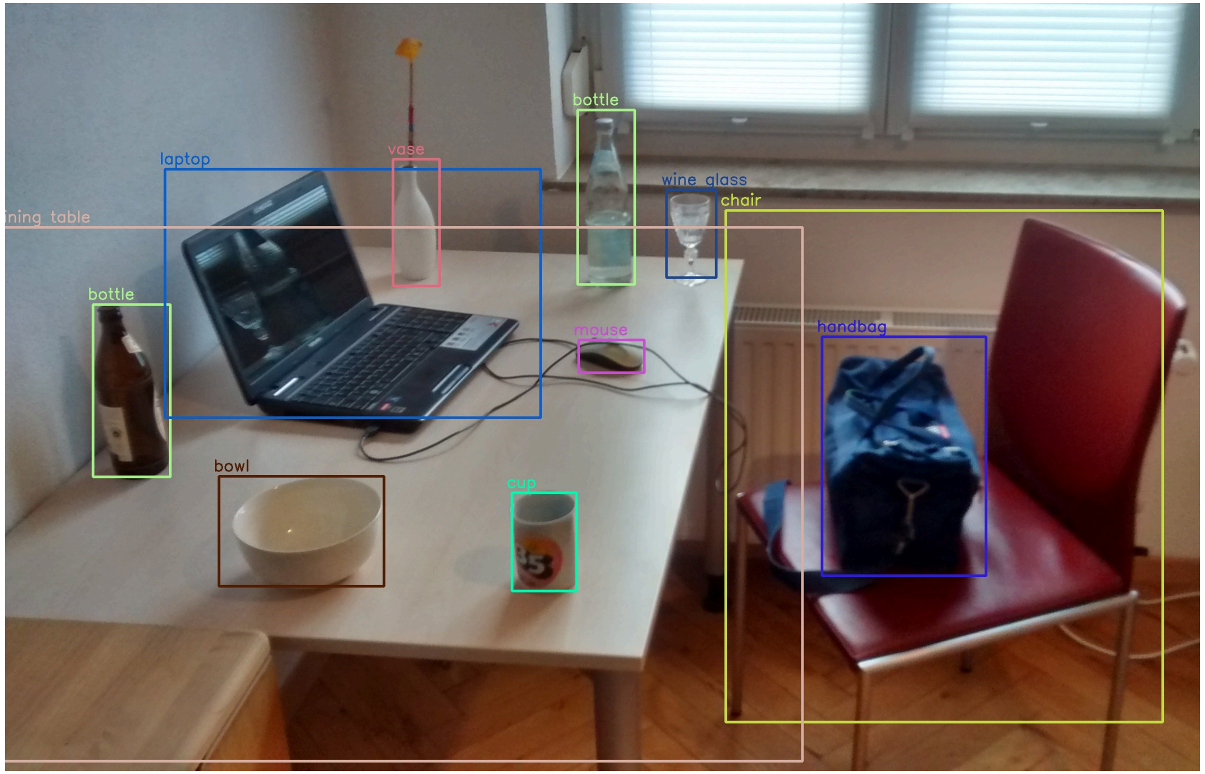
Georgia Tech

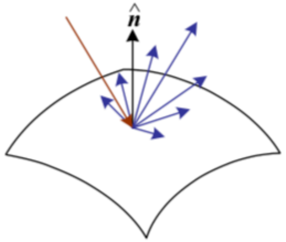
CS 4476: Computer Vision

Introduction to Object Recognition



Guest Lecturer: Judy Hoffman

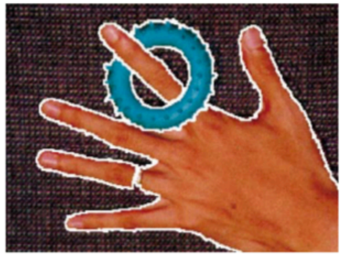
Slides by Lana Lazebnik except where indicated otherwise



2. Image Formation

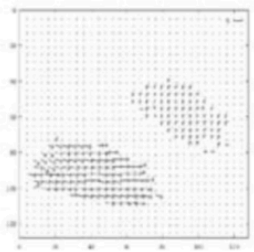
3. Image Processing

4. Features

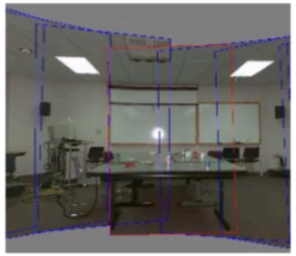


5. Segmentation

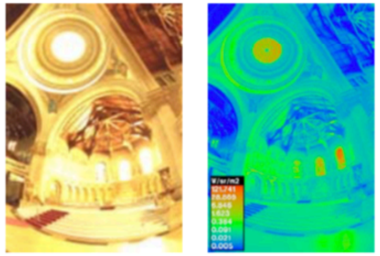
6-7. Structure from Motion



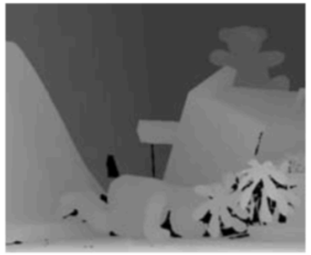
8. Motion



9. Stitching

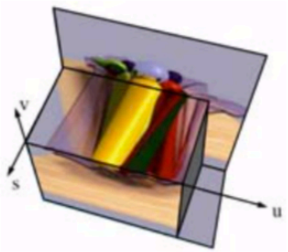


10. Computational Photography



11. Stereo

12. 3D Shape



13. Image-based Rendering

14. Recognition

Introduction to recognition

Source: Charley Harper

Outline

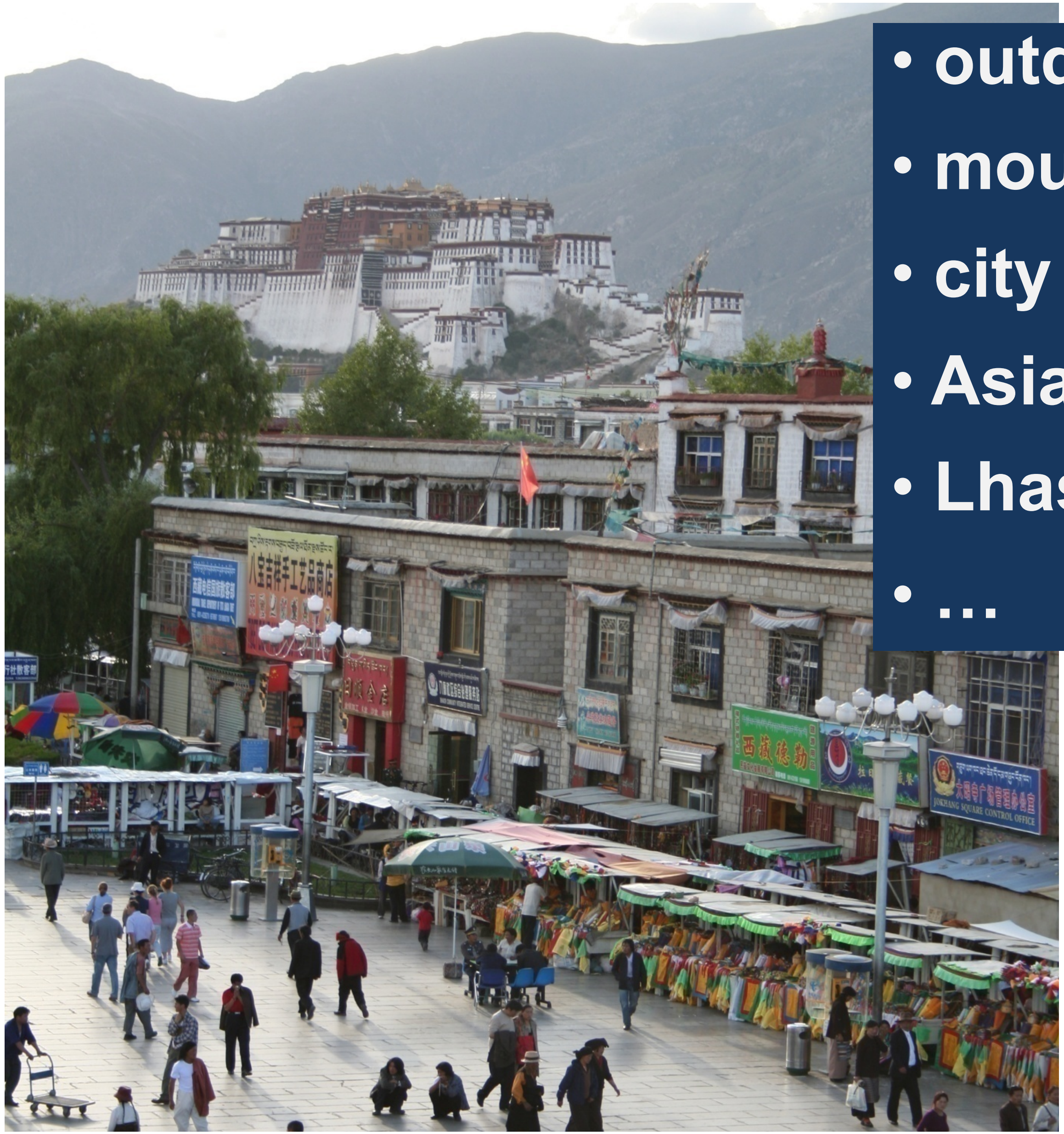
- Overview: recognition tasks
- Statistical learning approach
- Classic / Shallow Pipeline
  - “Bag of features” representation
  - Classifiers: nearest neighbor, linear, SVM
- Deep Pipeline
  - Neural Networks

Common Recognition Tasks

Adapted from
Fei-Fei Li

Image Classification and Tagging

What is this an image of?



- outdoor
- mountains
- city
- Asia
- Lhasa
- …

Adapted from
Fei-Fei Li

Object Detection

Localize!

find pedestrians

Adapted from Fei-Fei Li

Activity Recognition

What are they doing?

- walking
- shopping
- rolling a cart
- sitting
- talking
- ...

Adapted from Fei-Fei Li

Semantic Segmentation

Label Every Pixel

Adapted from
Fei-Fei Li

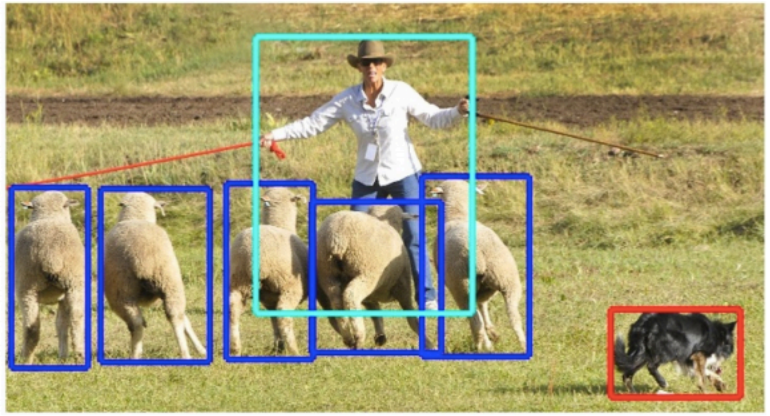
Semantic Segmentation

Label Every Pixel

Adapted from
Fei-Fei Li

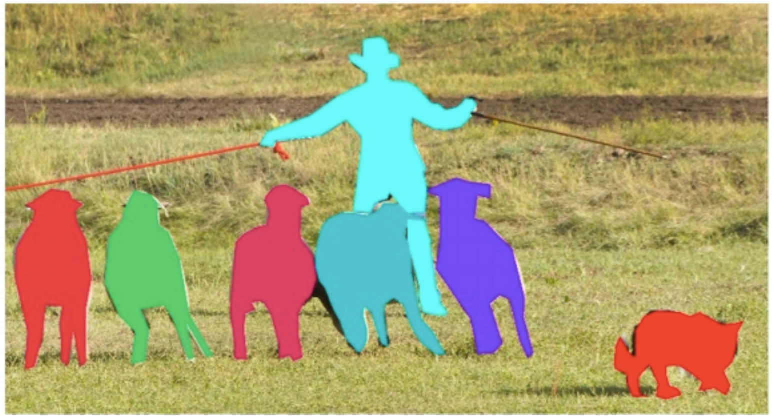
Detection, semantic and instance segmentation

image classification



object detection

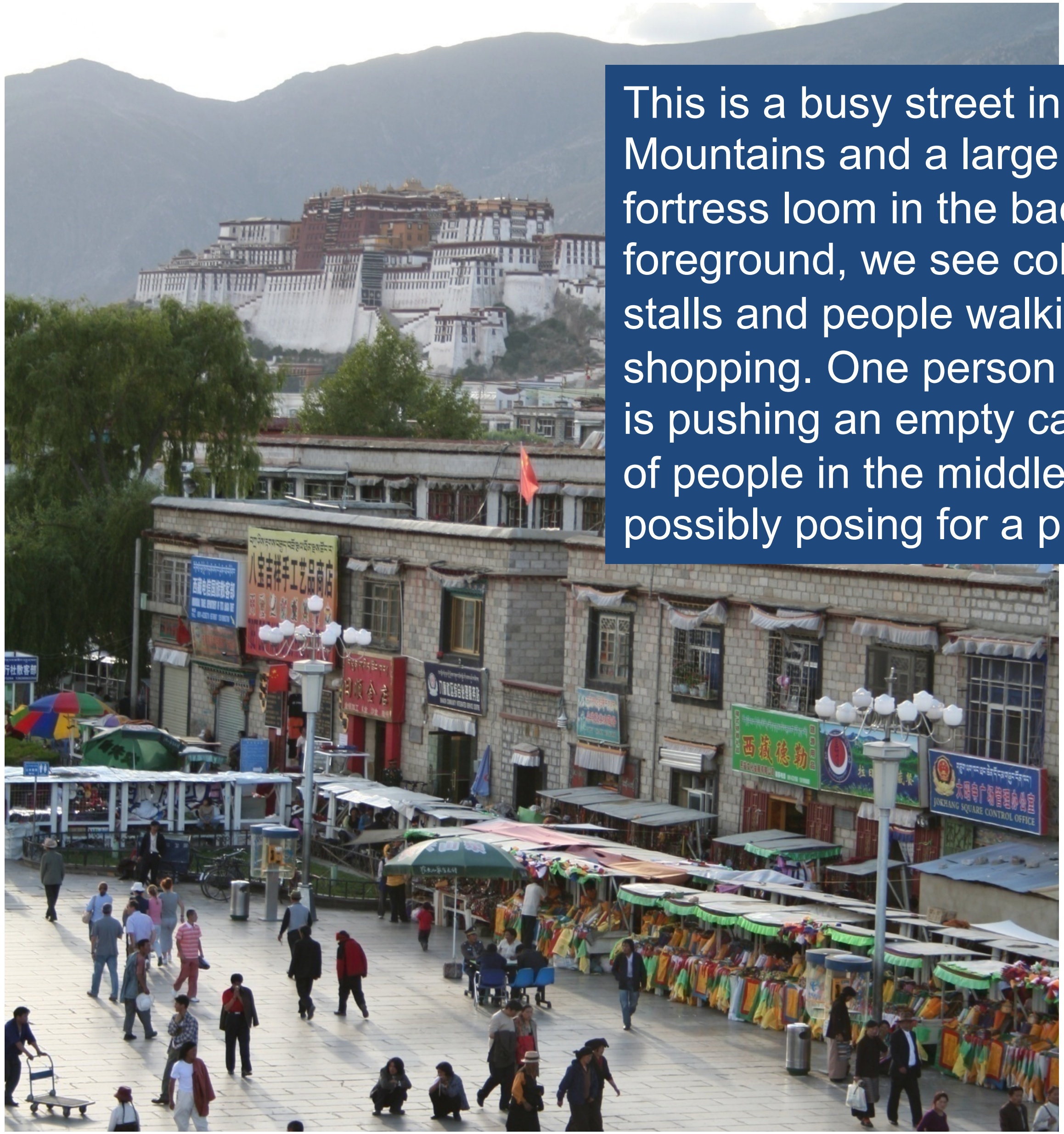
semantic segmentation



instance segmentation

Image source

Image Description



This is a busy street in an Asian city. Mountains and a large palace or fortress loom in the background. In the foreground, we see colorful souvenir stalls and people walking around and shopping. One person in the lower left is pushing an empty cart, and a couple of people in the middle are sitting, possibly posing for a photograph.

