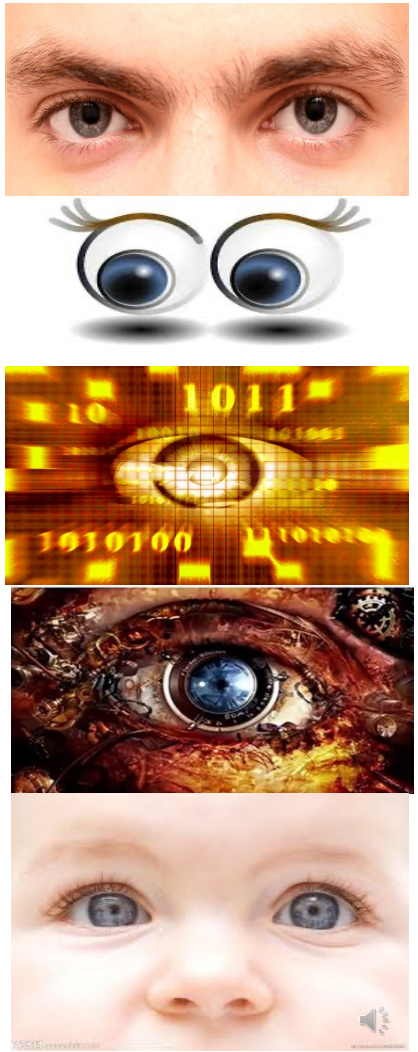
Computer Vision I:

Jingya Wang

Email: wangjingya@shanghaitech.edu.cn



Introduction to object recognition

Overview

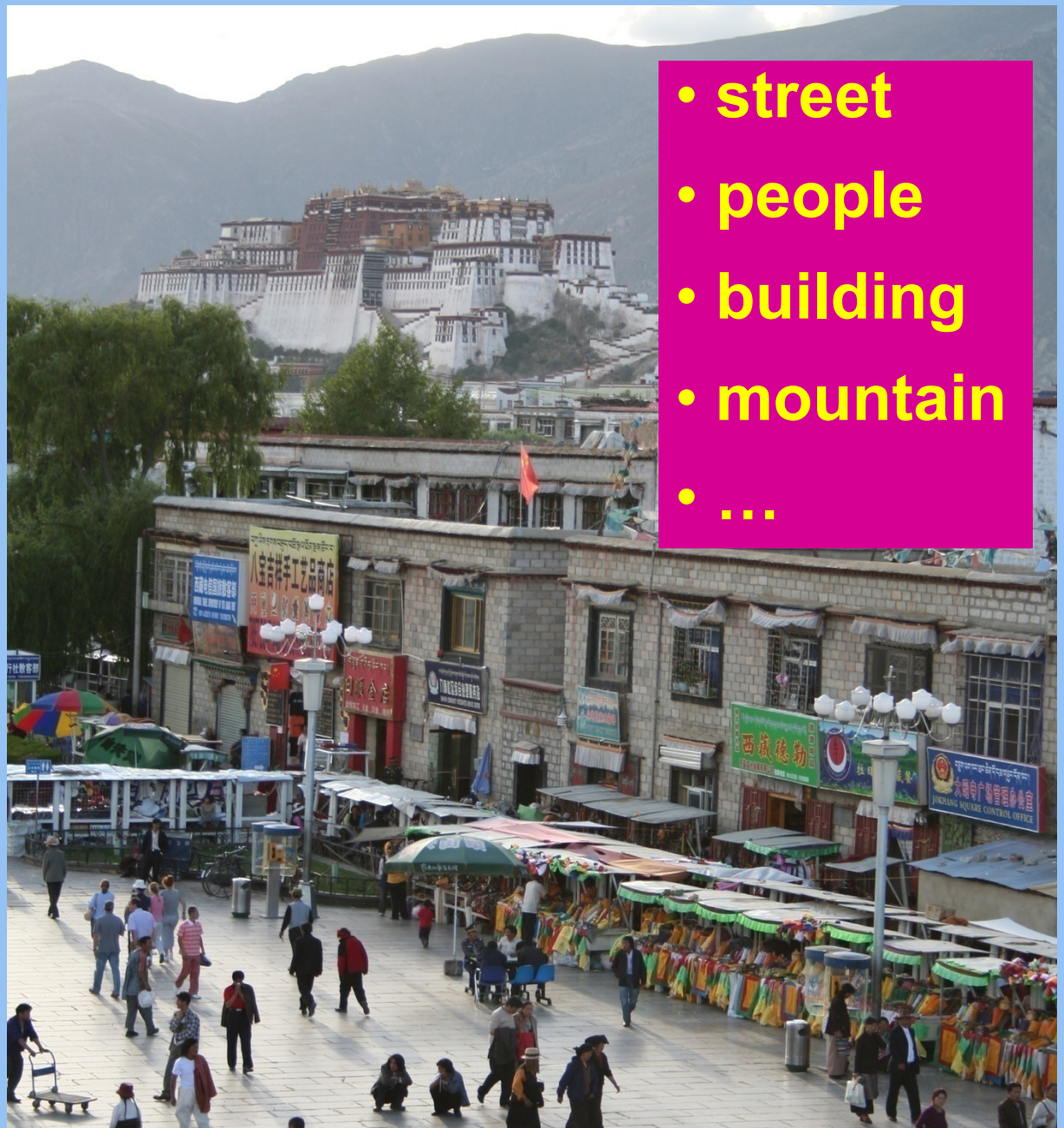
- Basic recognition tasks
- A statistical learning approach
- Traditional or “shallow” recognition pipeline
  - Bags of features
  - Classifiers
- Next time: neural networks and “deep” recognition pipeline

Common recognition tasks

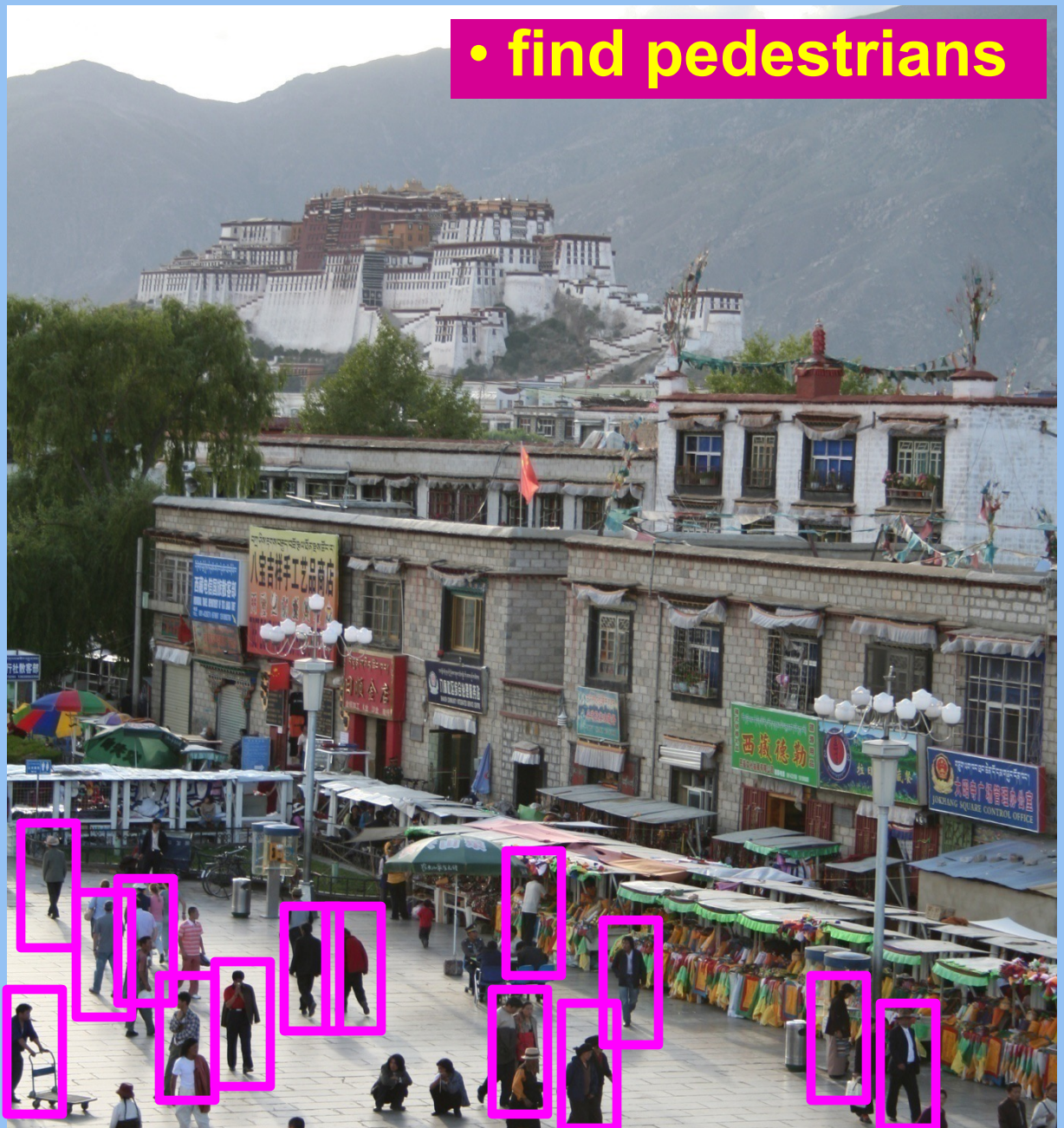
Image classification

- outdoor/indoor
- city/forest/factory/etc.

Image tagging



Object detection



Activity recognition

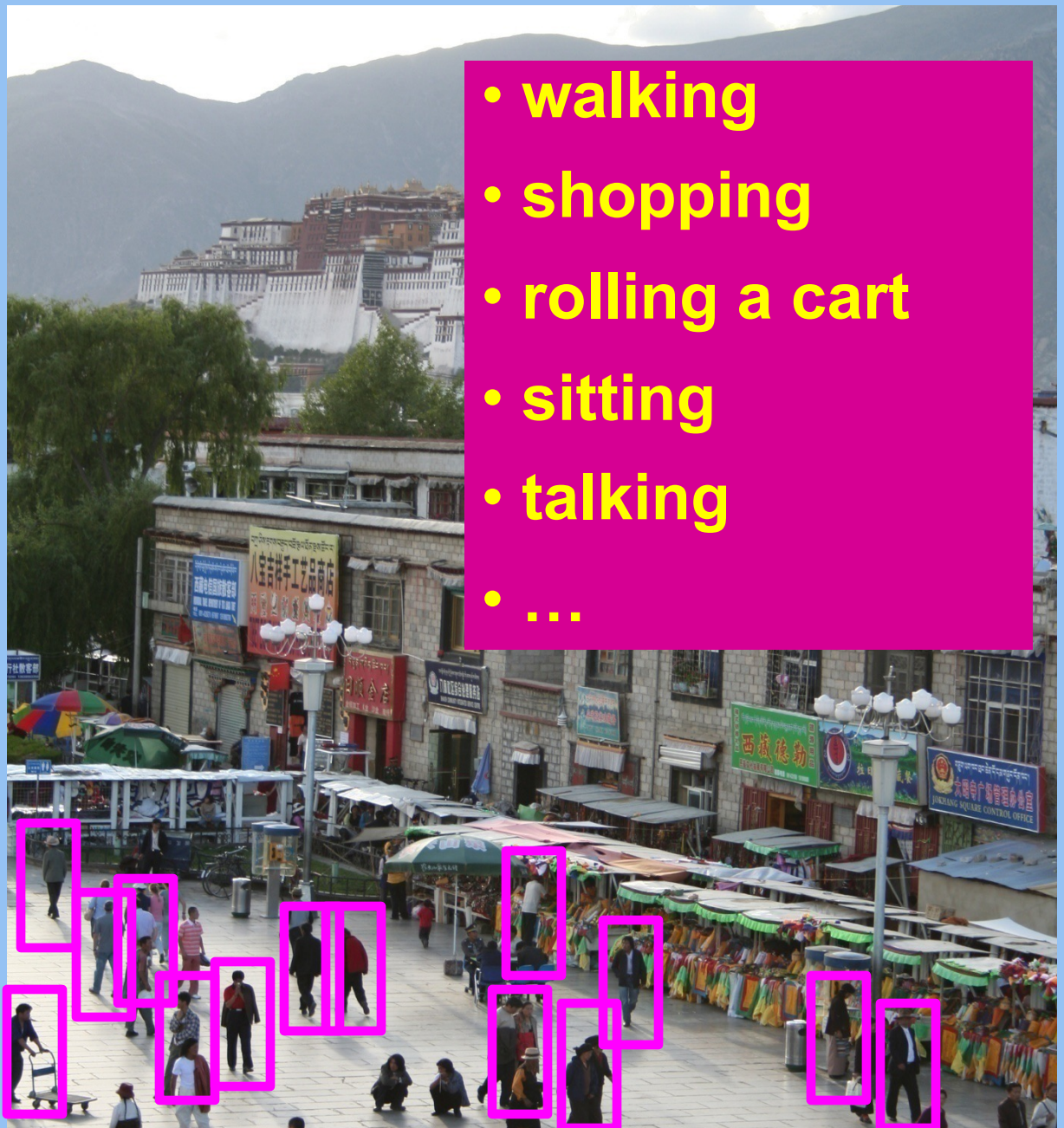
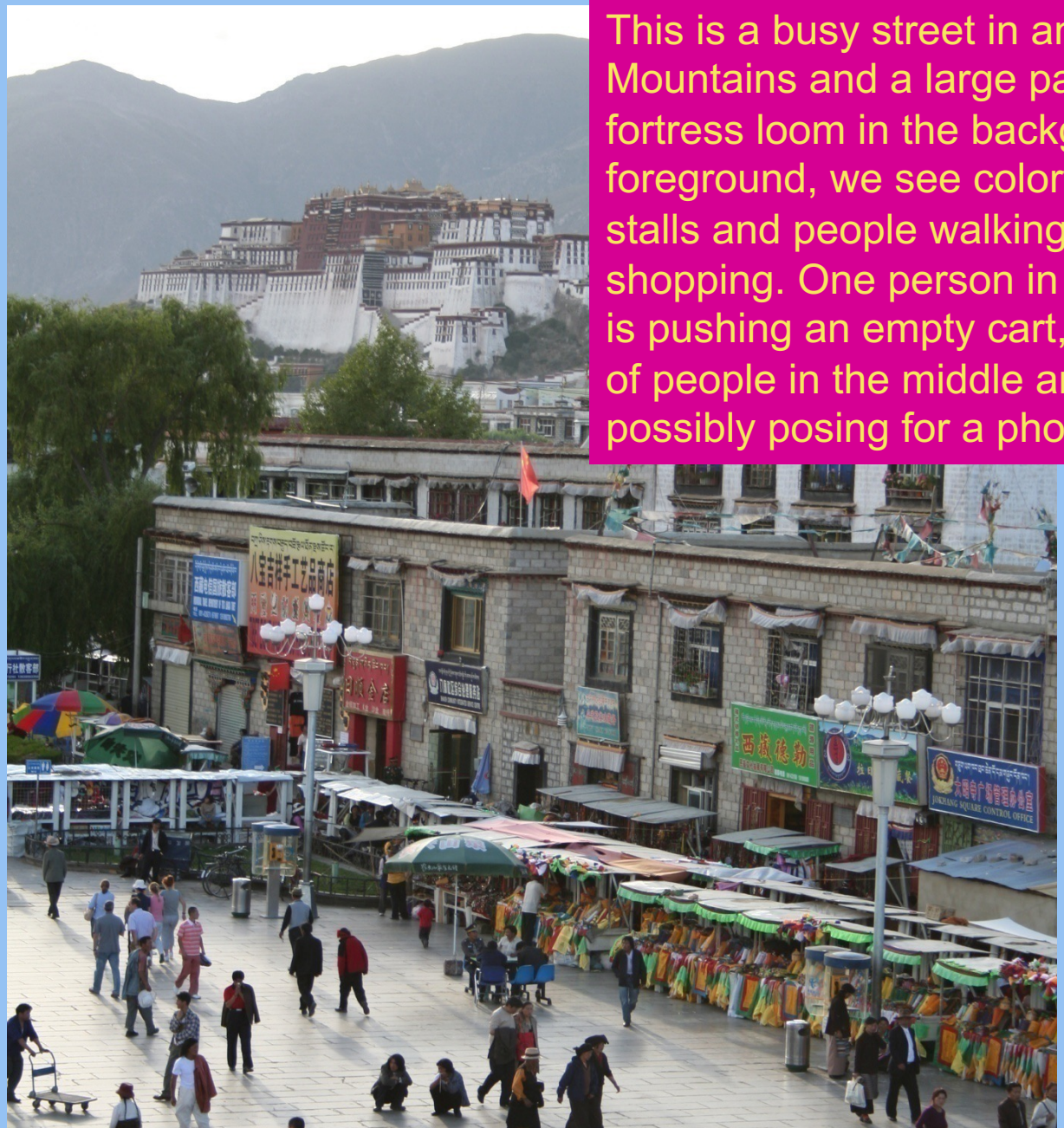


Image parsing

Image description

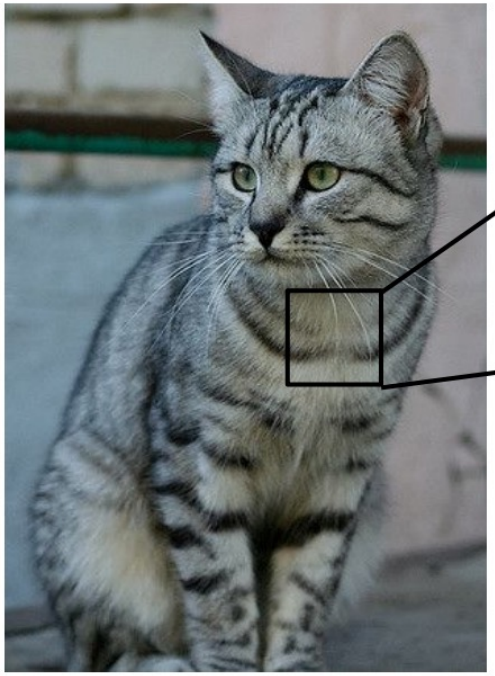


This is a busy street in an Asian city. Mountains and a large palace or fortress loom in the background. In the foreground, we see colorful souvenir stalls and people walking around and shopping. One person in the lower left is pushing an empty cart, and a couple of people in the middle are sitting,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

Image Classification: A core task in Computer Vision

cat

The Problem: Semantic Gap



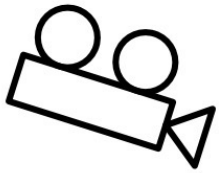

```
[[105 112 108 111 104  99 106  99  96 103 112 119 104  97  93  87]
 [ 91  98 102 106 104  79  98 103  99 105 123 136 110 105  94  85]
 [ 76  85  90 105 128 105  87  96  95  99 115 112 106 103  99  85]
 [ 99  81  81  93 120 131 127 100  95  98 102  99  96  93 101  94]
 [106  91  61  64  69  91  88  85 101 107 109  98  75  84  96  95]
 [114 108  85  55  55  69  64  54  64  87 112 129  98  74  84  91]
 [133 137 147 103  65  81  80  65  52  54  74  84 102  93  85  82]
 [128 137 144 140 109  95  86  70  62  65  63  63  60  73  86 101]
 [125 133 148 137 119 121 117  94  65  79  80  65  54  64  72  98]
 [127 125 131 147 133 127 126 131 111  96  89  75  61  64  72  84]
 [115 114 109 123 150 148 131 118 113 109 100  92  74  65  72  78]
 [ 89  93  90  97 108 147 131 118 113 114 113 109 106  95  77  80]
 [ 63  77  86  81  77  79 102 123 117 115 117 125 125 130 115  87]
 [ 62  65  82  89  78  71  80 101 124 126 119 101 107 114 131 119]
 [ 63  65  75  88  89  71  62  81 120 138 135 105  81  98 110 118]
 [ 87  65  71  87 106  95  69  45  76 130 126 107  92  94 105 112]
 [118  97  82  86 117 123 116  66  41  51  95  93  89  95 102 107]
 [164 146 112  80  82 120 124 104  76  48  45  66  88 101 102 109]
 [157 170 157 120  93  86 114 132 112  97  69  55  70  82  99  94]
 [130 128 134 161 139 100 109 118 121 134 114  87  65  53  69  86]
 [128 112  96 117 150 144 120 115 104 107 102  93  87  81  72  79]
 [123 107  96  86  83 112 153 149 122 109 104  75  80 107 112  99]
 [122 121 102  80  82  86  94 117 145 148 153 102  58  78  92 107]
 [122 164 148 103  71  56  78  83  93 103 119 139 102  61  69  84]]
```

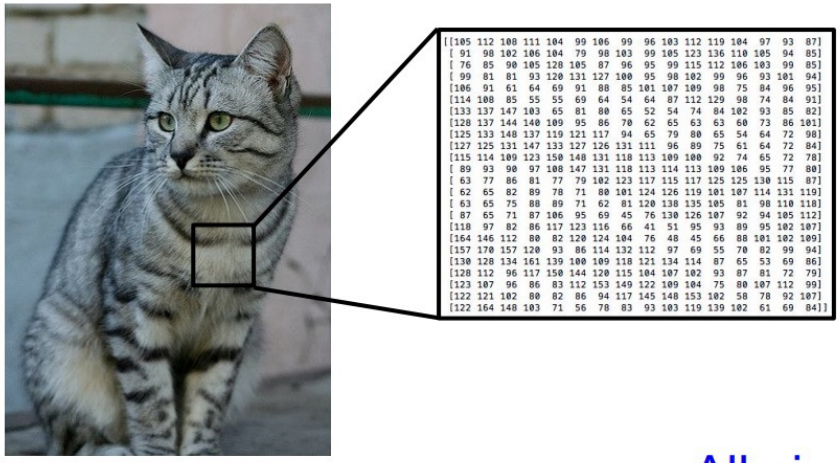
What the computer sees

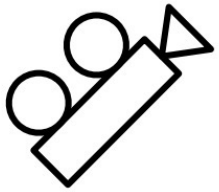
An image is a tensor of integers between [0, 255]:

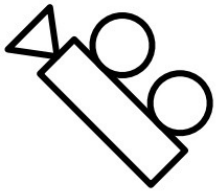
e.g. 800 x 600 x 3
(3 channels RGB)

Challenges: Viewpoint variation









All pixels change when the camera moves!

