Cognitive Services

LM Informatica & Data Science - 2nd semester - 6 CFU

Introduction to visual recognition, image classification

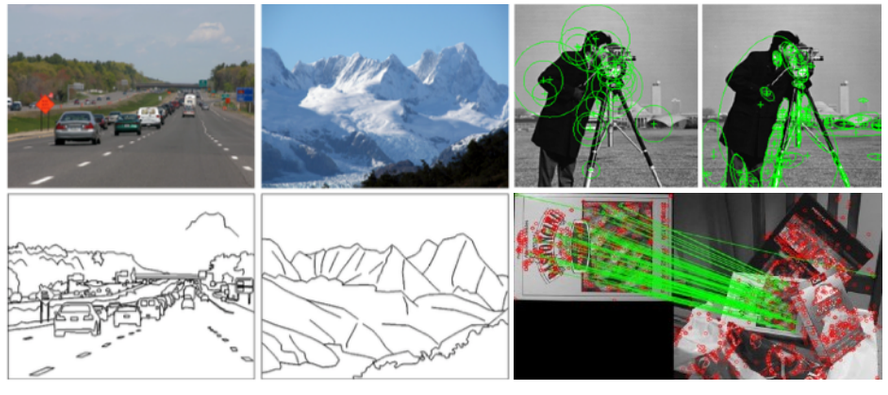
Lamberto Ballan

University of Padova, A.Y. 2018/19 May 7, 2019

What we learned in the last class

(a brief summary)

- Edge detection
- Image gradients
- Local invariant features
- DoG and SIFT descriptors



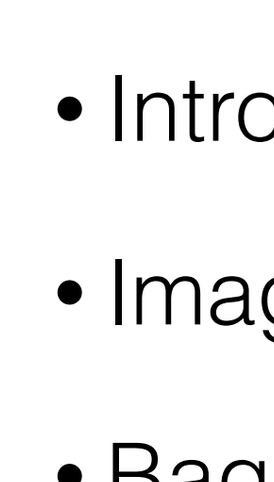
Background reading:

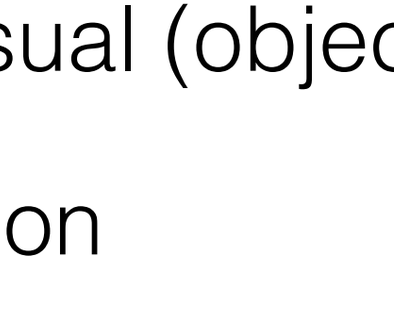
Forsyth and Ponce, Computer Vision, Chapter 8

D.G. Lowe, SIFT paper, IJCV 2014

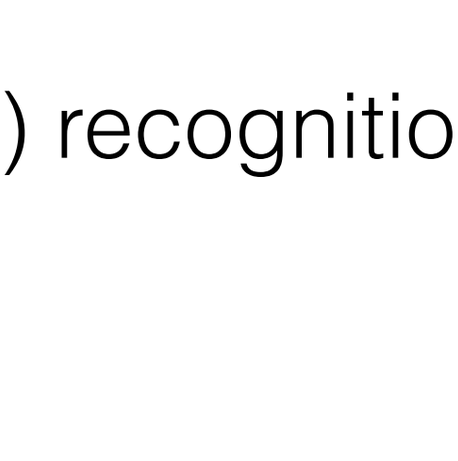
What we will learn today?

- Introduction to visual (object) recognition
- Image classification
- Bag of (visual) words
- Spatial pyramids

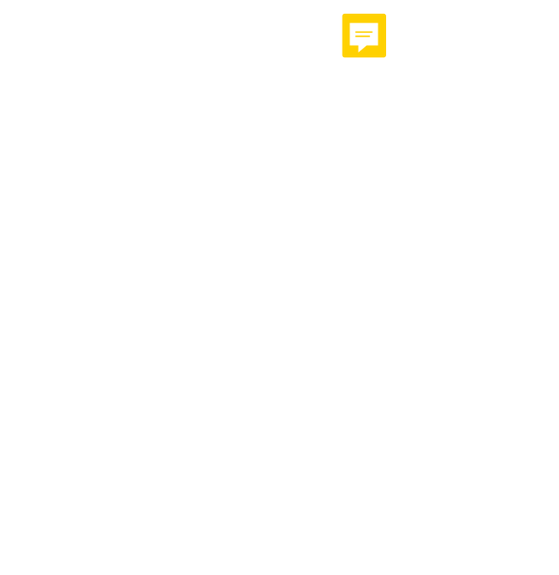




Input image

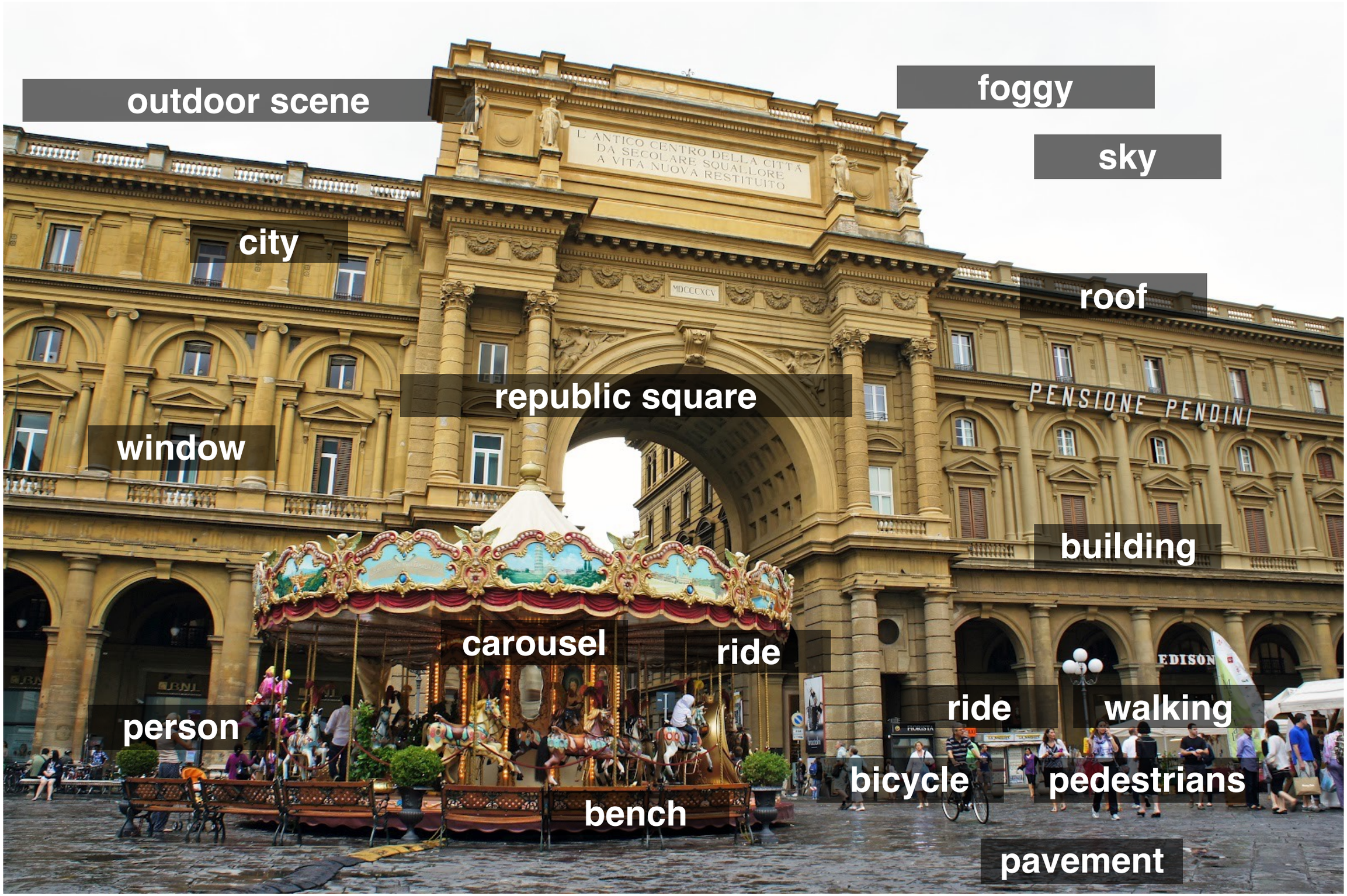


Bag of image patches



Feature space

Vision as a source of semantic information



Visual recognition

- From specific object instances to classes
- **Image classification**: a core task in computer vision

Image

Assume given set of discrete labels (classes)

{cat, dog, plane, car, person, bird, sky, …}

bird

Image classification

- Data-driven approach:
  - Collect a dataset of images and labels
  - Use Machine Learning to train a classifier
  - Evaluate the classifier on new images

Example: Caltech-101 http://www.vision.caltech.edu/Image_Datasets/Caltech101/

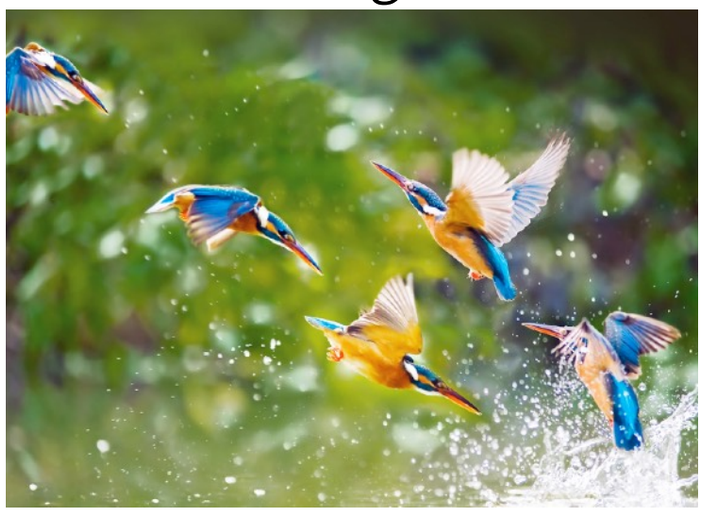





Image classification

- How to implement it?
- We need two major “ingredients”:
  - a way to *describe* images (global image representation)
  - a procedure to *compare* different images and *learn* a statistical model of a specific class

