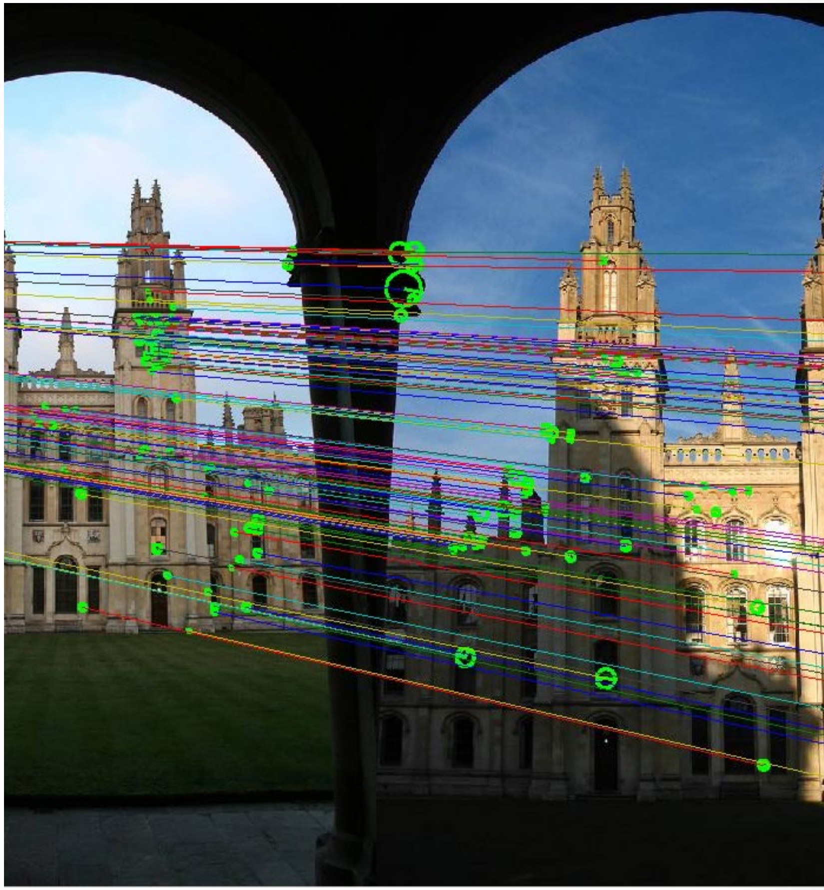
VISUAL SIGNAL PROCESS & PIPELINE

Chih-Chung Hsu (許志仲)
Assistant Professor
ACVLab, Institute of Data Science
National Cheng Kung University
https://cchsu.info