Challenges: Background Clutter

This image is CC0 1.0 public domain

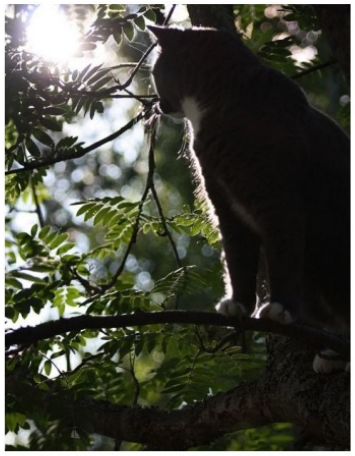
This image is CC0 1.0 public domain

Challenges: Illumination

This image is CC0 1.0 public domain

This image is CC0 1.0 public domain

This image is CC0 1.0 public domain



This image is CC0 1.0 public domain

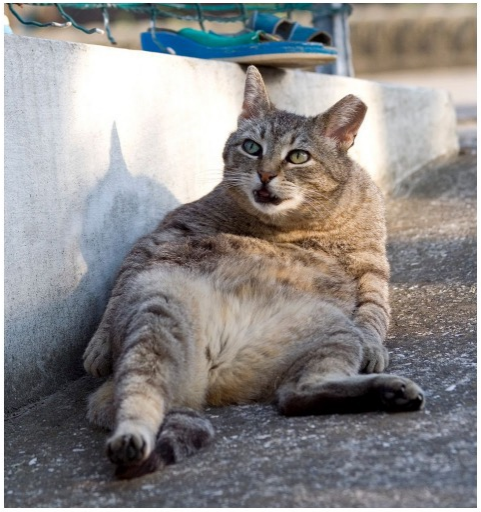
Challenges: Occlusion

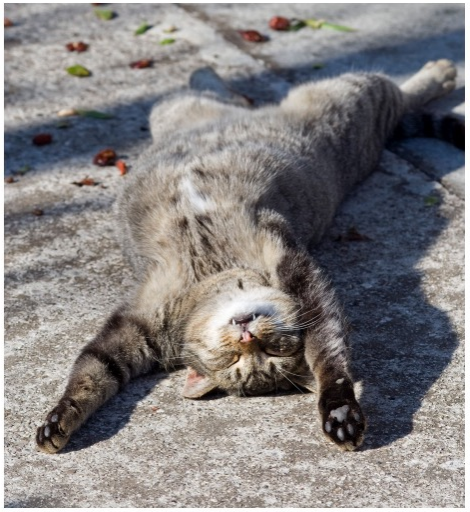
This image is CC0 1.0 public domain

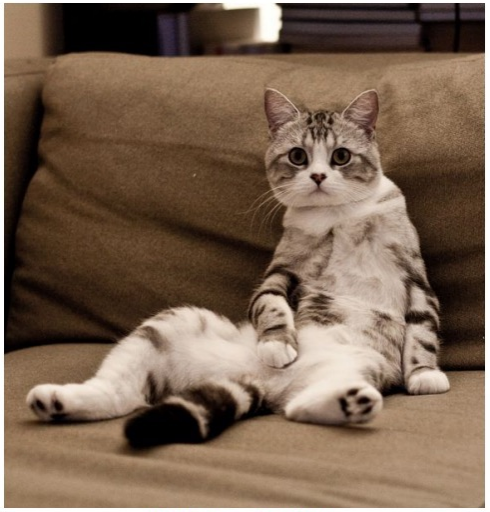
This image is CC0 1.0 public domain

This image by ionsson is licensed under CC-BY 2.0

Challenges: Deformation



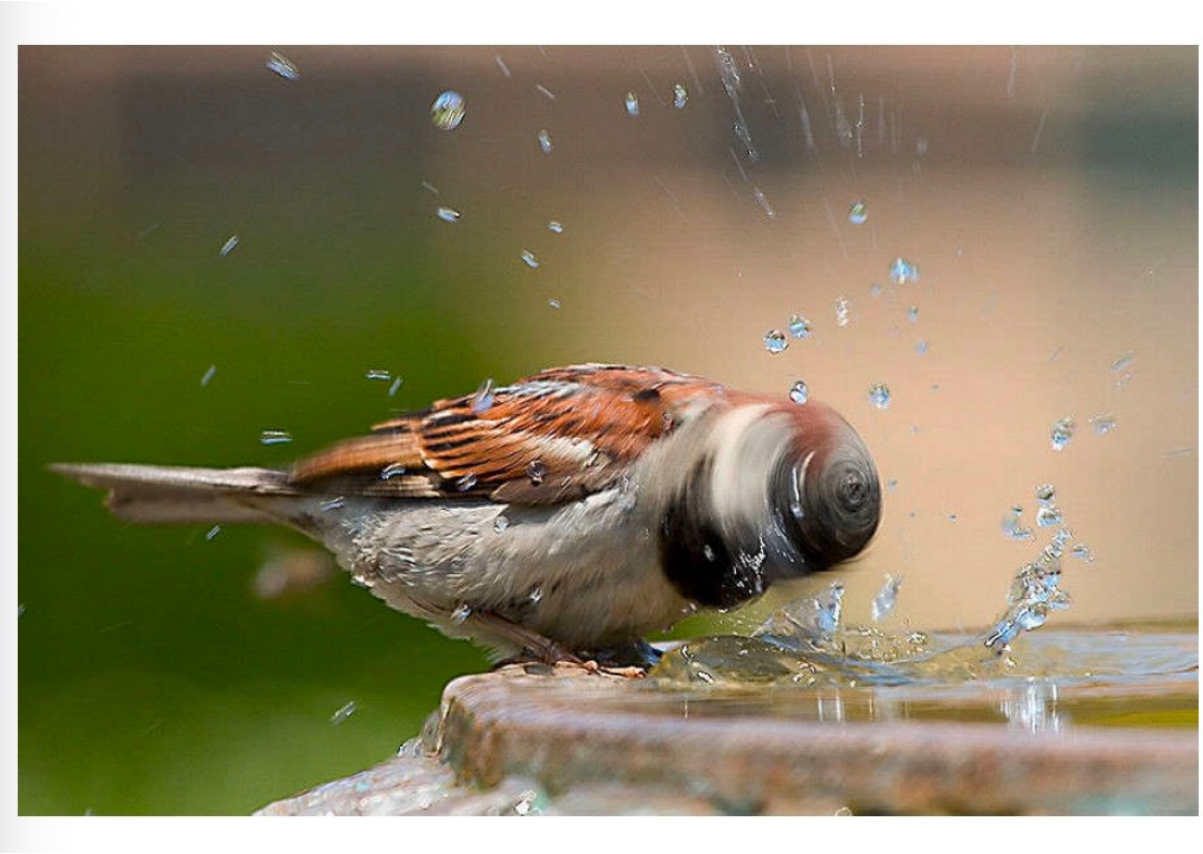




Challenges: scale

Source: S. Lazebnik

Challenges: motion blur



Source: S. Lazebnik

Challenges: Intraclass variation

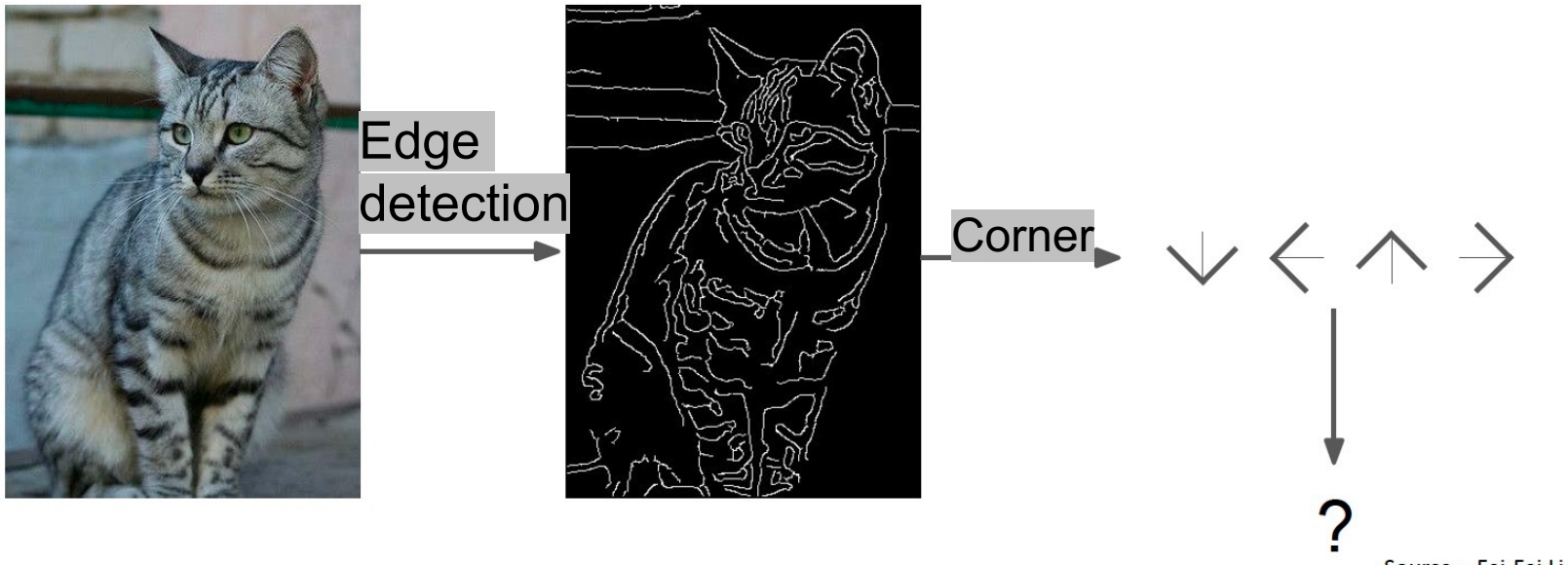
Challenges: Intraclass variation

Source: Fei-Fei

Solution

- Rule based methods? Hard-coding

```
def classify_image ( image ):
    # Do something magical here
    return class_label
```



Source: Fei-Fei Li

Machine Learning: Data-Driven Approach

1. Collect a dataset of images and labels
2. Use Machine Learning algorithms to train a classifier
3. Evaluate the classifier on new images

```
def train(images, labels):
  # Machine learning!
  return model
```

```
def predict(model, test_images):
  # Use model to predict labels
  return test_labels
```

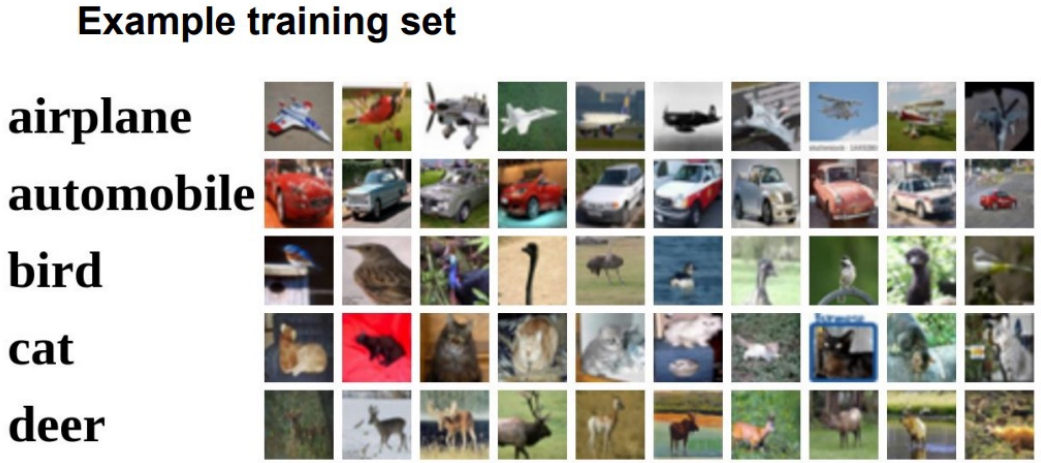


Image classification

The statistical learning framework

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

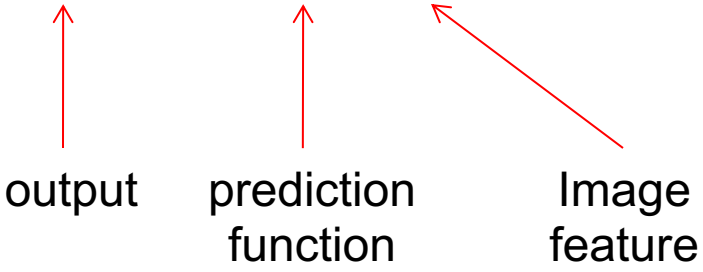
f() = “apple”

f() = “tomato”

f() = “cow”

The statistical learning framework

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



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

Steps

Training

Training Images

Training Labels

Image Features

Training

Learned model

Testing

Test Image

Image Features

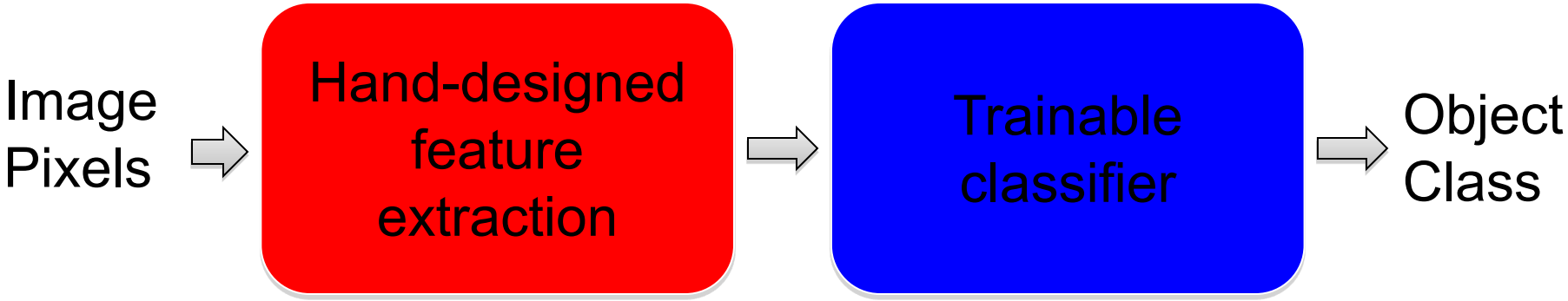
Learned model

Prediction

Slide credit: D. Hoiem

Traditional recognition pipeline

feature 不可学习, fixed!!



不是 data-driven

- Features are not learned
- Trainable classifier is often generic (e.g. SVM)

Bags of features: Motivation

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

Bags of features: Motivation

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

TF-IDF

↑ 重要性↑

词频

•**TF: Term Frequency**, which measures how frequently a term occurs in a document. Since every document is different in length, it is possible that a term would appear much more times in long documents than shorter ones. Thus, the term frequency is often divided by the document length (aka. the total number of terms in the document) as a way of normalization:

TF(t) = (Number of times term t appears in a document) / (Total number of terms in the document).

逆文档词频

过滤不重要的高频词

•**IDF: Inverse Document Frequency**, which measures how important a term is. While computing TF, all terms are considered equally important. However it is known that certain terms, such as "is", "of", and "that", may appear a lot of times but have little importance. Thus we need to weigh down the frequent terms while scale up the rare ones, by computing the following:

IDF(t) = log(Total number of documents / Number of documents with term t in it).

Example

Consider a document containing 100 words wherein the word *cat* appears 3 times. The term frequency (i.e., tf) for *cat* is ? $3/100$

Now, assume we have 10 million documents and the word *cat* appears in one thousand of these. Then, the inverse document frequency (i.e., idf) is calculated as ? $\log_{10}\left(\frac{10,000,000}{10^3}\right) = 4$

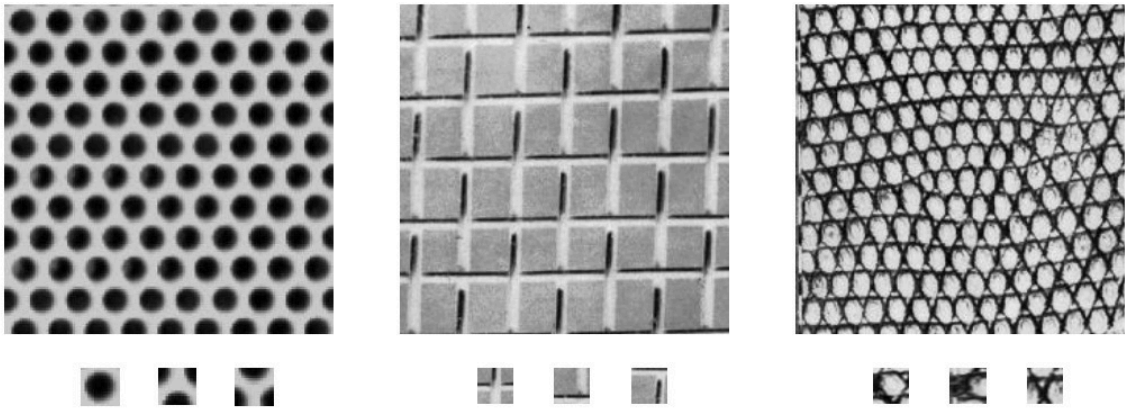
TF-IDF: $0.03 \times 4 = 0.12$

Bags of features

Origin 1: Texture recognition

Texture is characterized by the repetition of basic elements or *textons*

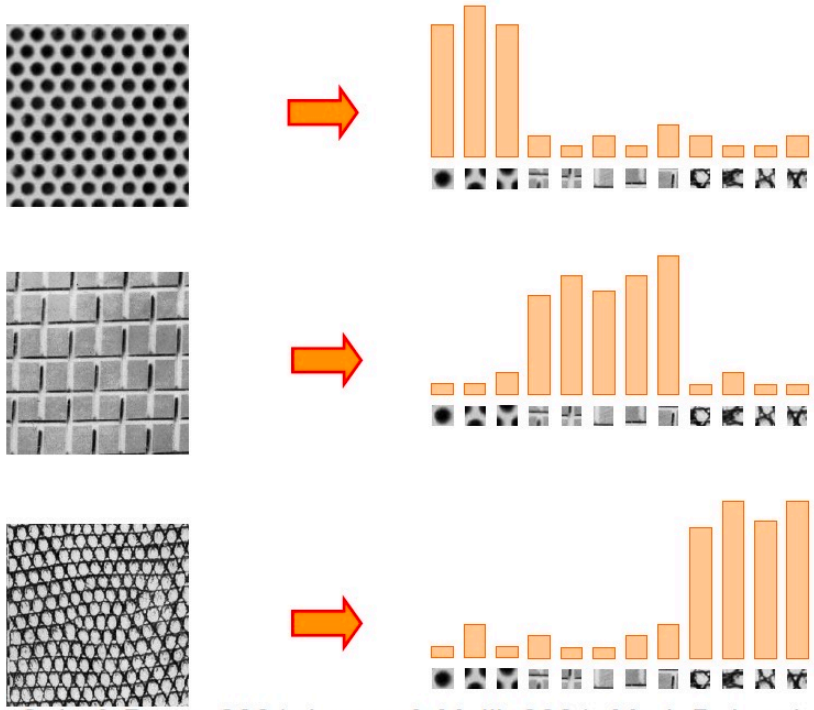
For stochastic textures, it is the identity of the textons, not their spatial arrangement, that matters



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

Source: Lazebnik

Origin 1: Texture recognition

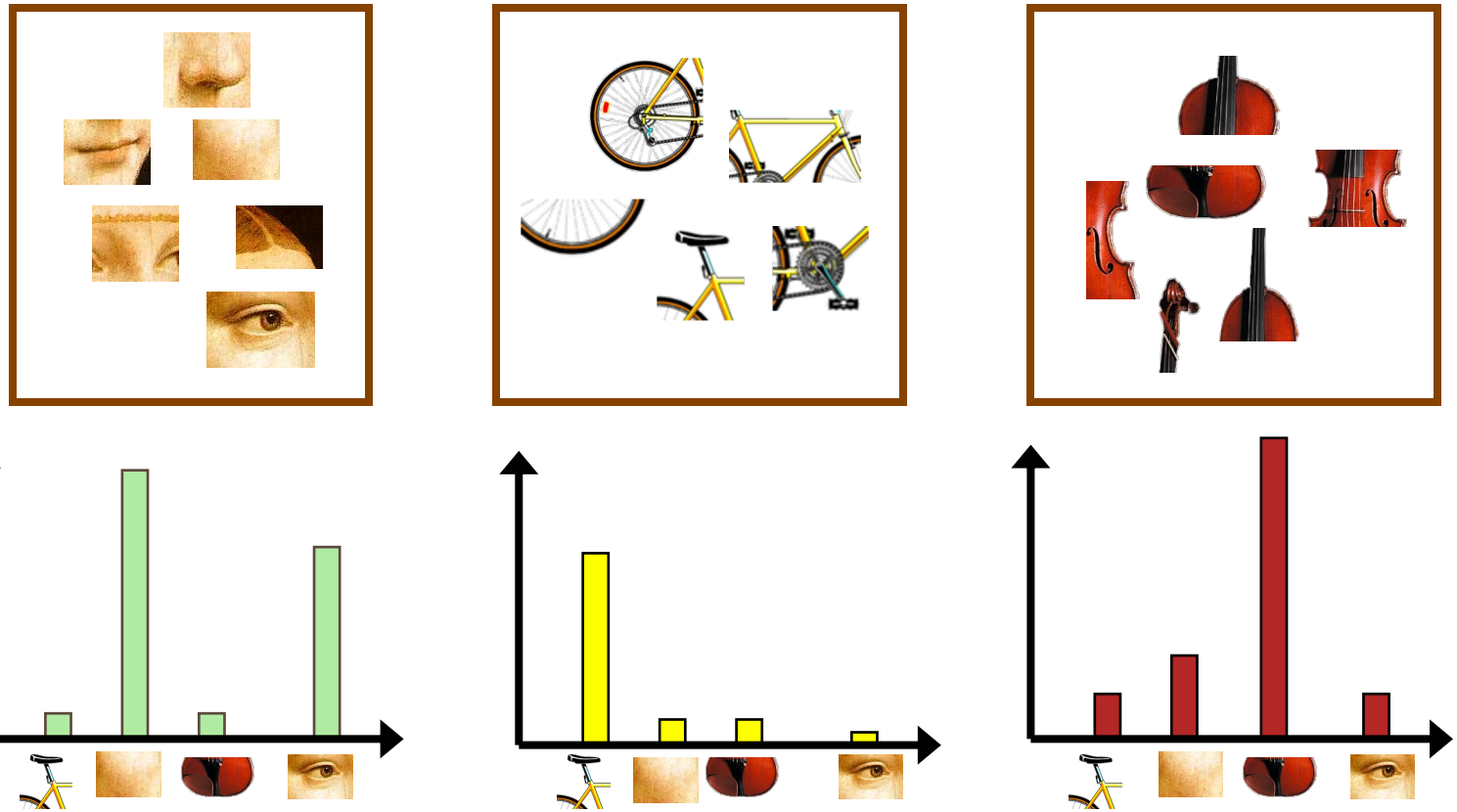


Julesz, 1981; Cula & Dana, 2001; Leung & Malik 2001; Mori, Belongie & Malik, 2001; Schmid 2001; Varma & Zisserman, 2002, 2003; Lazebnik, Schmid & Ponce, 2003

Source: Lazebnik

Traditional features: Bags-of-features

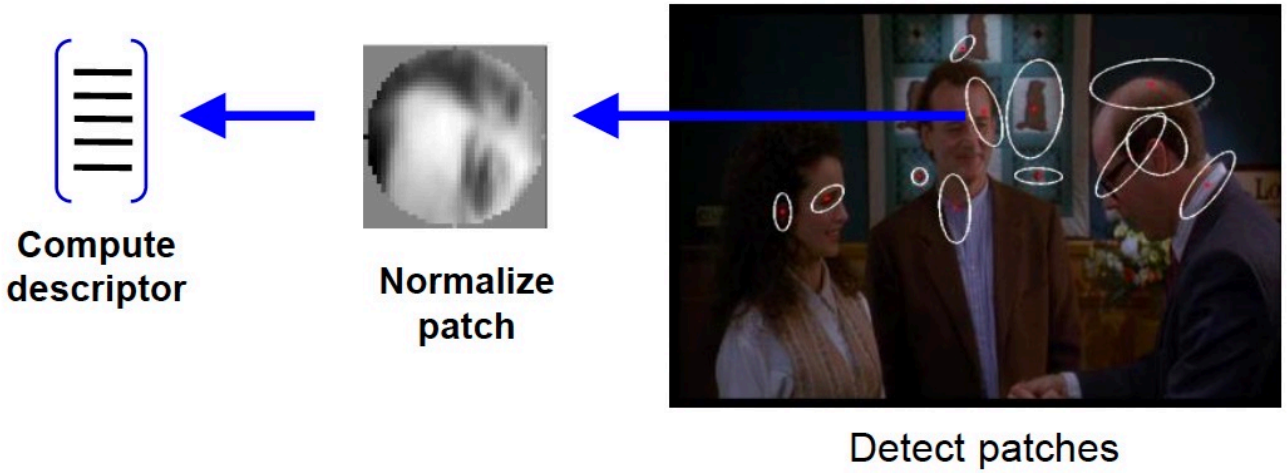
1. Extract local features
2. Learn “visual vocabulary”
3. Quantize local features using visual vocabulary
4. Represent images by frequencies of “visual words”