Image

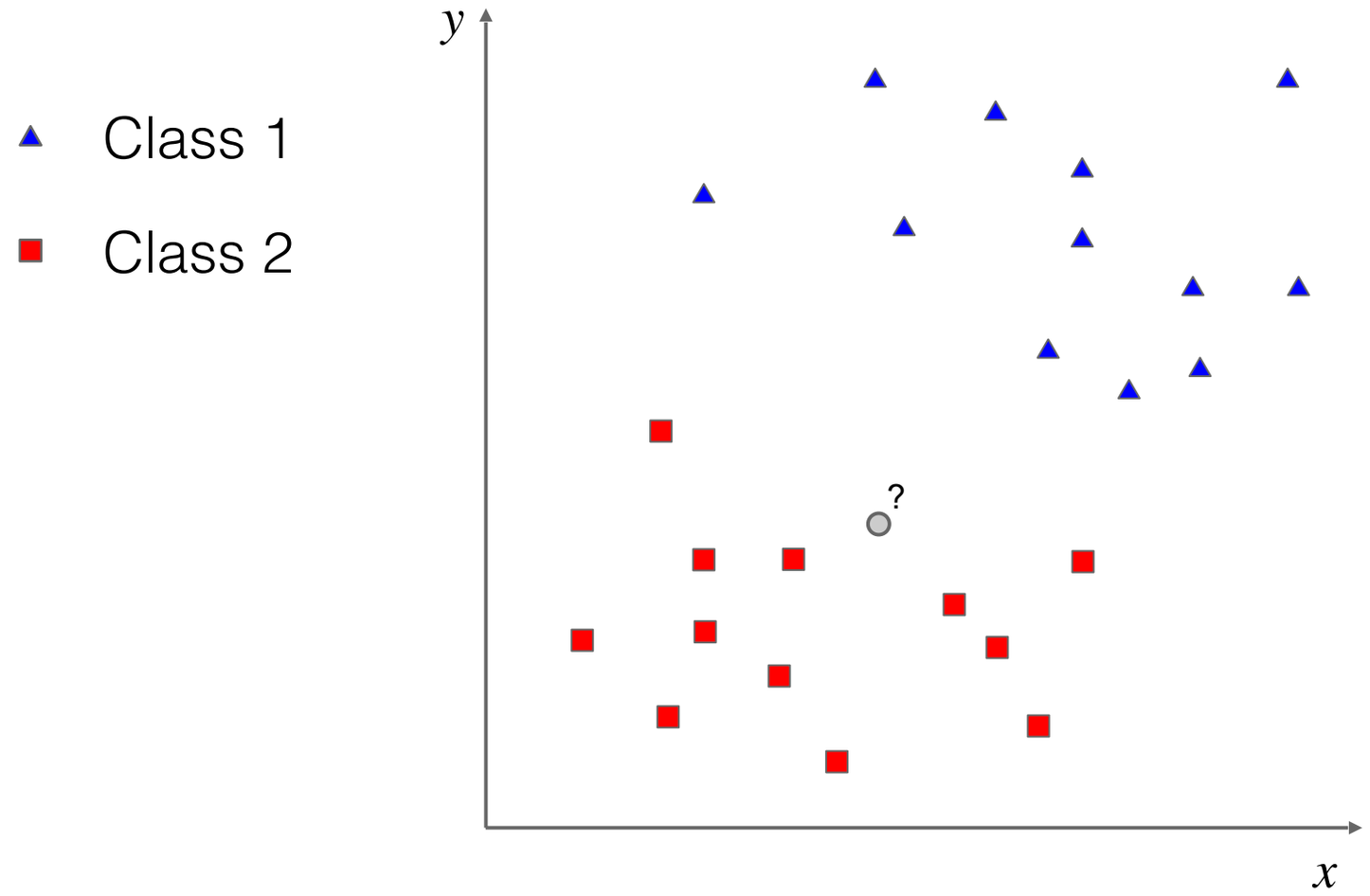


img_feat: $[c_1,\ldots,c_N]$ → 

→ bird

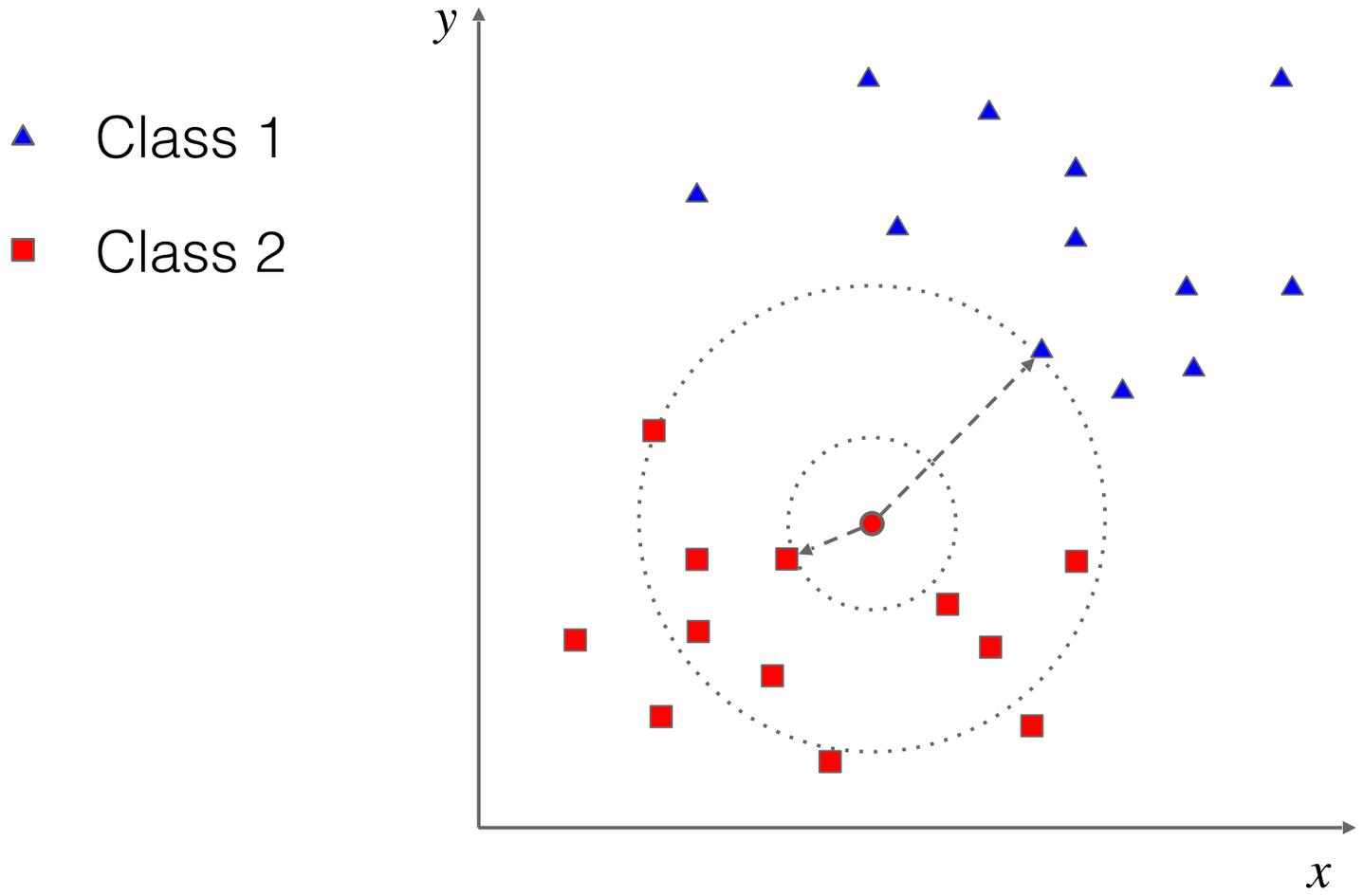
Nearest Neighbor classifier

- Which is the simplest approach?
- Nearest Neighbor! Intuition:



Nearest Neighbor classifier

- Which is the simplest approach?
- Nearest Neighbor! Intuition:

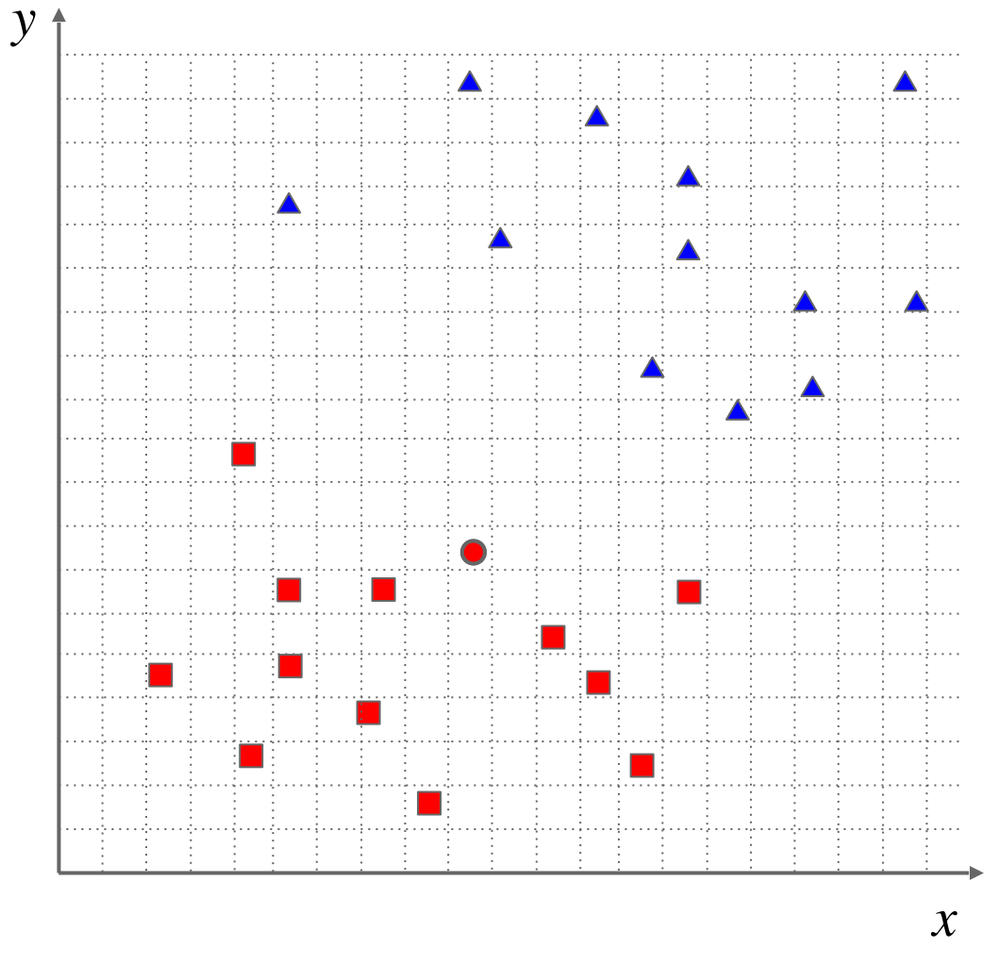


Nearest Neighbor classifier

- Compute distances (Euclidean):

$$d = \mathrm{sqrt}((x_q - x_p)^2 + (y_q - y_p)^2)$$

- Class 1
- Class 2



x_q	y_q
10	8

x_q	y_q	d	x_q	y_q	d
9	2	6.1	16	11	6.7
5	3	7.1	14	12	5.7
14	3	6.4	18	12	8.9
8	4	4.5	18	14	10
3	5	7.6	20	14	11.7
6	5	5.0	11	15	7.1
13	5	4.2	15	15	8.6
12	6	2.8	6	16	8.9
6	7	4.1	15	17	10.3
8	7	2.2	13	18	10.4
15	7	5.1	10	19	11
5	10	5.4	20	19	14.9

Nearest Neighbor classifier

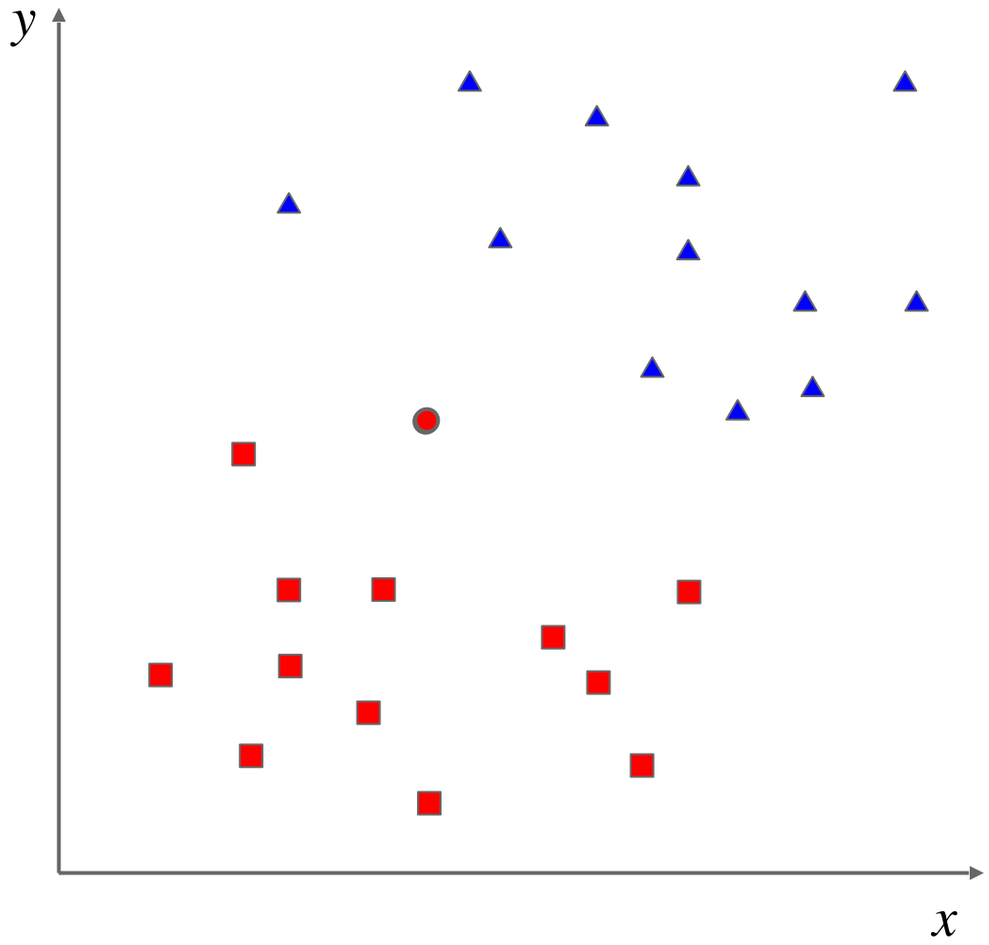
- Compute distances (Euclidean):

$$d = \text{sqrt}((x_q - x_p)^2 + (y_q - y_p)^2)$$

Class 1

Class 2

Is this guy blue or red?