Supervised learning

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

output

prediction function

Image feature

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

Slide credit: L. Lazebnik

Image Categorization

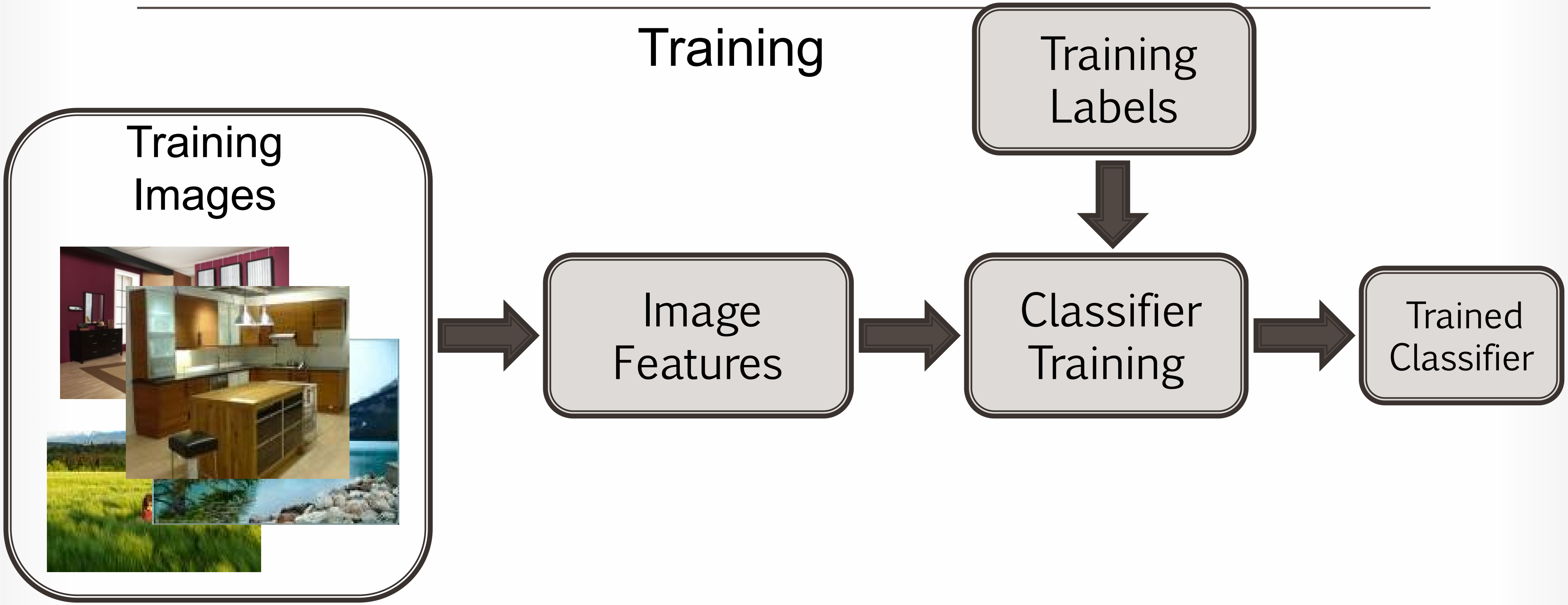


Image Categorization

Training

Training Labels

Training Images

Image Features

Classifier Training

Trained Classifier

Testing

Test Image

Image Features

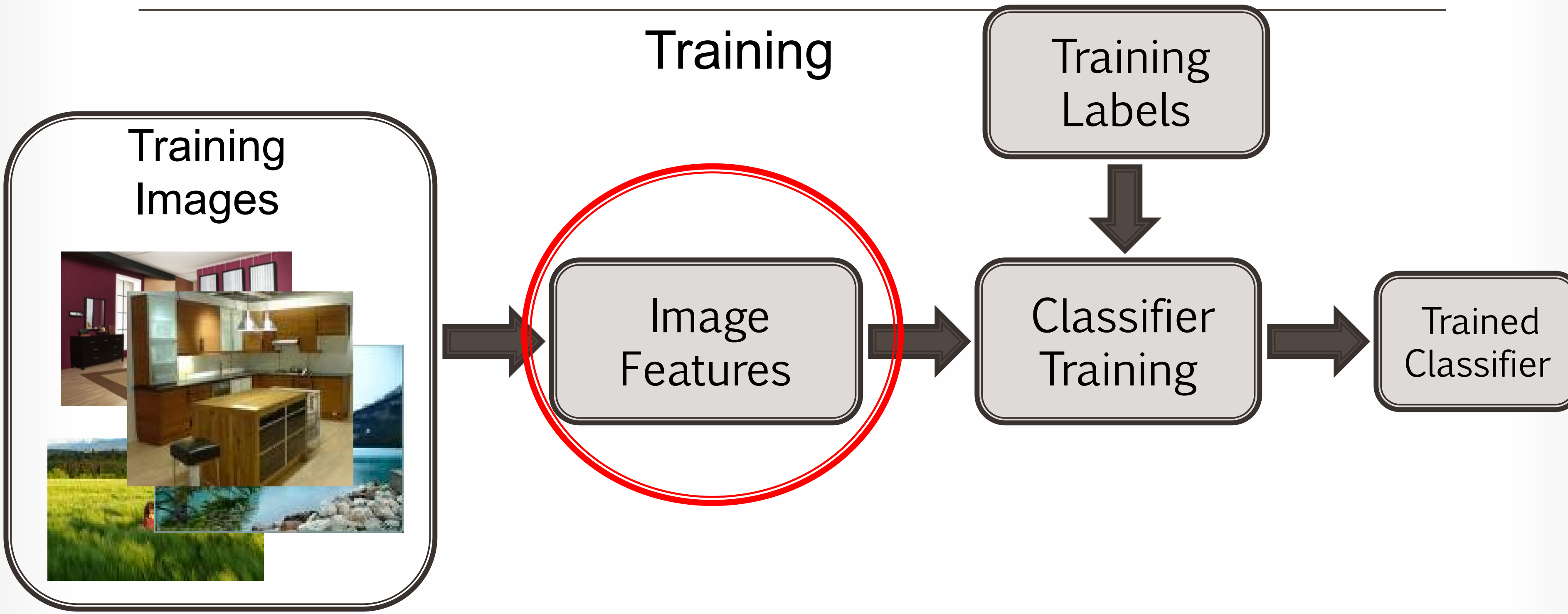
Trained Classifier

Prediction
Outdoor

Example: Scene Categorization

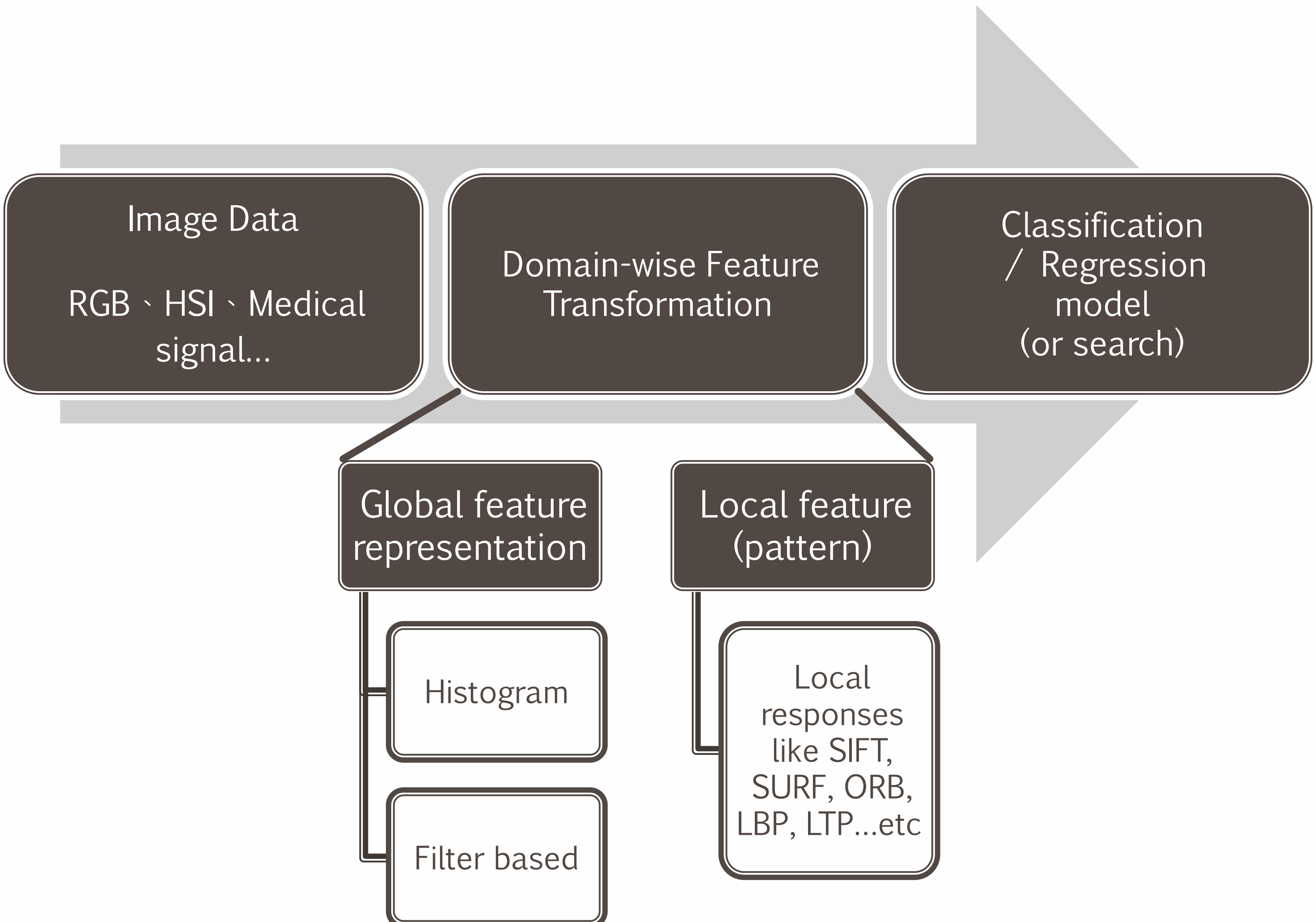
- Is this a kitchen?