1. Local feature extraction

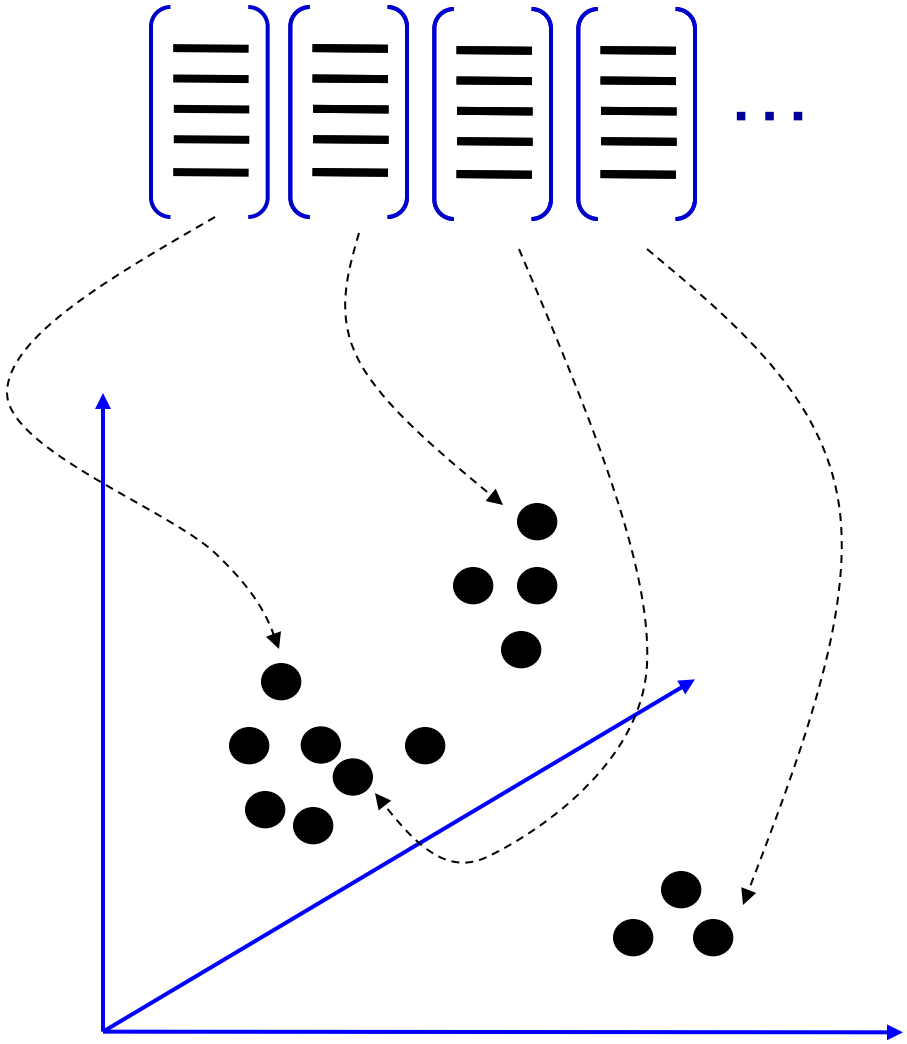
- Sample patches and extract descriptors

1. Feature extraction



Source: Josef Sivic

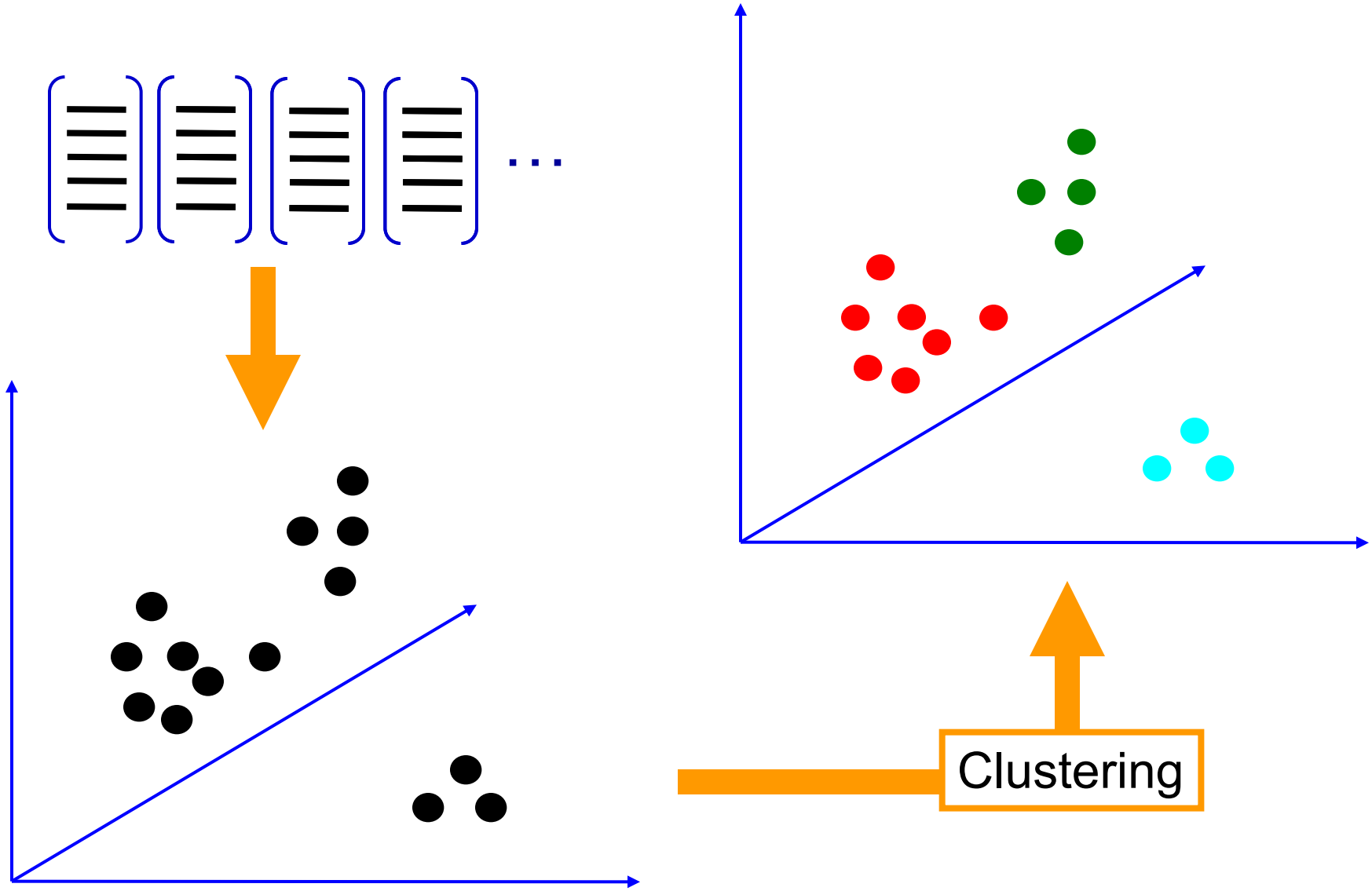
2. Learning the visual vocabulary



Extracted descriptors from the training set

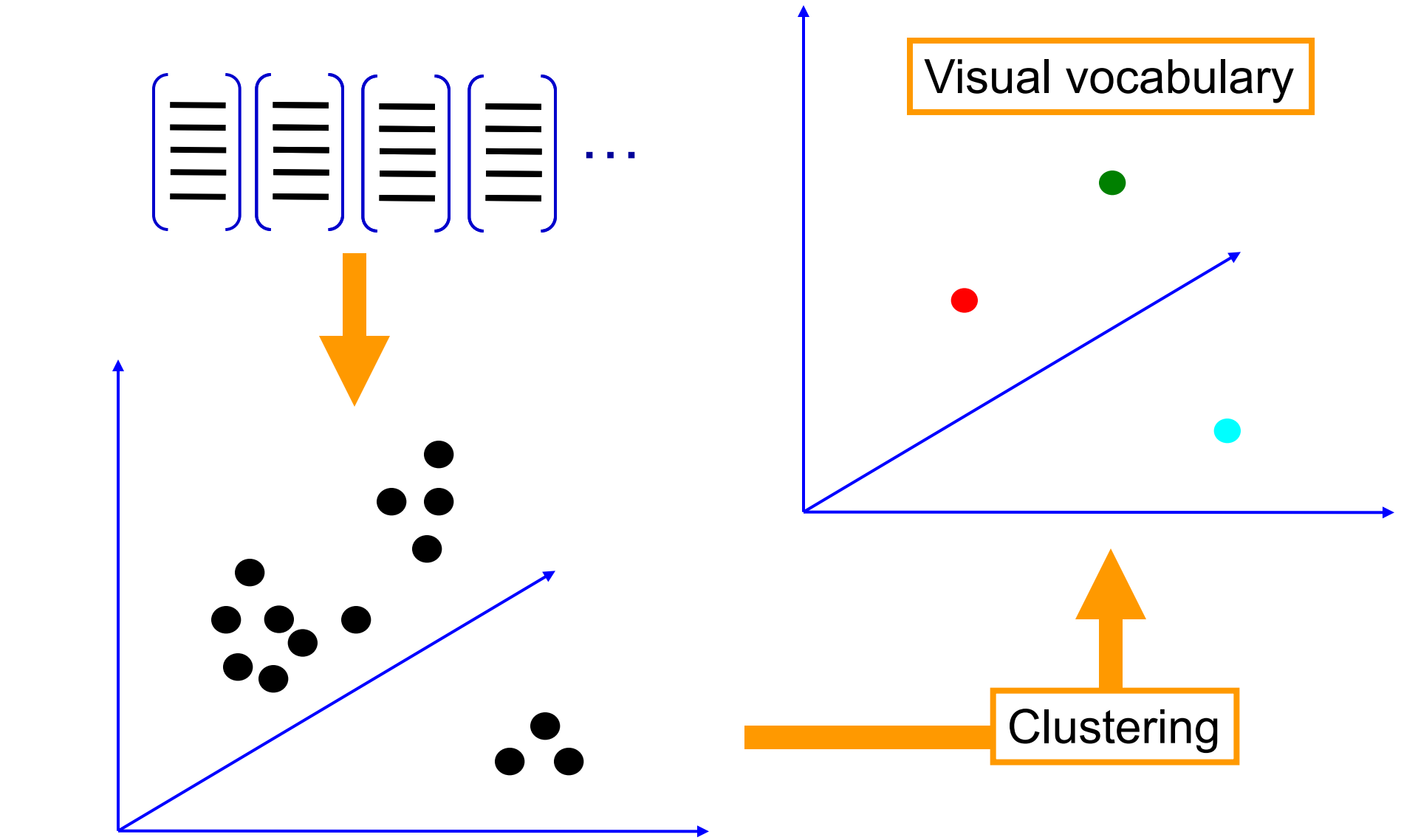
Slide credit: Josef Sivic

2. Learning the visual vocabulary



Slide credit: Josef Sivic

2. Learning the visual vocabulary



Slide credit: Josef Sivic

K-means

A clustering algorithm (unsupervised learning method)

Group similar things together.

K-means clustering

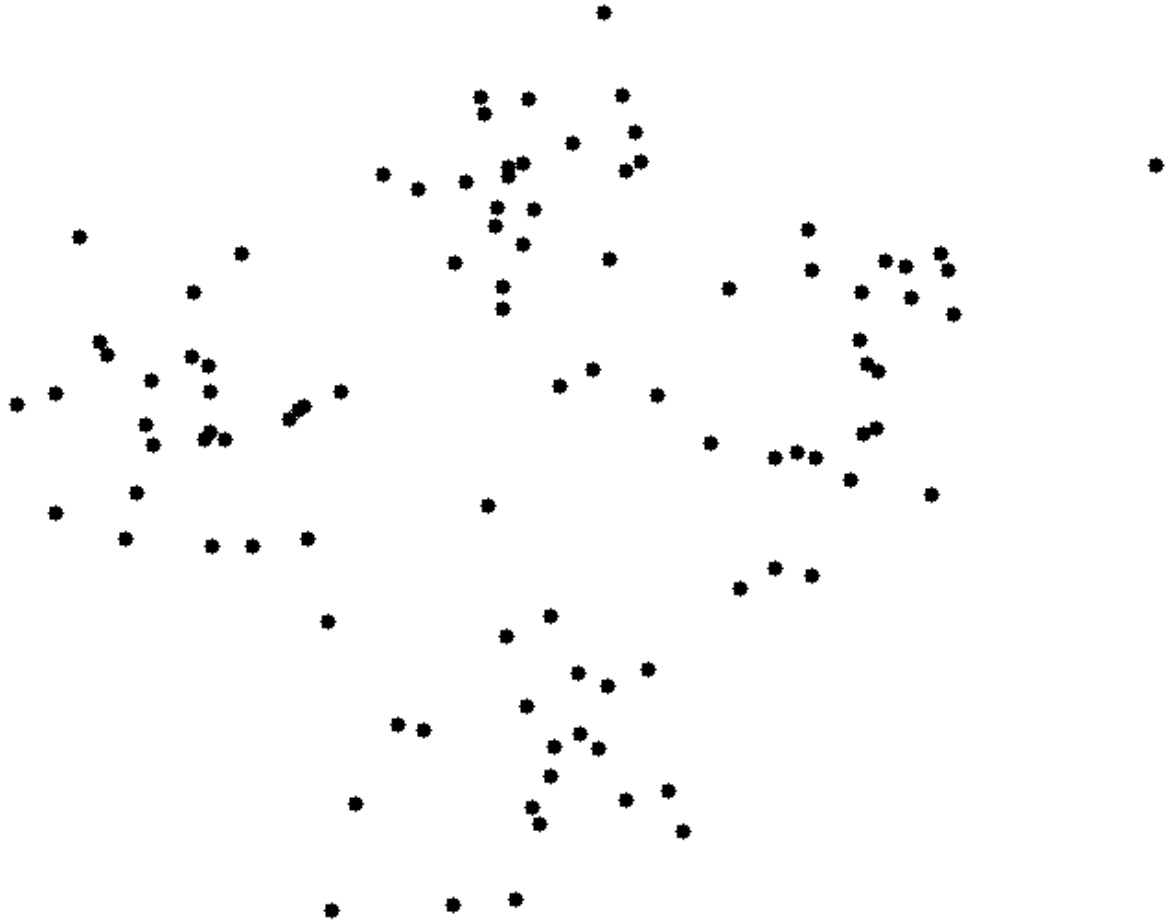
- Want to minimize sum of squared Euclidean distances between points $\mathbf{x}_i$ and their nearest cluster centers $\mathbf{m}_k$

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

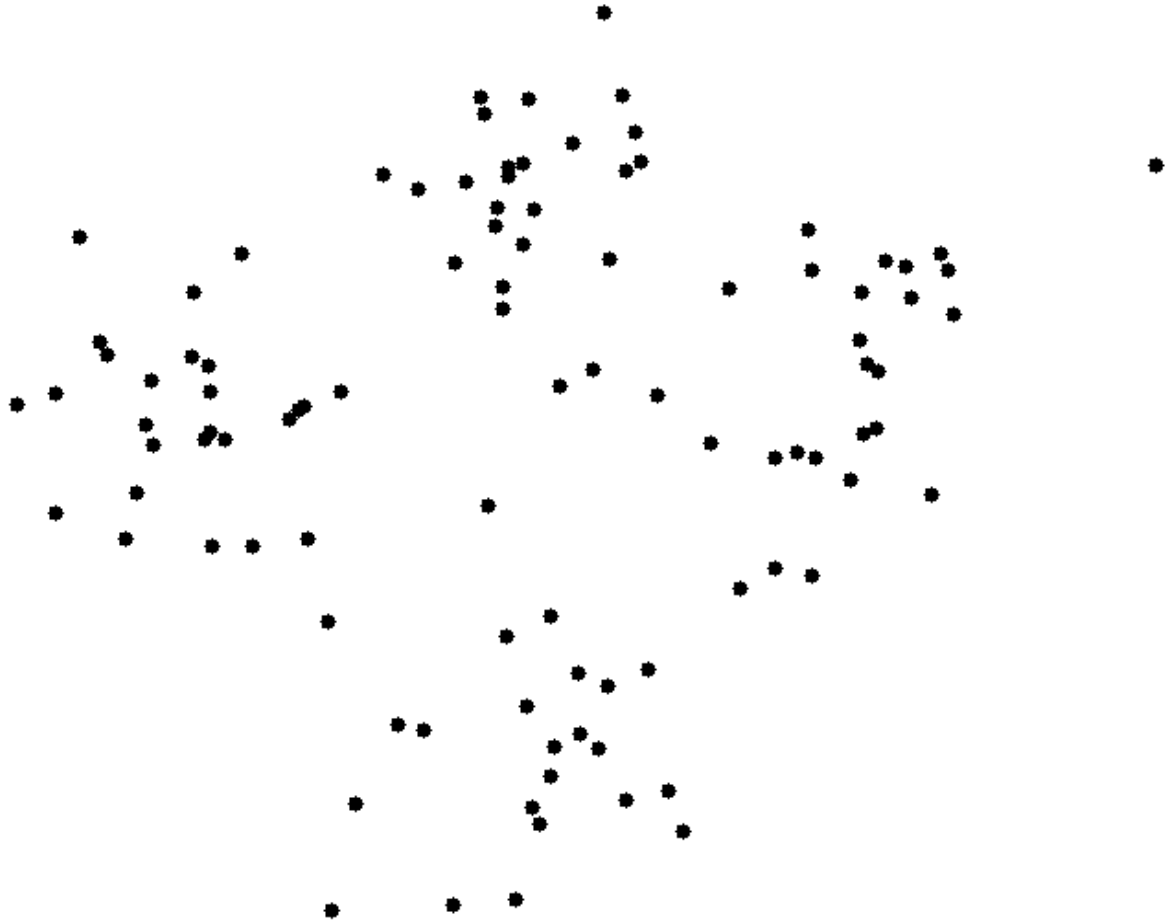
Algorithm:

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

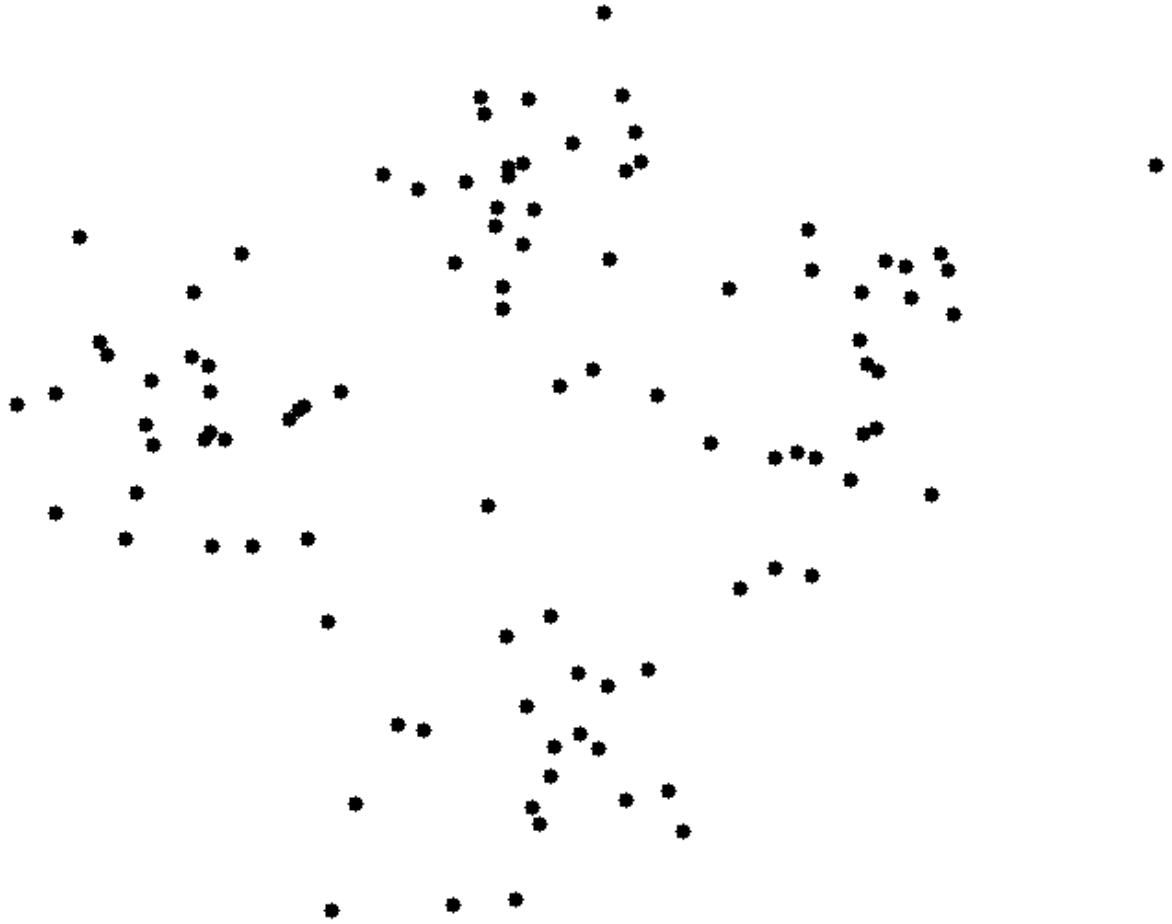
K-means demo



Source: http://shabal.in/visuals/kmeans/1.html
Another demo: http://www.kovan.ceng.metu.edu.tr/~maya/kmeans/