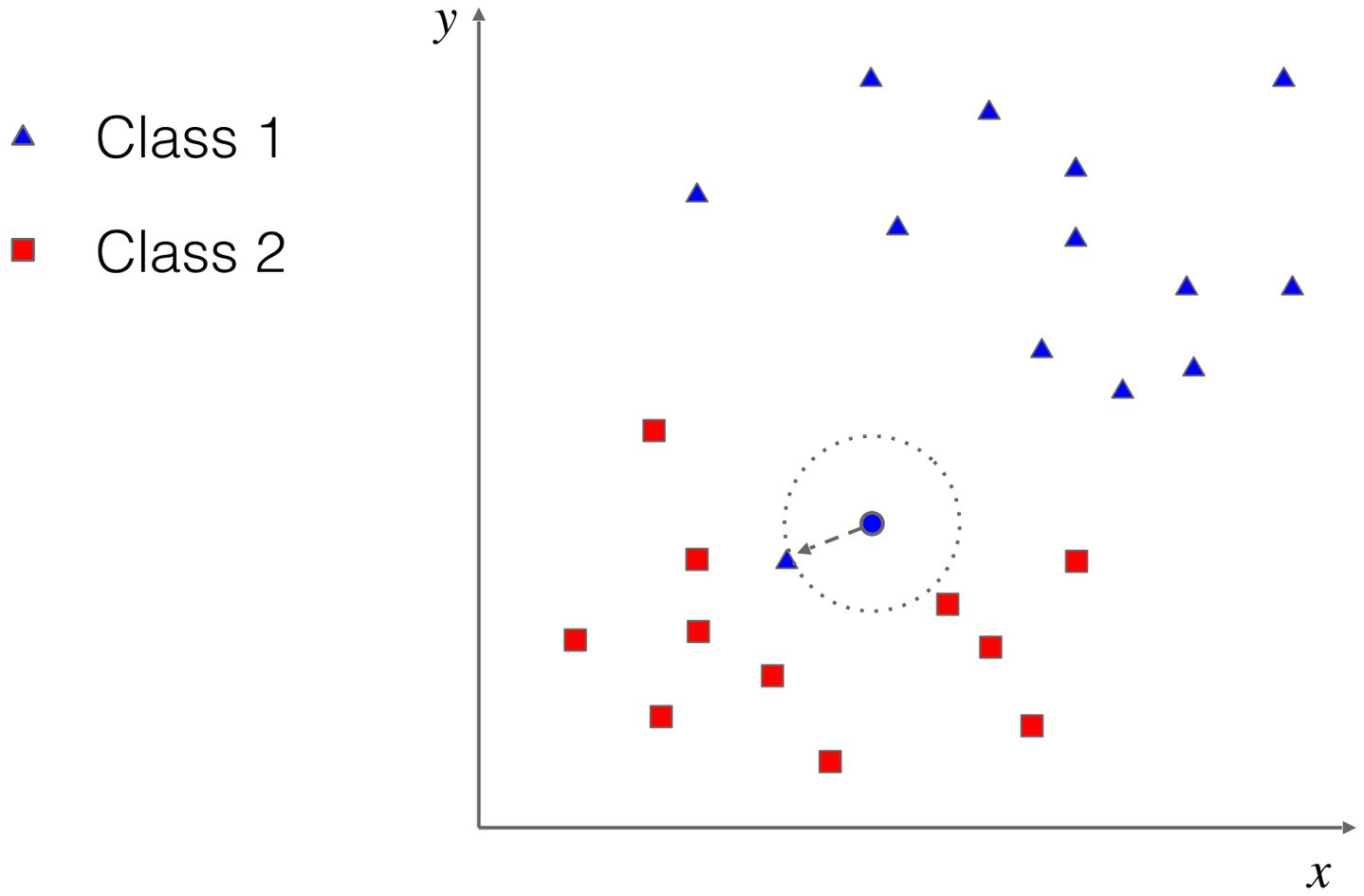


x_q	y_q
9	11

x_q	y_q	d	x_q	y_q	d
9	2	9.0	16	11	7.0
5	3	8.9	14	12	5.1
14	3	9.4	18	12	9.1
8	4	7.1	18	14	9.5
3	5	8.5	20	14	11.4
6	5	6.7	11	15	4.5
13	5	7.2	15	15	7.2
12	6	5.8	6	16	5.8
6	7	5.0	15	17	8.5
8	7	4.1	13	18	8.1
15	7	7.2	10	19	8.1
5	10	4.1	20	19	13.6

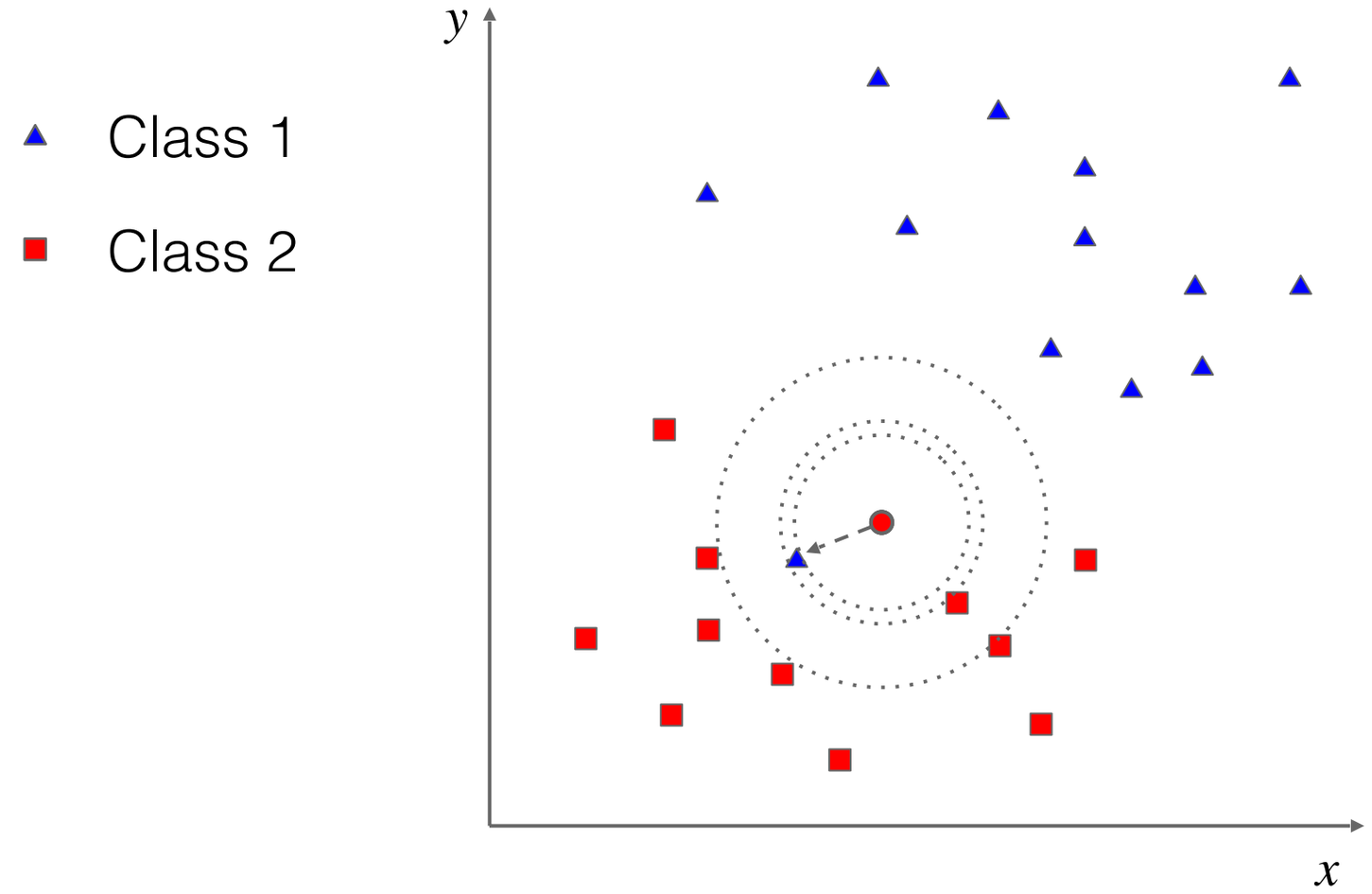
Nearest Neighbor classifier

- Properties: NN is sensitive to outliers



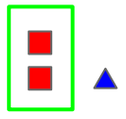
k-Nearest Neighbors classifier

- From NN to k-NN…



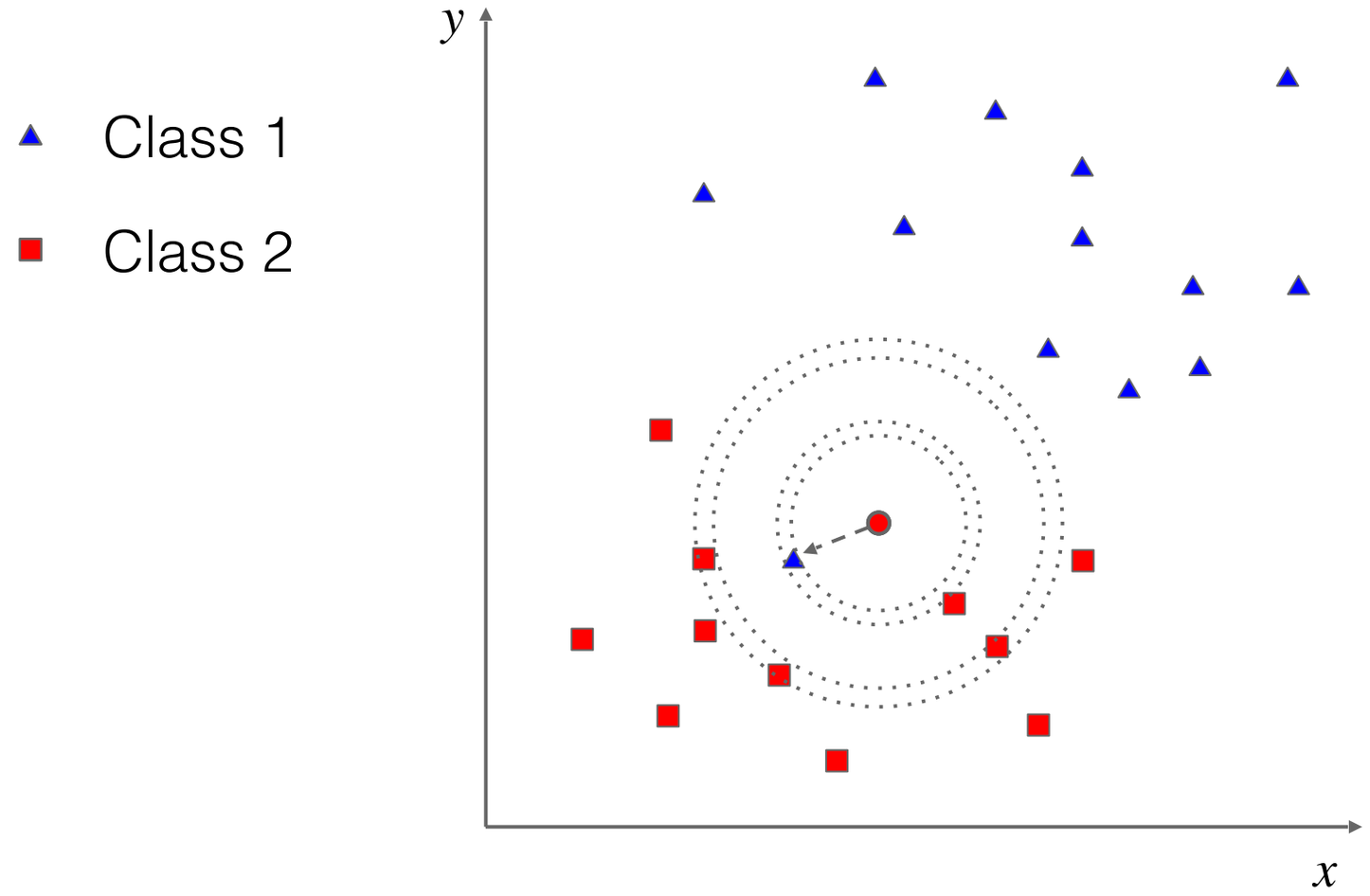
k=3
(a.k.a. 3-NN)

majority voting:



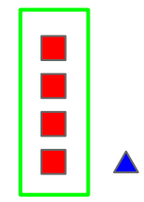
k-Nearest Neighbors classifier

- Parameter k has a very strong effect
  - Small k: sensitive to outliers



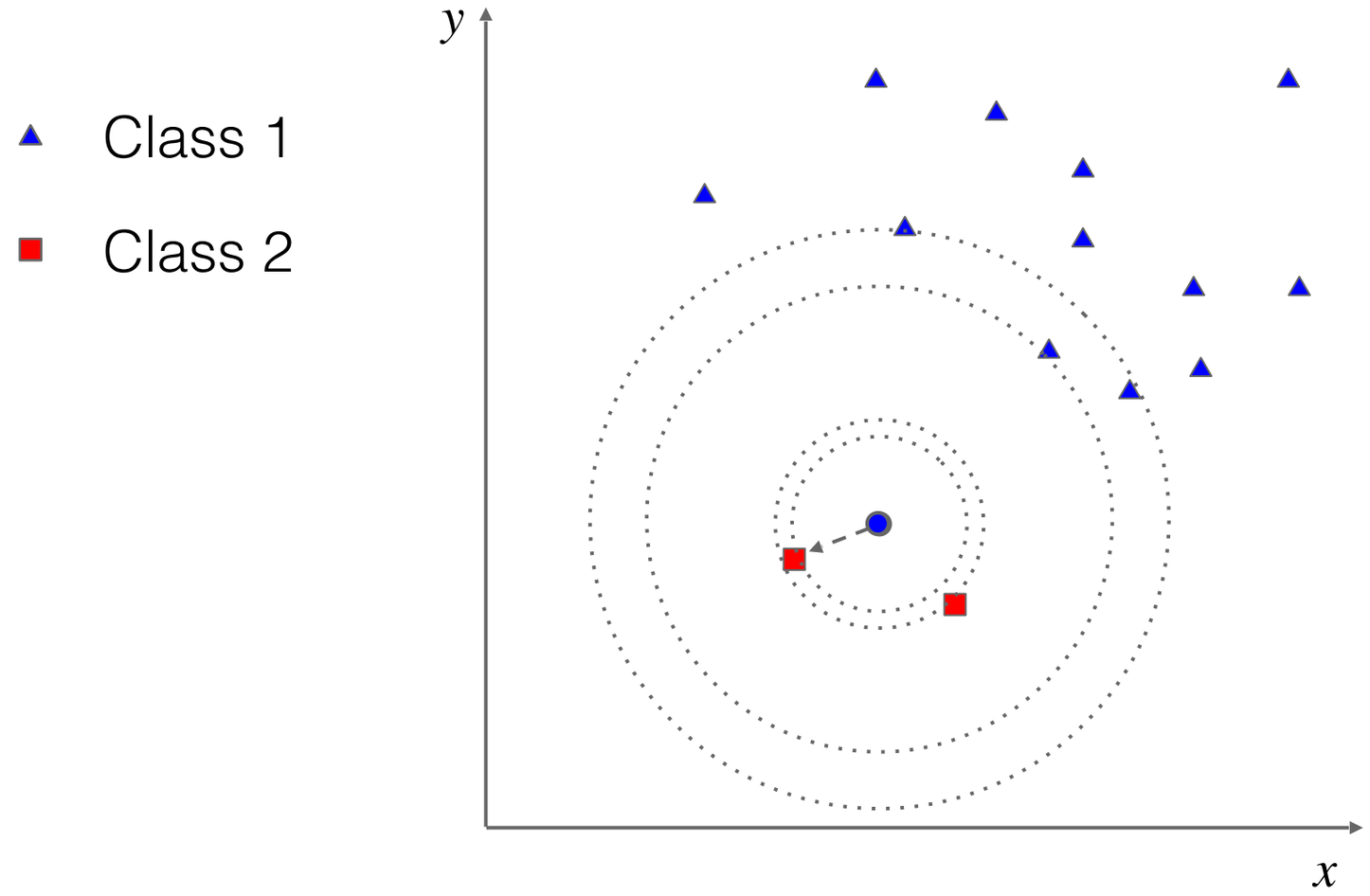
k=5
(a.k.a. 5-NN)

majority voting:



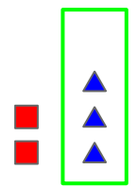
k-Nearest Neighbors classifier

- Parameter k has a very strong effect
  - Large k: everything is classified as the most frequent class



k=5
(a.k.a. 5-NN)

majority voting:



k-Nearest Neighbors classifier

- k-NN: large data is good!

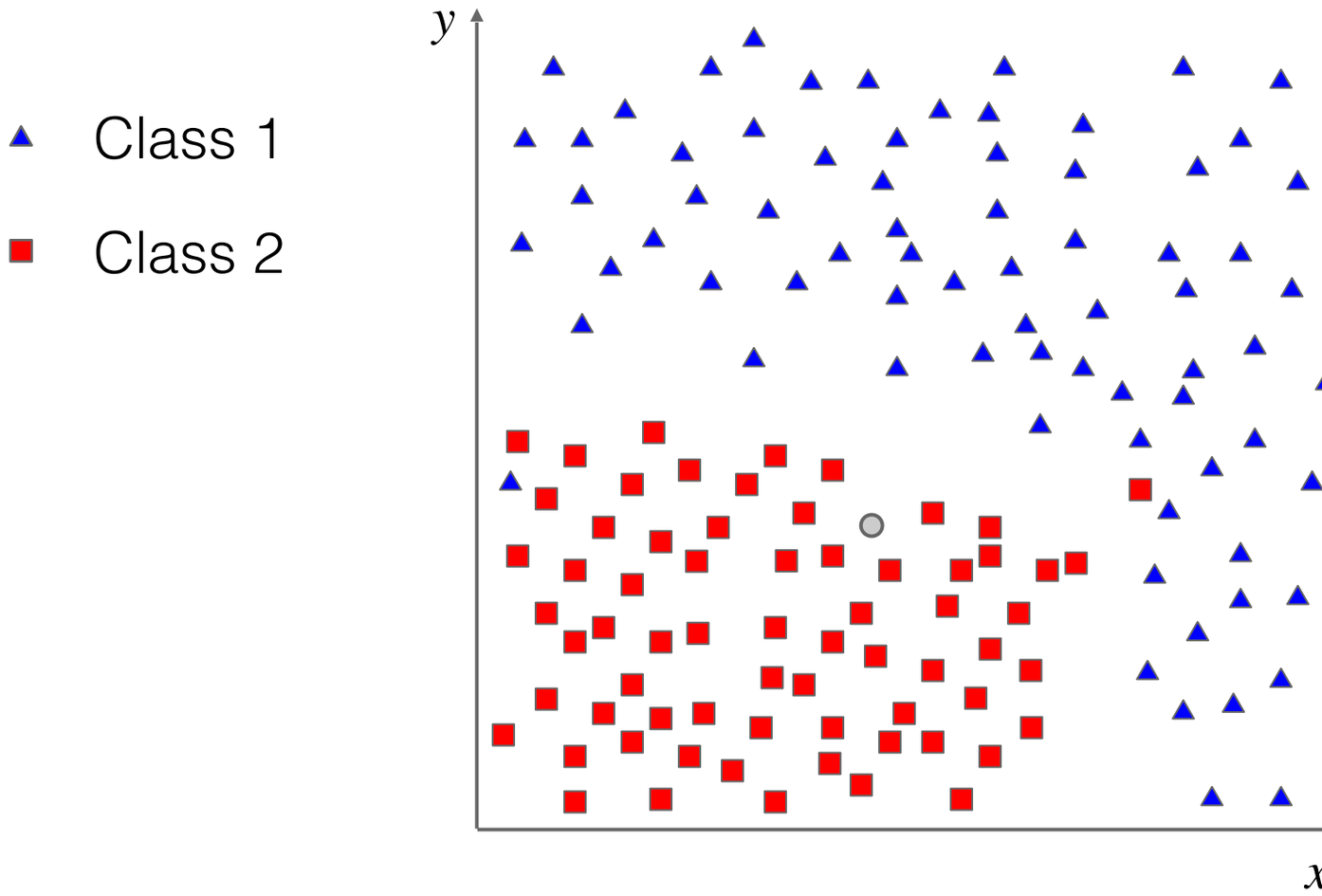


Image classification

- What about images? How to represent them?
  - The simplest representation is provided by the raw pixels

- A simple image classification pipeline:
  - Resize images to the same scale/resolution
  - Represent images as a 2D or 1D array of raw pixels
  - Define a distance metric (e.g. L1 or L2) to compare images
  - Use k-NN for classification

Image classification

- Example dataset: CIFAR10

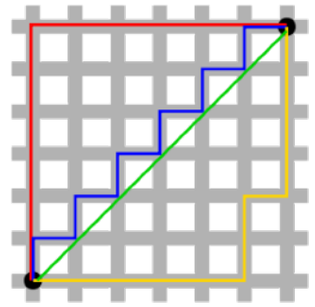
10 classes
50,000 training images
10,000 testing images

test images and nearest neighbors

Slide credit: Fei-Fei, Johnson, Yeung

Image classification

- Distance metric to compare images:
  - L1 distance *(Manhattan)*: $d_{L1}(I_1, I_2) = \sum_p |I_1^p - I_2^p|$



  - Example:

test image					training image				pixel-wise absolute value differences				
56	32	10	18	-	10	20	24	17	46	12	14	1	add → 456
90	23	128	133		8	10	89	100	82	13	39	33	
24	26	178	200		12	16	178	170	12	10	0	30	
2	0	255	220		4	32	233	112	2	32	22	108	

Slide credit: Fei-Fei, Johnson, Yeung

Image classification

- Example (Python implementation):

```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

For each test image:

- Find closest train image
- Predict label of nearest train image

Slide credit: Fei-Fei, Johnson, Yeung

Image classification

- Example (Python implementation):