Image features



BASELINE: IMAGE RECOGNITION WITH FEATURES

Data Pipeline for Image Recognition

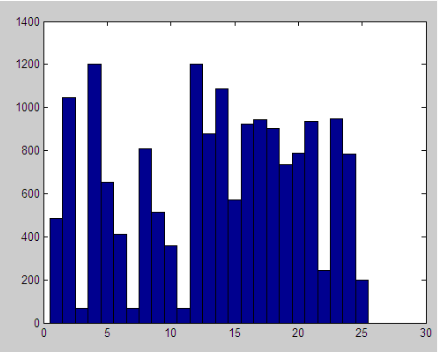


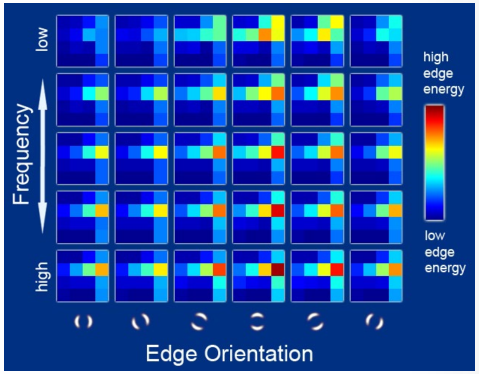
Feature-based Image Classification

- Global Feature
- It is nothing new.
  - Just use the feature extracted from images and followed by feeding the feature to the classifier.
- Local Feature
  - #features per image fixed.
    - Similar to global one
  - #features variant
    - Matching is necessary….

Features

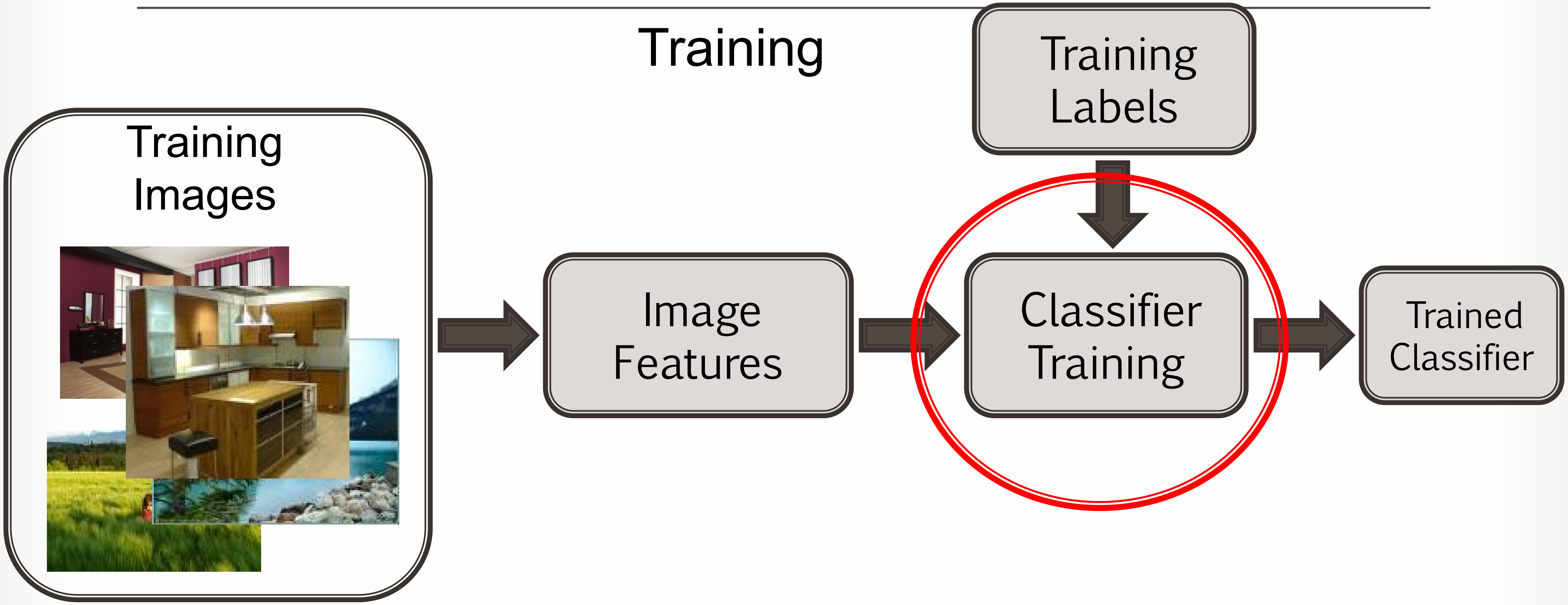
- Raw pixels
- Histograms
- GIST descriptors
- SIFT/SURF/LBP/HOG…
- Learning features from data