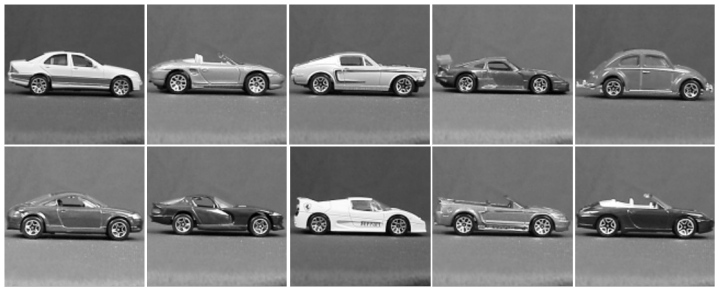


Local Minimum



Local Minimum

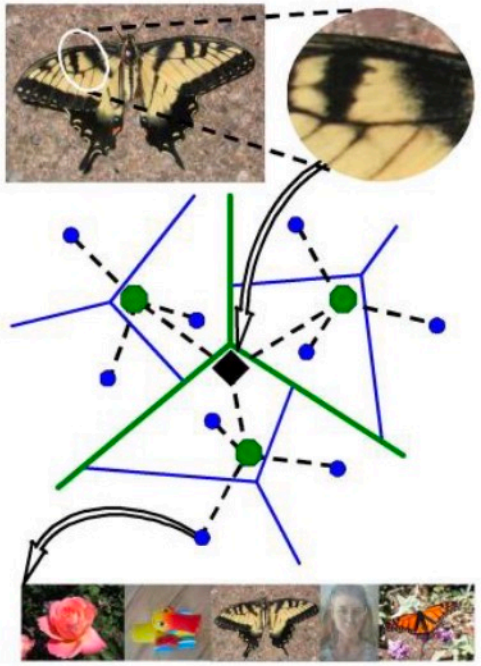
Example visual vocabulary



Source: B. Leibe

Visual vocabularies: Issues

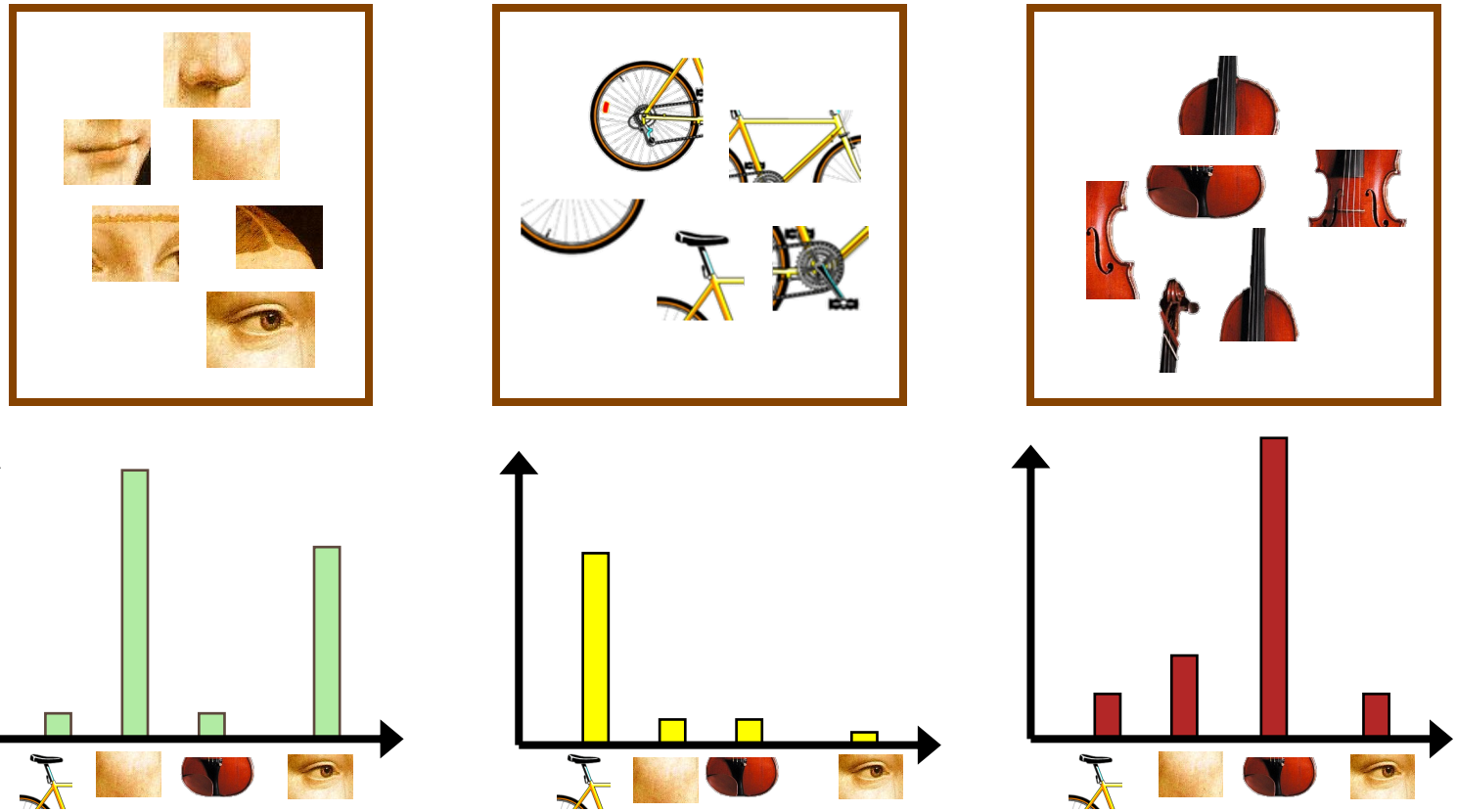
- How to choose vocabulary size?
  - Too small: visual words not representative of all patches
  - Too large: quantization artifacts, overfitting
- Computational efficiency
  - Vocabulary trees
    (Nister & Stewenius, 2006)



Source: Lazebnik

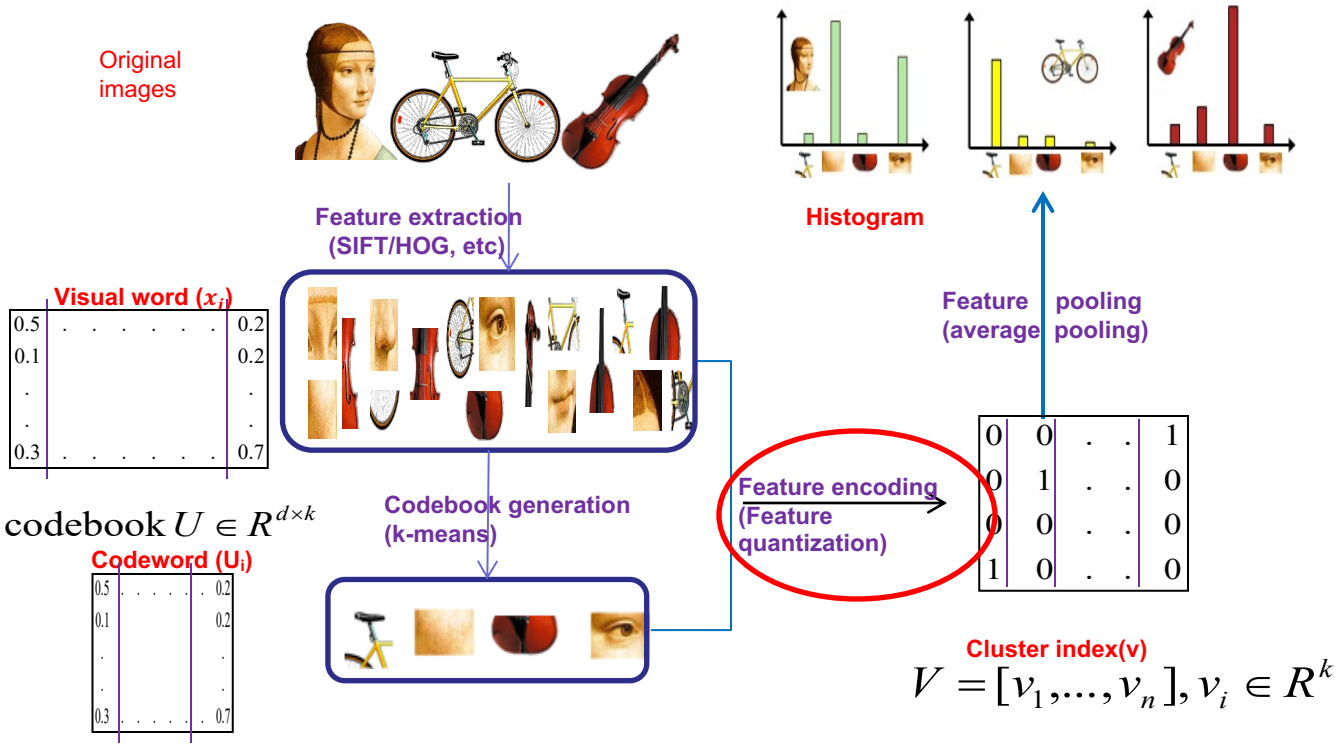
Bag-of-features steps

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



Bag-of-(Visual)Words (BoW) model:

(i) Feature extraction; (ii) Dictionary learning and Feature encoding; (iii) pooling



(credited to Fei-Fei Li)

Bags of features: Motivation

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

Spatial pyramid representation

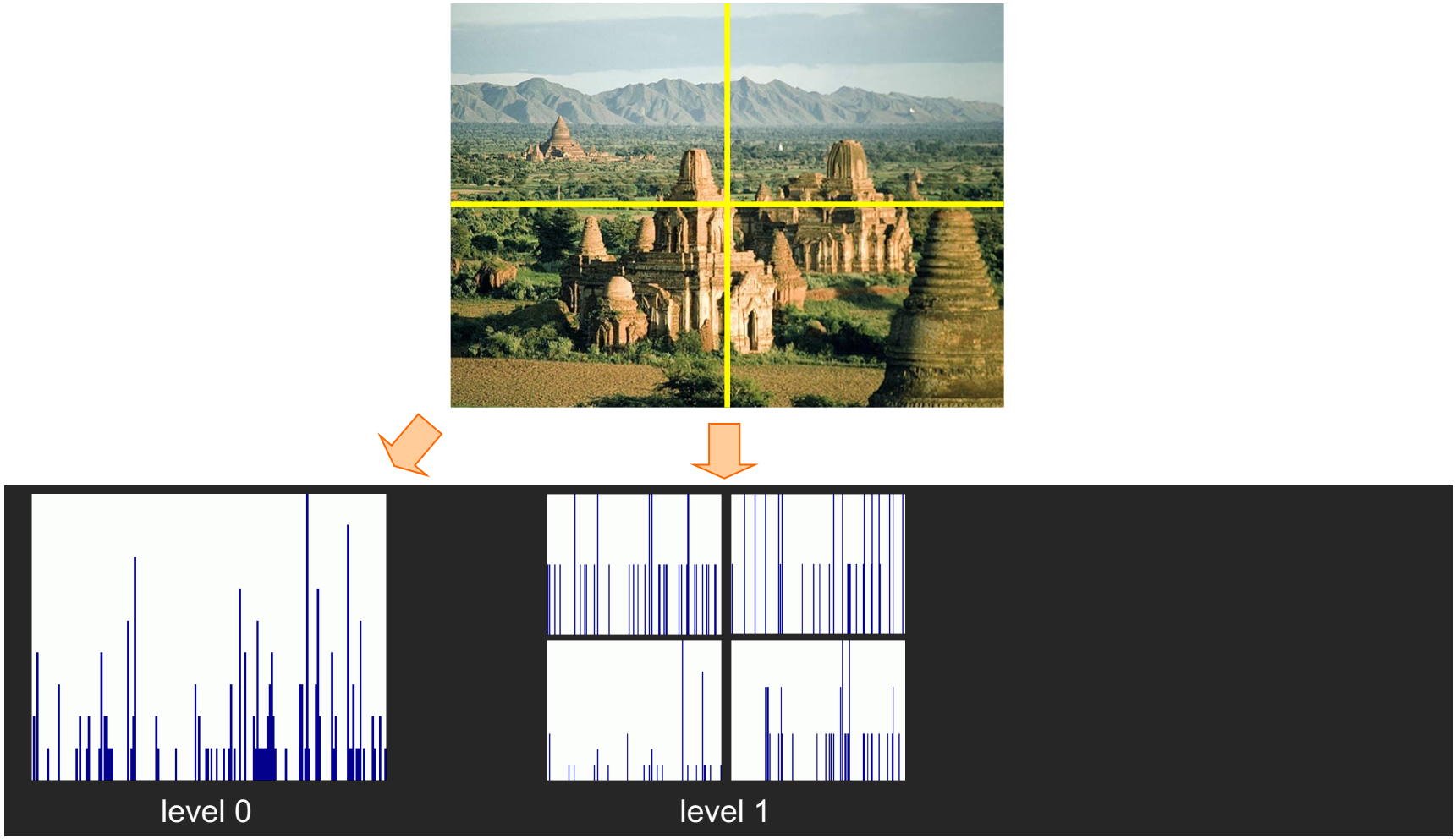
Lazebnik, Schmid & Ponce (CVPR 2006)

Extension of a bag of features

Locally orderless representation at several levels of resolution

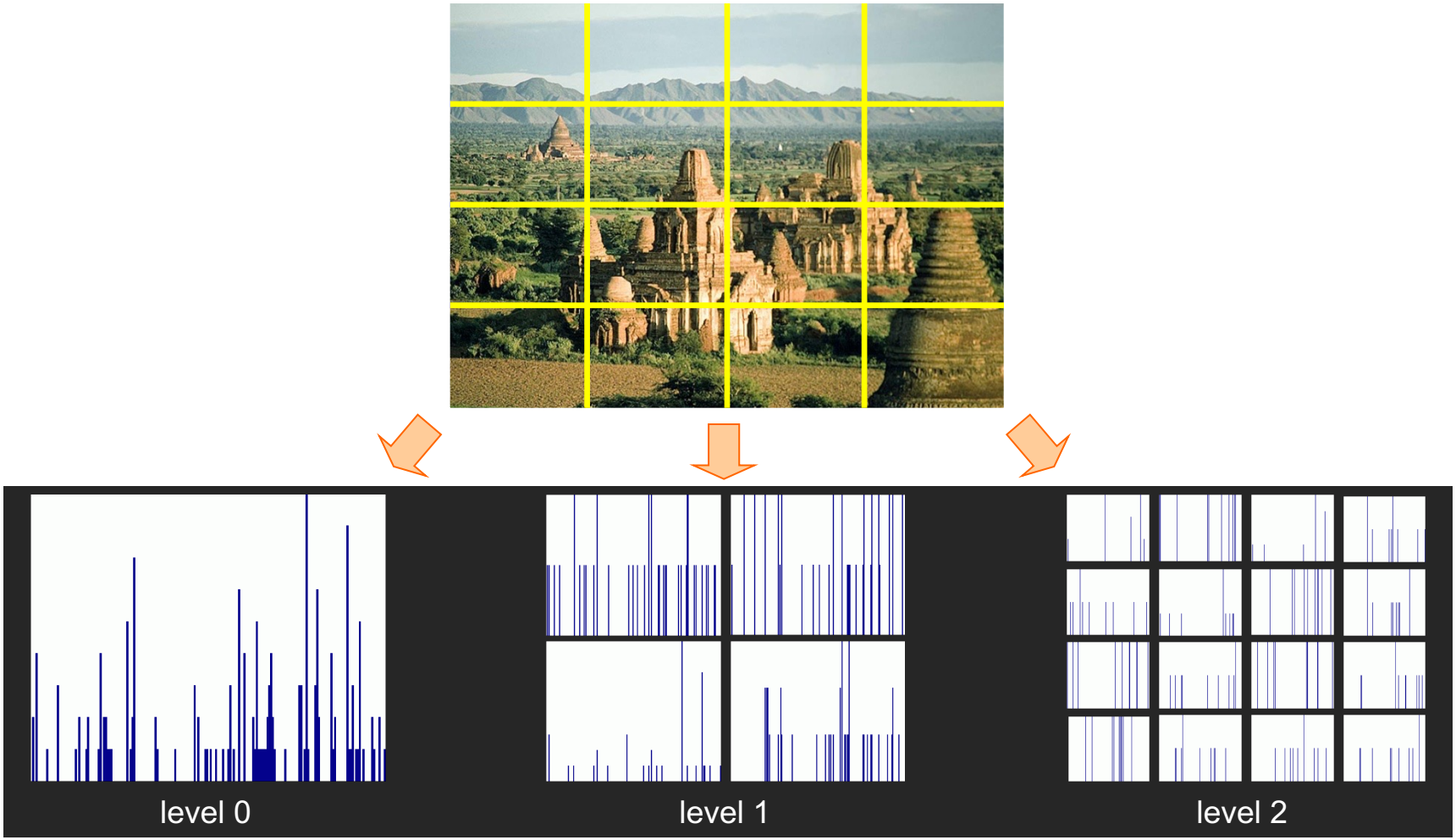
Lazebnik, Schmid & Ponce (CVPR 2006)

Spatial pyramids



Lazebnik, Schmid & Ponce (CVPR 2006)

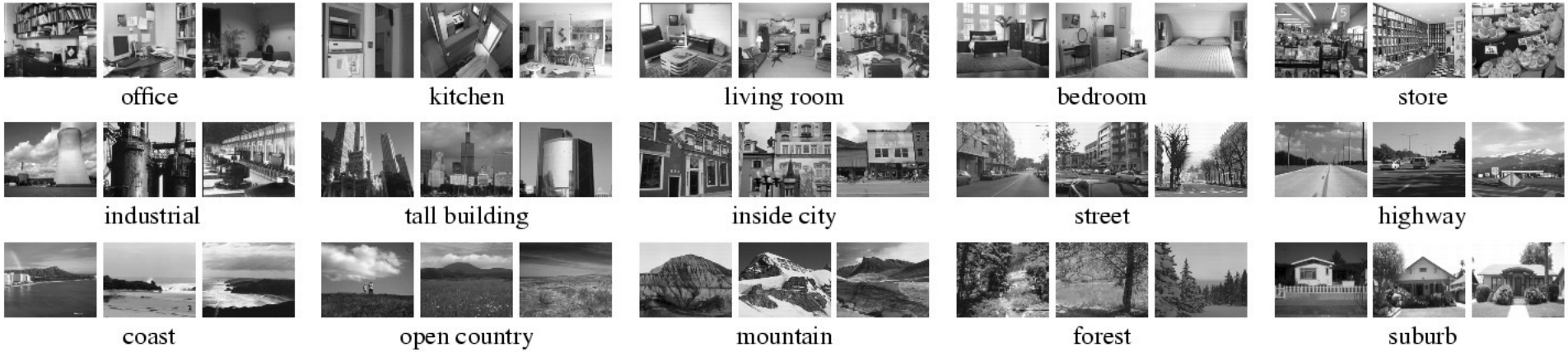
Spatial pyramids



Lazebnik, Schmid & Ponce (CVPR 2006)

Spatial pyramids

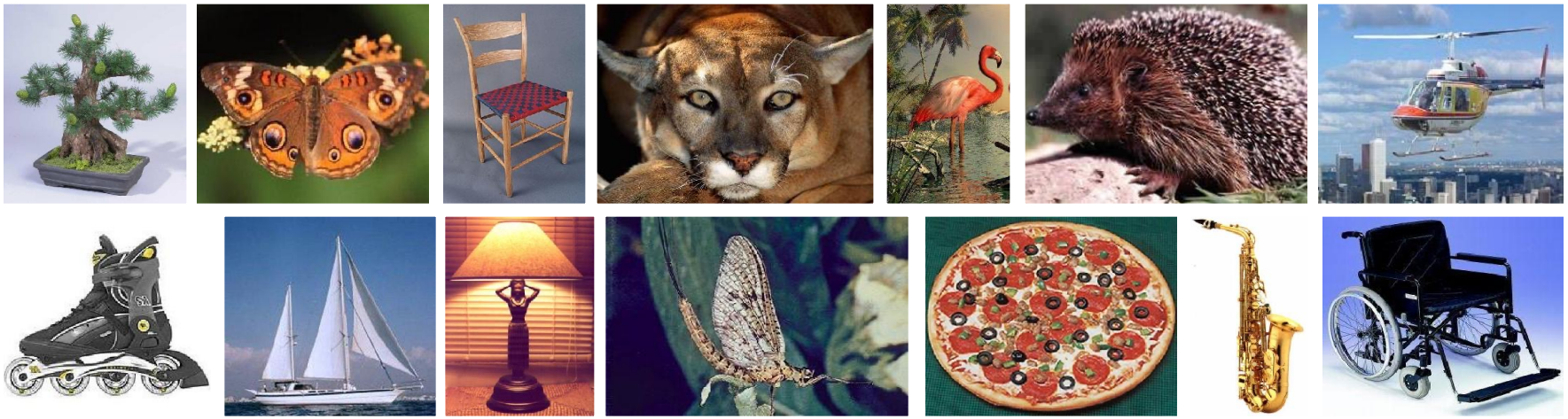
- Scene classification results



	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

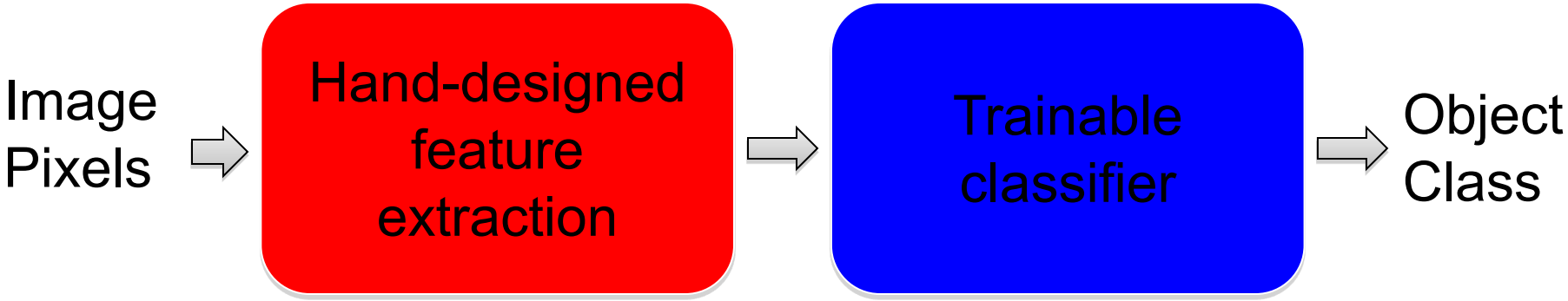
Spatial pyramids

- Caltech101 classification results

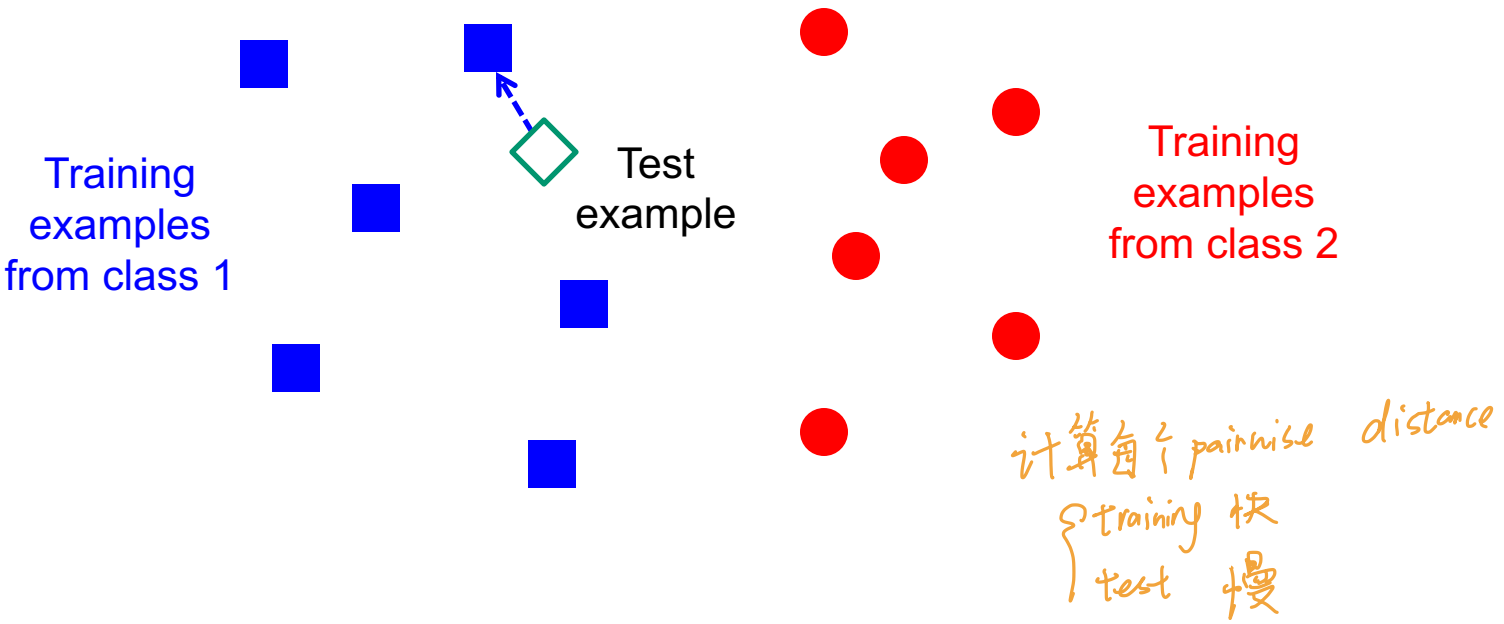


	Weak features (16)		Strong features (200)	
Level	Single-level	Pyramid	Single-level	Pyramid
0	15.5 ±0.9		41.2 ±1.2	
1	31.4 ±1.2	32.8 ±1.3	55.9 ±0.9	57.0 ±0.8
2	47.2 ±1.1	49.3 ±1.4	63.6 ±0.9	**64.6** ±0.8
3	52.2 ±0.8	**54.0** ±1.1	60.3 ±0.9	64.6 ±0.7

Traditional recognition pipeline



Classifiers: Nearest neighbor



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

All we need is a distance function for our inputs

No training required!