```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
```

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

A: Train O(1), Test O(N)

Q: Is this good or bad?

A: This is bad! We want classifiers that are fast at prediction (slow for training is ~ok)

Slide credit: Fei-Fei, Johnson, Yeung

Image classification

- What can we do better?
  - We can design/use a better image representation
  - We can rely on better functions to compare images
  - We can design/use better classifiers

Bag of (visual) words model

- Intuition: build an intermediate representation based on the number of occurrences of *"visual primitives"*

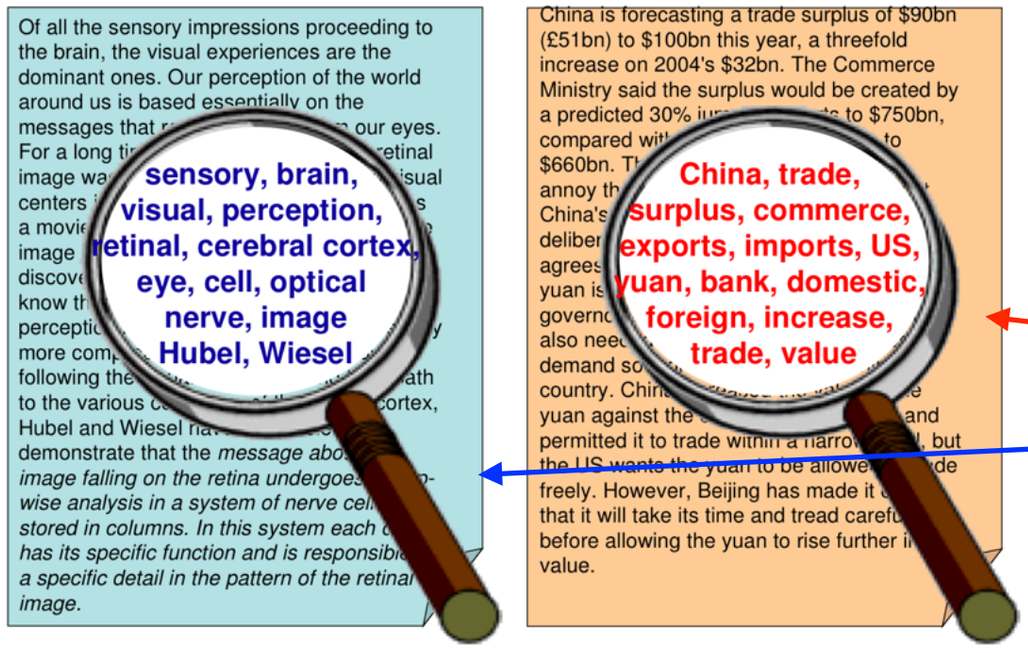
Object

Bag of 'words'

Image credit: Fei-Fei, Fergus, Torralba

Bag of words model

- Origin: bag of words model for text categorization
  - *the task is to assign a textual document to one or more categories based on its content*



Q: What's the main topic?

- Approach: a document is represented as an *unordered* collection of words disregarding grammar & word order

Image credit: Fei-Fei, Fergus, Torralba

Bag of (visual) words / Bag of features

- The very same idea can be applied for image classification (object recognition)

Face
Flower
Building

- Works pretty well for image-level classification and also for recognising object instances

Sivic & Zisserman (2003, 2005), Csurka et al. (2004), Grauman & Darrell (2005), Niebles et al. (2006, 2008), Ballan et al. (2009, 2012)

Bag of features: pipeline

- Three main steps:
  - Feature detection (sampling) and description
    - *take a bunch of images and extract local visual features*
  - Codebook (visual vocabulary) formation and image representation
    - *Build up a dictionary, i.e. our intermediate representation*