Adapted from
Fei-Fei Li

Image classification

The statistical learning framework

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

f() = “apple”

f() = “tomato”

f() = “cow”

The statistical learning framework

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

"apple"

output   prediction function   feature representation

Training

Given labeled *training set*
$\{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_N, y_N)\}$

Learn the prediction function f, by minimizing prediction error on *training set*

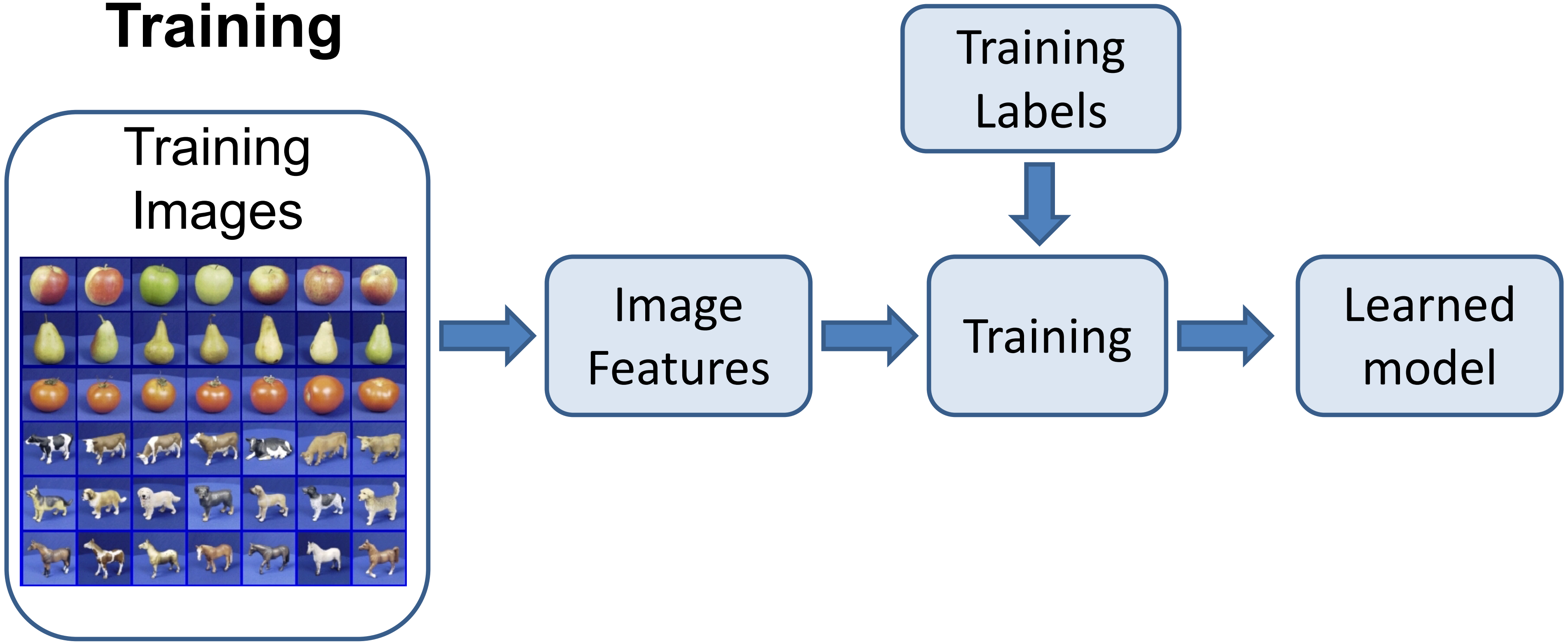
Testing

Given unlabeled *test instance*
$\boldsymbol{x}$

Predict the output label y as
$y = f(\boldsymbol{x})$

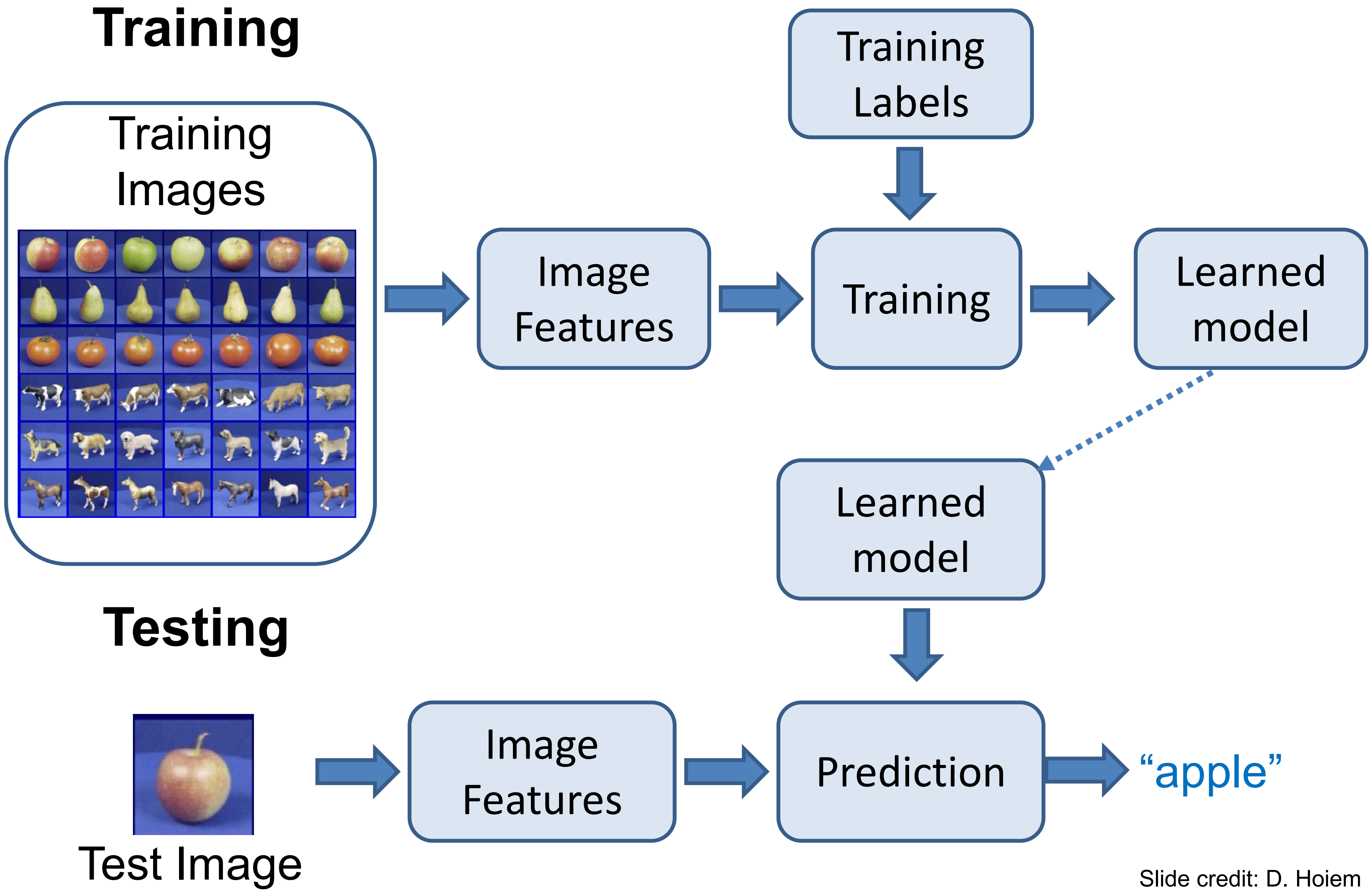
Steps

Training



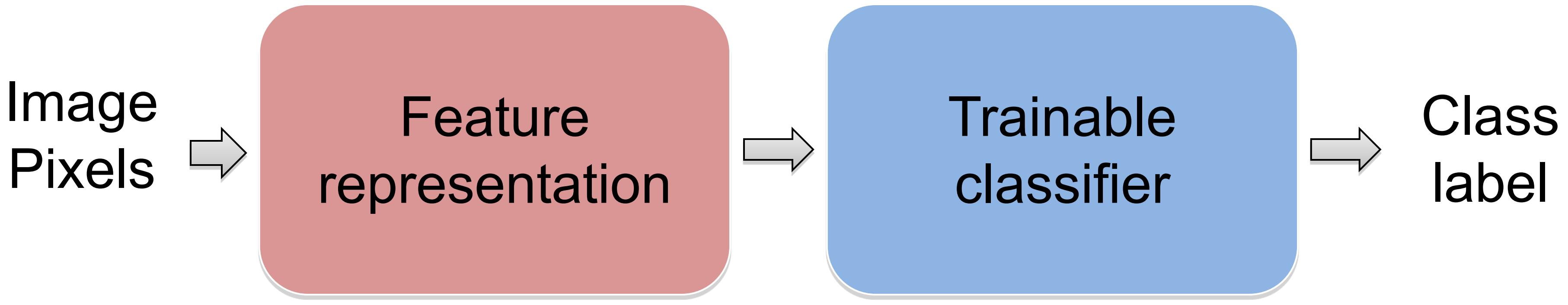
Slide credit: D. Hoiem

Steps



Slide credit: D. Hoiem

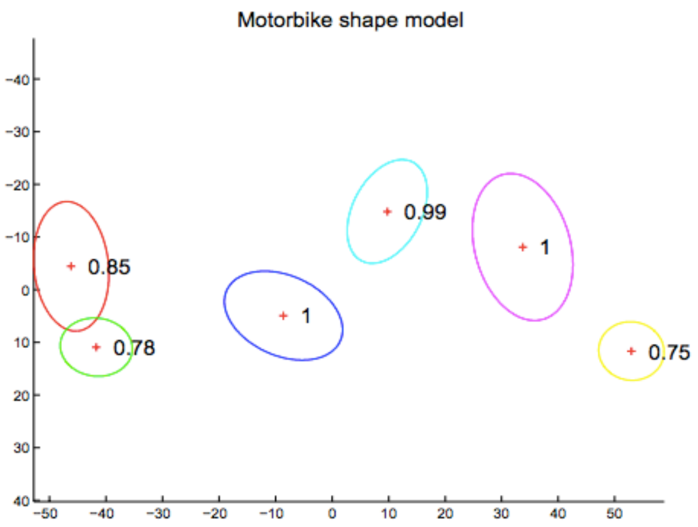
“Classic” recognition pipeline

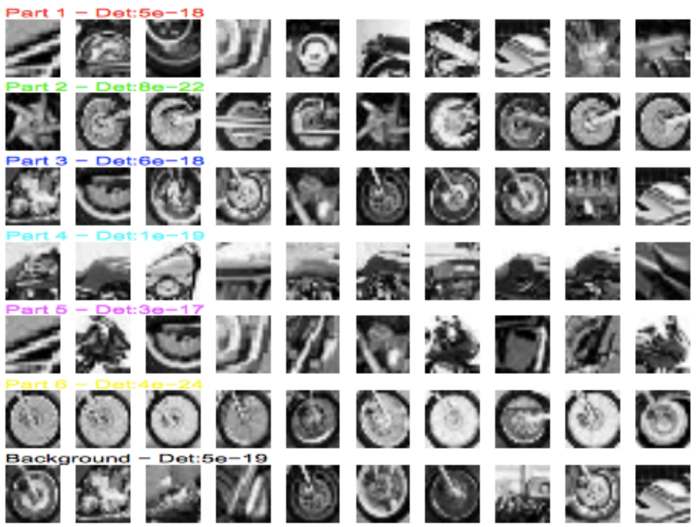


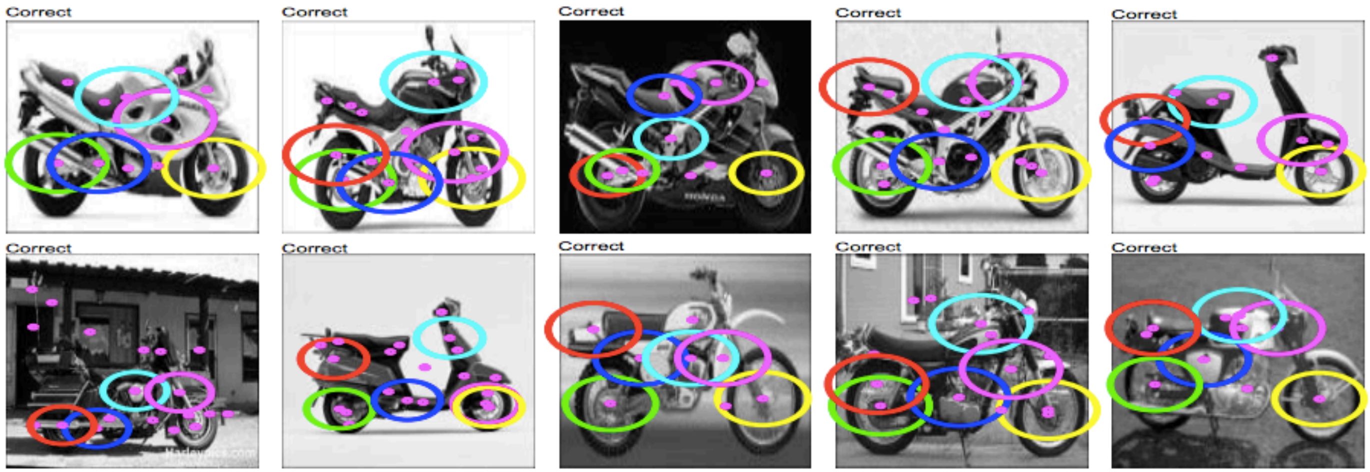
- Hand-crafted feature representation
- Off-the-shelf trainable classifier