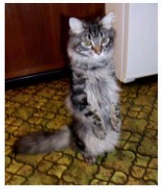
First classifier: **Nearest Neighbor**

deer bird plane cat car

Training data with labels

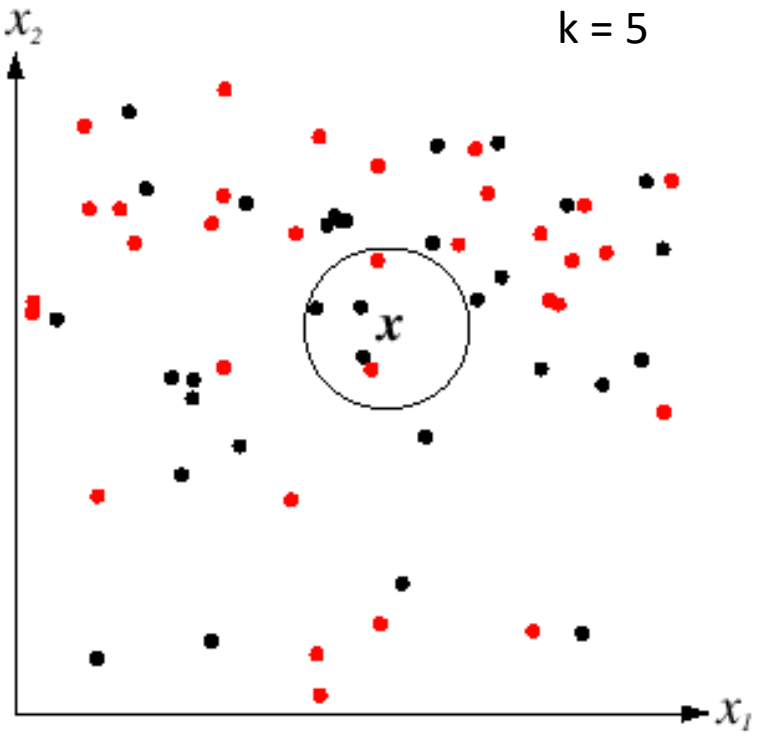
?

query data

Distance Metric $\left|\right.$  ,  $\left|\right. \rightarrow \mathbb{R}$

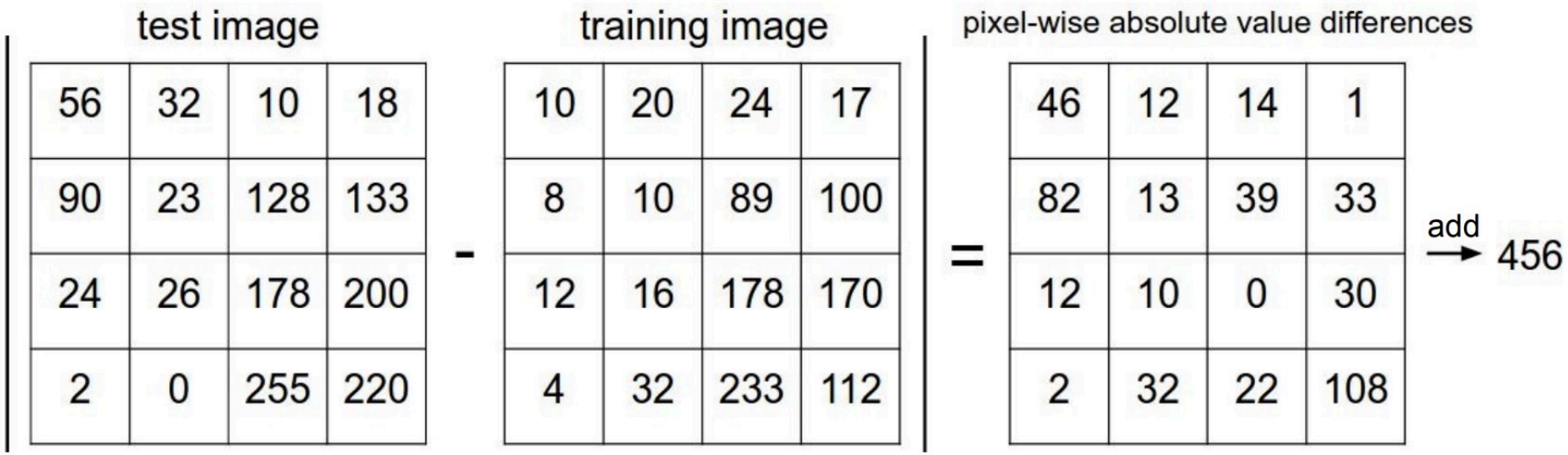
K-nearest neighbor classifier

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



Distance Metric to compare images

L1 distance: $$d_1(I_1, I_2) = \sum_p |I_1^p - I_2^p|$$



```python
import numpy as np

class NearestNeighbor:
  def __init__(self):
    pass

  def train(self, X, y):
    """ X is N x D where each row is an example. Y is 1-dimension of size N """
    # the nearest neighbor classifier simply remembers all the training data
    self.Xtr = X
    self.ytr = y

  def predict(self, X):
    """ X is N x D where each row is an example we wish to predict label for """
    num_test = X.shape[0]
    # lets make sure that the output type matches the input type
    Ypred = np.zeros(num_test, dtype = self.ytr.dtype)

    # loop over all test rows
    for i in xrange(num_test):
      # find the nearest training image to the i'th test image
      # using the L1 distance (sum of absolute value differences)
      distances = np.sum(np.abs(self.Xtr - X[i,:]), axis = 1)
      min_index = np.argmin(distances) # get the index with smallest distance
      Ypred[i] = self.ytr[min_index] # predict the label of the nearest example

    return Ypred
```

Nearest Neighbor classifier

无参数 -> 无学习过程 Memoize

时间都用在 prediction

Nearest Neighbor classifier

```python
import numpy as np

class NearestNeighbor:
  def __init__(self):
    pass

  def train(self, X, y):
    """ X is N x D where each row is an example. Y is 1-dimension of size N """
    # the nearest neighbor classifier simply remembers all the training data
    self.Xtr = X
    self.ytr = y

  def predict(self, X):
    """ X is N x D where each row is an example we wish to predict label for """
    num_test = X.shape[0]
    # lets make sure that the output type matches the input type
    Ypred = np.zeros(num_test, dtype = self.ytr.dtype)

    # loop over all test rows
    for i in xrange(num_test):
      # find the nearest training image to the i'th test image
      # using the L1 distance (sum of absolute value differences)
      distances = np.sum(np.abs(self.Xtr - X[i,:]), axis = 1)
      min_index = np.argmin(distances) # get the index with smallest distance
      Ypred[i] = self.ytr[min_index] # predict the label of the nearest example

    return Ypred
```

Memorize training data

Nearest Neighbor classifier

```python
import numpy as np

class NearestNeighbor:
  def __init__(self):
    pass

  def train(self, X, y):
    """ X is N x D where each row is an example. Y is 1-dimension of size N """
    # the nearest neighbor classifier simply remembers all the training data
    self.Xtr = X
    self.ytr = y

  def predict(self, X):
    """ X is N x D where each row is an example we wish to predict label for """
    num_test = X.shape[0]
    # lets make sure that the output type matches the input type
    Ypred = np.zeros(num_test, dtype = self.ytr.dtype)

    # loop over all test rows
    for i in xrange(num_test):
      # find the nearest training image to the i'th test image
      # using the L1 distance (sum of absolute value differences)
      distances = np.sum(np.abs(self.Xtr - X[i,:]), axis = 1)
      min_index = np.argmin(distances) # get the index with smallest distance
      Ypred[i] = self.ytr[min_index] # predict the label of the nearest example

    return Ypred
```

For each test image:
  Find closest train image
  Predict label of nearest image

```python
import numpy as np

class NearestNeighbor:
  def __init__(self):
    pass

  def train(self, X, y):
    """ X is N x D where each row is an example. Y is 1-dimension of size N """
    # the nearest neighbor classifier simply remembers all the training data
    self.Xtr = X
    self.ytr = y

  def predict(self, X):
    """ X is N x D where each row is an example we wish to predict label for """
    num_test = X.shape[0]
    # lets make sure that the output type matches the input type
    Ypred = np.zeros(num_test, dtype = self.ytr.dtype)

    # loop over all test rows
    for i in xrange(num_test):
      # find the nearest training image to the i'th test image
      # using the L1 distance (sum of absolute value differences)
      distances = np.sum(np.abs(self.Xtr - X[i,:]), axis = 1)
      min_index = np.argmin(distances) # get the index with smallest distance
      Ypred[i] = self.ytr[min_index] # predict the label of the nearest example

    return Ypred
```

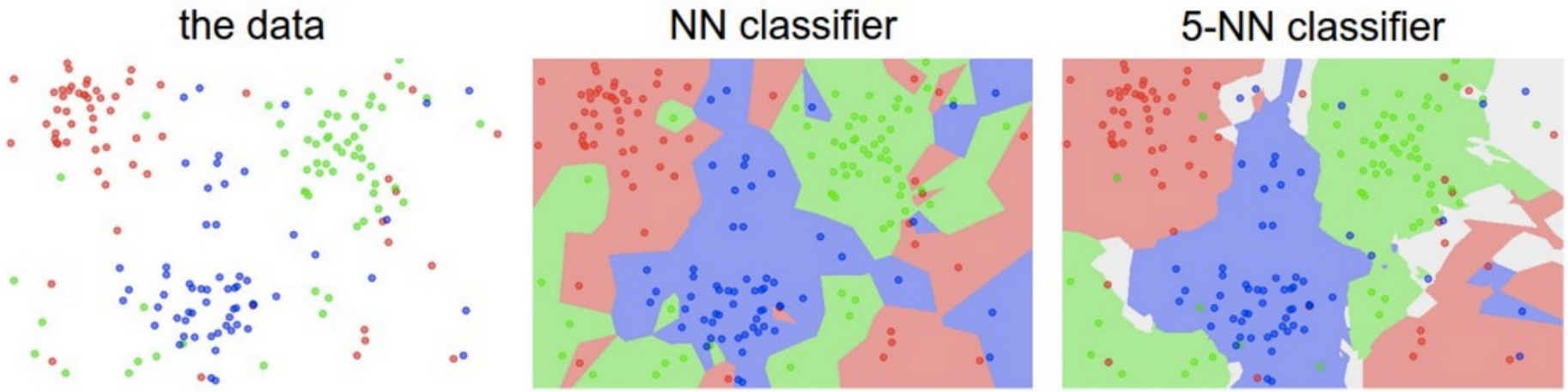
Nearest Neighbor classifier

Q: With N examples, how fast are training and prediction?

Ans: Train O(1),
predict O(N)

This is bad: we want classifiers that are **fast** at prediction; **slow** for training is ok

K-nearest neighbor classifier



Which classifier is more robust to *outliers*?

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

Hyperparameters

What is the best value of **k** to use?
What is the best **distance** to use?

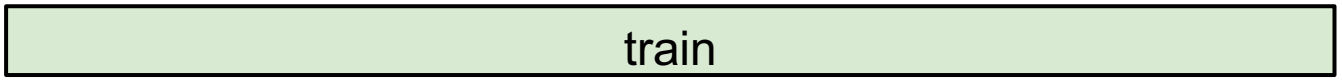
These are **hyperparameters**: choices about the algorithms themselves.

Very problem/dataset-dependent.
Must try them all out and see what works best.

Setting Hyperparameters

cross-validation

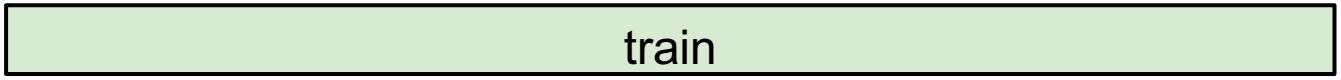
Idea #1: Choose hyperparameters that work best on the **training data**



Setting Hyperparameters

Idea #1: Choose hyperparameters that work best on the **training data**

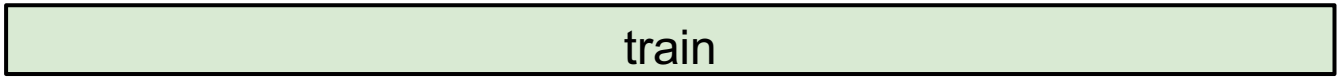
BAD: K = 1 always works perfectly on training data



Setting Hyperparameters

Idea #1: Choose hyperparameters that work best on the **training data**

BAD: K = 1 always works perfectly on training data

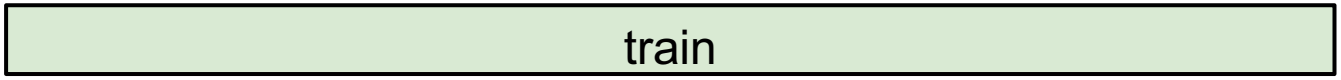


Idea #2: choose hyperparameters that work best on **test** data

Setting Hyperparameters

Idea #1: Choose hyperparameters that work best on the **training data**

BAD: K = 1 always works perfectly on training data



Idea #2: choose hyperparameters that work best on **test** data

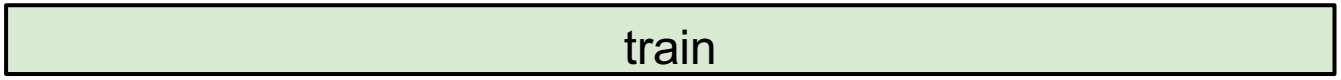
BAD: No idea how algorithm will perform on new data

Never do this!

Setting Hyperparameters

Idea #1: Choose hyperparameters that work best on the **training data**

BAD: K = 1 always works perfectly on training data



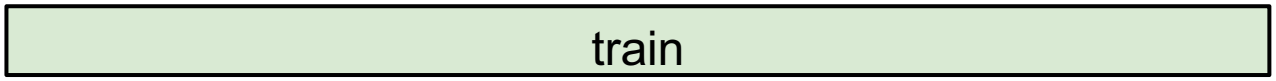
Idea #2: choose hyperparameters that work best on **test** data

BAD: No idea how algorithm will perform on new data

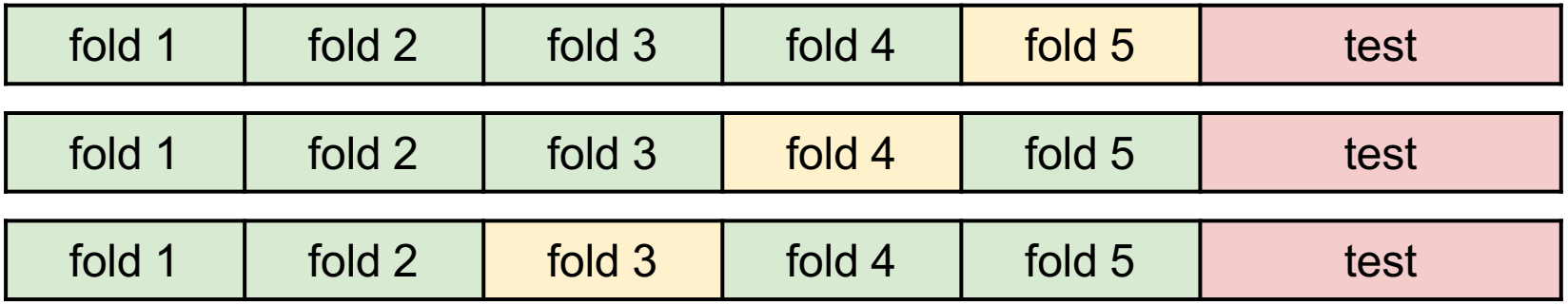
Idea #3: Split data into **train**, **val**; choose hyperparameters on val and evaluate on test

Better!

Setting Hyperparameters



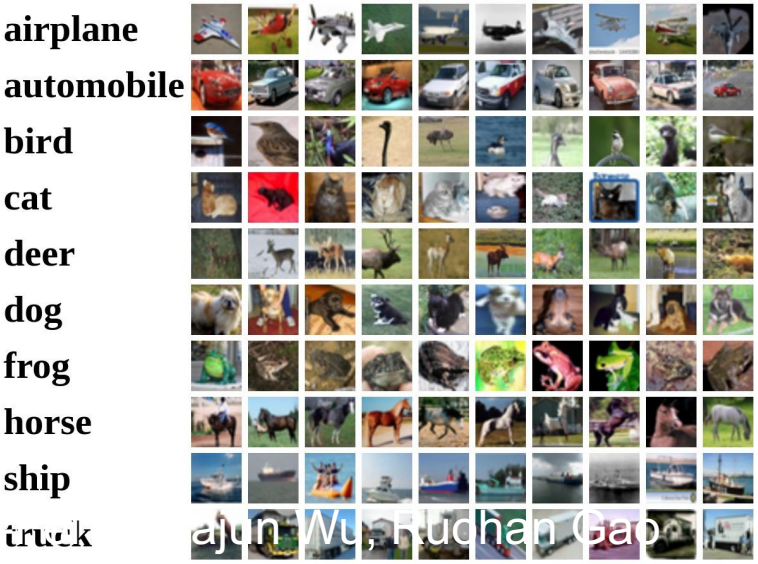
Idea #4: **Cross-Validation**: Split data into **folds**,
try each fold as validation and average the results



Useful for small datasets, but not used too frequently in deep learning

Example Dataset: **CIFAR10**

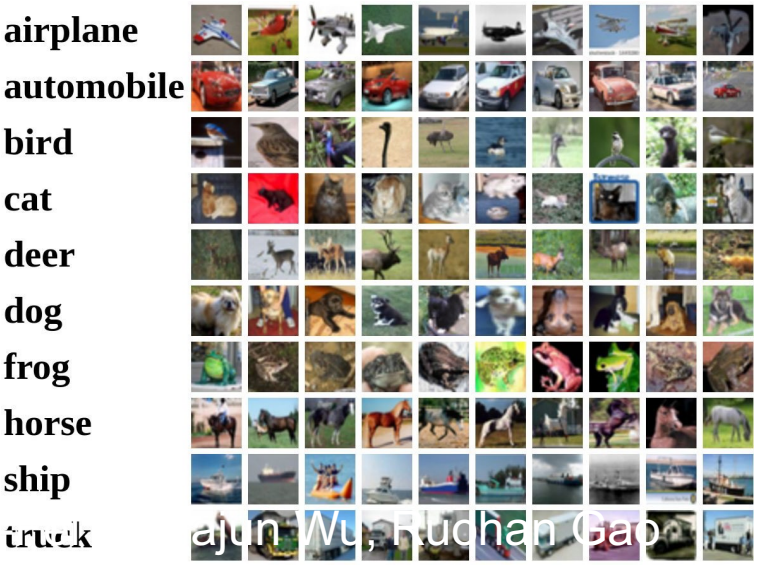
10 classes
50,000 training images
10,000 testing images



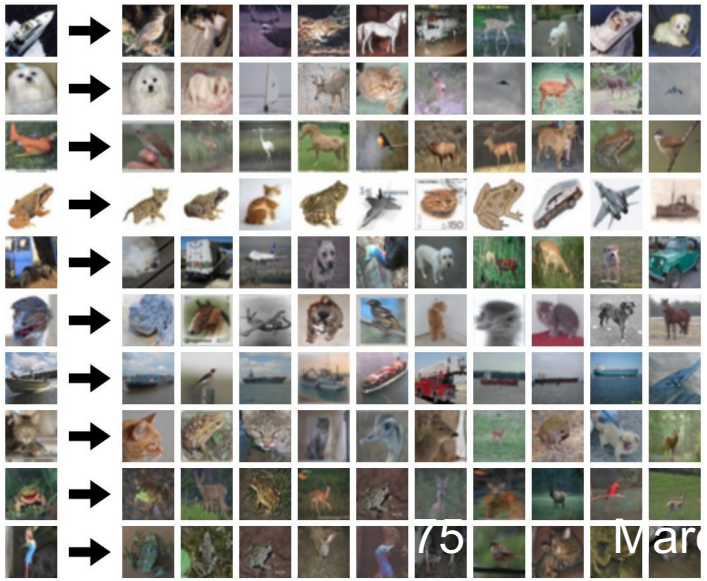
Alex Krizhevsky, “Learning Multiple Layers of Features from Tiny Images”, Technical Report, 2009.

Example Dataset: **CIFAR10**

10 classes
50,000 training images
10,000 testing images

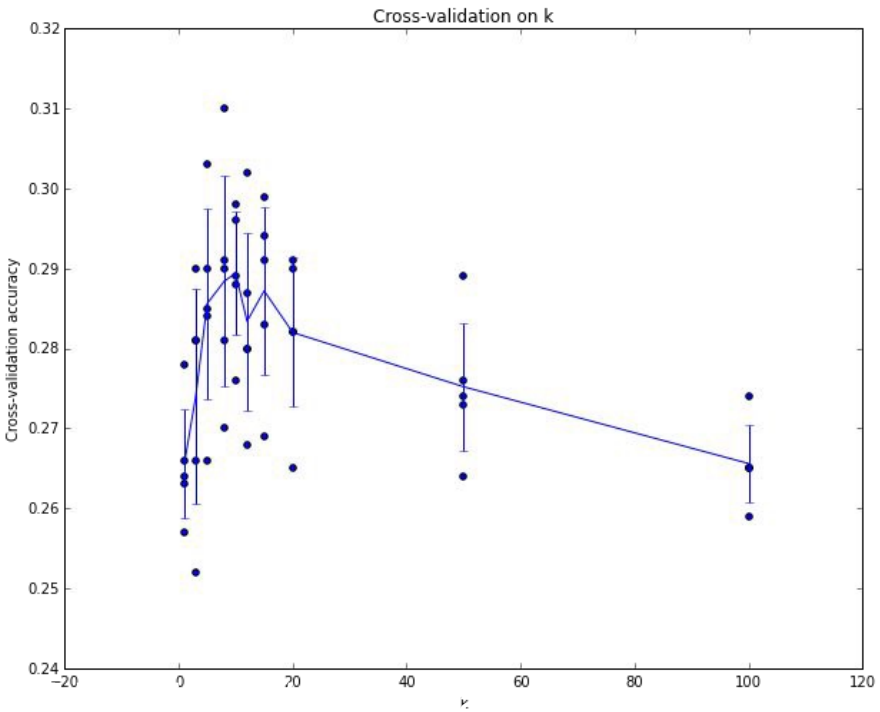


Test images and nearest neighbors



Alex Krizhevsky, “Learning Multiple Layers of Features from Tiny Images”, Technical Report, 2009.