    - *For each feature find the closest visual word in the visual dictionary and build a frequency histogram*
  - Learning and recognition

Tutorials / short courses:

“Recognizing and Learning Object Categories” - Fei-Fei, Fergus, Torralba [web]

“Hands on Advanced BoW Models for Visual Recognition” - Ballan, Seidenari [web]

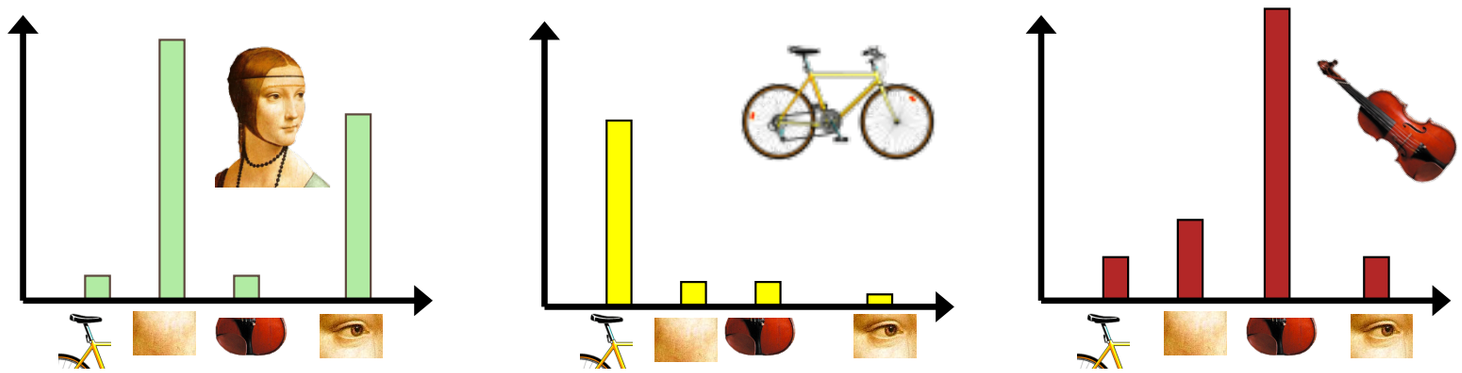
Bag of features: pipeline

- Feature extraction:

- Codebook formation:

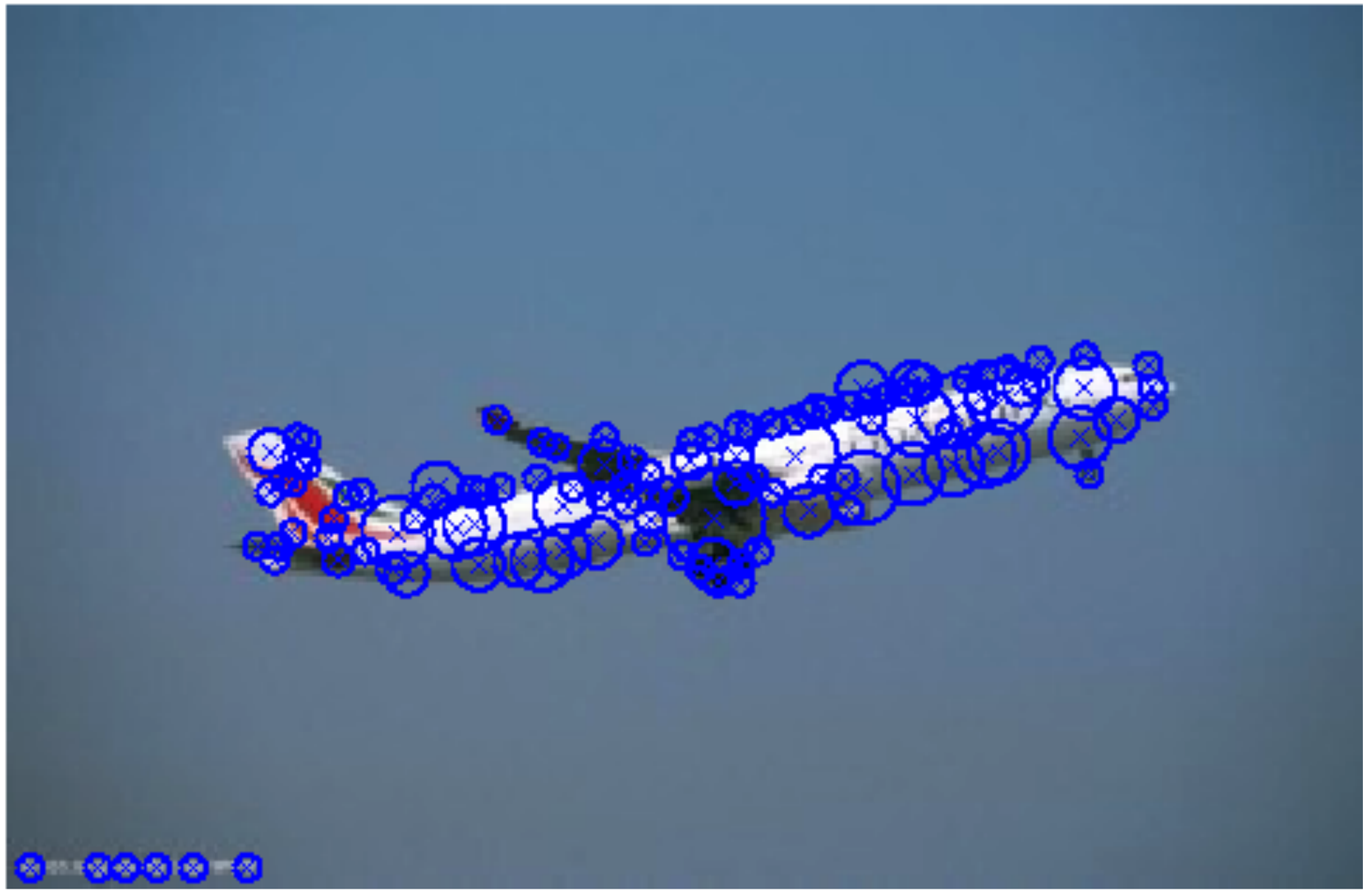
- Quantize features using visual vocabulary

- Represent images by frequencies of “visual words”:



Feature extraction (sampling)

- Keypoint detectors (e.g. DoG, Harris, MSER, …)



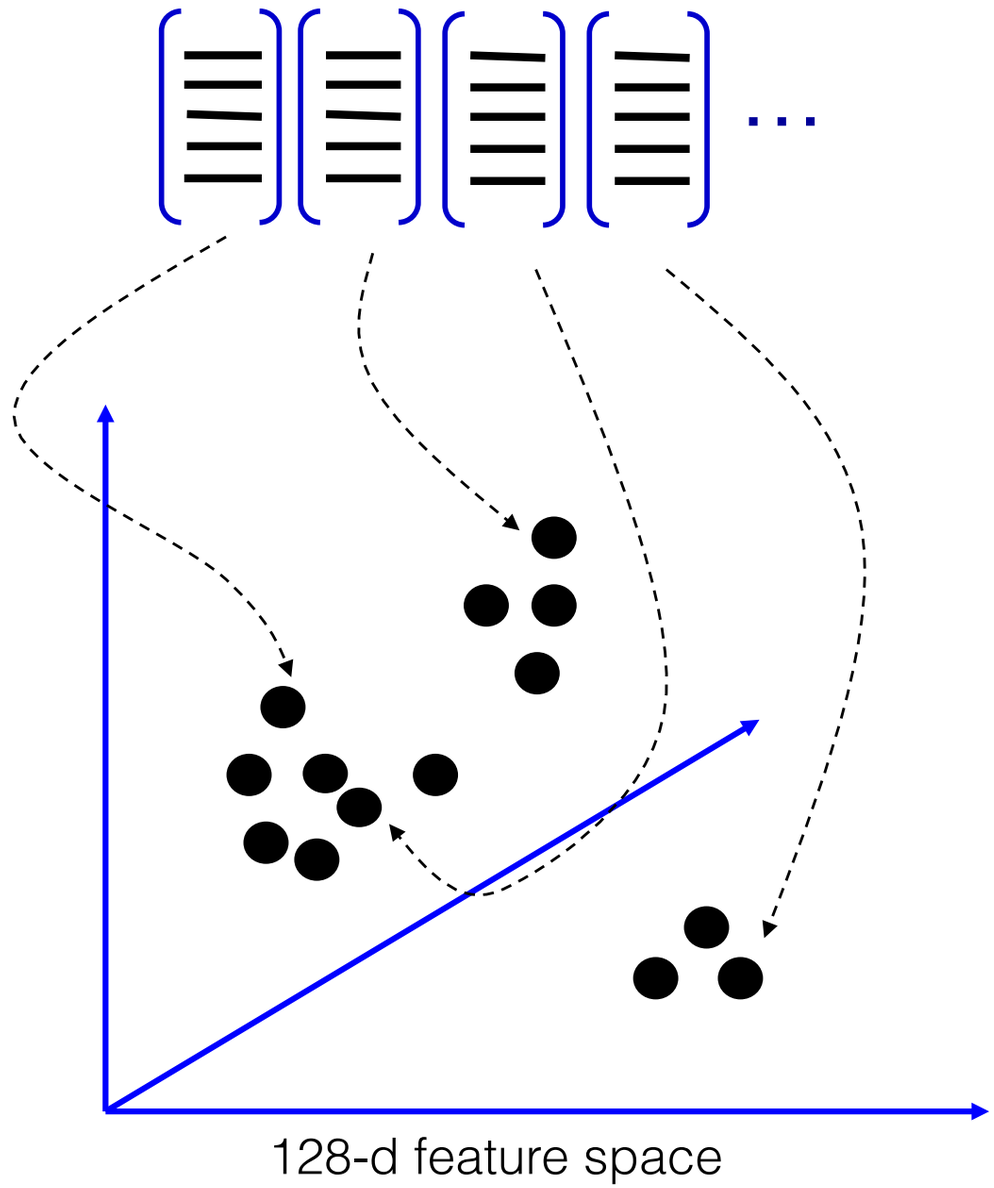
Feature extraction (sampling)

- Dense sampling using a regular grid

  (or other simple methods, e.g. random sampling)

Codebook formation

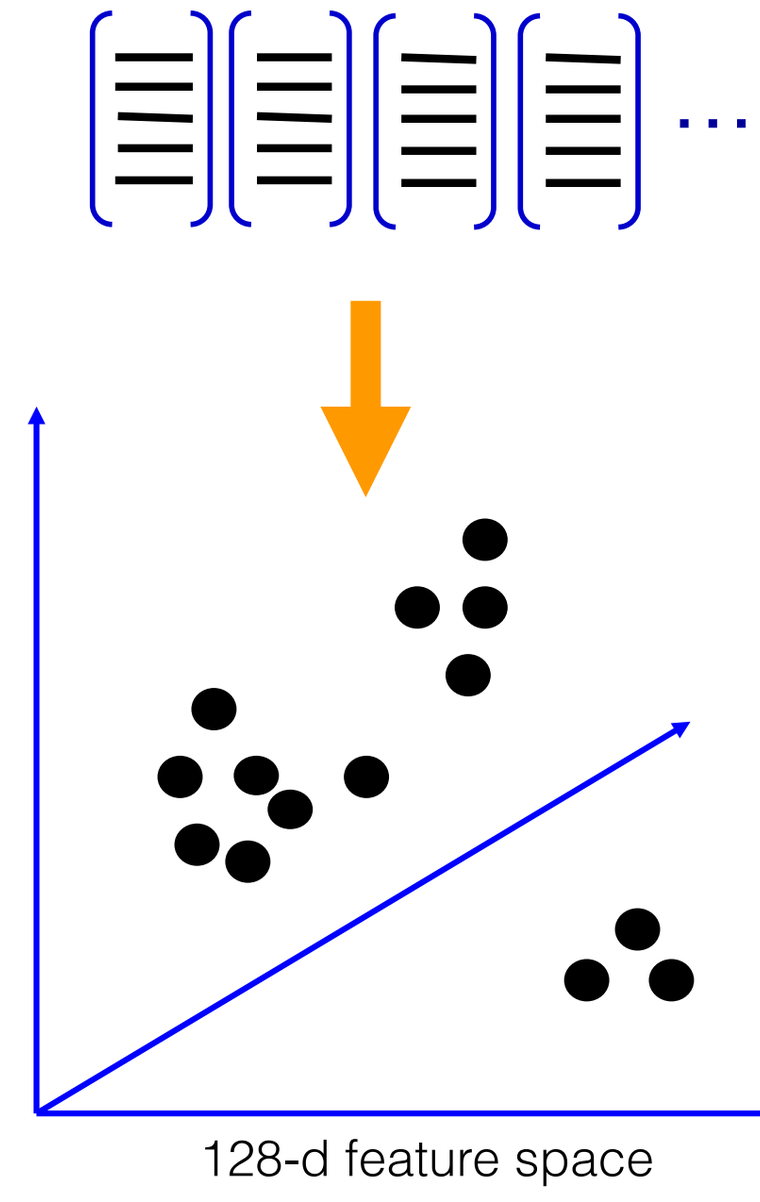
- Learning the visual vocabulary:

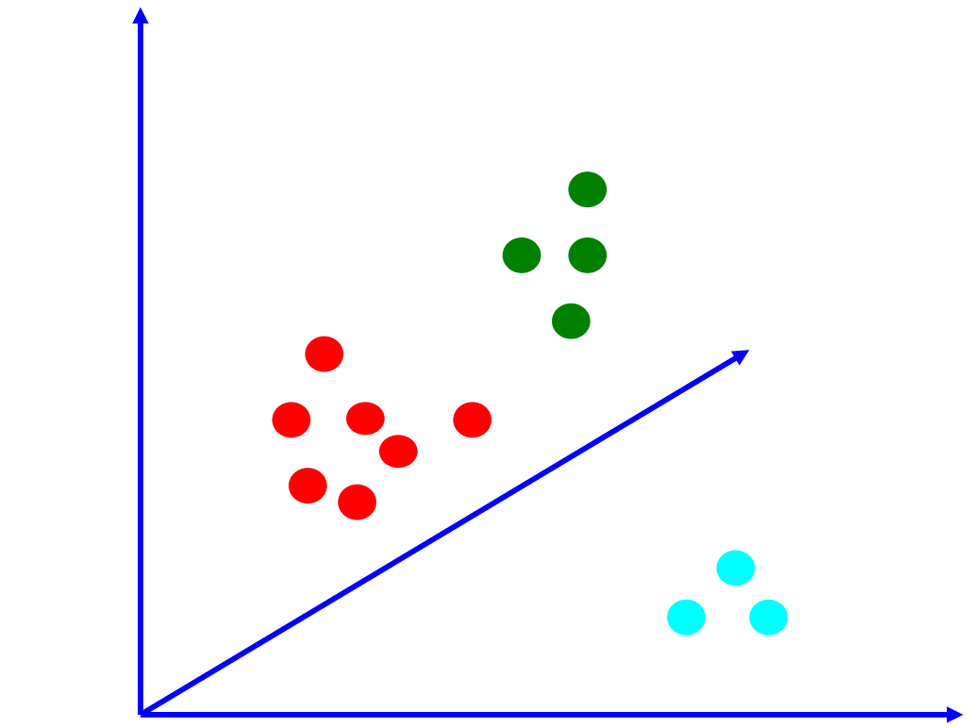


Slide credit: J. Sivic

Codebook formation

- Learning the visual vocabulary:

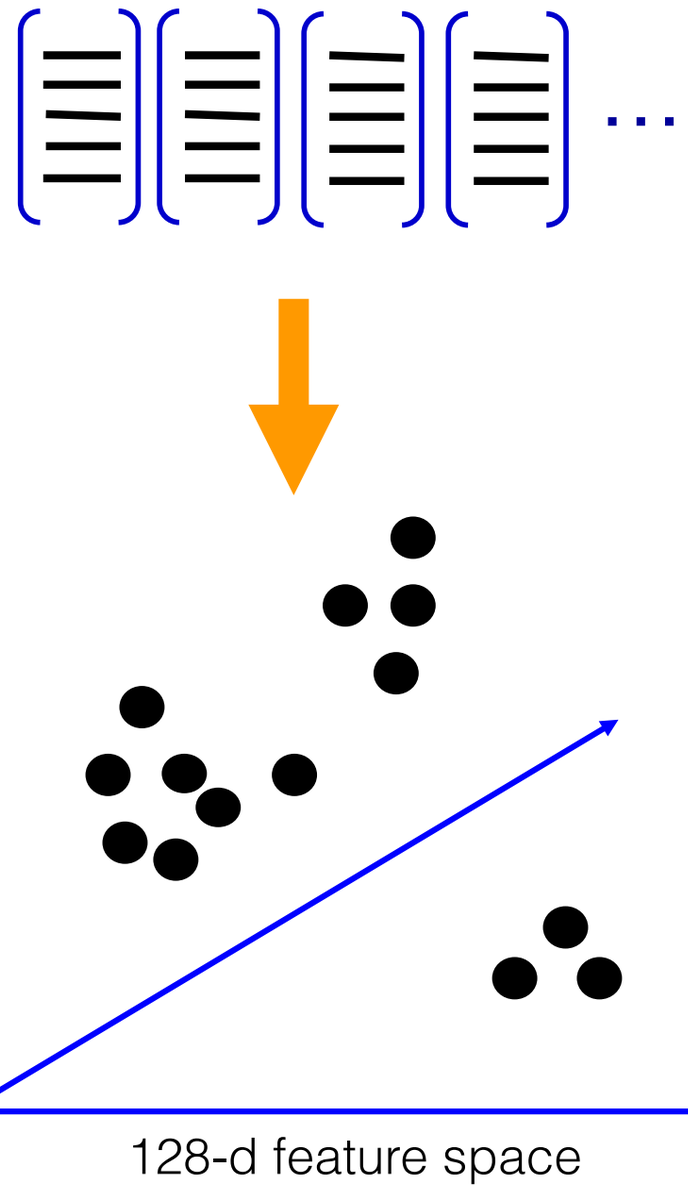




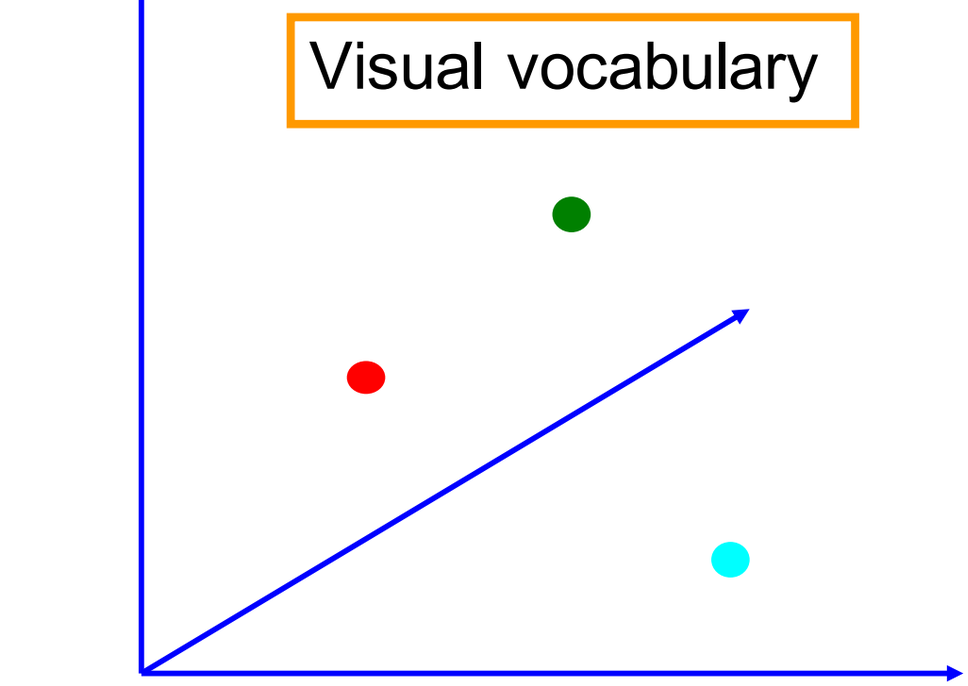
Slide credit: J. Sivic

Codebook formation

- Learning the visual vocabulary:



128-d feature space



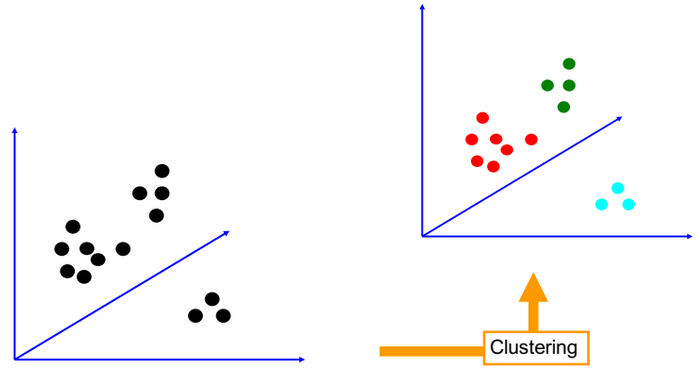
Slide credit: J. Sivic

Clustering recap: k-means

- **Goal:** minimize sum of squared Euclidean distances between points x_i and their nearest cluster centers m_k

$$D(X, M) = \sum_{k} \sum_{i \in k} (x_i - m_k)^2$$

- (Lloyd's) 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



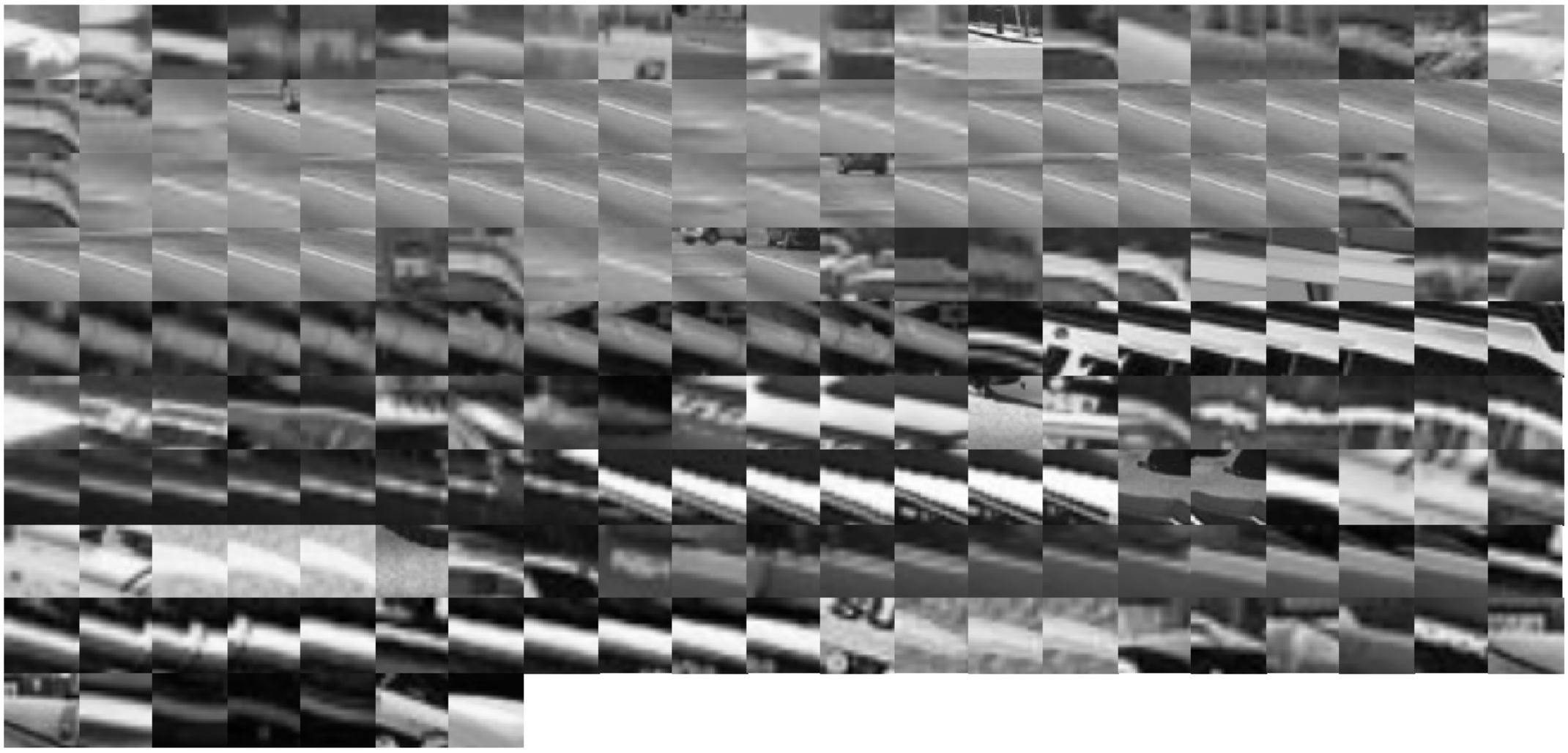
From clustering to vector quantization

- Clustering is a common method for learning a visual vocabulary or codebook
  - This is an unsupervised learning process
  - Each cluster center becomes a *codeword* (or *codevector*)