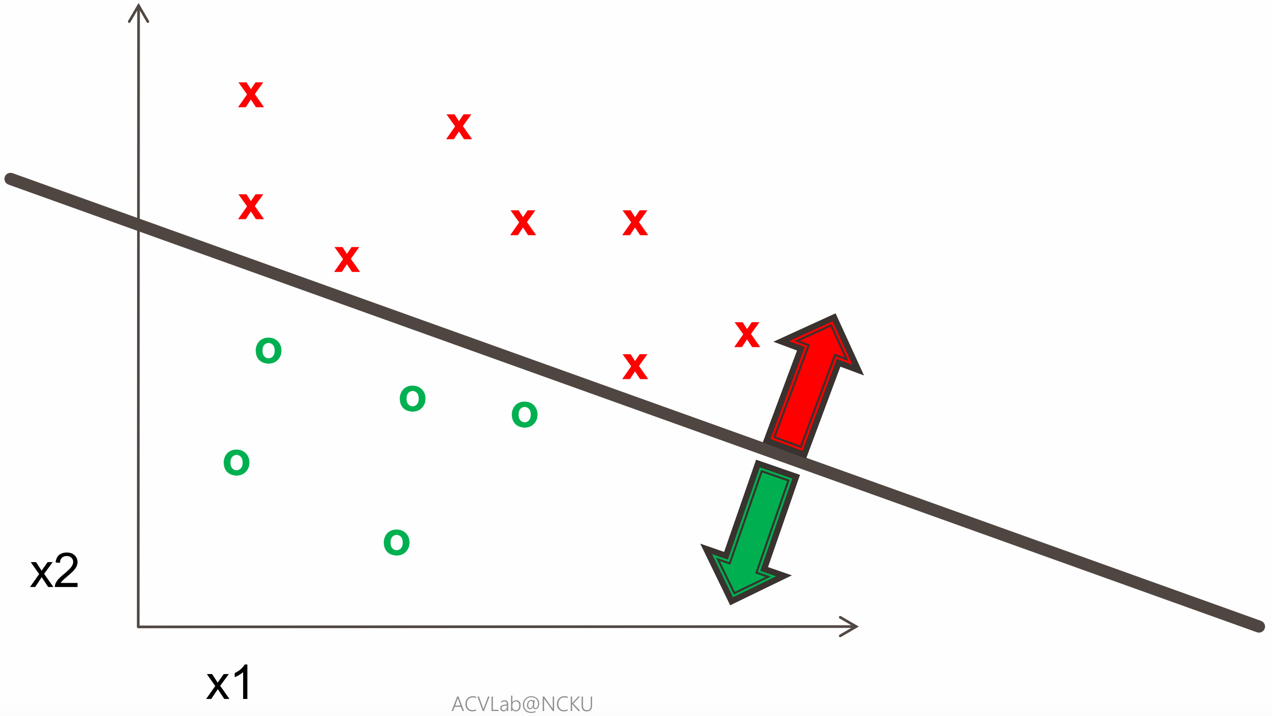
Slide credit: L. Lazebnik

Classifiers

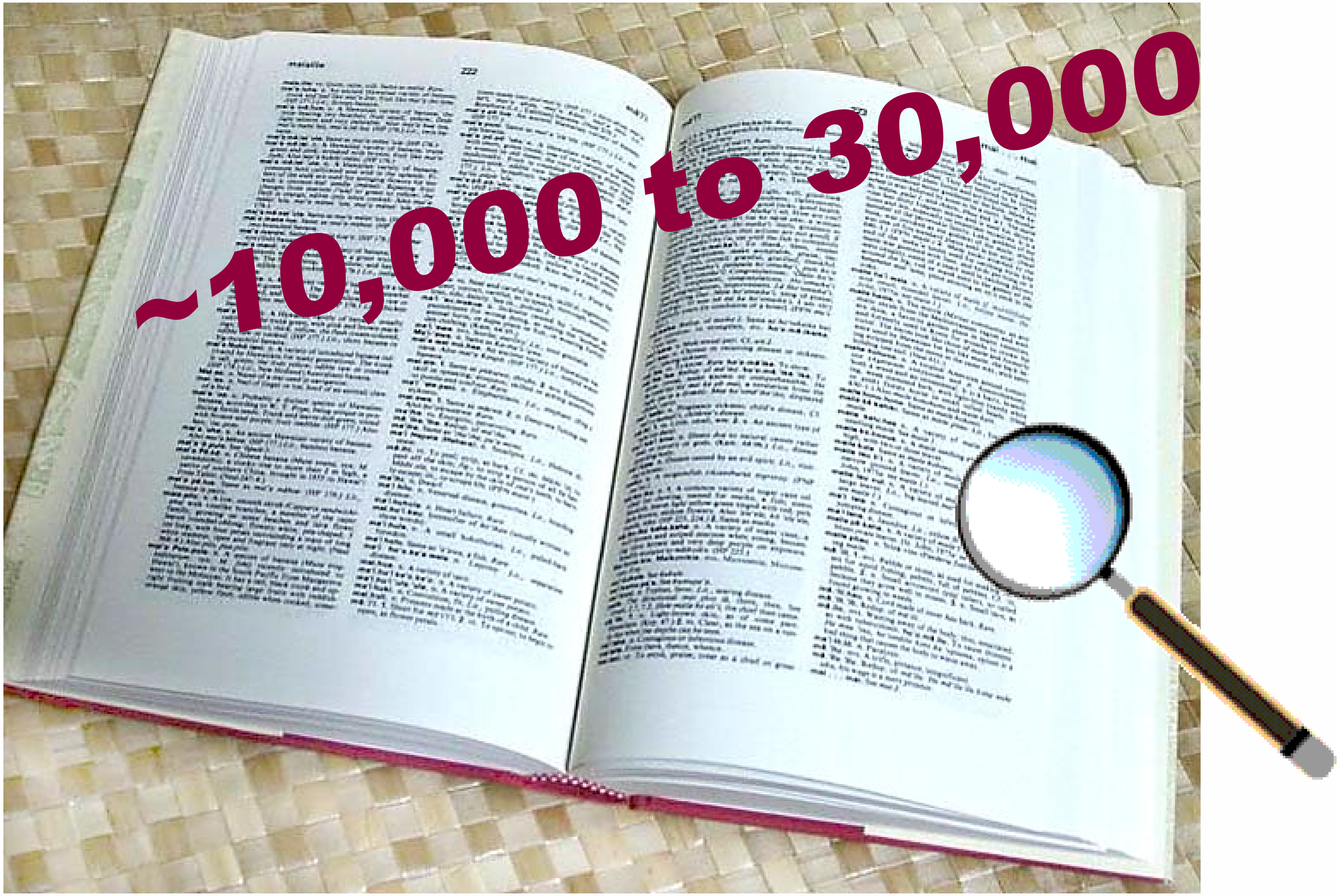


Learning a classifier

Given some set of features with corresponding labels, learn a function to predict the labels from the features



How many visual object categories are there?



Biederman 1987


~10,000 to 30,000

OBJECTS

- ANIMALS
  - .....
  - VERTEBRATE
    - MAMMALS
      - TAPIR
      - BOAR
    - BIRDS
      - GROUSE
- PLANTS
- INANIMATE (無生命)
  - NATURAL