Setting Hyperparameters



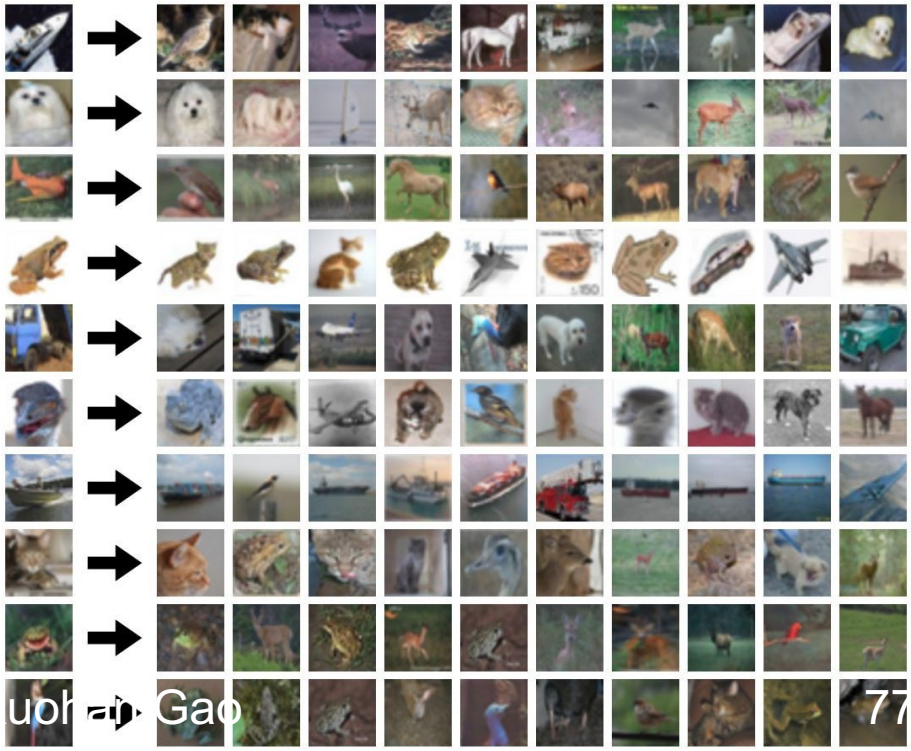
Example of
5-fold cross-validation
for the value of **k.**

Each point: single
outcome.

The line goes
through the mean, bars
indicated standard
deviation

(Seems that k ~= 7 works best
for this data)

What does this look like?



What does this look like?

k-Nearest Neighbor with pixel distance **never used.**

- Distance metrics on pixels are not informative

Original

Occluded

Shifted (1 pixel)

Tinted

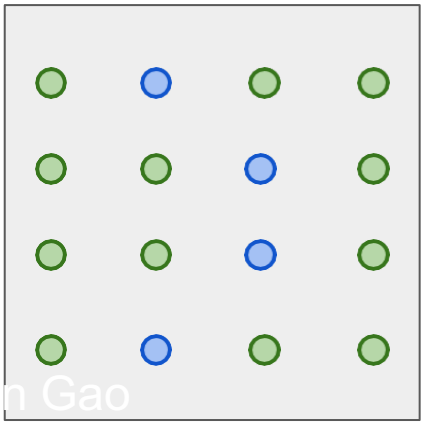
(All three images on the right have the same pixel distances to the one on the left)

k-Nearest Neighbor with pixel distance **never used.**

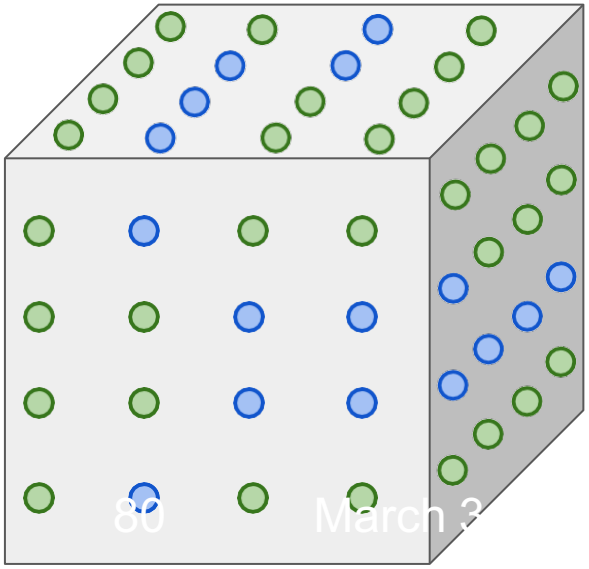
- Curse of dimensionality

Dimensions = 1
Points = 4

Dimensions = 2
Points = 4^2



Dimensions = 3
Points = 4^3



K-Nearest Neighbors: Summary

In **image classification** we start with a **training set** of images and labels, and must predict labels on the **test set**

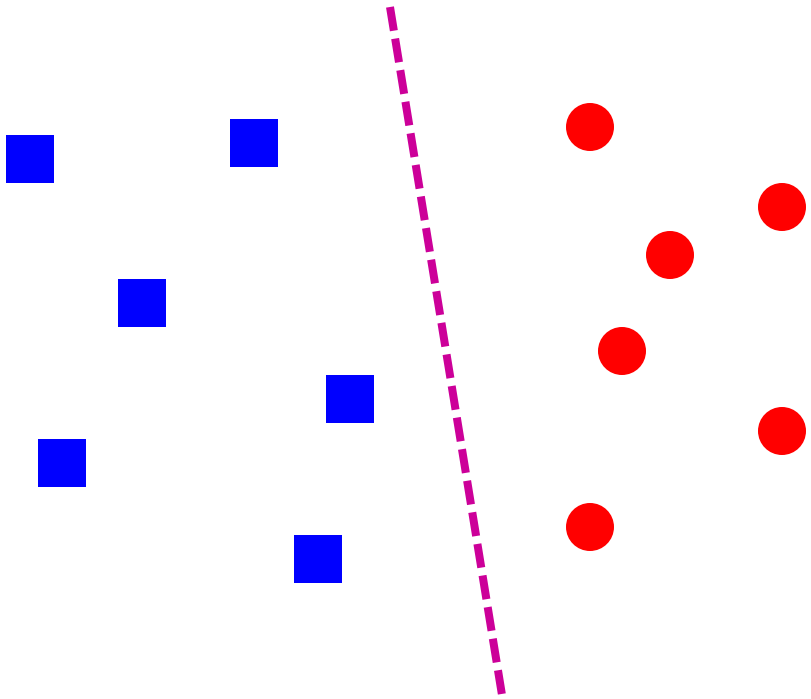
The **K-Nearest Neighbors** classifier predicts labels based on the K nearest training examples

Distance metric and K are **hyperparameters**

Choose hyperparameters using the **validation set**;

Only run on the test set once at the very end!

Linear classifiers



Find a *linear function* to separate the classes:

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

>0 1

<0 -1

Parametric Approach: Linear Classifier

$$f(x,W) = Wx$$

Image

Array of **32x32x3** numbers
(3072 numbers total)

f(**x**,**W**)

10 numbers giving
class scores

W

parameters
or weights

Parametric Approach: Linear Classifier

$$f(x, W) = Wx$$

3072x1

10x1

10x3072

Image

Array of **32x32x3** numbers
(3072 numbers total)

f(**x**, **W**)

10 numbers giving class scores

W

parameters or weights

Parametric Approach: Linear Classifier

$$f(x, W) = Wx + b$$

3072x1 (x)

10x1 (b)

10x1 (f(x,W))

10x3072 (W)

Image

Array of **32x32x3** numbers
(3072 numbers total)

f(**x**,**W**)

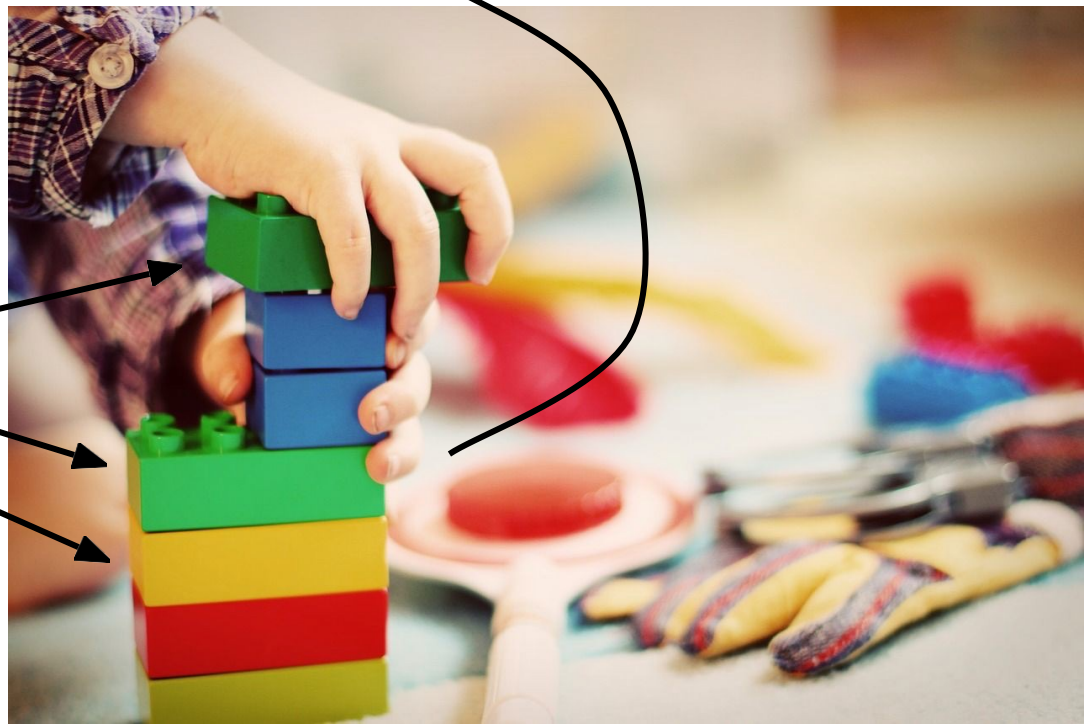
10 numbers giving class scores

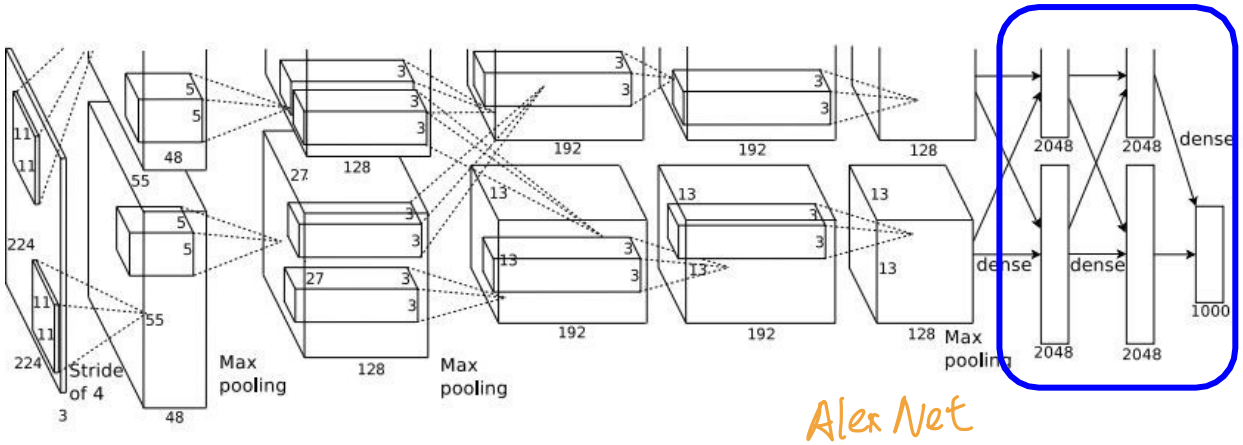
W

parameters or weights

Neural Network

Linear classifiers

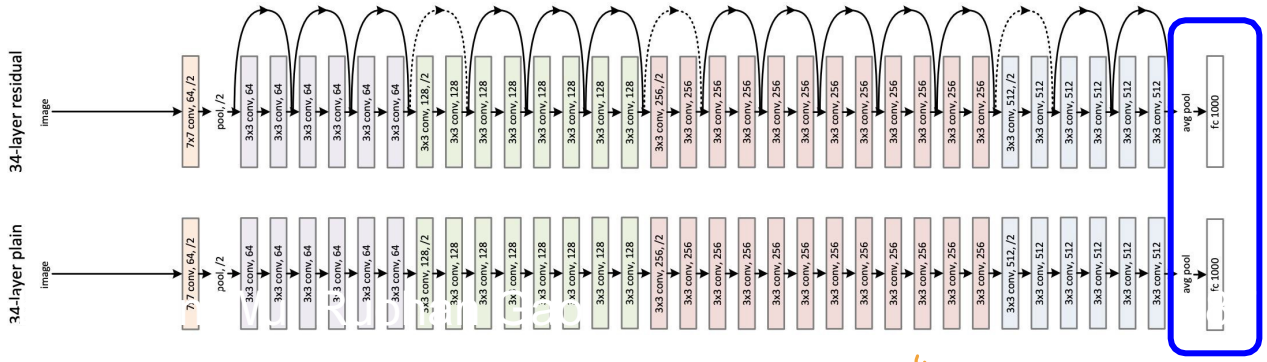




Alex Net

[Krizhevsky et al. 2012]

Linear layers



[He et al. 2015]

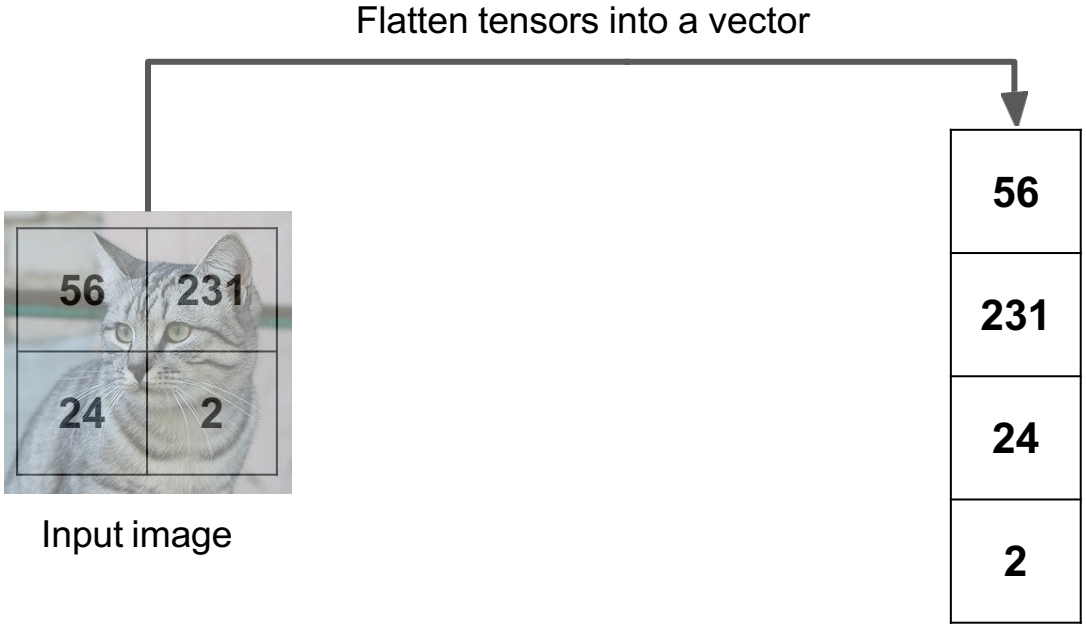
Res Net

Recall CIFAR10

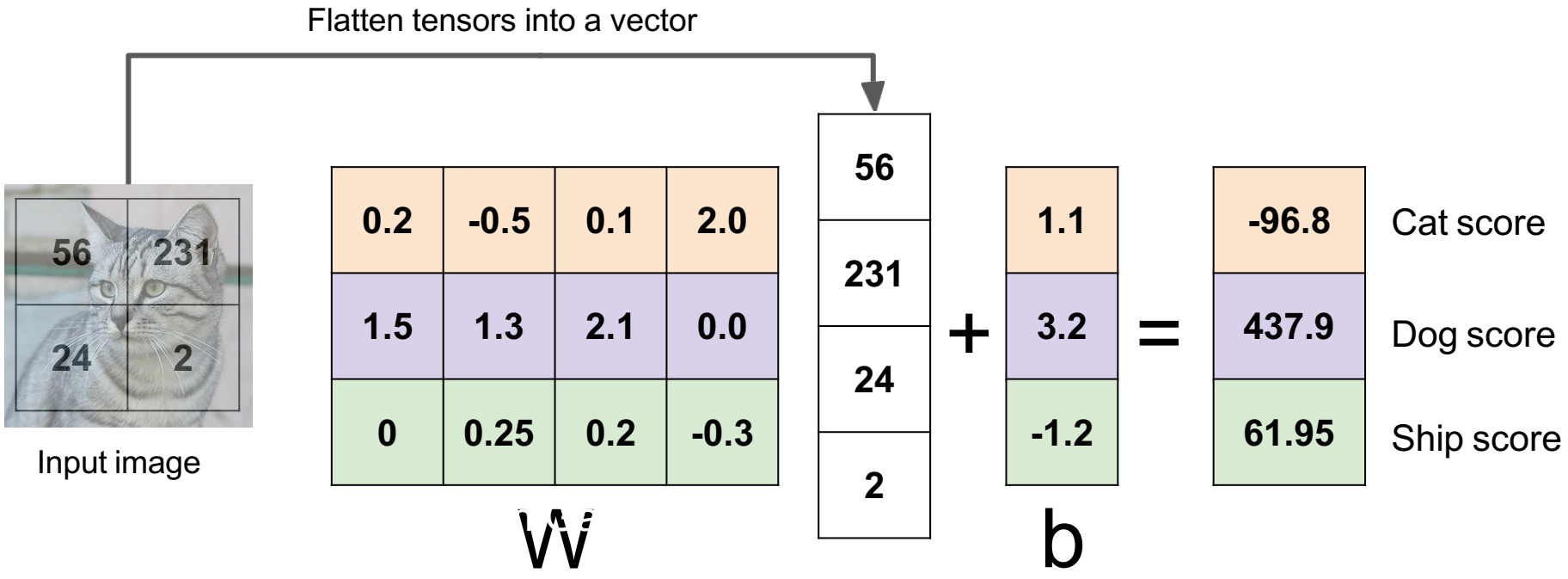
50,000 training images
each image is **32x32x3**

10,000 test images.

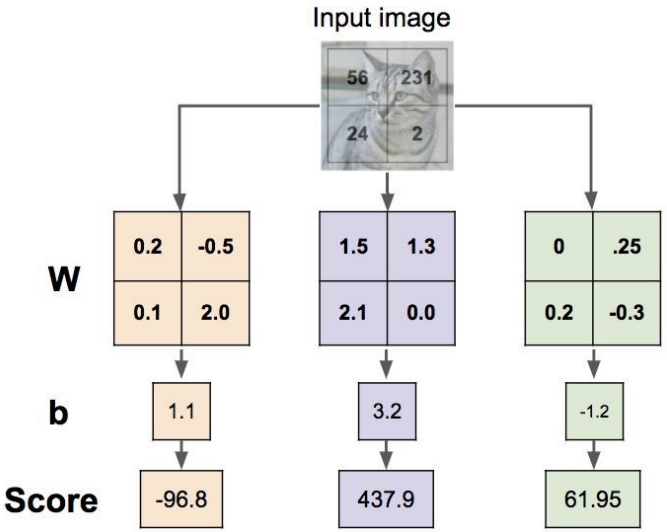
Example with an image with 4 pixels, and 3 classes (cat/dog/ship)



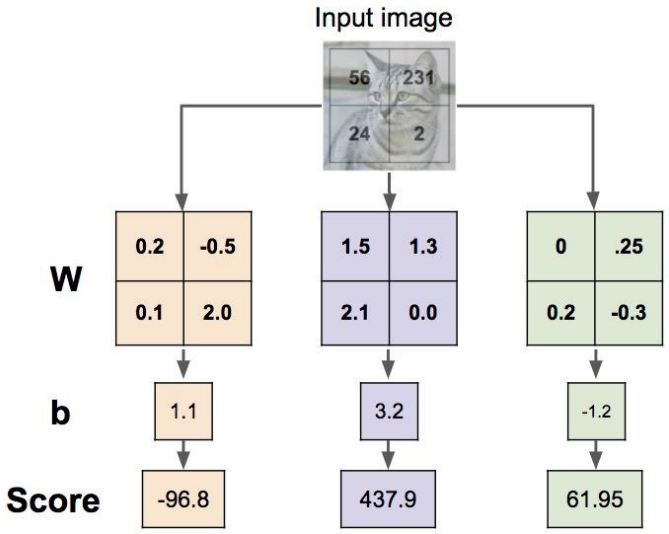
Example with an image with 4 pixels, and 3 classes (cat/dog/ship) Algebraic Viewpoint