“Classic” representation: Bag of features

Motivation 1: Part-based models

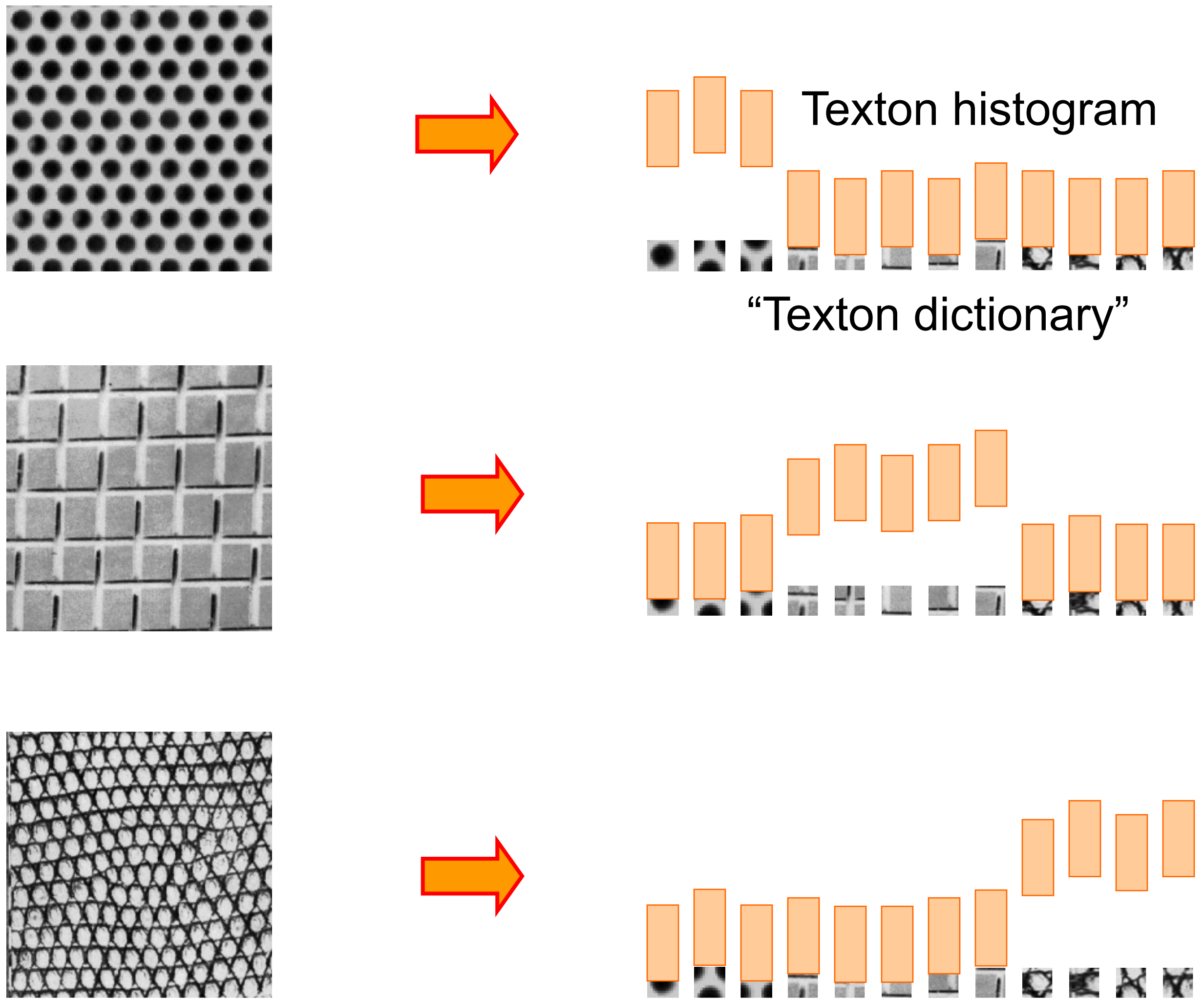






Weber, Welling & Perona (2000), Fergus, Perona & Zisserman (2003)

Motivation 2: Texture models



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

Motivation 3: Bags of words

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

Motivation 3: Bags of words

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

Motivation 3: Bags of words

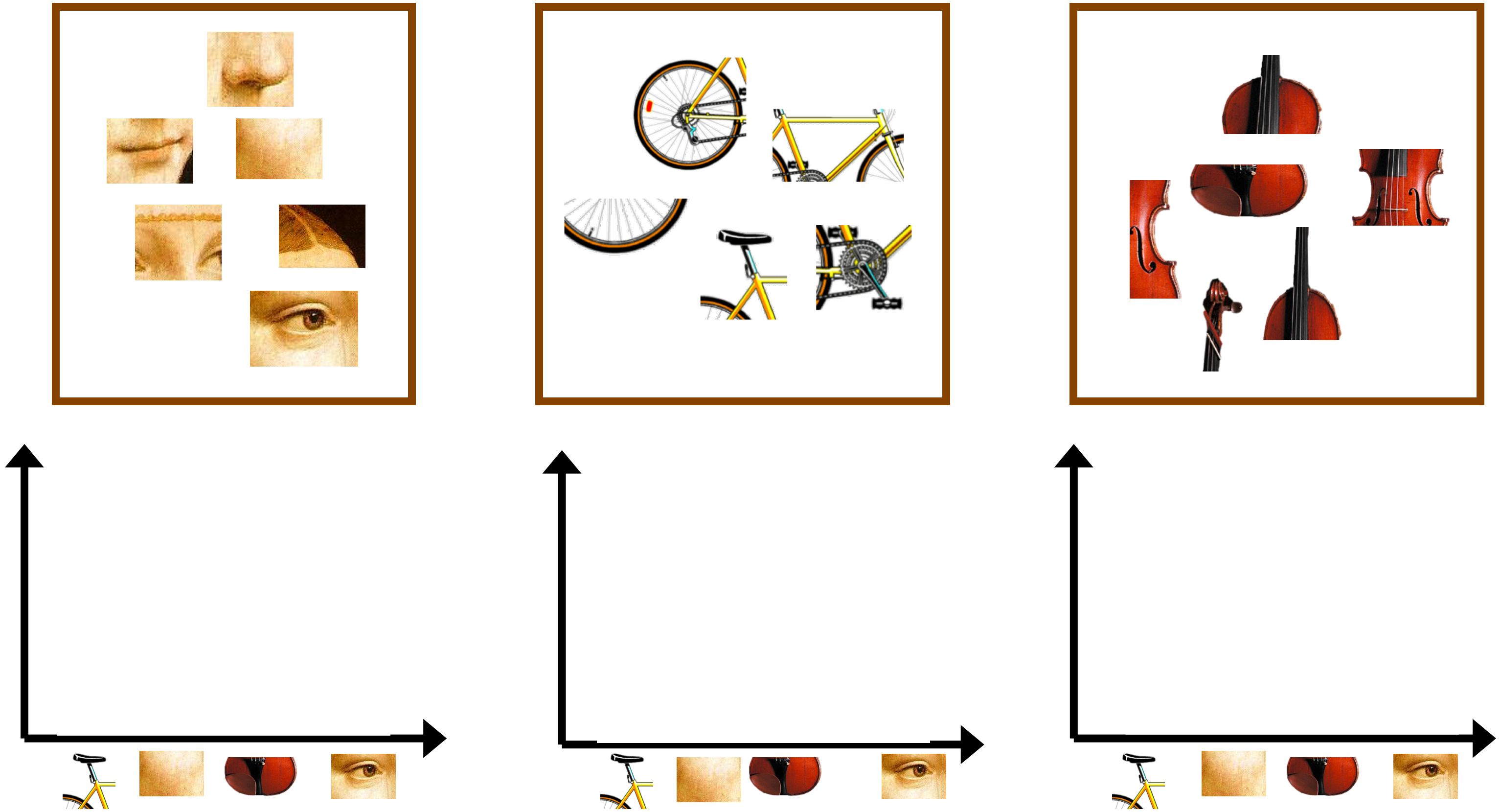
Orderless document representation: frequencies of words from a dictionary Salton & McGill (1983)

Motivation 3: Bags of words

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

Bag of features: Outline

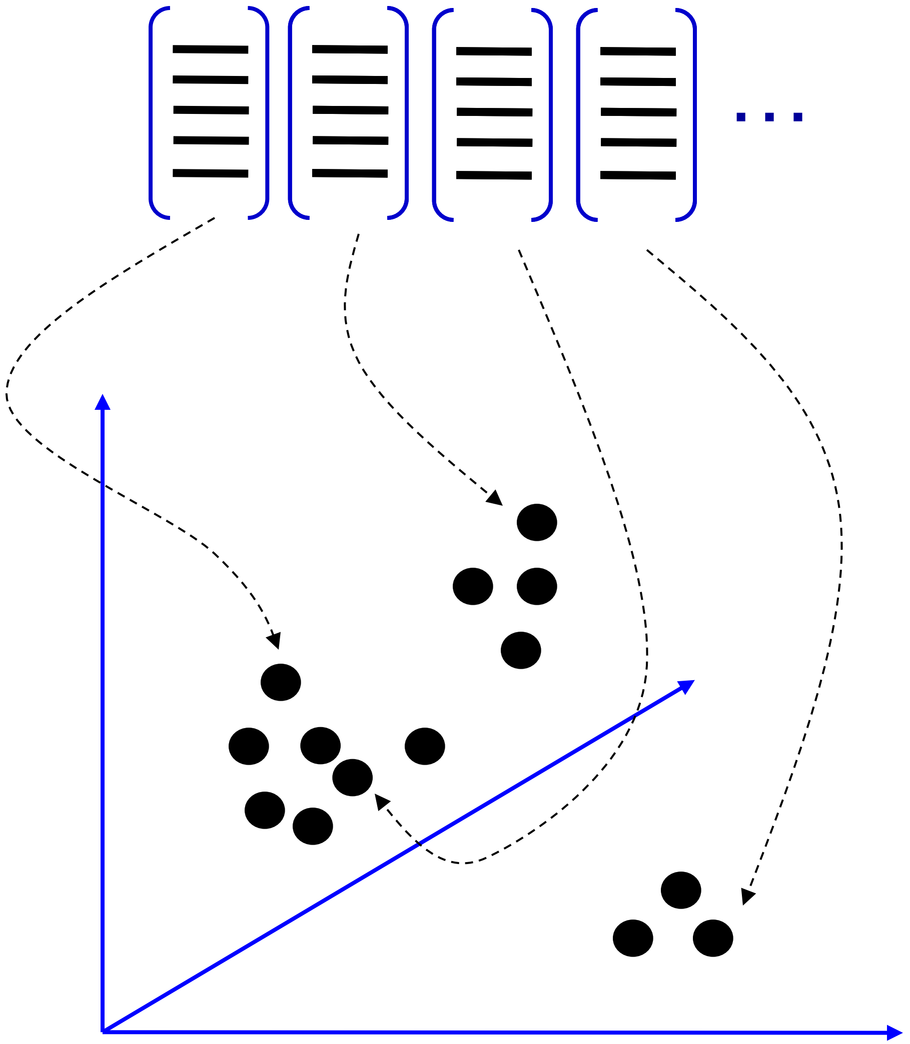
1. Extract local features
2. Learn "visual vocabulary"
3. Quantize local features using visual vocabulary
4. Represent images by frequencies of "visual words"