  - MAN-MADE
    - CAMERA

Specific recognition tasks

Scene categorization or classification

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

Image annotation / tagging / attributes

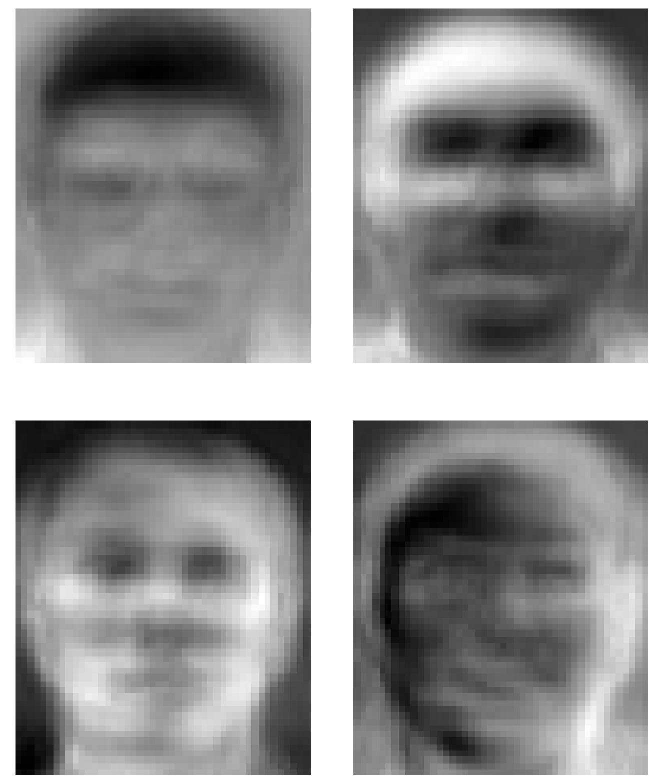
- street
- people
- building
- mountain
- tourism
- cloudy
- brick
- …

Object detection

Image parsing / semantic segmentation

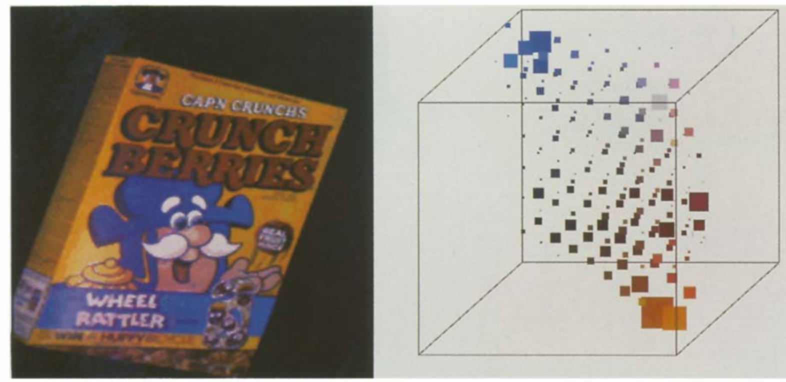
Scene understanding?

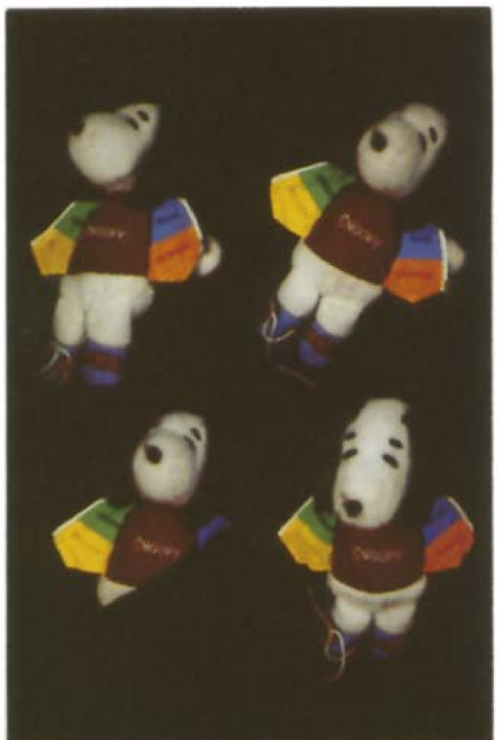
Eigenfaces (Turk & Pentland, 1991)