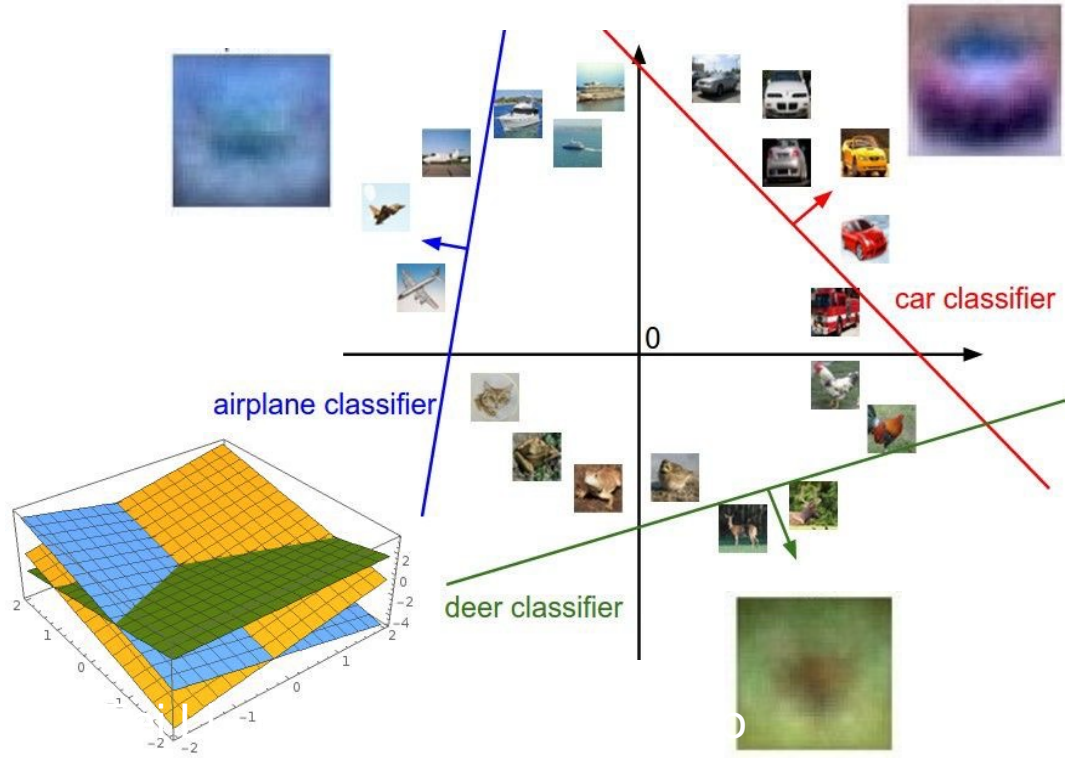
Interpreting a Linear Classifier



Interpreting a Linear Classifier: Visual Viewpoint



Interpreting a Linear Classifier: Geometric Viewpoint



$$f(x,W) = Wx + b$$

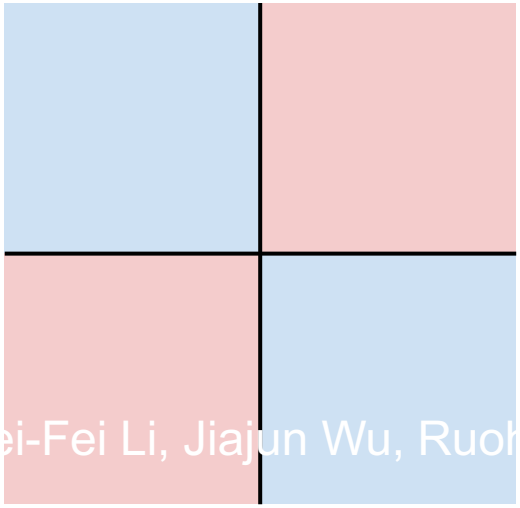
Array of **32x32x3** numbers
(3072 numbers total)

Plot created using Wolfram Cloud

Hard cases for a linear classifier

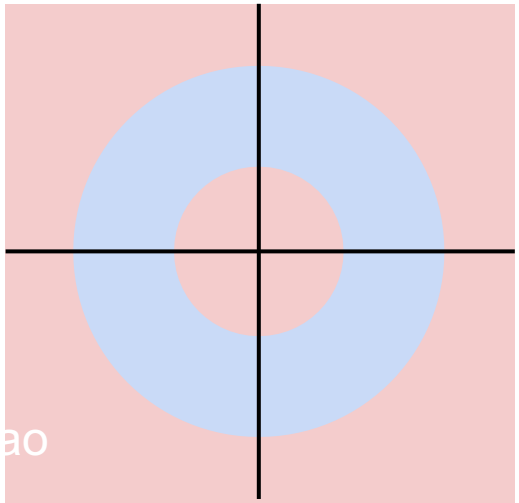
Class 1:
First and third quadrants

Class 2:
Second and fourth quadrants



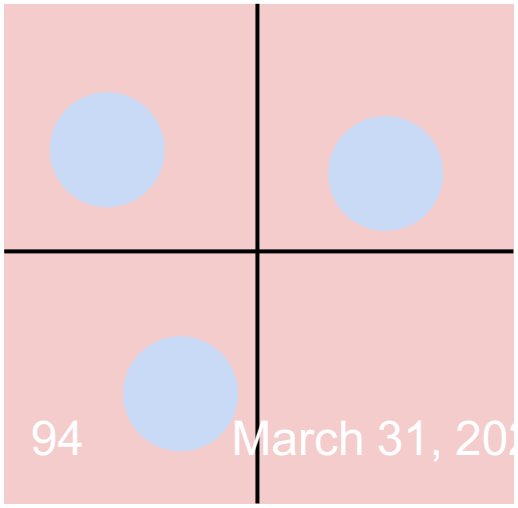
Class 1:
1 <= L2 norm <= 2

Class 2:
Everything else



Class 1:
Three modes

Class 2:
Everything else



Linear Classifier – Choose a good W

airplane	-3.45	-0.51	3.42
automobile	-8.87	**6.04**	4.64
bird	0.09	5.31	2.65
cat	**2.9**	-4.22	5.1
deer	4.48	-4.19	2.64
dog	8.02	3.58	5.55
frog	3.78	4.49	**-4.34**
horse	1.06	-4.37	-1.5
ship	-0.36	-2.09	-4.79
truck	-0.72	-2.93	6.14

TODO:

1. Define a **loss function** that quantifies our unhappiness with the scores across the training data.

2.Come up with a way of efficiently finding the parameters that minimize the loss function. **(optimization)**

Suppose: 3 training examples, 3 classes.
With some W the scores $f(x, W) = Wx$ are:



cat	**3.2**	1.3	2.2
car	5.1	**4.9**	2.5
frog	-1.7	2.0	**-3.1**

Suppose: 3 training examples, 3 classes.
With some W the scores $f(x, W) = Wx$ are:



cat	**3.2**	1.3	2.2
car	5.1	**4.9**	2.5
frog	-1.7	2.0	**-3.1**

A **loss function** tells how good our current classifier is

Suppose: 3 training examples, 3 classes.
With some W the scores $f(x, W) = Wx$ are:



cat	**3.2**	1.3	2.2
car	5.1	**4.9**	2.5
frog	-1.7	2.0	**-3.1**

A **loss function** tells how good our current classifier is

Given a dataset of examples

$$\{(x_i, y_i)\}_{i=1}^{N}$$

Where x_i is image and
y_i is (integer) label

Suppose: 3 training examples, 3 classes.
With some W the scores $f(x, W) = Wx$ are:



cat	**3.2**	1.3	2.2
car	5.1	**4.9**	2.5
frog	-1.7	2.0	**-3.1**

A **loss function** tells how good our current classifier is

Given a dataset of examples

$$\{(x_i, y_i)\}_{i=1}^{N}$$

Where x_i is image and
y_i is (integer) label

Loss over the dataset is a average of loss over examples:

$$L = \frac{1}{N} \sum_i L_i(f(x_i, W), y_i)$$

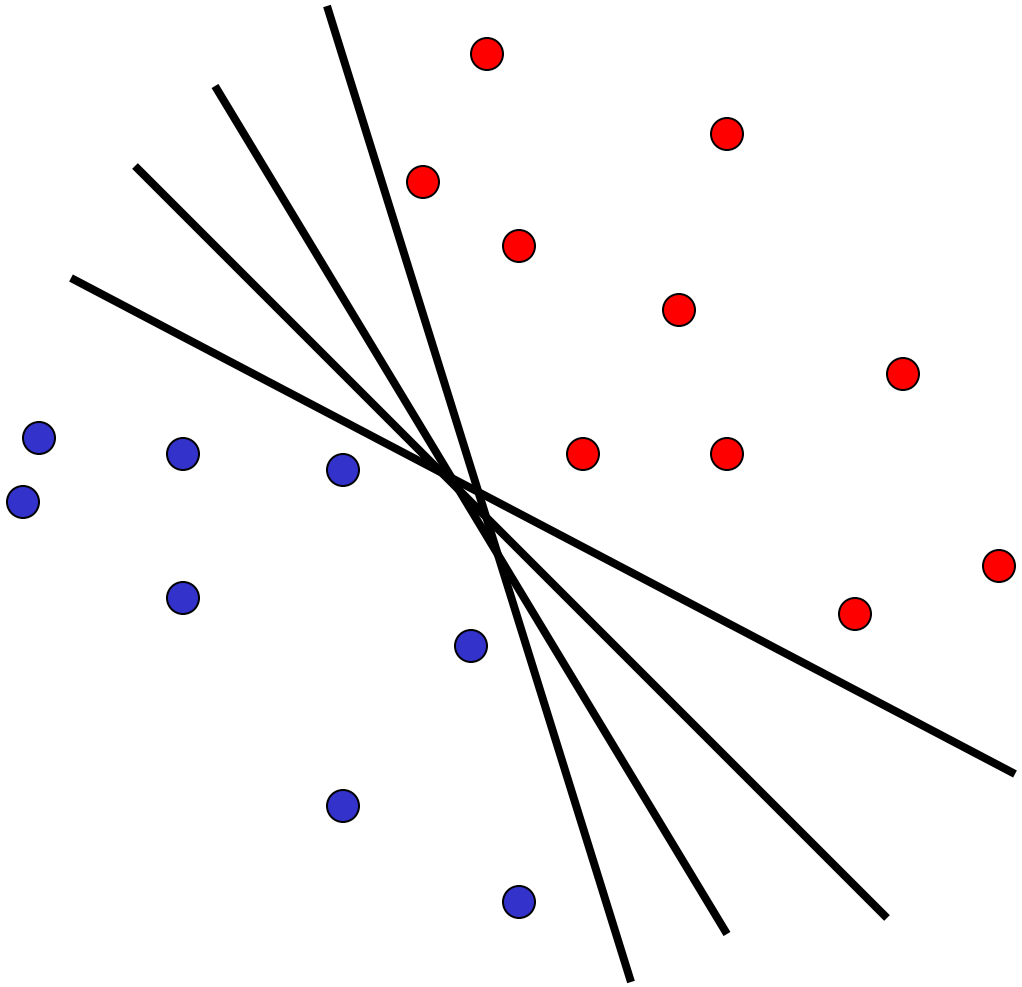
Nearest neighbor vs. linear classifiers

- NN pros:
  - Simple to implement
  - Decision boundaries not necessarily linear
  - Works for any number of classes
  - *Nonparametric* method
- NN cons:
  - Need good distance function
  - Slow at test time
- Linear pros:
  - Low-dimensional *parametric* representation
  - Very fast at test time
- Linear cons:
  - Works for two classes
  - How to train the linear function?
  - What if data is not linearly separable?

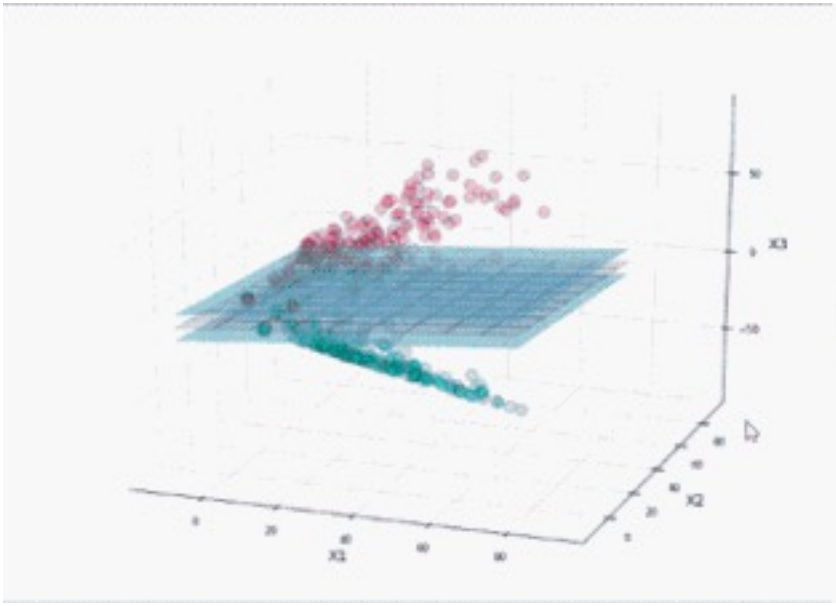
Support vector machines

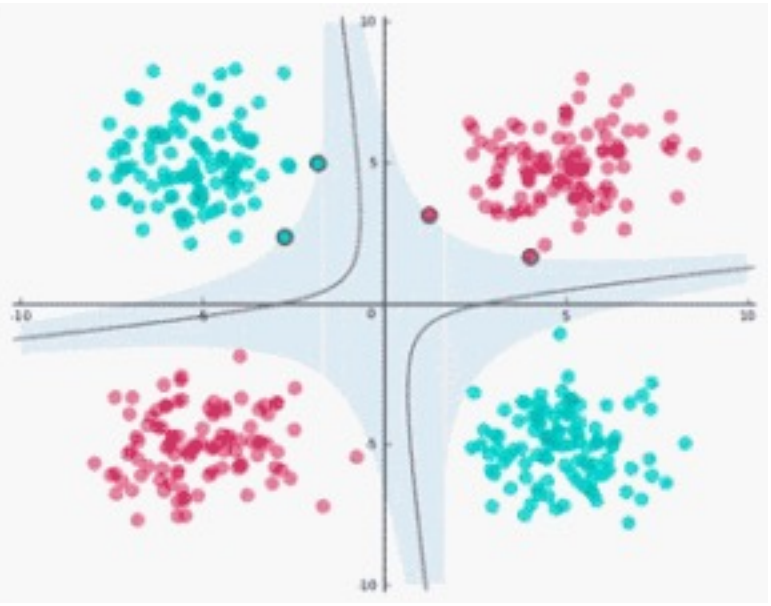
① 找超平面切两类
② 让其与最近点最远

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



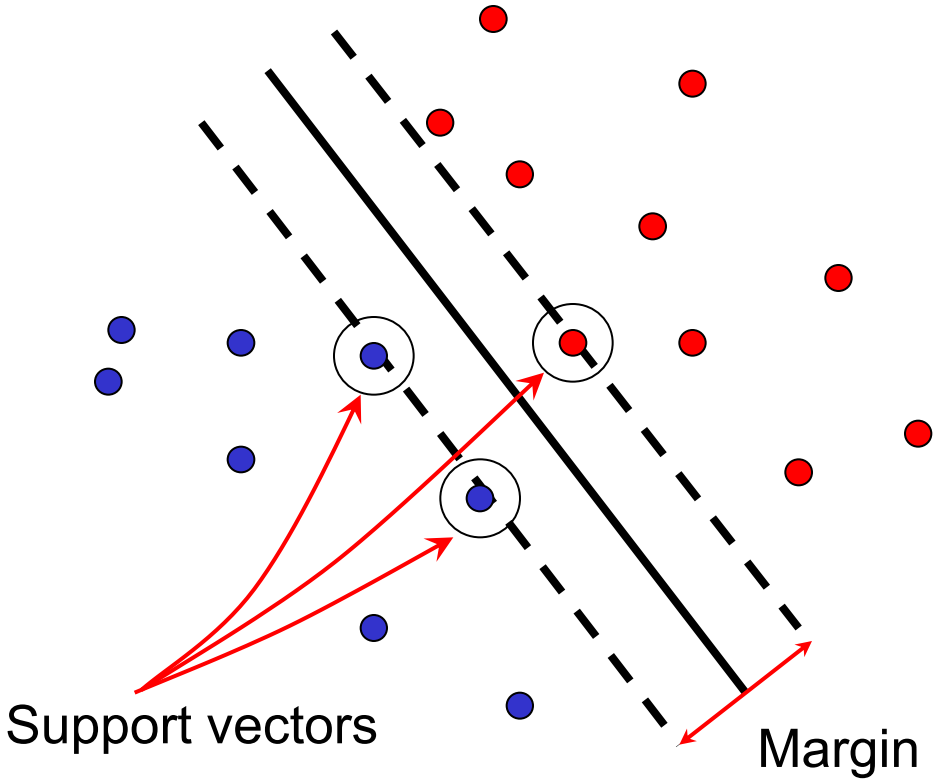
Which separator is best?


X1
X2
X3


10
5
0
-5
-10
-10
-5
0
5
10

Support vector machines

- Find hyperplane that maximizes the *margin* between the positive and negative examples



$\mathbf{x}_i$ positive $(y_i = 1)$: $\mathbf{x}_i \cdot \mathbf{w} + b \geq 1$

$\mathbf{x}_i$ negative $(y_i = -1)$: $\mathbf{x}_i \cdot \mathbf{w} + b \leq -1$

For support vectors, $\mathbf{x}_i \cdot \mathbf{w} + b = \pm 1$

Distance between point and hyperplane: $\dfrac{|\mathbf{x}_i \cdot \mathbf{w} + b|}{\|\mathbf{w}\|}$

Therefore, the margin is $2 / \|\mathbf{w}\|$

C. Burges, A Tutorial on Support Vector Machines for Pattern Recognition, Data Mining and Knowledge Discovery, 1998

Finding the maximum margin hyperplane

1. Maximize margin $2 / \|\mathbf{w}\|$
2. Correctly classify all training data:

$$\mathbf{x}_i \text{ positive } (y_i = 1): \quad \mathbf{x}_i \cdot \mathbf{w} + b \geq 1$$
$$\mathbf{x}_i \text{ negative } (y_i = -1): \quad \mathbf{x}_i \cdot \mathbf{w} + b \leq -1$$

Quadratic optimization problem:

$$\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|^2 \quad \text{subject to} \quad y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1$$

C. Burges, A Tutorial on Support Vector Machines for Pattern Recognition, Data Mining and Knowledge Discovery, 1998

SVM parameter learning

- Separable data: $\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|^2 \quad \text{subject to} \quad y_i(\mathbf{w}\cdot\mathbf{x}_i + b) \geq 1$

  Maximize margin

  Classify training data correctly

- Non-separable data:

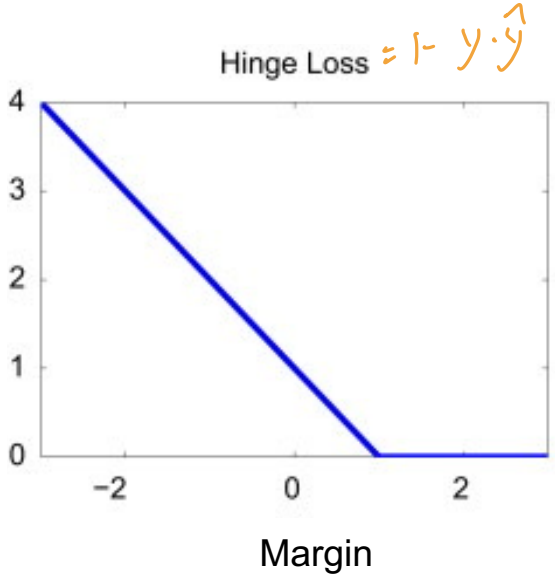
$$\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n}\max\left(0, 1 - y_i(\mathbf{w}\cdot\mathbf{x}_i + b)\right)$$

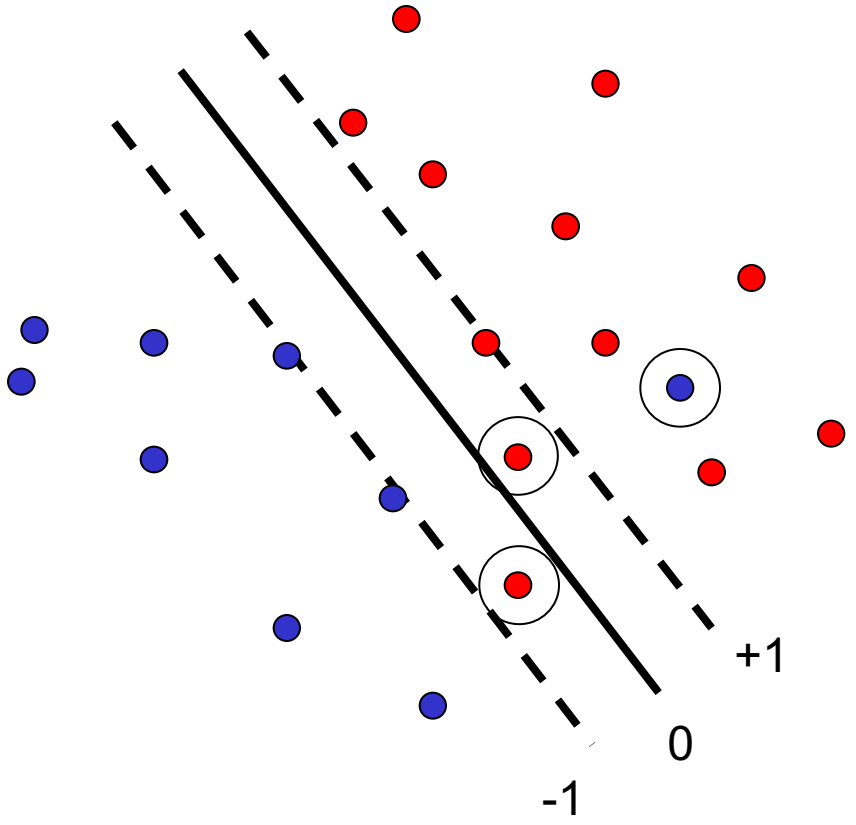
  Maximize margin

  Minimize classification mistakes

SVM parameter learning

$$\min_{\mathbf{w},b} \quad \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n}\max\left(0,1-y_i(\mathbf{w}\cdot\mathbf{x}_i+b)\right)$$





Demo: http://cs.stanford.edu/people/karpathy/svmjs/demo

Nonlinear SVMs

- General idea: the original input space can always be mapped to some higher-dimensional feature space where the training set is separable

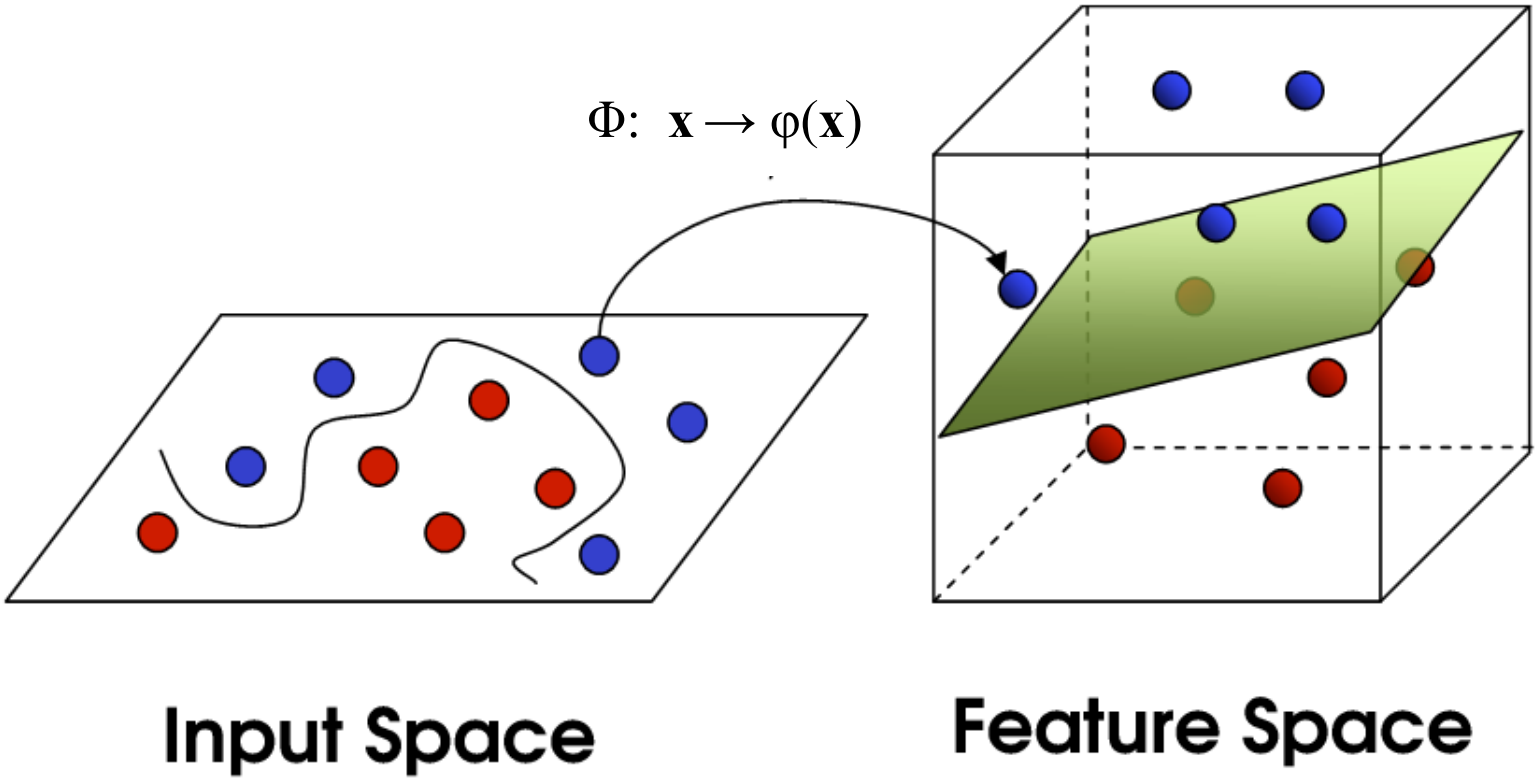
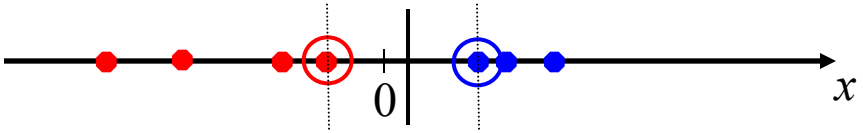