1. Local feature extraction

Sample patches and extract descriptors

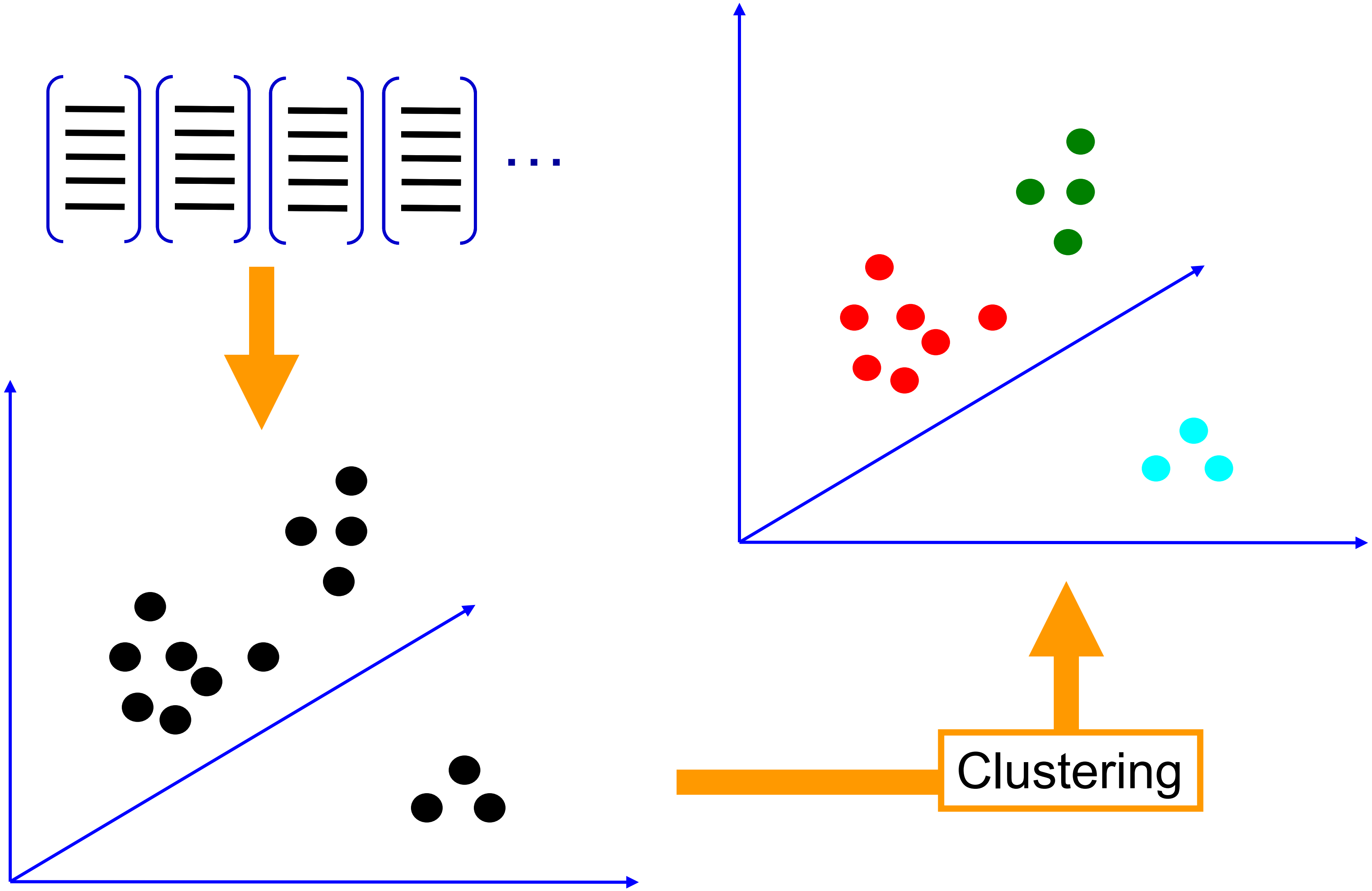
2. Learning the visual vocabulary



Extracted descriptors from the training set

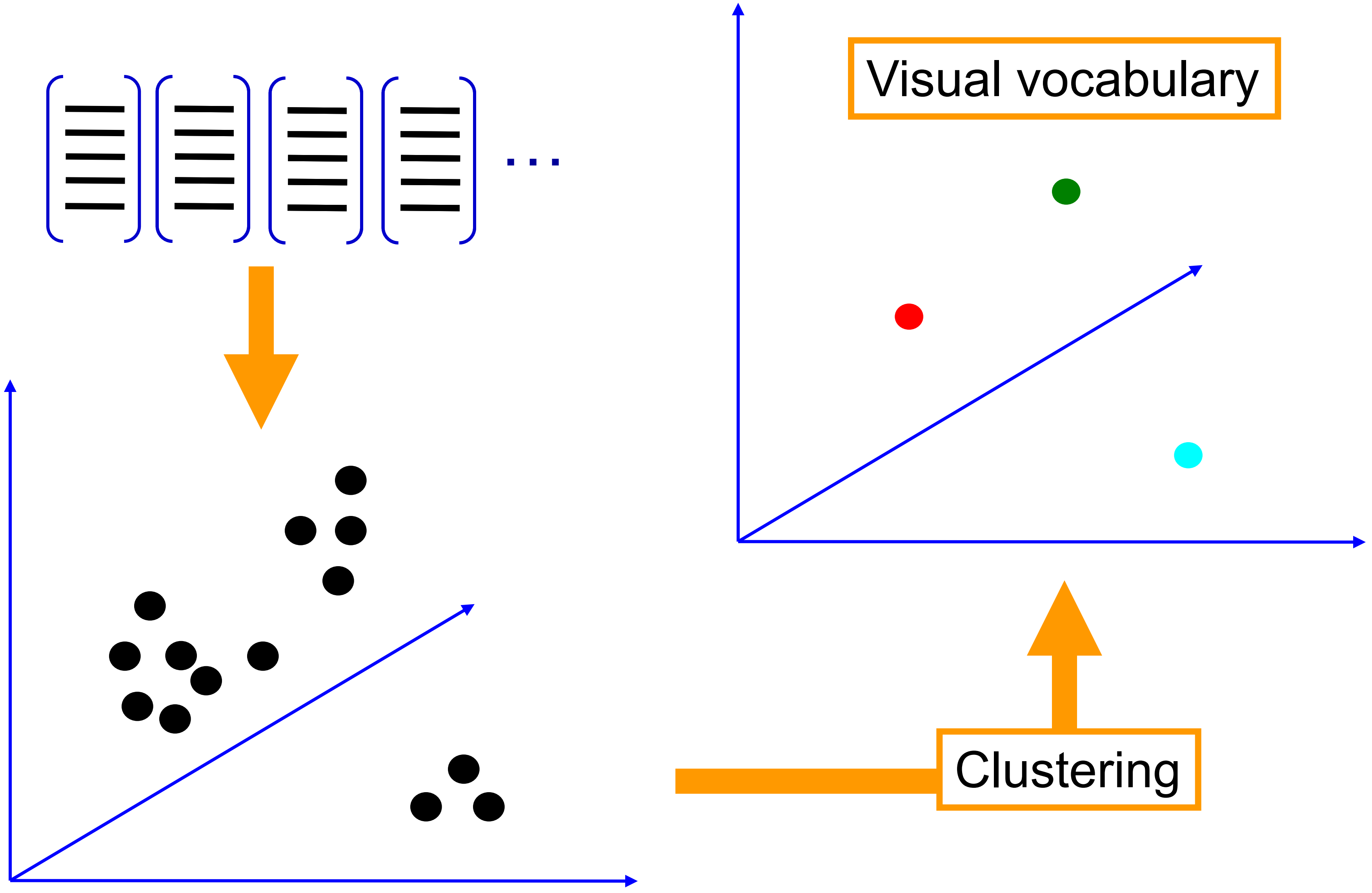
Slide credit: Josef Sivic

2. Learning the visual vocabulary



Slide credit: Josef Sivic

2. Learning the visual vocabulary



Slide credit: Josef Sivic

Recall: K-means clustering

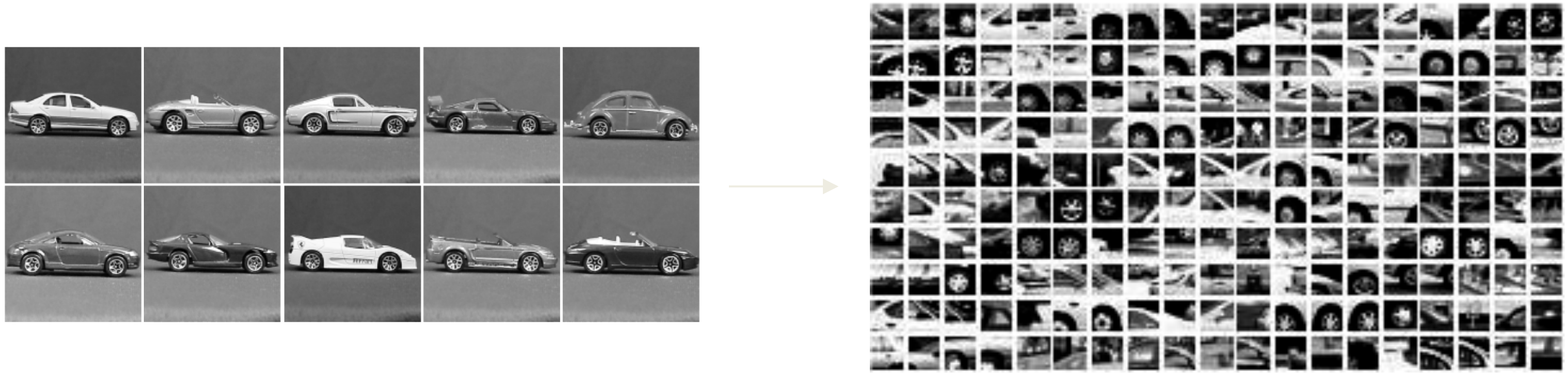
Goal: minimize sum of squared Euclidean distances between features $\mathbf{x}_i$ and their nearest cluster centers $\mathbf{m}_k$

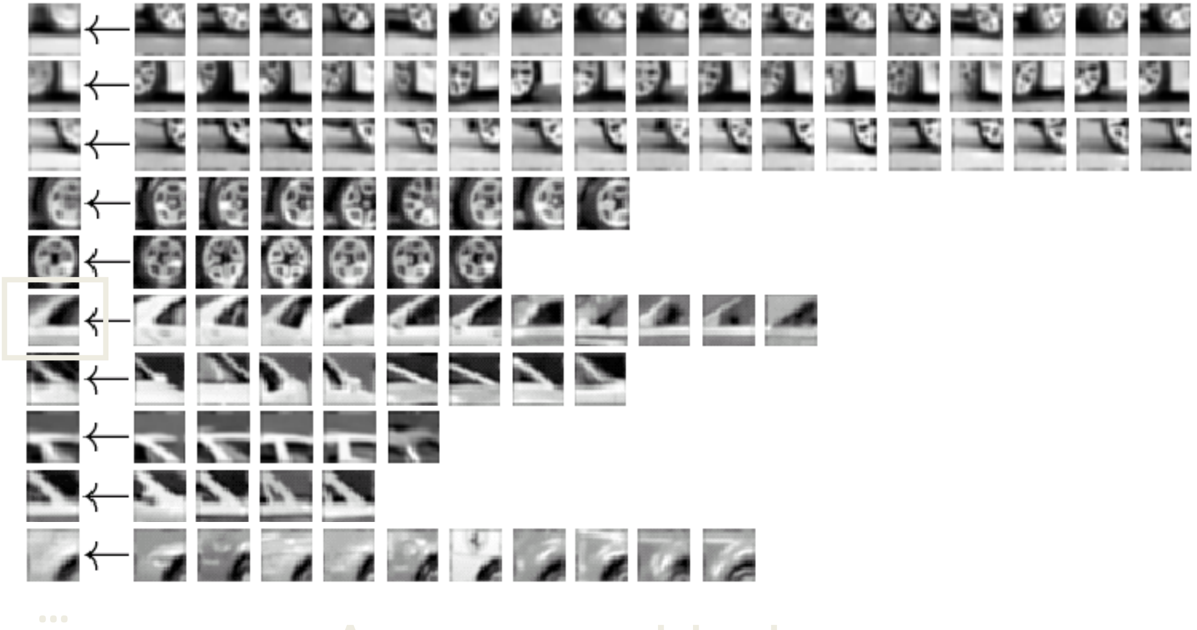
$$D(X, M) = \sum_{\text{cluster } k} \sum_{\substack{\text{point } i \text{ in} \\ \text{cluster } k}} (\mathbf{x}_i - \mathbf{m}_k)^2$$

Algorithm:

- Randomly initialize K cluster centers
- Iterate until convergence:
  - Assign each feature to the nearest center
  - Recompute each cluster center as the mean of all features assigned to it

Recall: Visual vocabularies



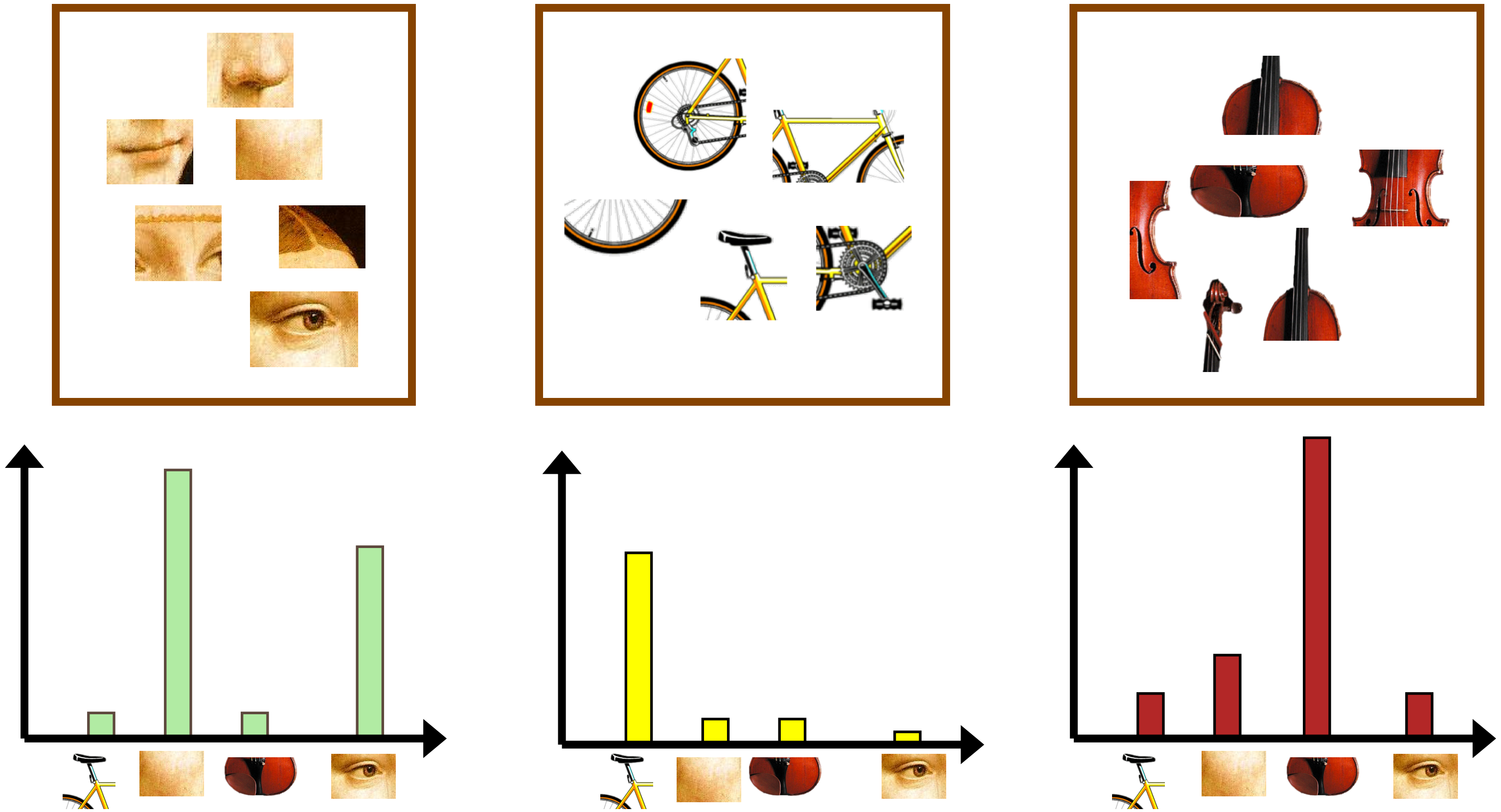


Appearance codebook

Source: B. Leibe

Bag of features: Outline

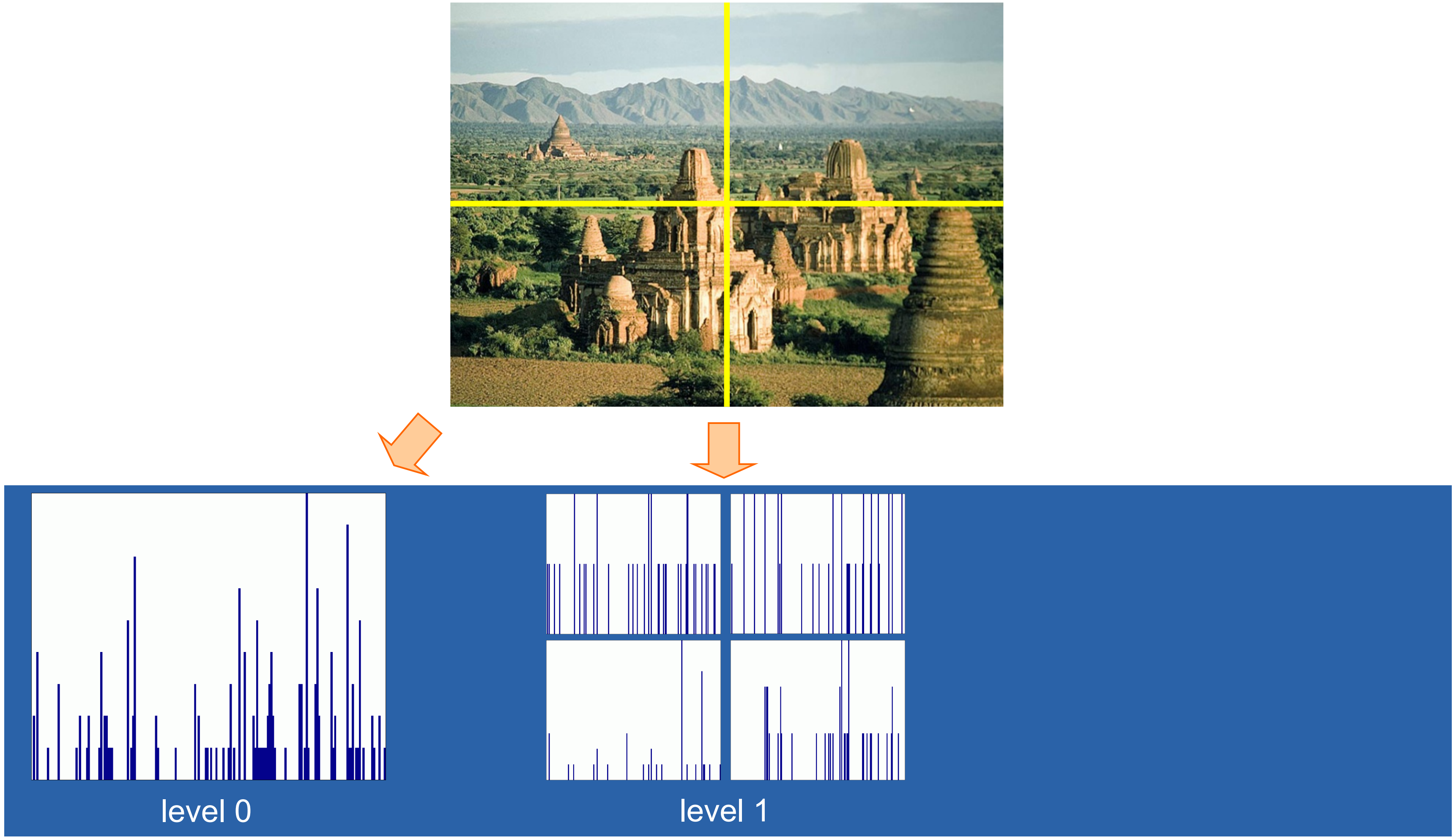
1. Extract local features
2. Learn "visual vocabulary"
3. **Quantize local features using visual vocabulary**
4. **Represent images by frequencies of "visual words"**