  - Codebook can be learned on separate training set
    - Provided the training set is sufficiently representative, the codebook will be “universal”
- The codebook is used for quantising visual 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

Slide credit: J.C. Niebles

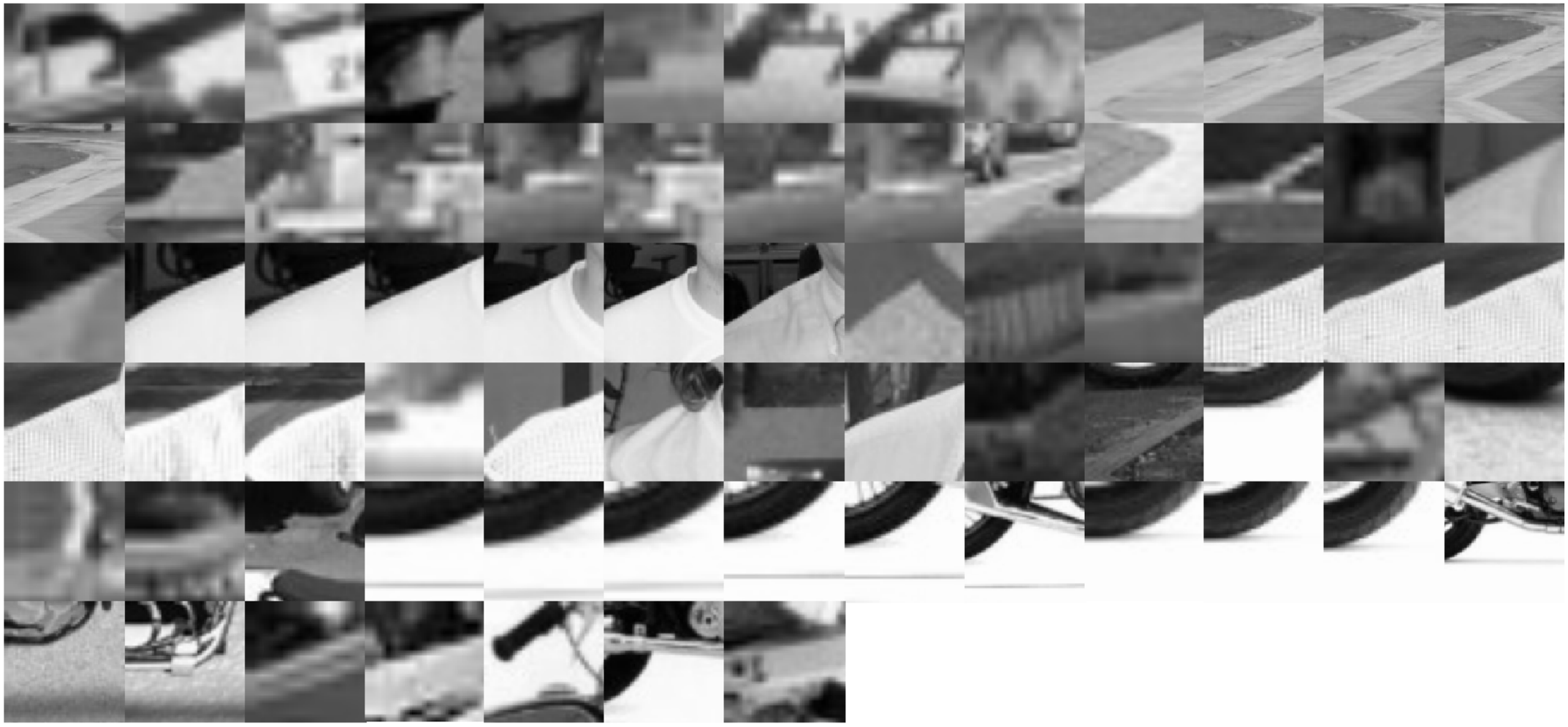
Codebook formation: visual words

- Visual word example 1: what's inside a cluster



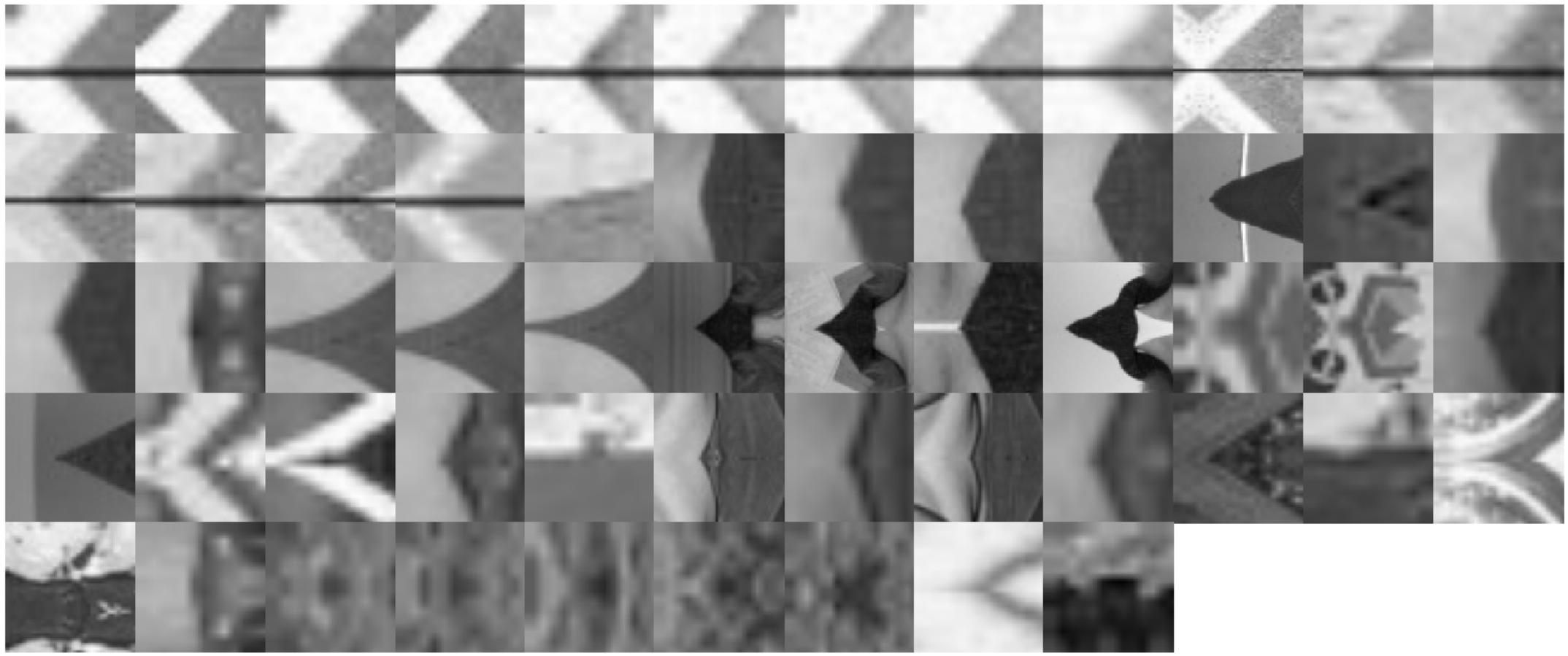
Codebook formation: visual words

- Visual word example 2: what's inside a cluster



Codebook formation: visual words

- Visual word example 3: what's inside a cluster



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

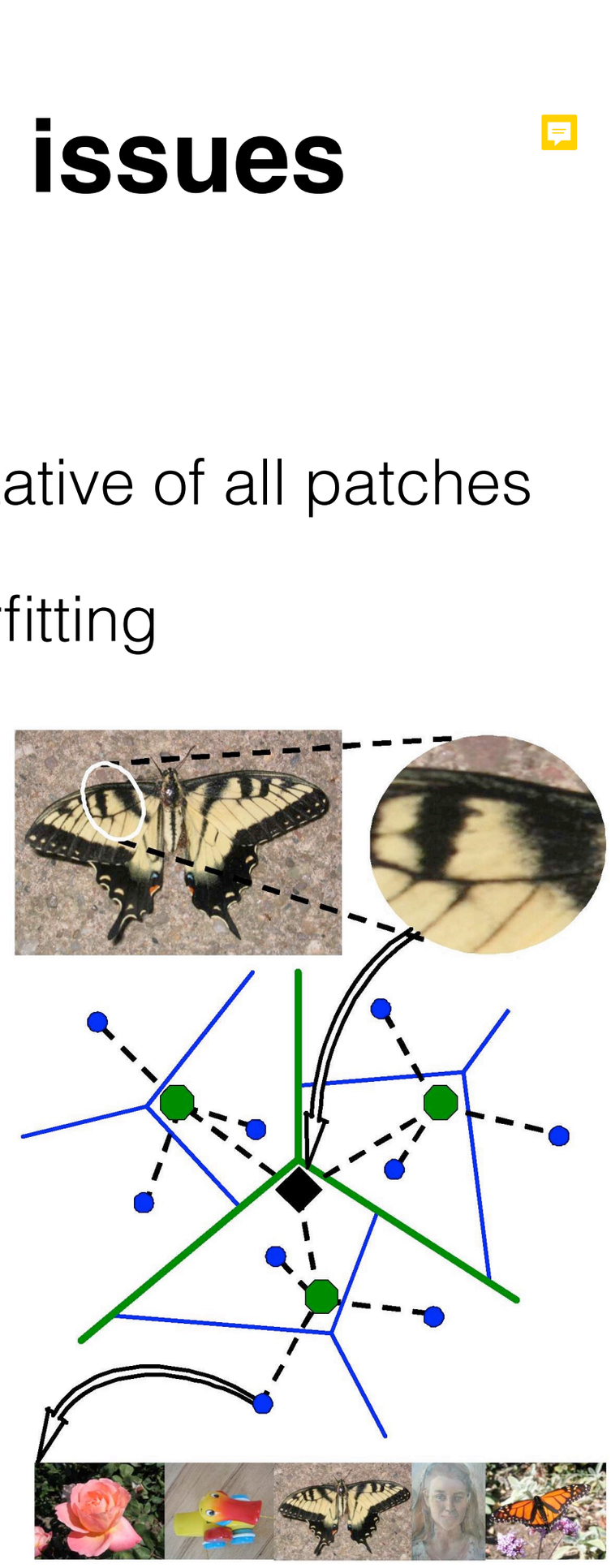
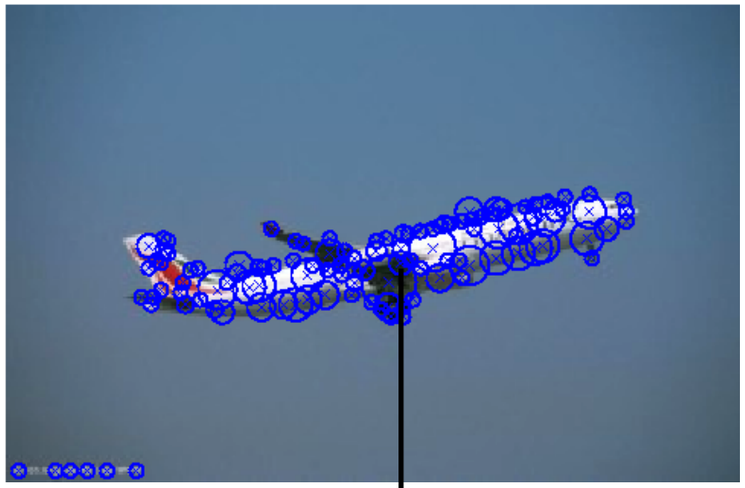


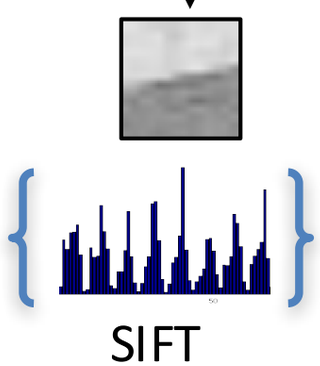
Image credit: D. Nister

Nister and Stewenius, “Scalable Recognition with a Vocabulary Tree”, CVPR 2006

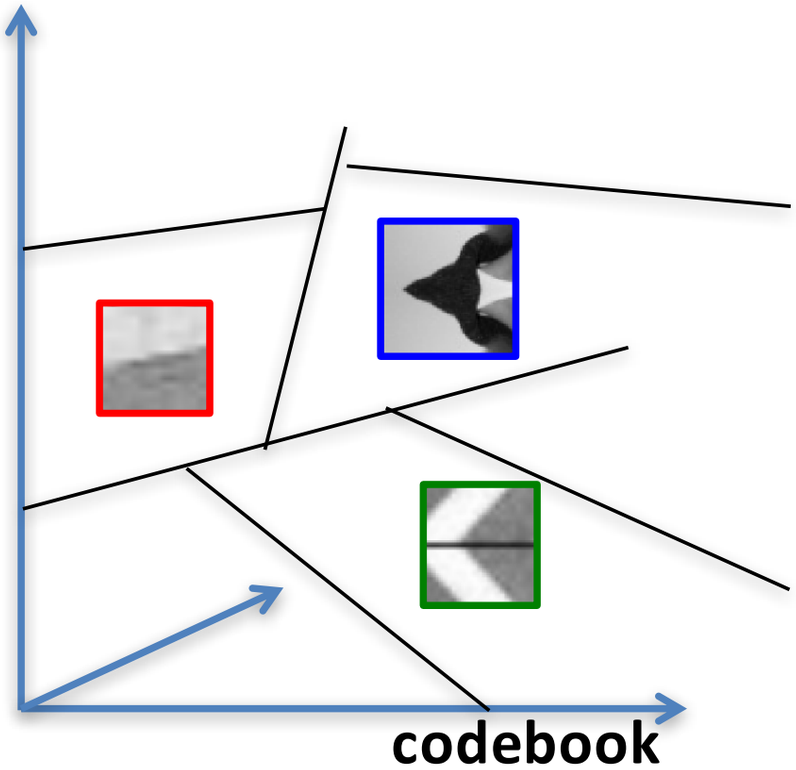
Features quantization

- Assign each feature to the most representative visual word





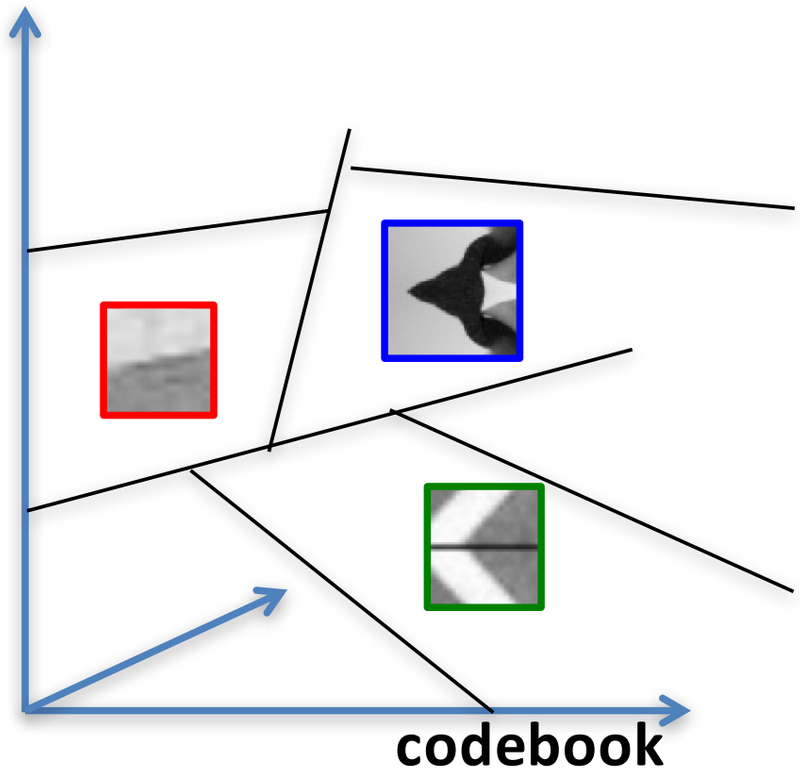
assignment

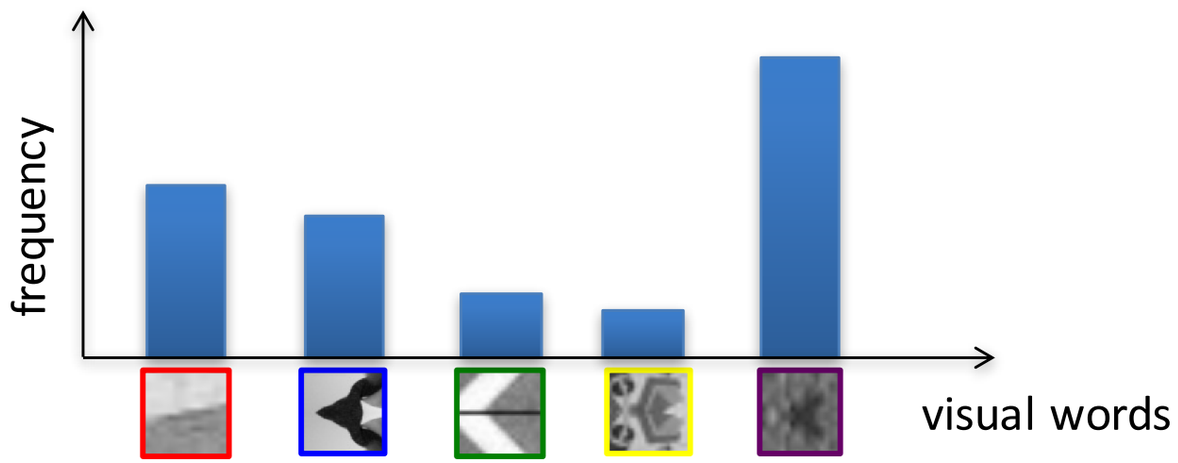


- Hard assignment
- Soft assignment

Image representation

- Compute histograms of visual word frequencies





Uses of BoW representation

- Image classification: treat as feature vector for standard classifiers
  - e.g. k-Nearest Neighbors, Support Vector Machine
- Large-scale image search / retrieval
  - BoW vectors have been useful in matching an image to a large database of object instances
- Cluster BoW vectors over image collection
  - i.e. discover visual themes

Weighting the (visual) words

- Just as with text, some visual words are more discriminative than others
  - the bigger fraction of the documents a word appears in, the less useful it is for matching