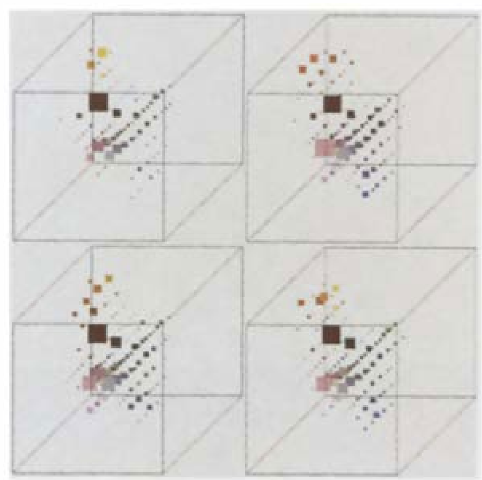


Experimental	Correct/Unknown Recognition Percentage		
Condition	Lighting	Orientation	Scale
Forced classification	96/0	85/0	64/0
Forced 100% accuracy	100/19	100/39	100/60
Forced 20% unknown rate	100/20	94/20	74/20

Color Histograms





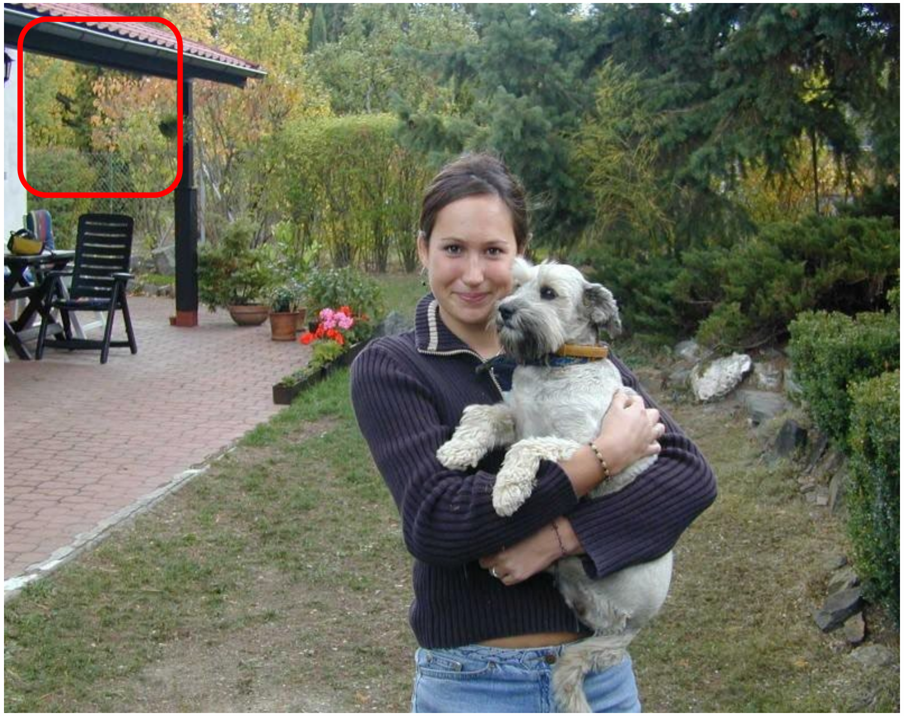


Swain and Ballard, Color Indexing, IJCV 1991.