Spatial pyramids

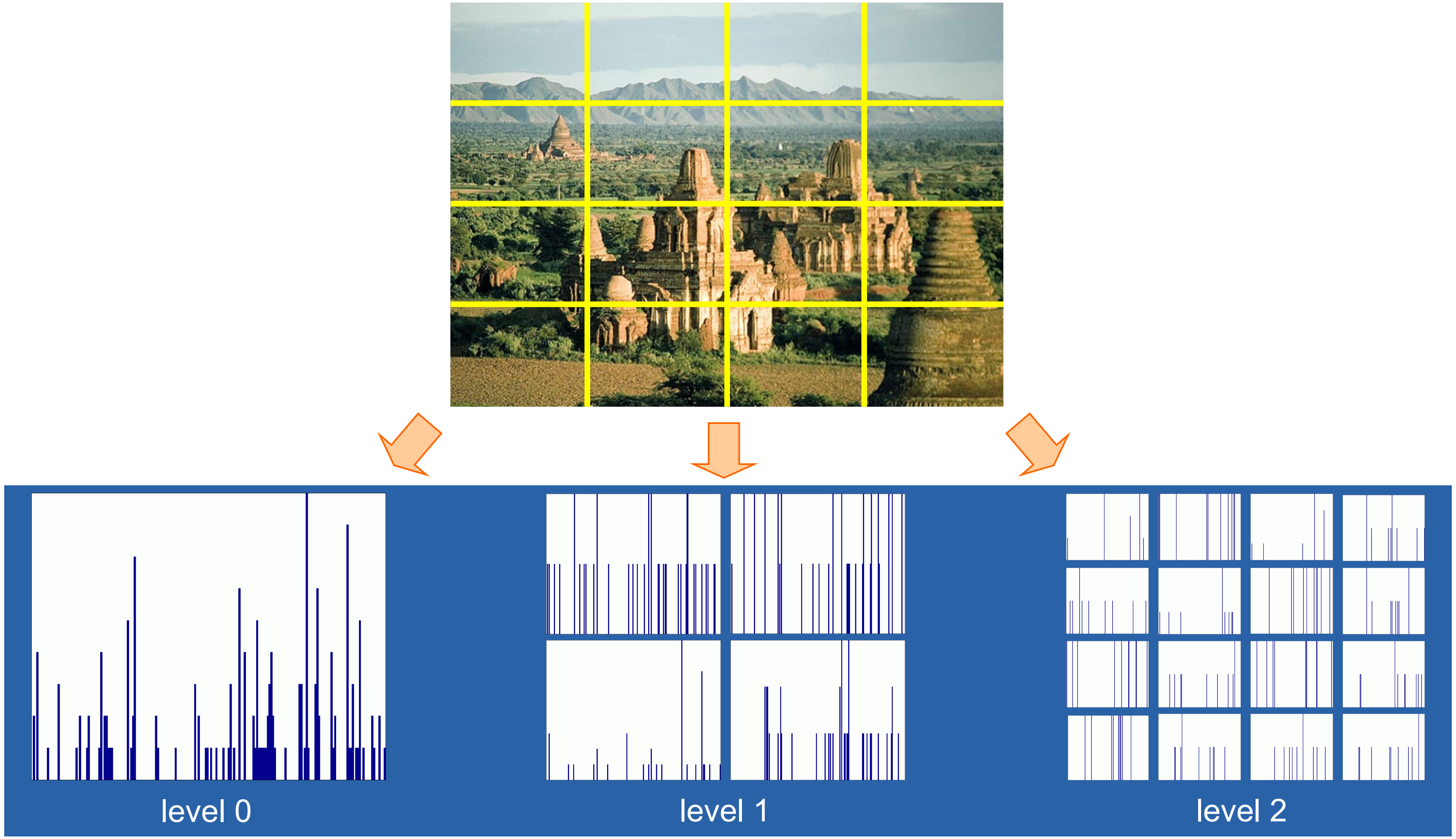
Lazebnik, Schmid & Ponce (CVPR 2006)

Spatial pyramids



Lazebnik, Schmid & Ponce (CVPR 2006)

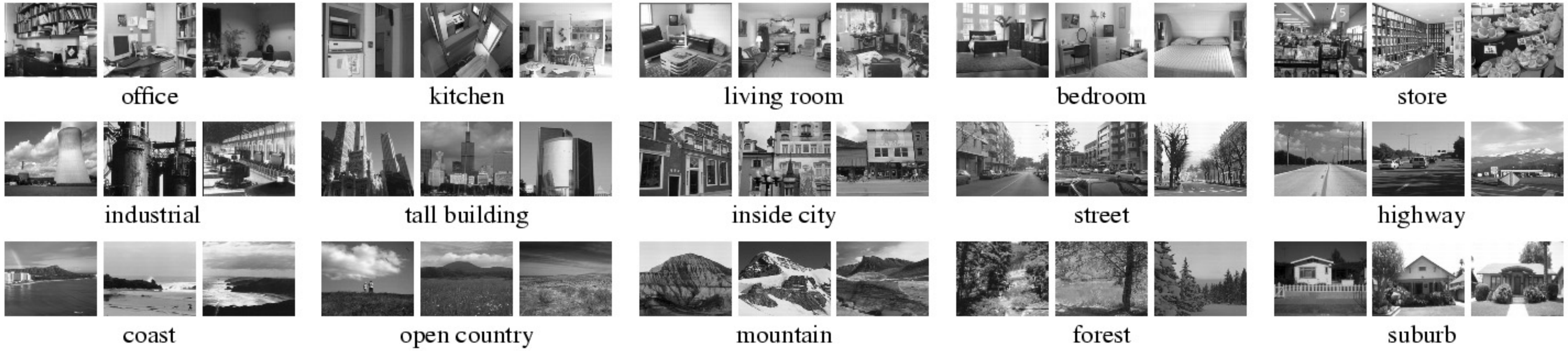
Spatial pyramids



Lazebnik, Schmid & Ponce (CVPR 2006)

Spatial pyramids

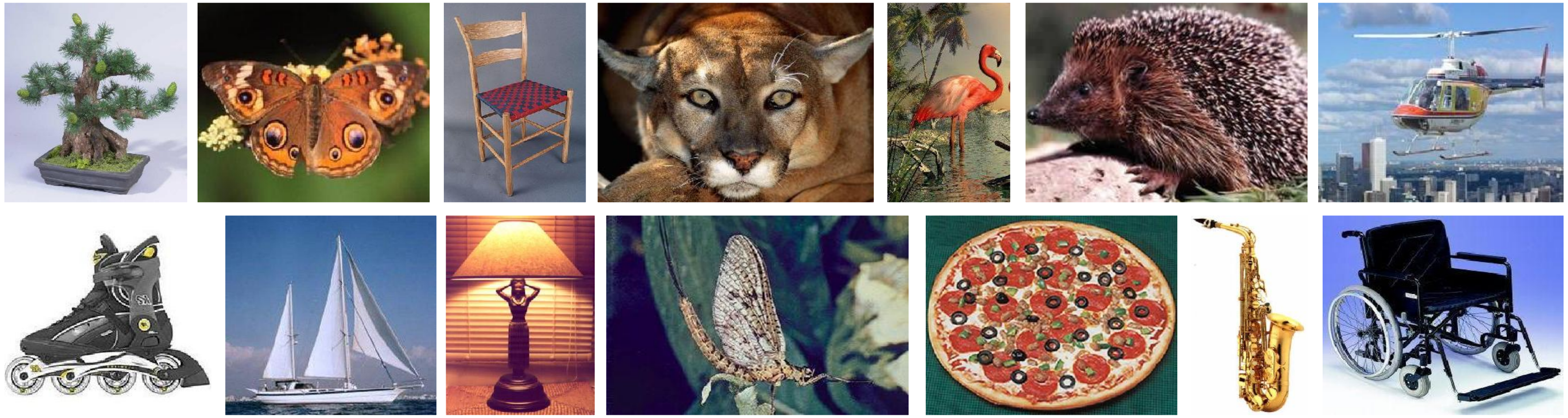
Scene classification results



	Weak features (vocabulary size: 16)		Strong features (vocabulary size: 200)	
Level	Single-level	Pyramid	Single-level	Pyramid
0 (1×1)	45.3 ±0.5		72.2 ±0.6	
1 (2×2)	53.6 ±0.3	56.2 ±0.6	77.9 ±0.6	79.0 ±0.5
2 (4×4)	61.7 ±0.6	64.7 ±0.7	79.4 ±0.3	**81.1** ±0.3
3 (8×8)	63.3 ±0.8	**66.8** ±0.6	77.2 ±0.4	80.7 ±0.3

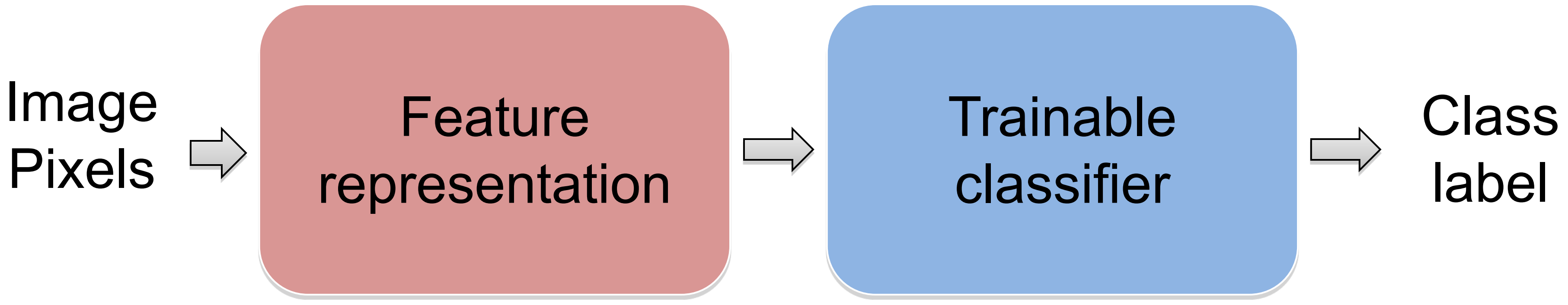
Spatial pyramids

Caltech101 classification results



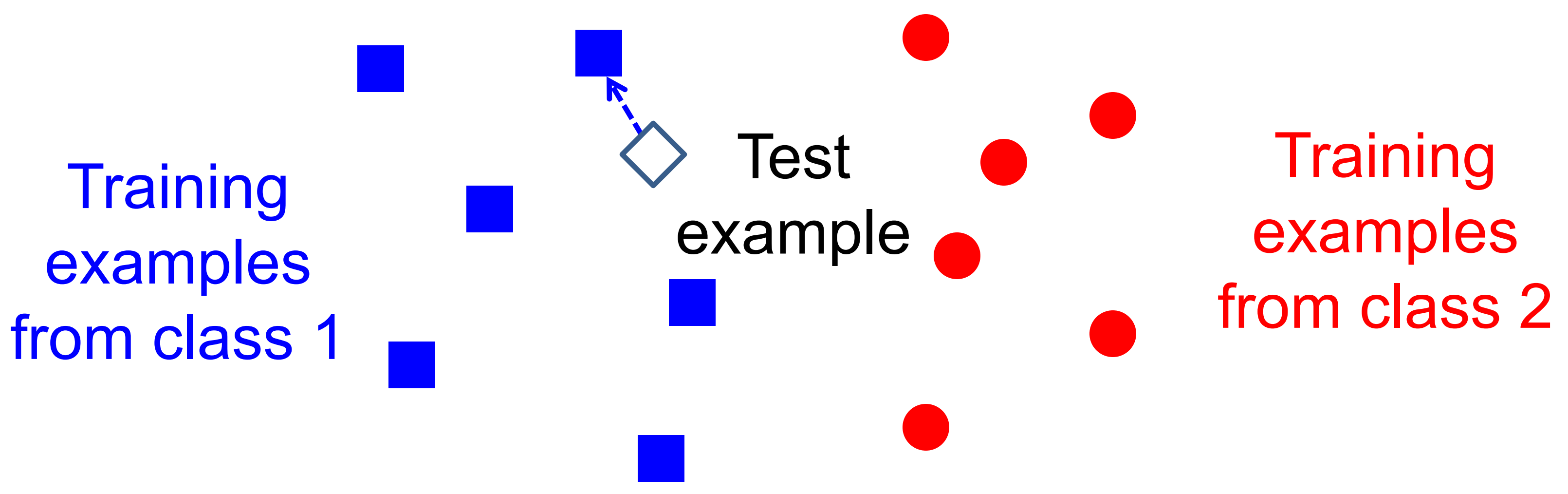
	Weak features (16)		Strong features (200)	
Level	Single-level	Pyramid	Single-level	Pyramid
0	15.5 ±0.9		41.2 ±1.2	
1	31.4 ±1.2	32.8 ±1.3	55.9 ±0.9	57.0 ±0.8
2	47.2 ±1.1	49.3 ±1.4	63.6 ±0.9	**64.6** ±0.8
3	52.2 ±0.8	**54.0** ±1.1	60.3 ±0.9	64.6 ±0.7

“Classic” recognition pipeline



- Hand-crafted feature representation
- Off-the-shelf trainable classifier

Classifiers: Nearest neighbor



f(**x**) = label of the training example nearest to **x**

- All we need is a distance or similarity function for our inputs
- No training required!