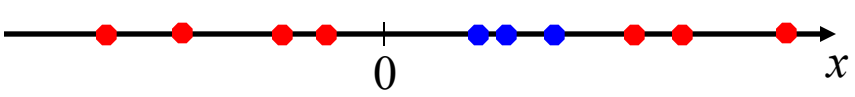
Image source

Nonlinear SVMs

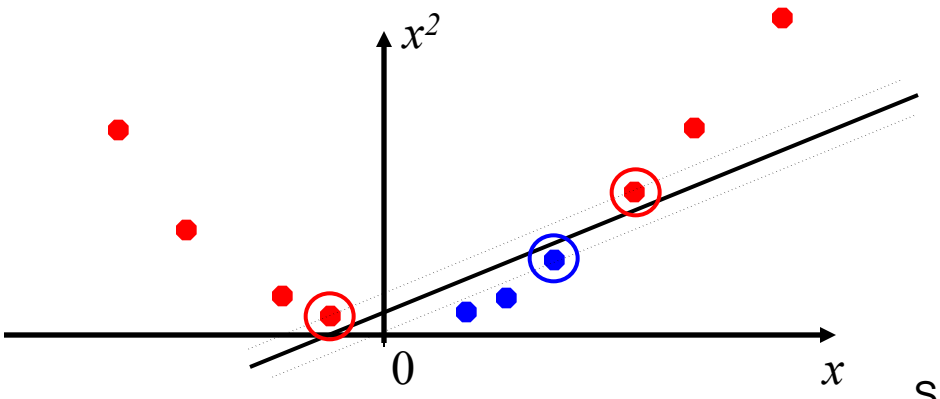
- Linearly separable dataset in 1D:



- Non-separable dataset in 1D:



- We can map the data to a *higher-dimensional space*:



Slide credit: Andrew Moore

The kernel trick

- General idea: the original input space can always be mapped to some higher-dimensional feature space where the training set is separable

- **The kernel trick:** instead of explicitly computing the lifting transformation $\varphi(\mathbf{x})$, define a kernel function K such that

$$K(\mathbf{x}, \mathbf{y}) = \varphi(\mathbf{x}) \cdot \varphi(\mathbf{y})$$

The kernel trick

- Linear SVM decision function:

$$\mathbf{w} \cdot \mathbf{x} + b = \sum_i \alpha_i y_i \mathbf{x}_i \cdot \mathbf{x} + b$$

learned weight

Support vector

C. Burges, A Tutorial on Support Vector Machines for Pattern Recognition, Data Mining and Knowledge Discovery, 1998

The kernel trick

- Linear SVM decision function:

$$\mathbf{w} \cdot \mathbf{x} + b = \sum_i \alpha_i y_i \mathbf{x}_i \cdot \mathbf{x} + b$$

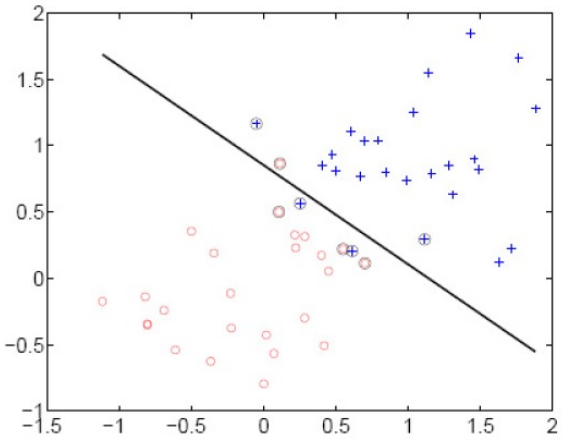
- Kernel SVM decision function:

$$\sum_i \alpha_i y_i \varphi(\mathbf{x}_i) \cdot \varphi(\mathbf{x}) + b = \sum_i \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}) + b$$

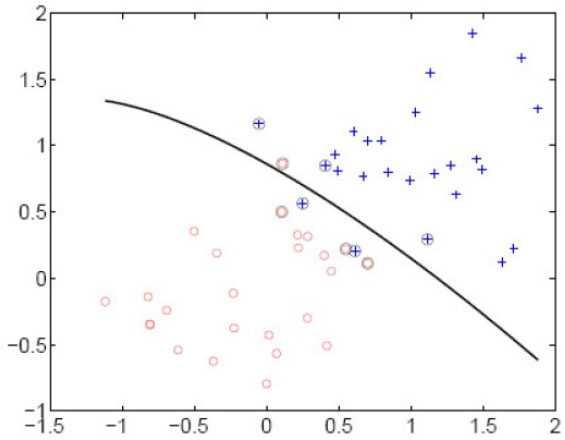
- This gives a nonlinear decision boundary in the original feature space

C. Burges, A Tutorial on Support Vector Machines for Pattern Recognition, Data Mining and Knowledge Discovery, 1998

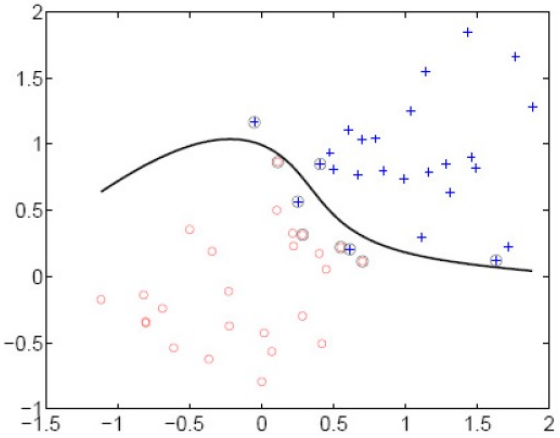
Polynomial kernel: $K(\mathbf{x}, \mathbf{y}) = (c + \mathbf{x} \cdot \mathbf{y})^d$



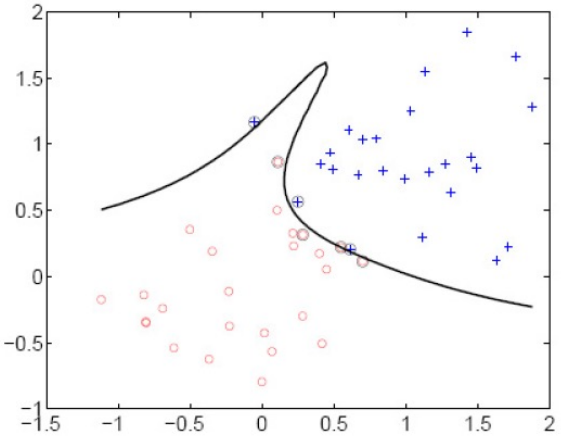
linear



2^{nd} order polynomial



4^{th} order polynomial

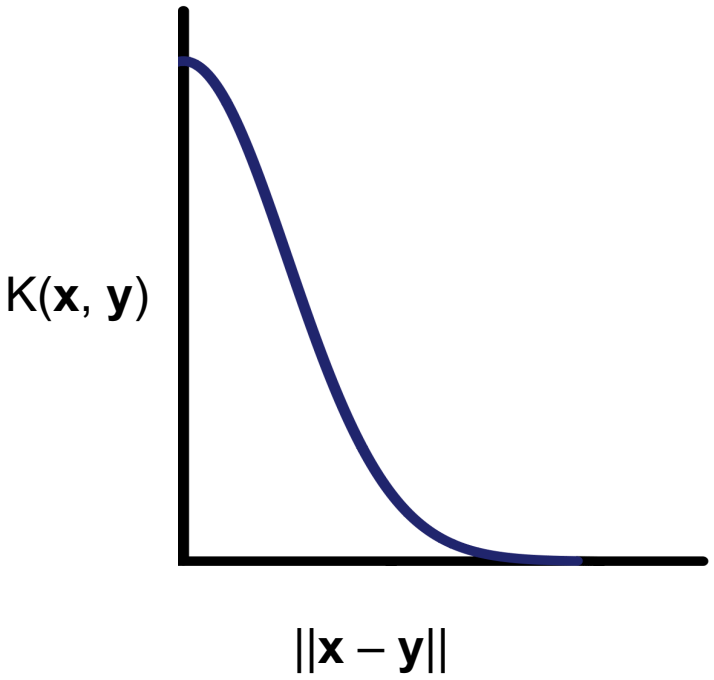


8^{th} order polynomial

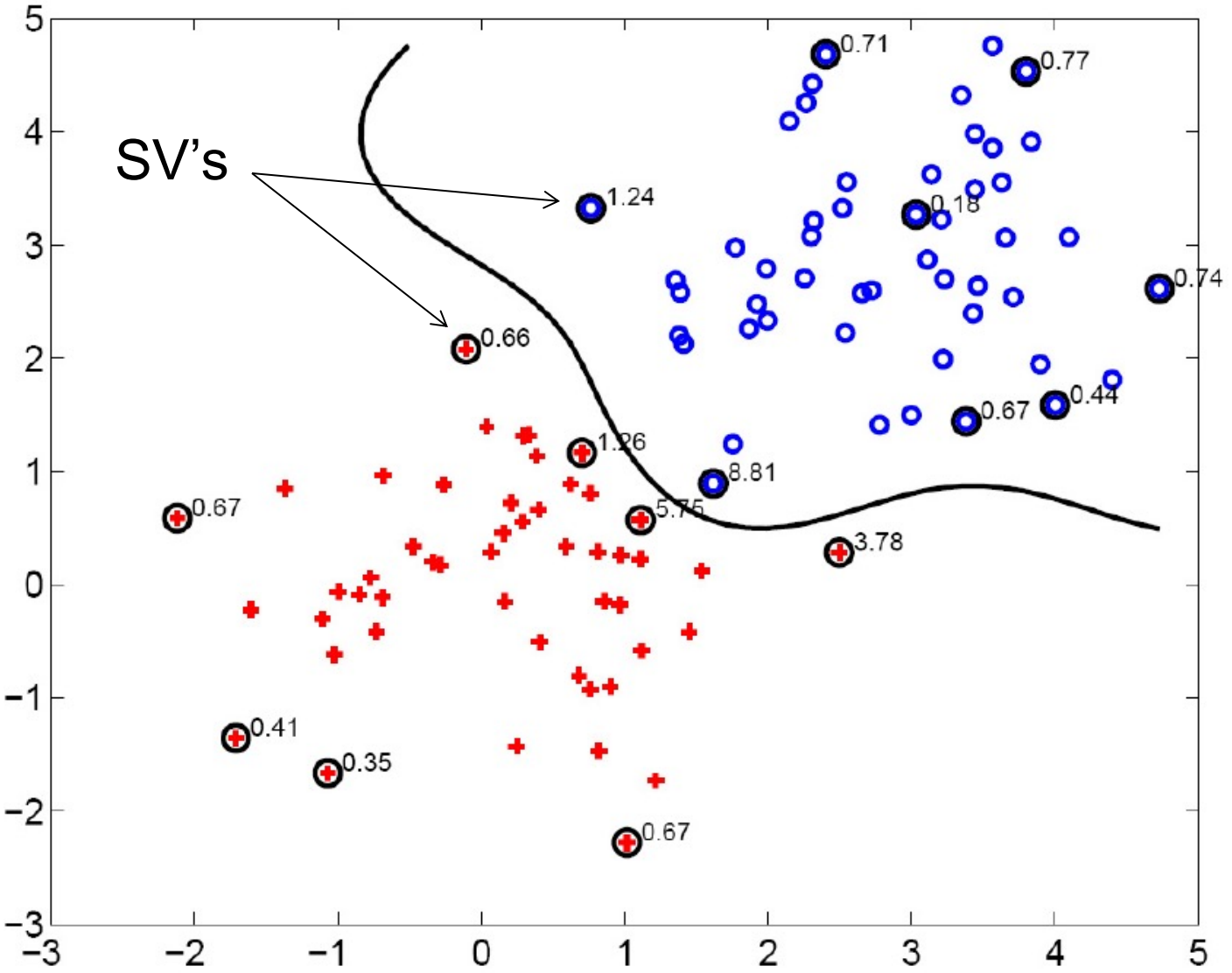
Gaussian kernel

- Also known as the radial basis function (RBF) kernel:

$$K(\mathbf{x}, \mathbf{y}) = \exp\left(-\frac{1}{\sigma^2} \|\mathbf{x} - \mathbf{y}\|^2 \right)$$



Gaussian kernel



SVMs: Pros and cons

- Pros
  - Kernel-based framework is very powerful, flexible
  - Training is convex optimization, globally optimal solution can be found
  - Amenable to theoretical analysis
  - SVMs work very well in practice, even with very small training sample sizes

- Cons
  - No “direct” multi-class SVM, must combine two-class SVMs (e.g., with one-vs-others)
  - Computation, memory (esp. for nonlinear SVMs)

一个SVM 只能二分类

Generalization

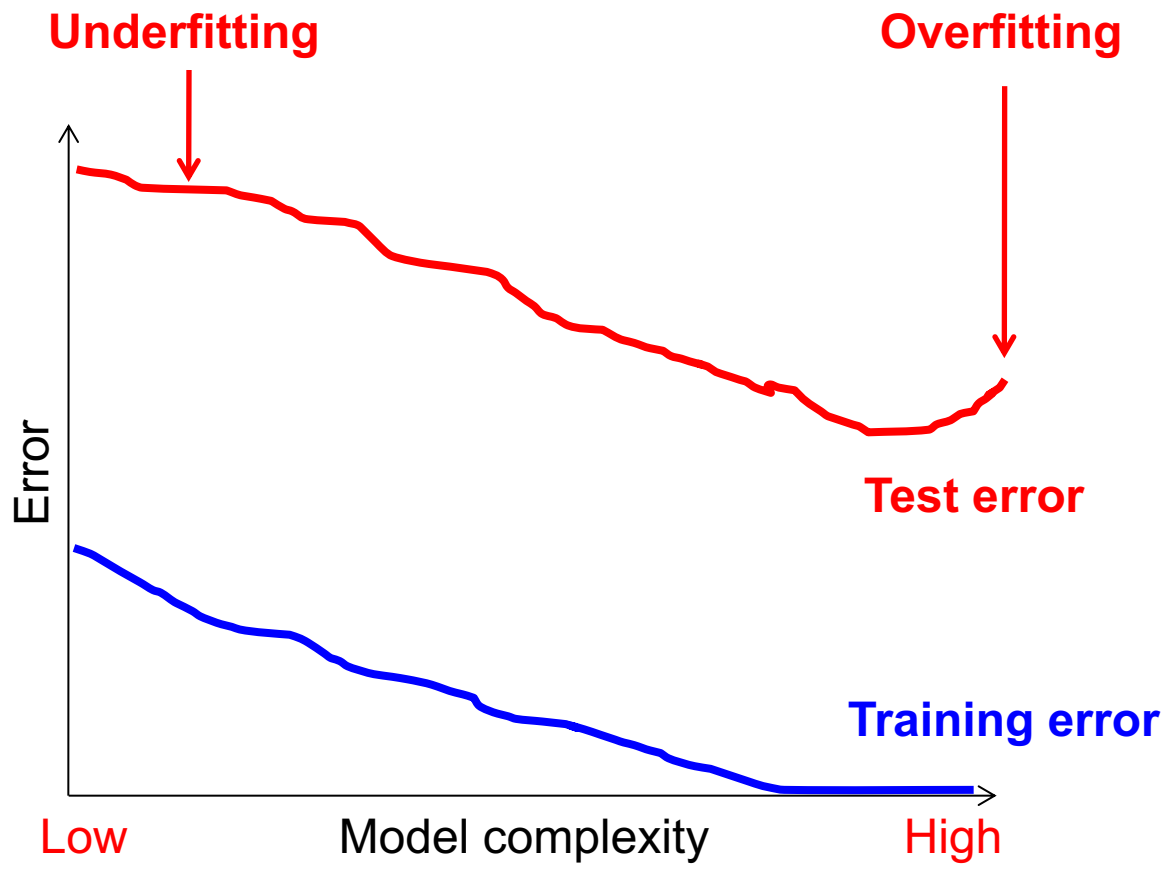
- Generalization refers to the ability to correctly classify never before seen examples
- Can be controlled by turning “knobs” that affect the complexity of the model

Training set (labels known)

Test set (labels unknown)

Diagnosing generalization ability

- **Training error:** how does the model perform on the data on which it was trained?
- **Test error:** how does it perform on never before seen data?



Source: D. Hoiem

Underfitting and overfitting

- **Underfitting:** training and test error are both *high*
  - Model does an equally poor job on the training and the test set
  - Either the training procedure is ineffective or the model is too "simple" to represent the data
- **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

Underfitting | Good generalization | Overfitting

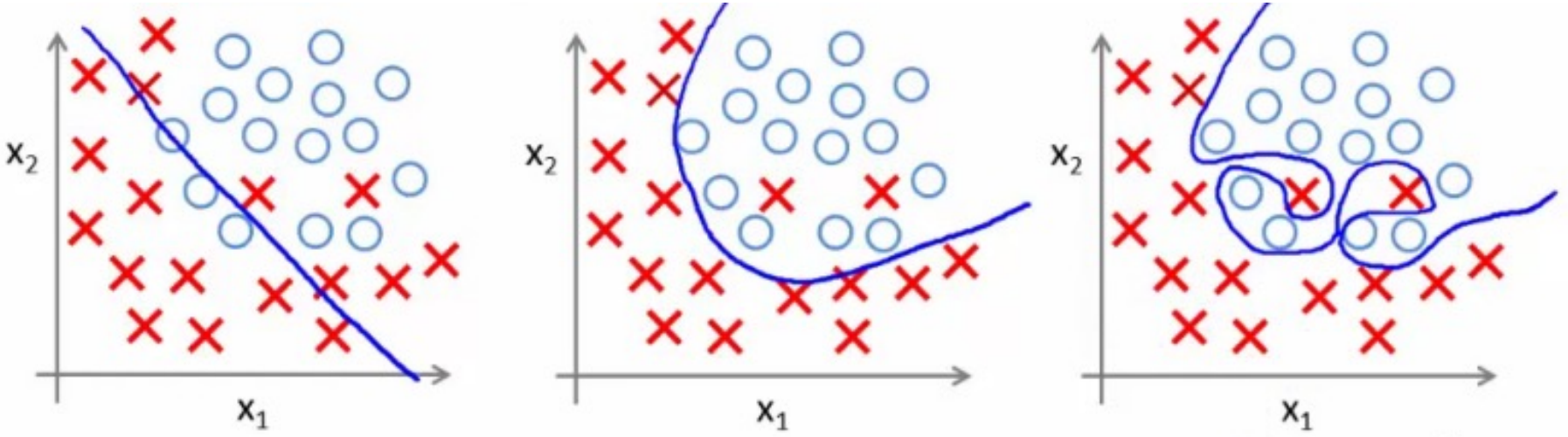
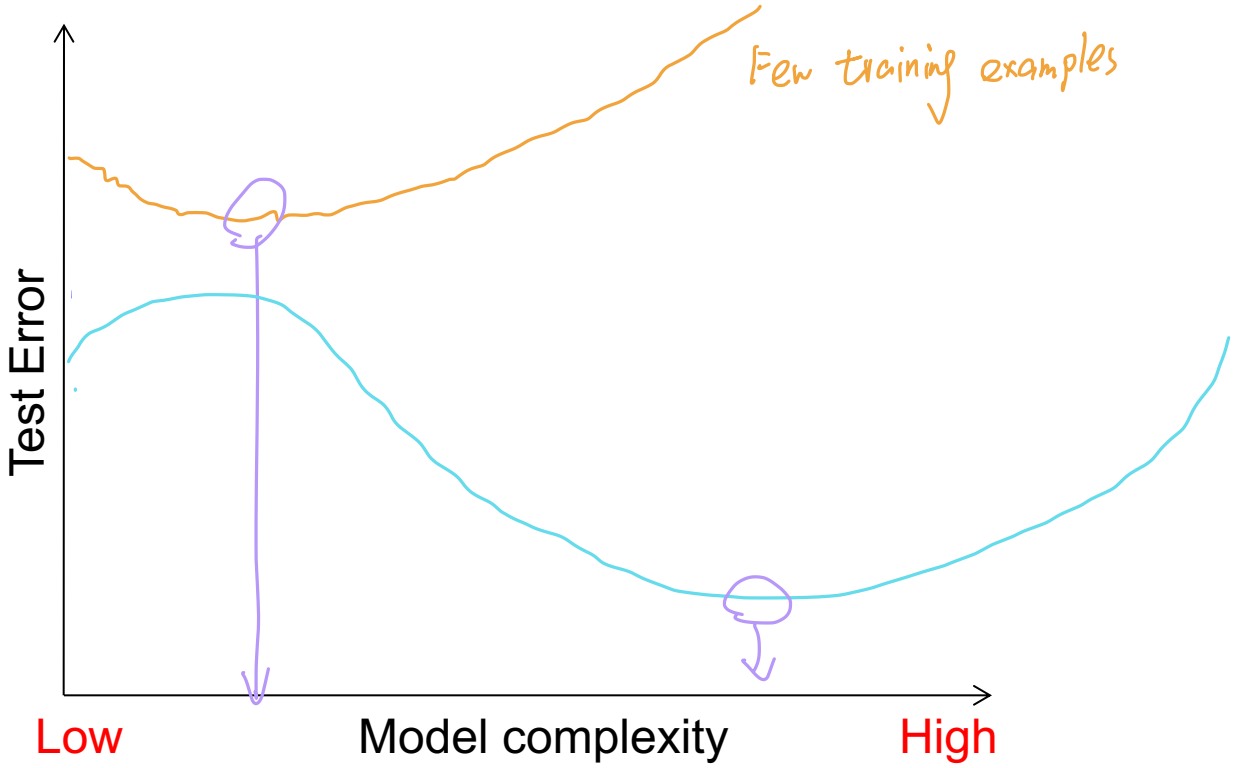


Figure source

Effect of training set size



Source: D. Hoiem

Validation

- Split the data into **training**, **validation**, and **test** subsets
- Use training set to **optimize model parameters**
- Use validation test to **choose the best model**
- Use test set only to **evaluate performance**