the, and, or *vs* ***bank, brain, perception***

- TF-IDF: instead of computing a regular histogram distance, we weight each word by it's Inverse Document Frequency

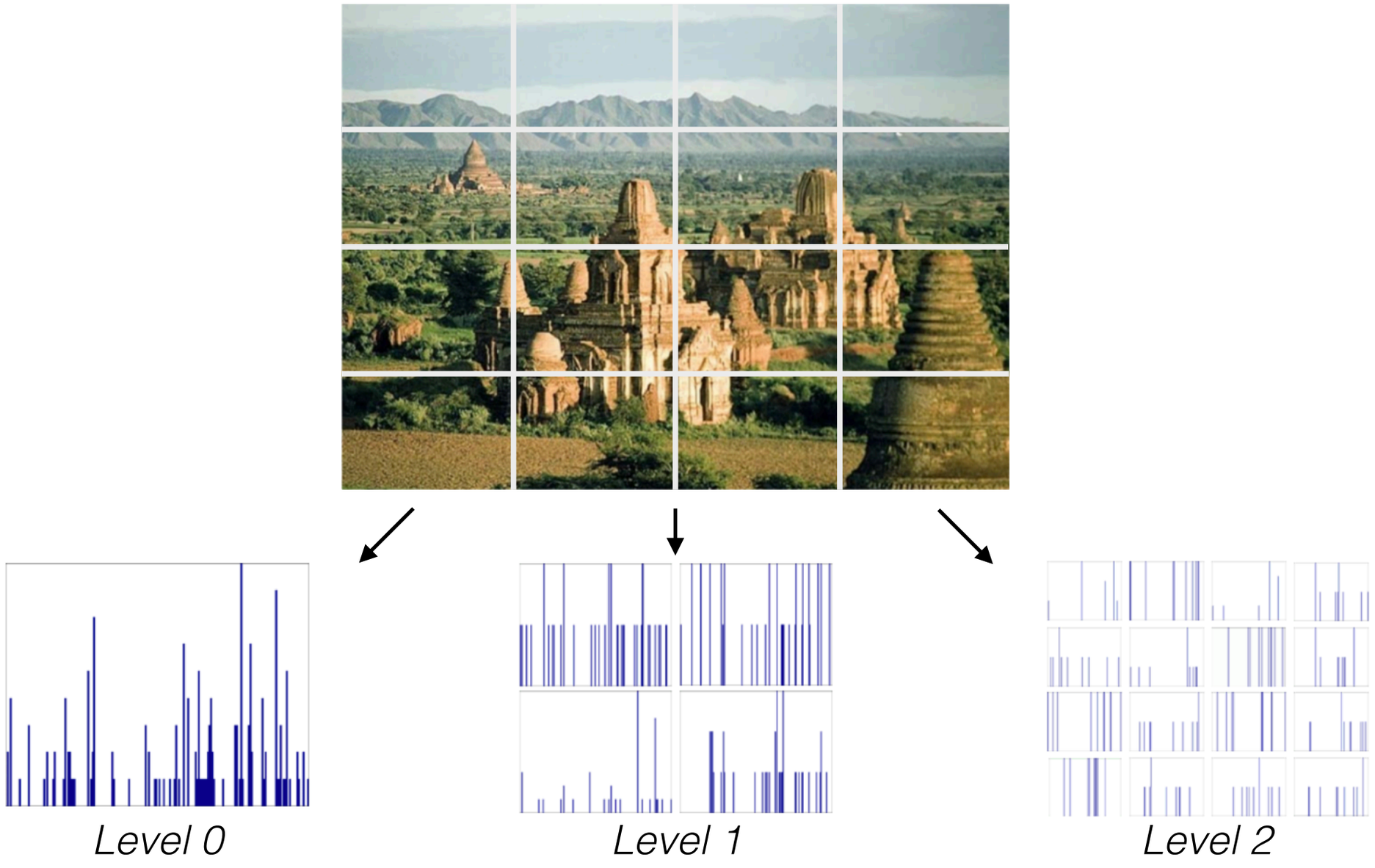
- IDF of word *j*: $\log \frac{\text{number of documents}}{\text{number of documents in which } j \text{ appears}}$

What about spatial information?

?

Spatial pyramids

- Intuition: locally orderless representation at several levels of spatial resolution



Slide credit: S. Lazebnik

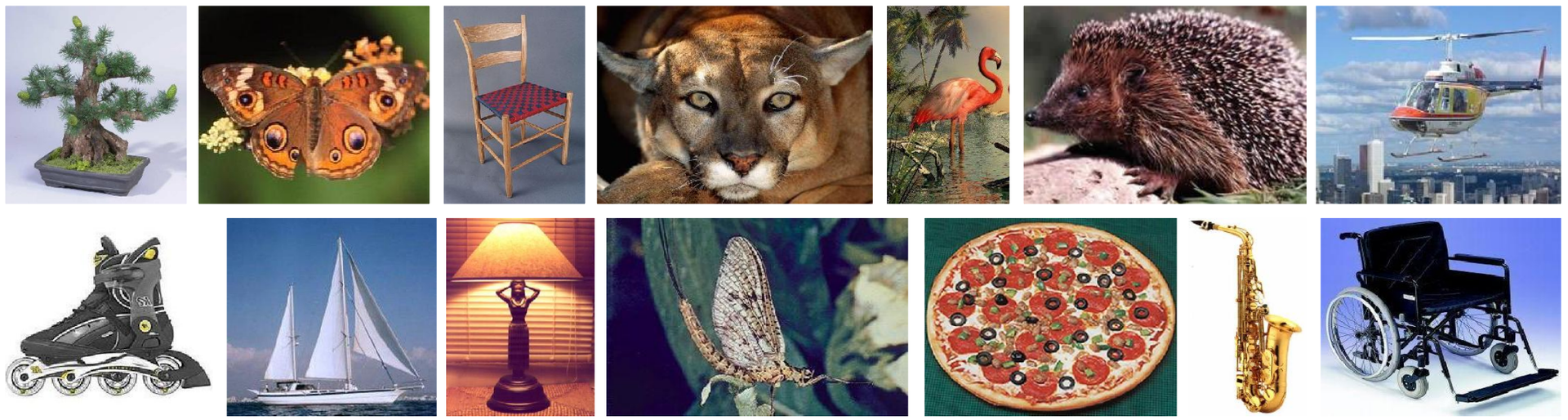
Spatial pyramids

- Very useful and effective for representing images:
  - Pyramid is built by using multiple copies of image
  - Each level in the pyramid is 1/4 of the size of previous level
  - The lowest level is of the highest resolution
  - The highest level is of the lowest resolution
- You can apply the same idea using different image partitions

S.Lazebnik, C.Schmid, J.Ponce, “Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories”, CVPR 2006

Spatial pyramids

- Example: experimental results on Caltech-101



Multi-class classification results (30 training images per class)

	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

Slide credit: S. Lazebnik

Object detection

- **Problem**: *recognizing* and *localizing* generic objects from various categories, such as cars, people, etc.

- Challenges:
  - illumination
  - viewpoint
  - deformations
  - Intra-class variability

Object detection

- Sliding window: slide through the image and check if there is an object at every location
- Should slide over:
  - locations
  - scales
  - windows size

Object detection

- How do we evaluate object detection?

predictions

ground truth

Overlap > *Th* (0.5) between predicted boxes and ground truth