Functions for comparing histograms

- L1 distance: $$D(h_1,h_2)=\sum_{i=1}^{N}|\,h_1(i)-h_2(i)\,|$$
- χ^2 distance: $$D(h_1,h_2)=\sum_{i=1}^{N}\frac{\left(h_1(i)-h_2(i)\right)^2}{h_1(i)+h_2(i)}$$
- Quadratic distance (*cross-bin distance*):

$$D(h_1,h_2)=\sum_{i,j}A_{ij}(h_1(i)-h_2(j))^2$$

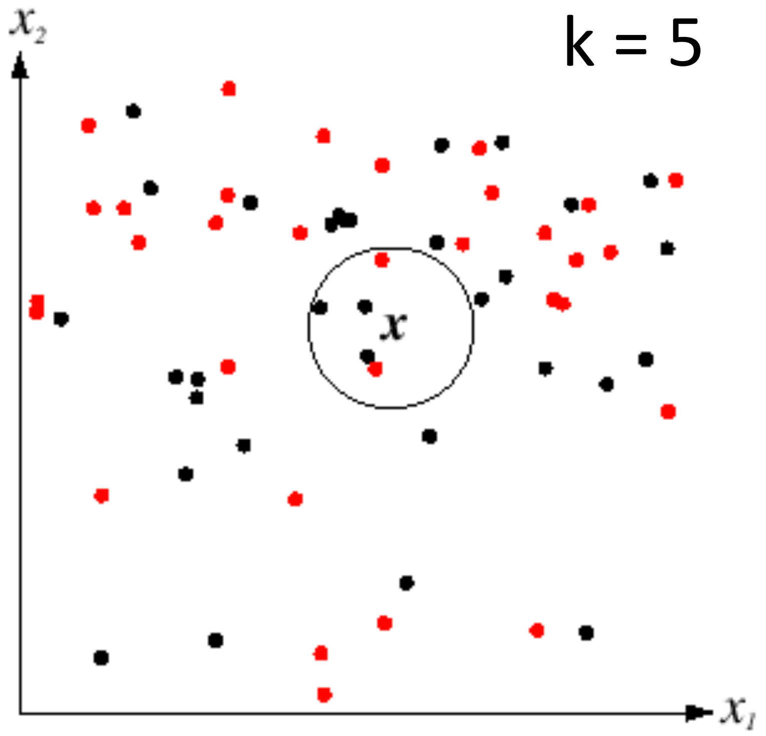
- Histogram intersection (similarity function):

$$I(h_1,h_2)=\sum_{i=1}^{N}\min(h_1(i),h_2(i))$$

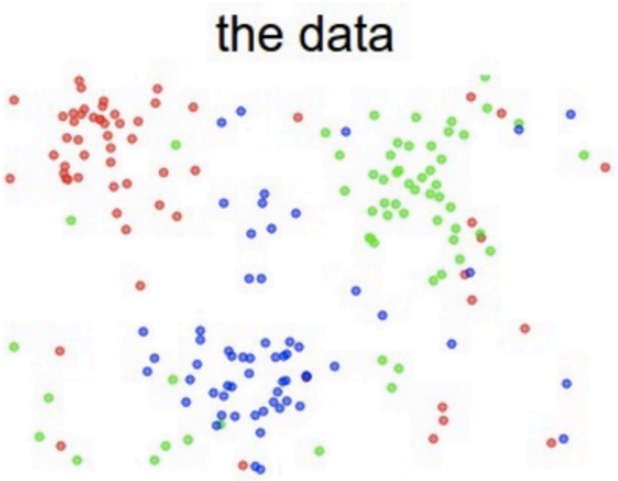
K-nearest neighbor classifier

- For a new point, find the k closest points from training data
- Vote for class label with labels of the k points

What is the label for *x*?



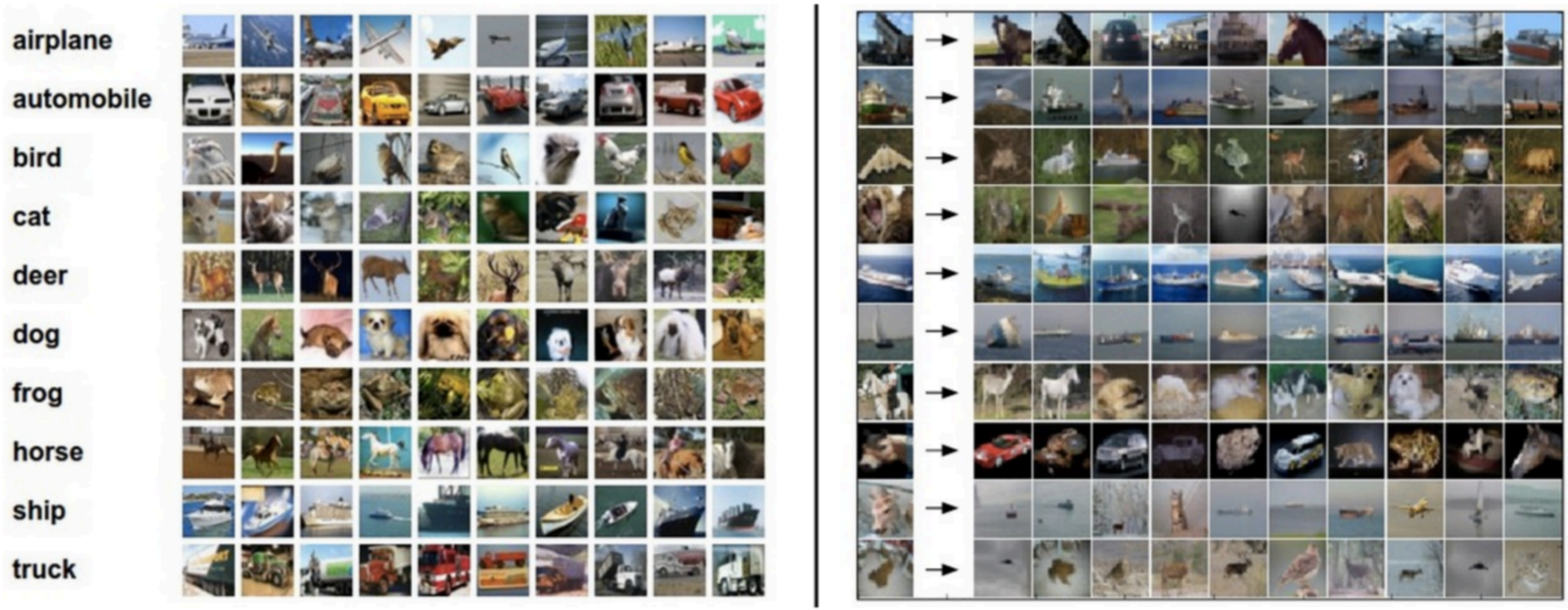
Quiz: K-nearest neighbor classifier



Which classifier is more robust to *outliers*?

Credit: Andrej Karpathy, http://cs231n.github.io/classification/

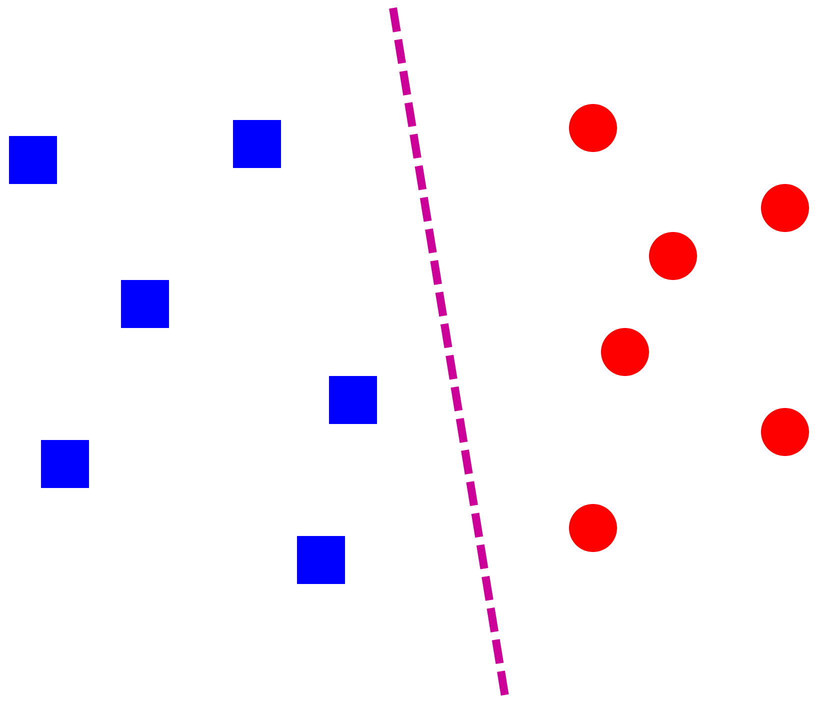
K-nearest neighbor classifier



Left: Example images from the CIFAR-10 dataset. Right: first column shows a few test images and next to each we show the top 10 nearest neighbors in the training set according to pixel-wise difference.

Credit: Andrej Karpathy, http://cs231n.github.io/classification/

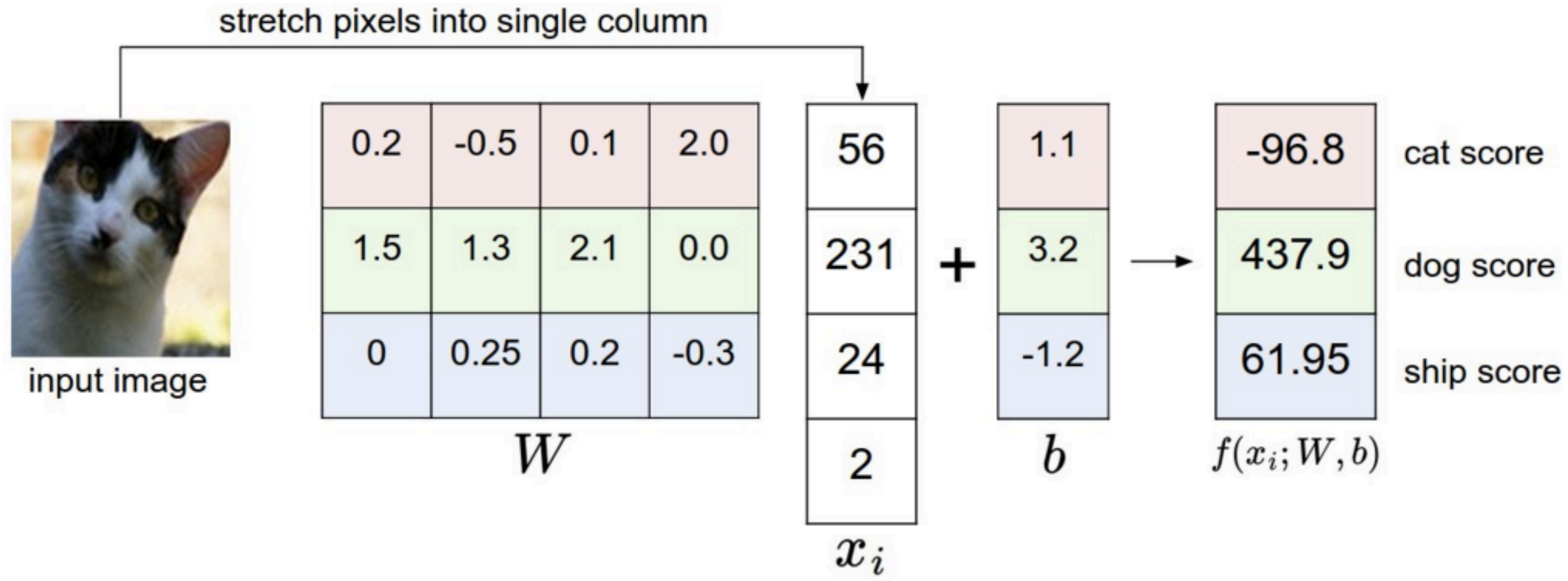
Linear classifiers

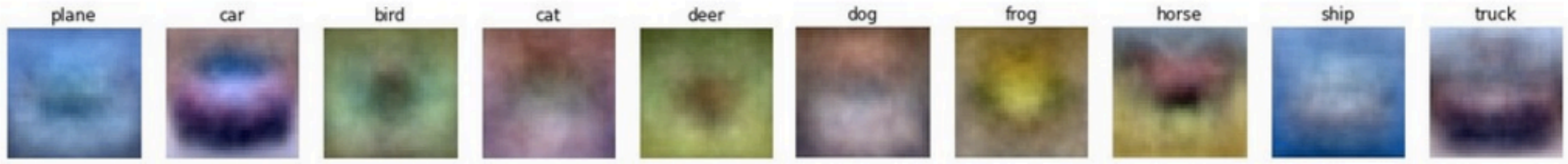


Find a *linear function* to separate the classes:

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

Visualizing linear classifiers





Source: Andrej Karpathy, http://cs231n.github.io/linear-classify/

Nearest neighbor vs. linear classifiers

Nearest Neighbors

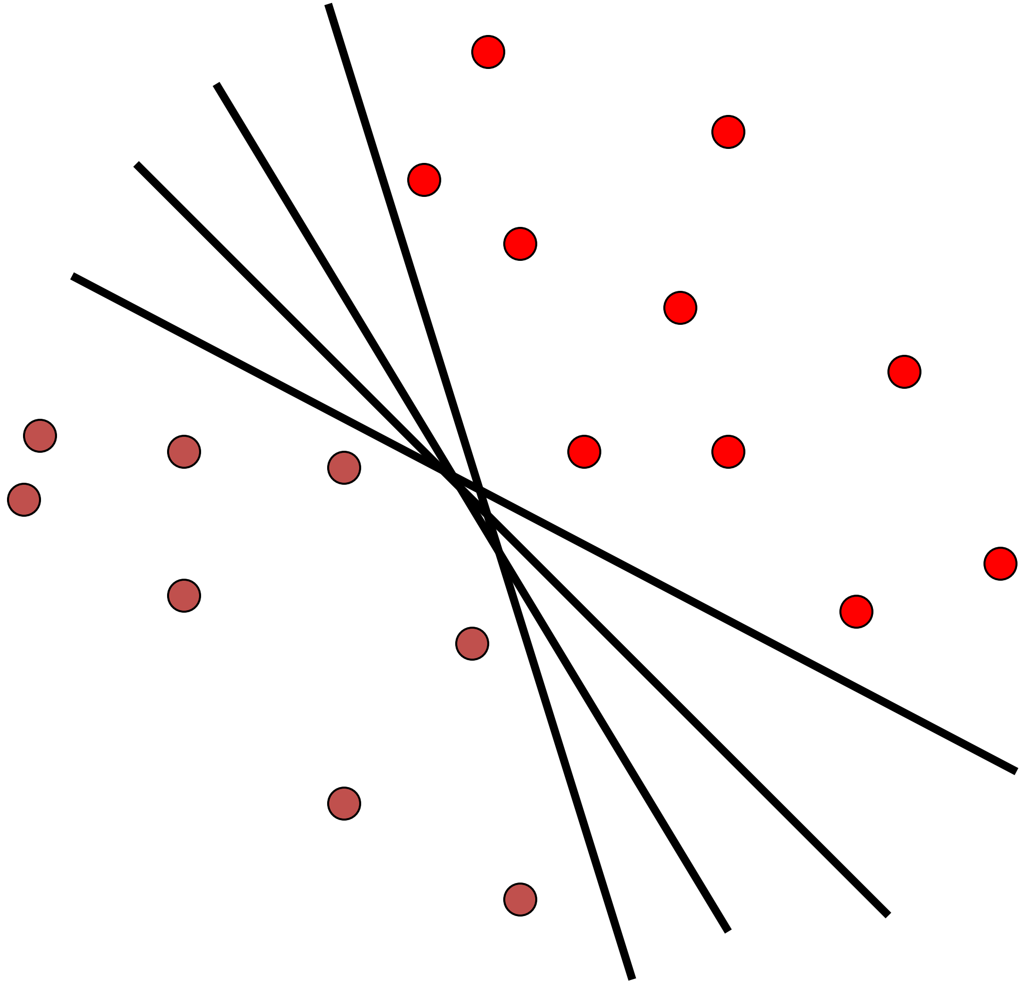
- Pros:
  - Simple to implement
  - Decision boundaries not necessarily linear
  - Works for any number of classes
  - *Nonparametric* method
- Cons:
  - Need good distance function
  - Slow at test time

Linear Models

- Pros:
  - Low-dimensional *parametric* representation
  - Very fast at test time
- Cons:
  - Works for two classes
  - How to train the linear function?
  - What if data is not linearly separable?