History of ideas in recognition

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

Sliding window approaches



History of ideas in recognition

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

Local features for object instance recognition

D. Lowe (1999, 2004)

Large-scale image search

Combining local features, indexing, and spatial constraints

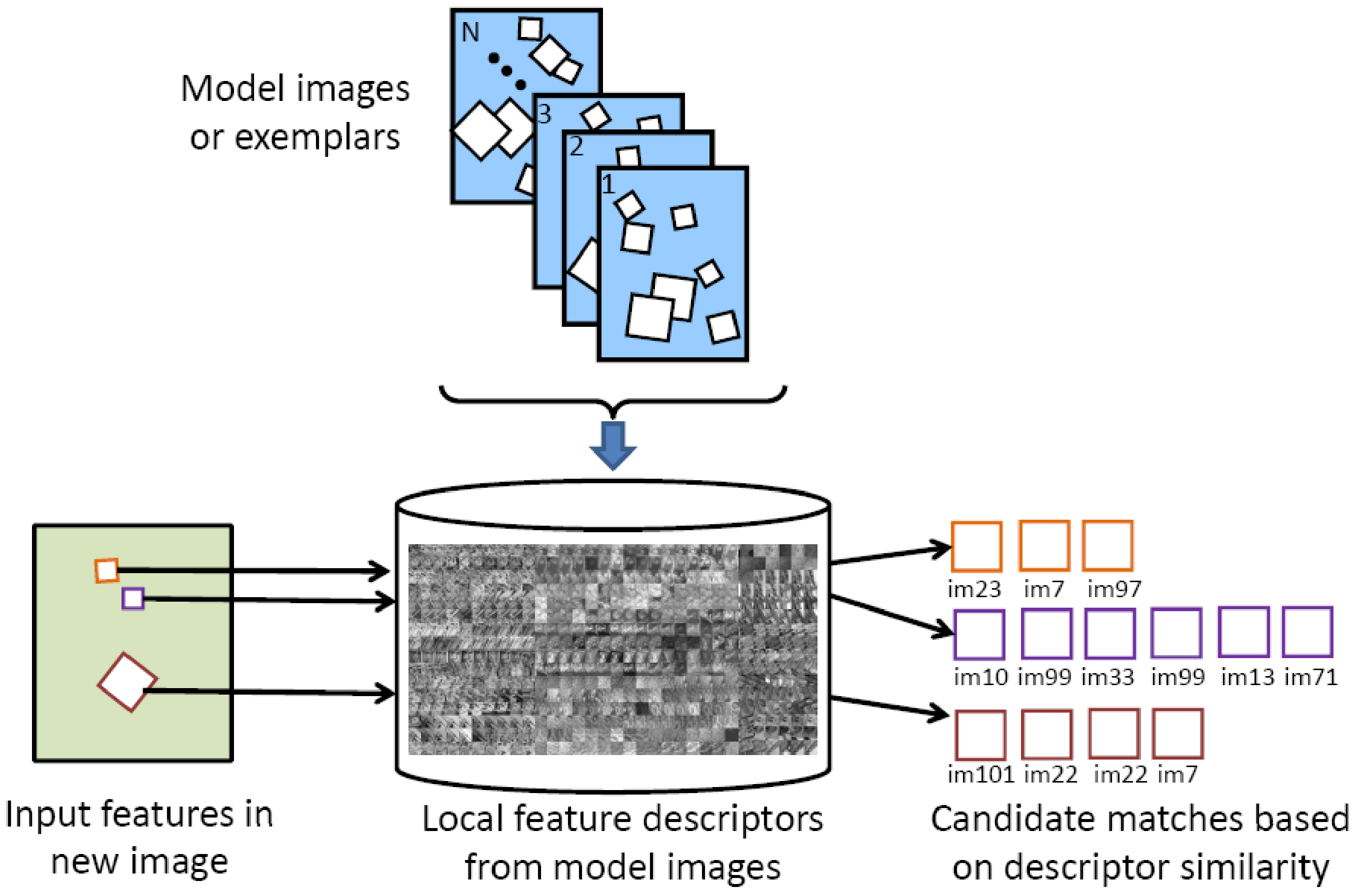
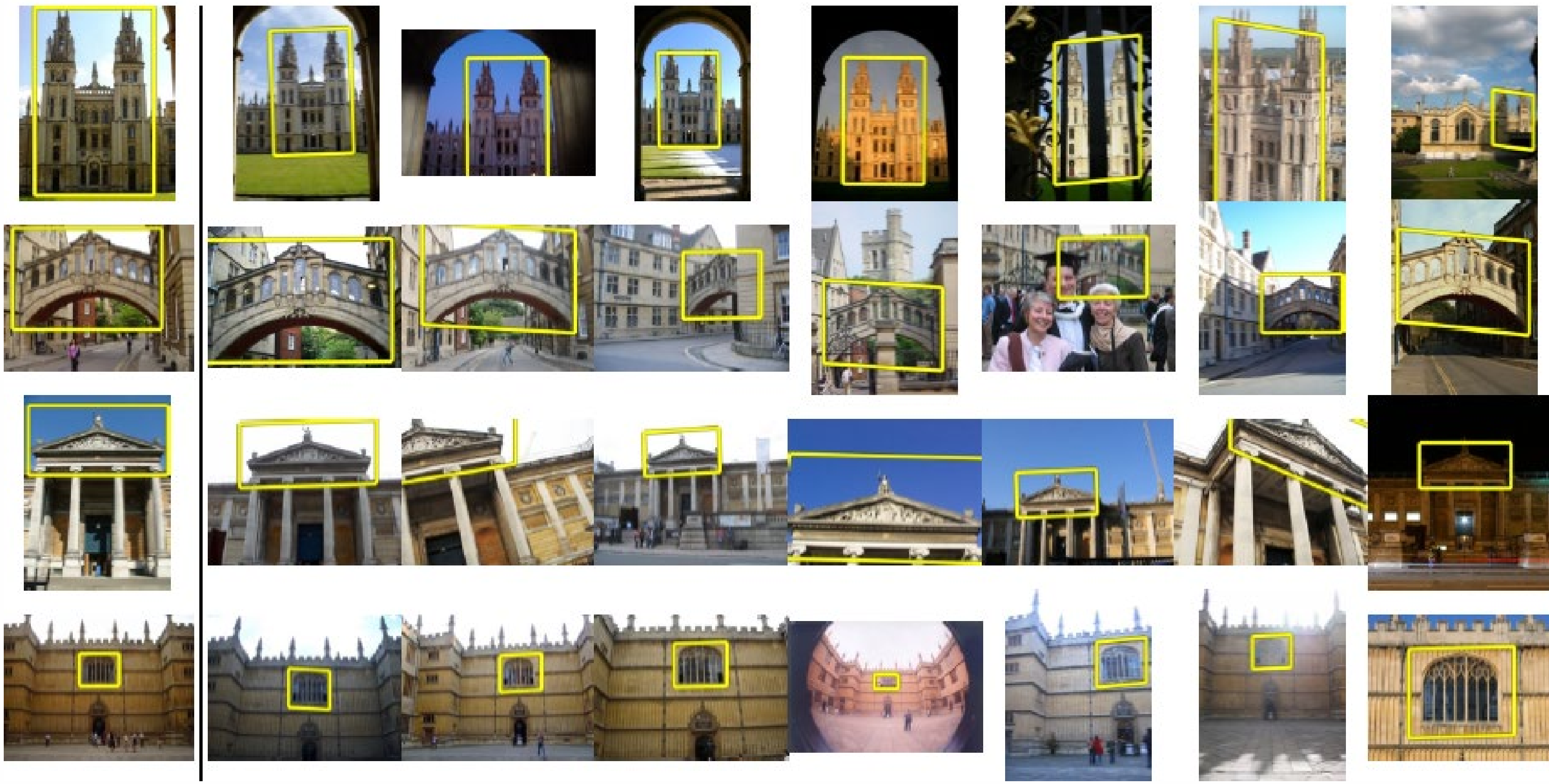


Image credit: K. Grauman and B. Leibe

Large-scale image search

Combining local features, indexing, and spatial constraints



Philbin et al. '07

Large-scale image search

Combining local features, indexing, and spatial constraints

Google Goggles in Action

Click the icons below to see the different ways Google Goggles can be used.