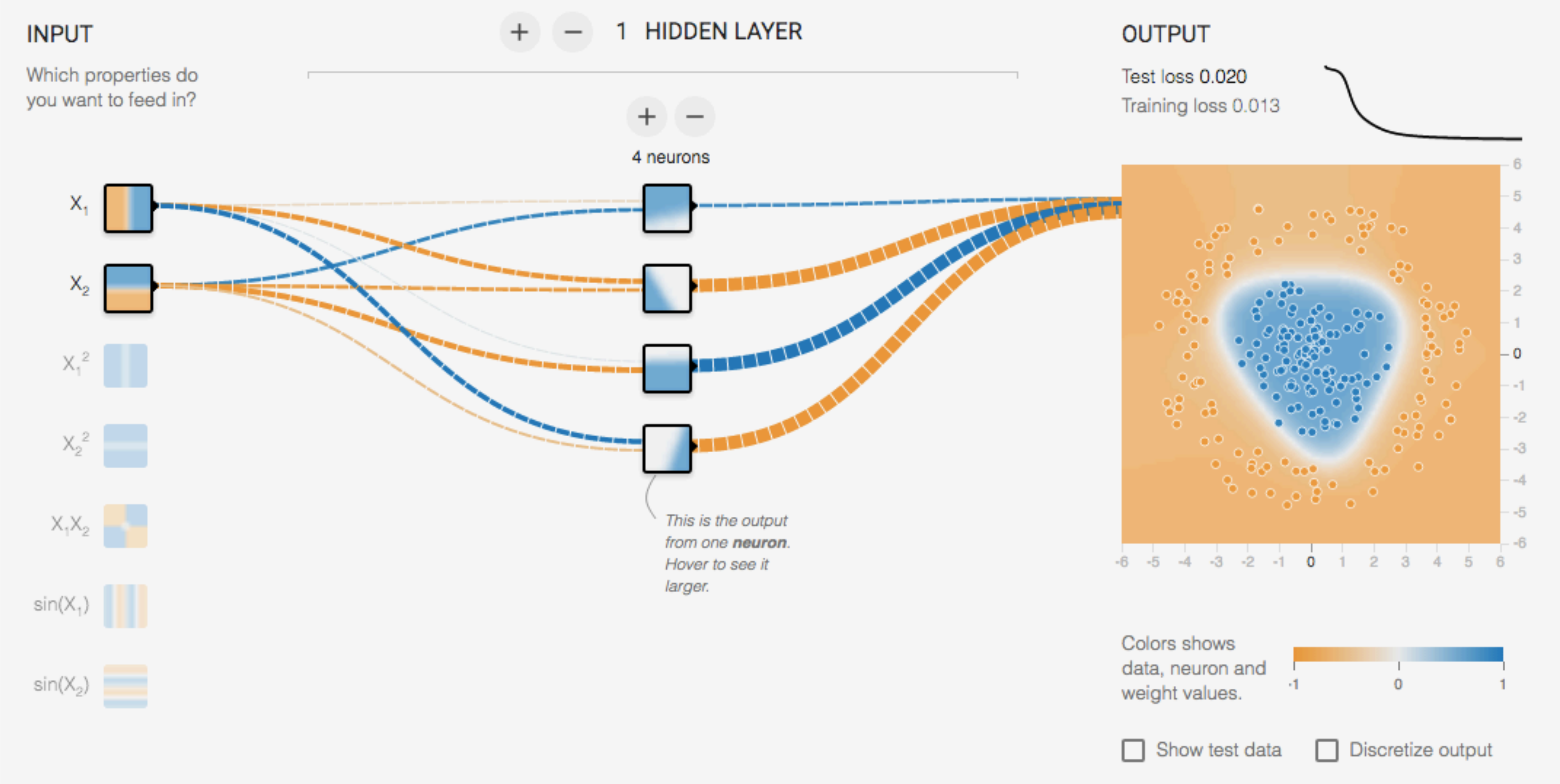
Linear classifiers

When the data is linearly separable, there may be more than one separator (hyperplane)



Which separator is best?

Review: Neural Networks

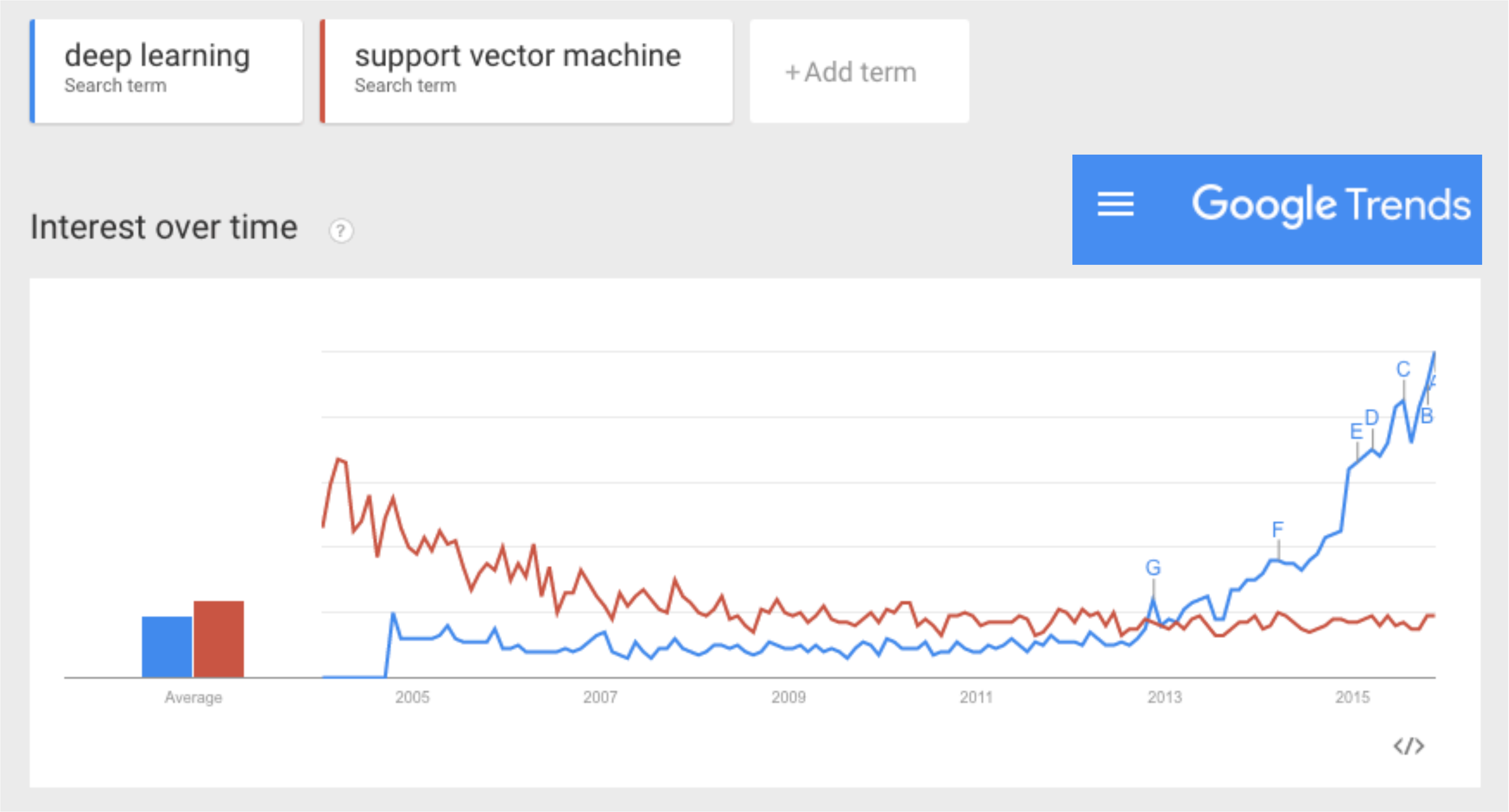


http://playground.tensorflow.org/

"Deep" recognition pipeline

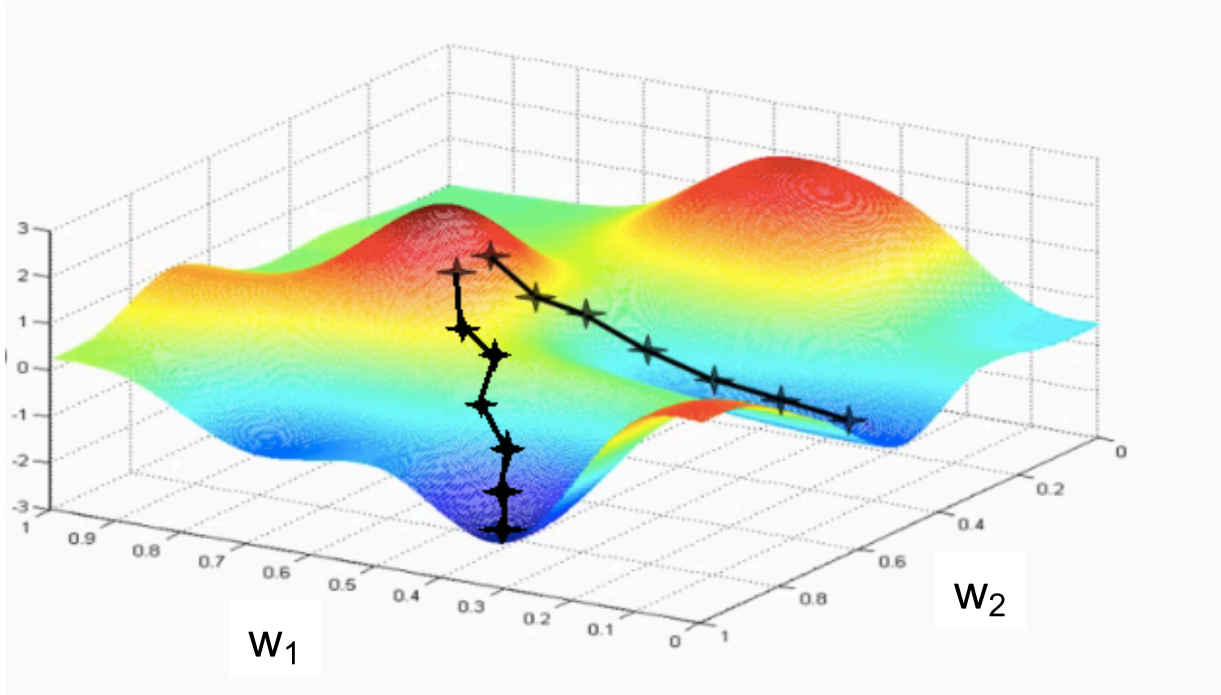
- Learn a *feature hierarchy* from pixels to classifier
- Each layer extracts features from the output of previous layer
- Train all layers jointly

“Deep” vs. “shallow” (SVMs) Learning



Training of multi-layer networks

- Find network weights to minimize the prediction loss between true and estimated labels of training examples:
- $E(\mathbf{w}) = \sum_i l(\mathbf{x}_i, y_i; \mathbf{w})$
- Update weights by **gradient descent:** $\mathbf{w} \leftarrow \mathbf{w} - \alpha \frac{\partial E}{\partial \mathbf{w}}$



Training of multi-layer networks

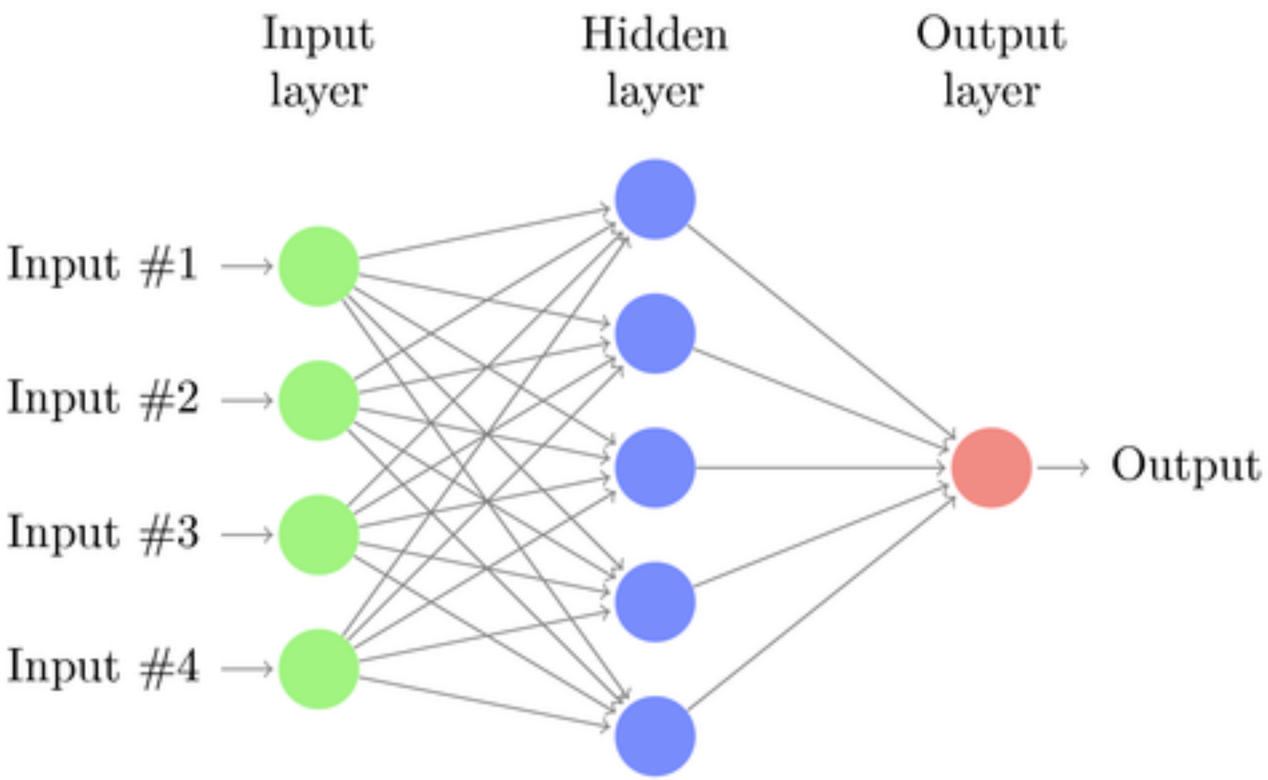
- Find network weights to minimize the prediction loss between true and estimated labels of training examples:

- $E(\mathbf{w}) = \sum_i l(\mathbf{x}_i, y_i; \mathbf{w})$
- Update weights by **gradient descent:** $\mathbf{w} \leftarrow \mathbf{w} - \alpha \dfrac{\partial E}{\partial \mathbf{w}}$

- **Back-propagation:** gradients are computed in the direction from output to input layers and combined using chain rule
- **Stochastic gradient descent:** compute the weight update w.r.t. one training example (or a small batch of examples) at a time, cycle through training examples in random order in multiple epochs

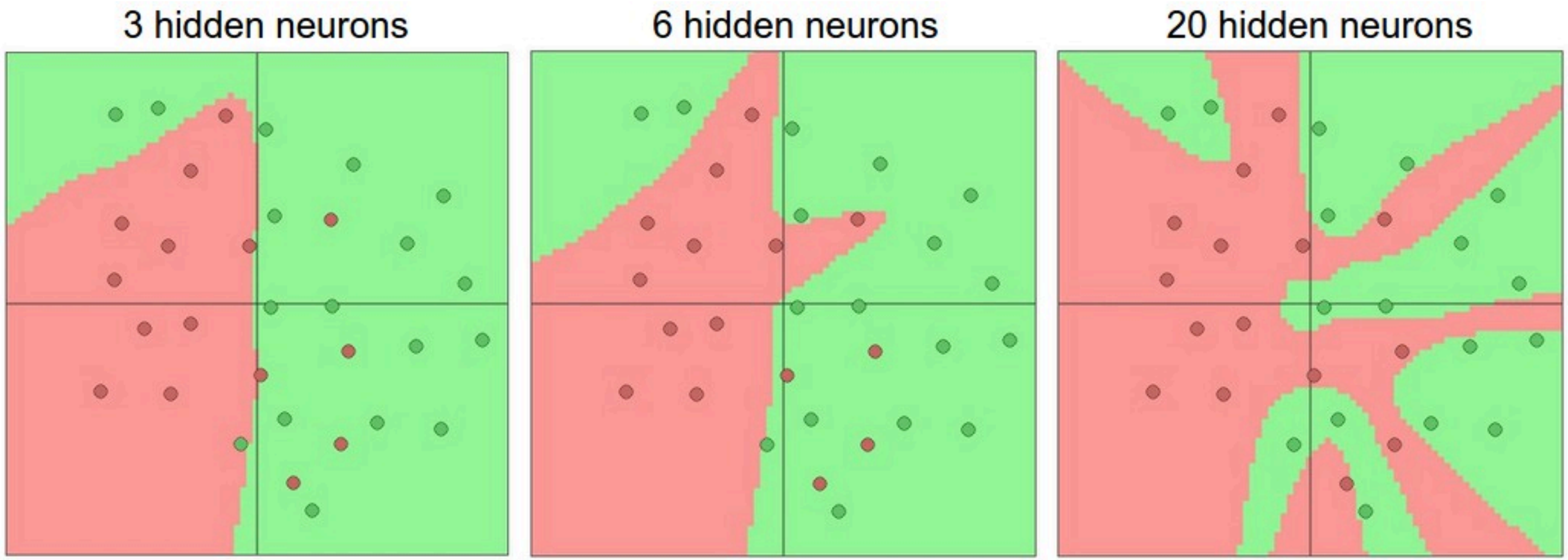
Network with a single hidden layer

- Neural networks with at least one hidden layer are [*universal function approximators*]



Network with a single hidden layer

Hidden layer size and *network capacity*:



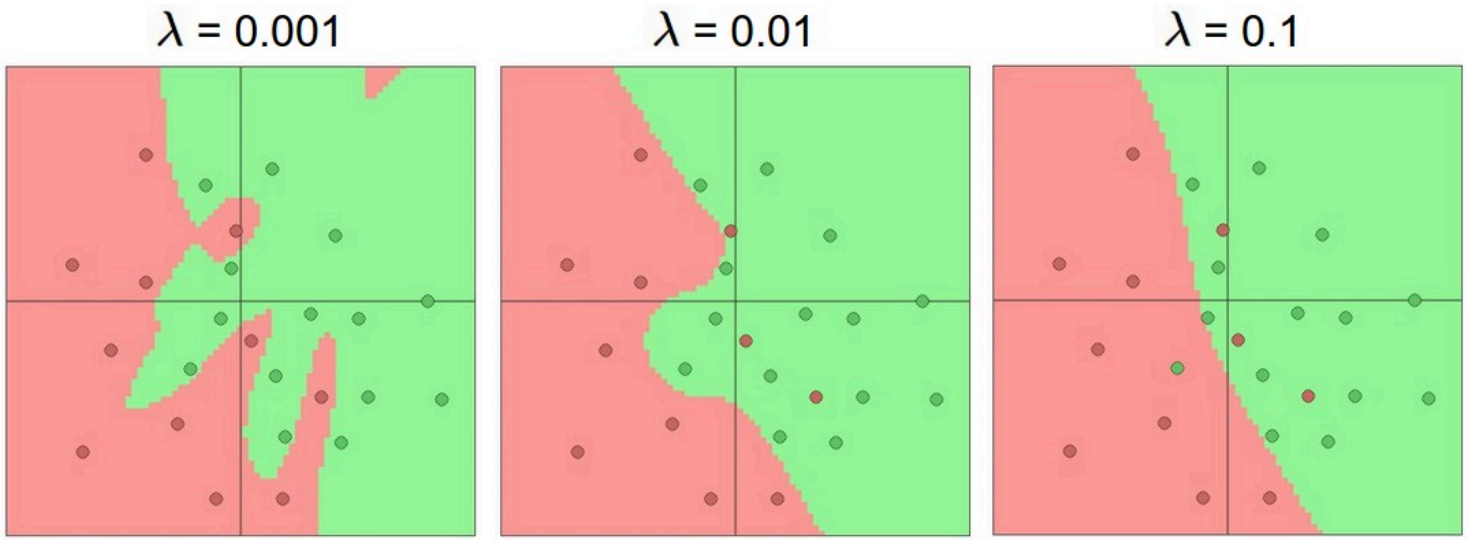
Source: http://cs231n.github.io/neural-networks-1/

Regularization

- It is common to add a penalty (e.g., quadratic) on weight magnitudes to the objective function:

$$E(\mathbf{w}) = \sum_i l(\mathbf{x}_i, y_i; \mathbf{w}) + \lambda \|\mathbf{w}\|^2$$

  – Quadratic penalty encourages network to use all of its inputs “a little” rather than a few inputs “a lot”



Source: http://cs231n.github.io/neural-networks-1/

Dealing with multiple classes

- If we need to classify inputs into C different classes, we put C units in the last layer to produce C *one-vs.-others* scores $f_1, f_2, \dots, f_C$
- Apply *softmax* function to convert these scores to probabilities:

$$\text{softmax}(f_1, \dots, f_c) = \left(\frac{\exp(f_1)}{\sum_j \exp(f_j)}, \dots, \frac{\exp(f_C)}{\sum_j \exp(f_j)} \right)$$

If one of the inputs is much larger than the others, then the corresponding softmax value will be close to 1 and others will be close to 0

- Use log likelihood (*cross-entropy*) loss:
- $l(\mathbf{x}_i, y_i; \mathbf{w}) = -\log P_{\mathbf{w}}(y_i \mid \mathbf{x}_i)$

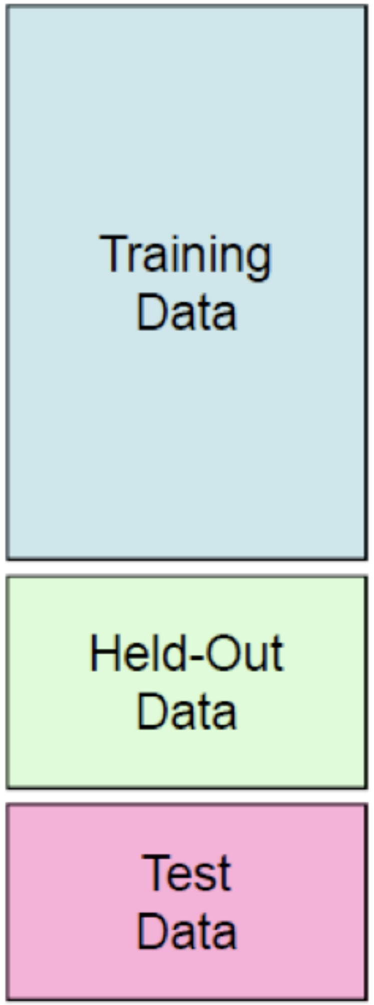
Neural networks: Pros and cons

- Pros
  - Flexible and general function approximation framework
  - Can build extremely powerful models by adding more layers
- Cons
  - Hard to analyze theoretically (e.g., training is prone to local optima)
  - Huge amount of training data, computing power may be required to get good performance
  - The space of implementation choices is huge (network architectures, parameters)

Best practices for training classifiers

- Goal: obtain a classifier with **good generalization** or performance on never before seen data

1. Learn *parameters* on the ***training set***
2. Tune *hyperparameters* (implementation choices) on the *held out* ***validation set***
3. Evaluate performance on the ***test set***
   – Crucial: do not peek at the test set when iterating steps 1 and 2!



Bias-variance tradeoff

- Prediction error of learning algorithms has two main components:
  - **Bias:** error due to simplifying model assumptions
  - **Variance:** error due to randomness of training set
- **Bias-variance tradeoff** can be controlled by turning "knobs" that determine model complexity

High bias, low variance

Low bias, high variance

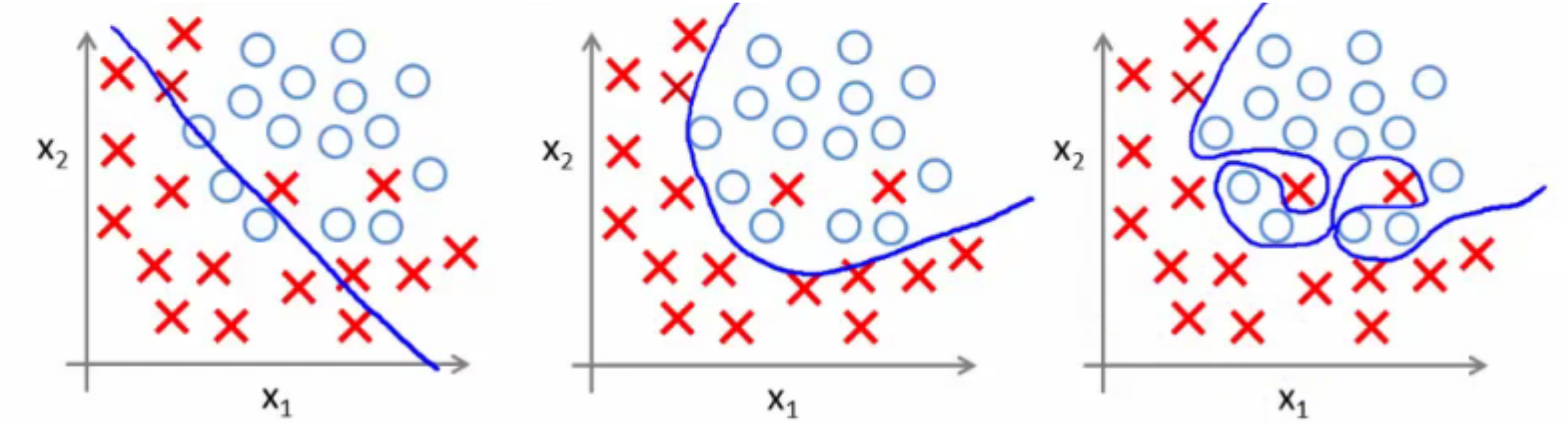


Figure source

Underfitting and overfitting

- **Underfitting:** training and test error are both *high*
  - Model does an equally poor job on the training and the test set
  - The model is too “simple” to represent the data or the model is not trained well
- **Overfitting:** Training error is *low* but test error is *high*
  - Model fits irrelevant characteristics (noise) in the training data
  - Model is too complex or amount of training data is insufficient

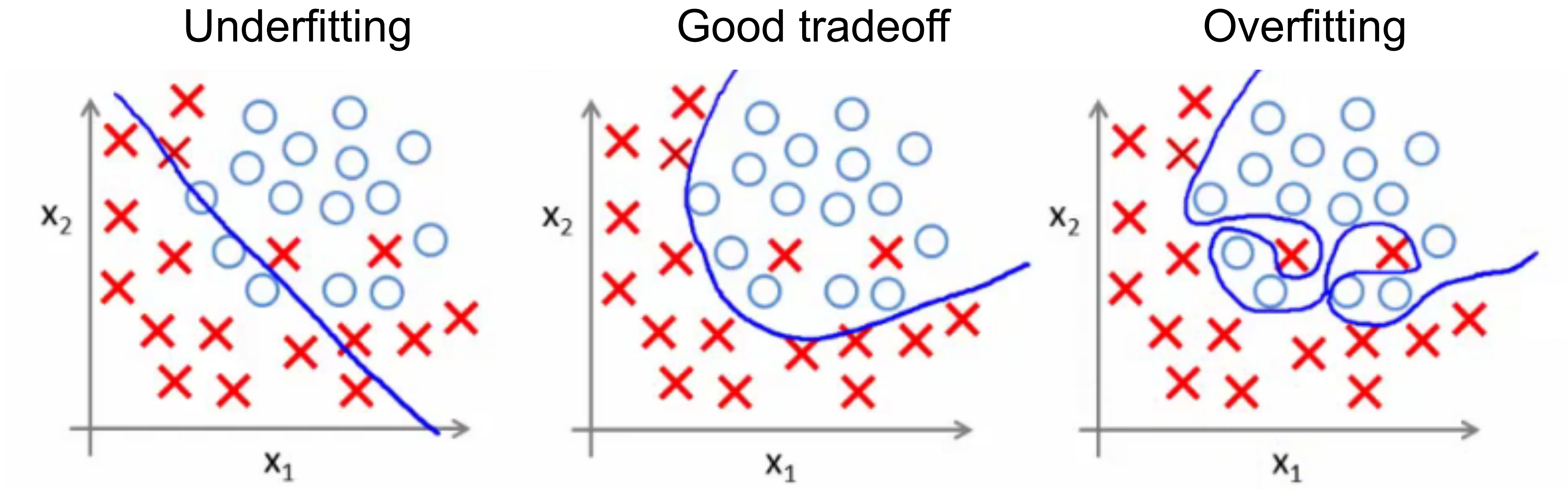


Figure source