Available on phones that run Android 1.6+ (i.e. Donut or Eclair)

History of ideas in recognition

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

Parts-and-shape models

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

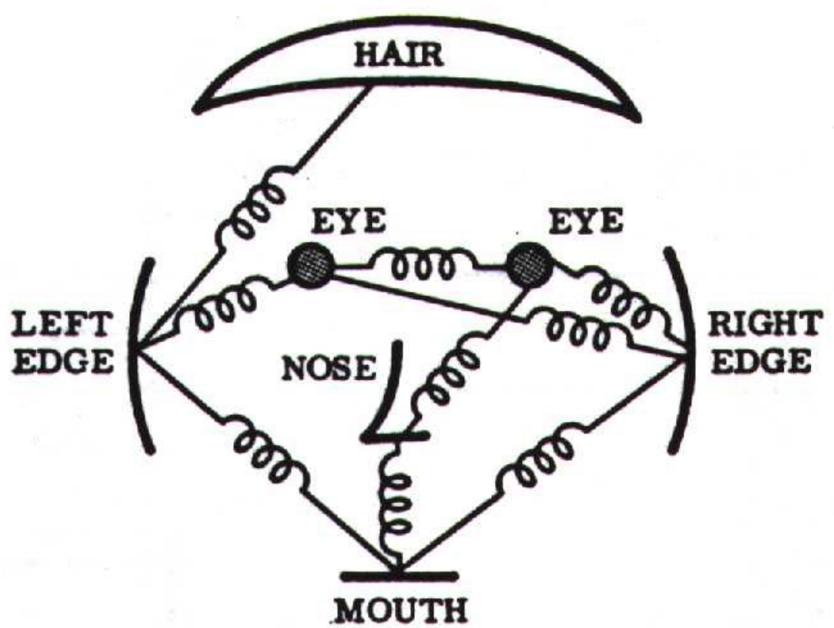
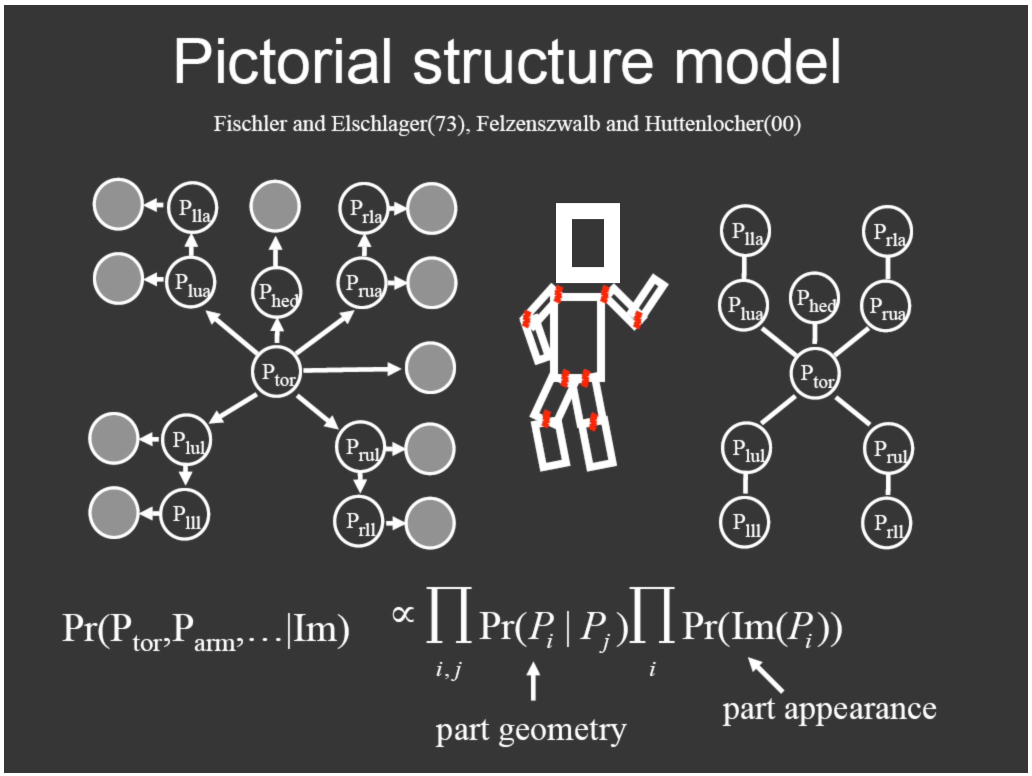
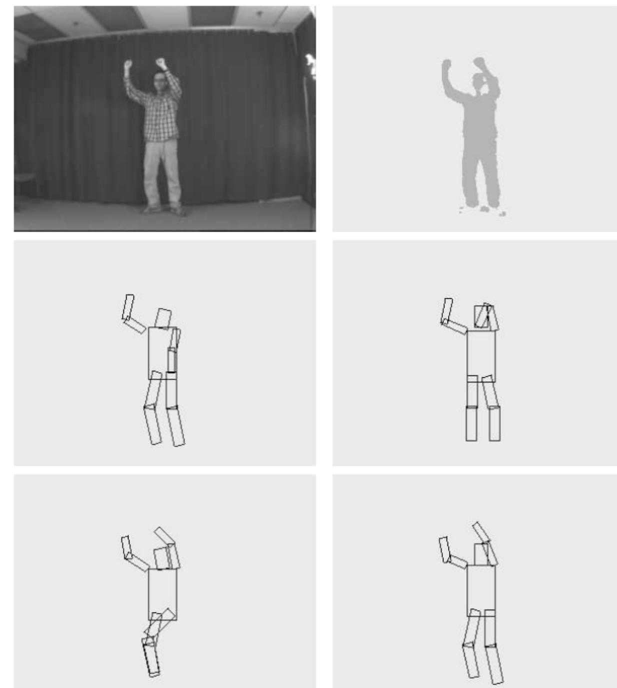


Figure from [Fischler & Elschlager 73]

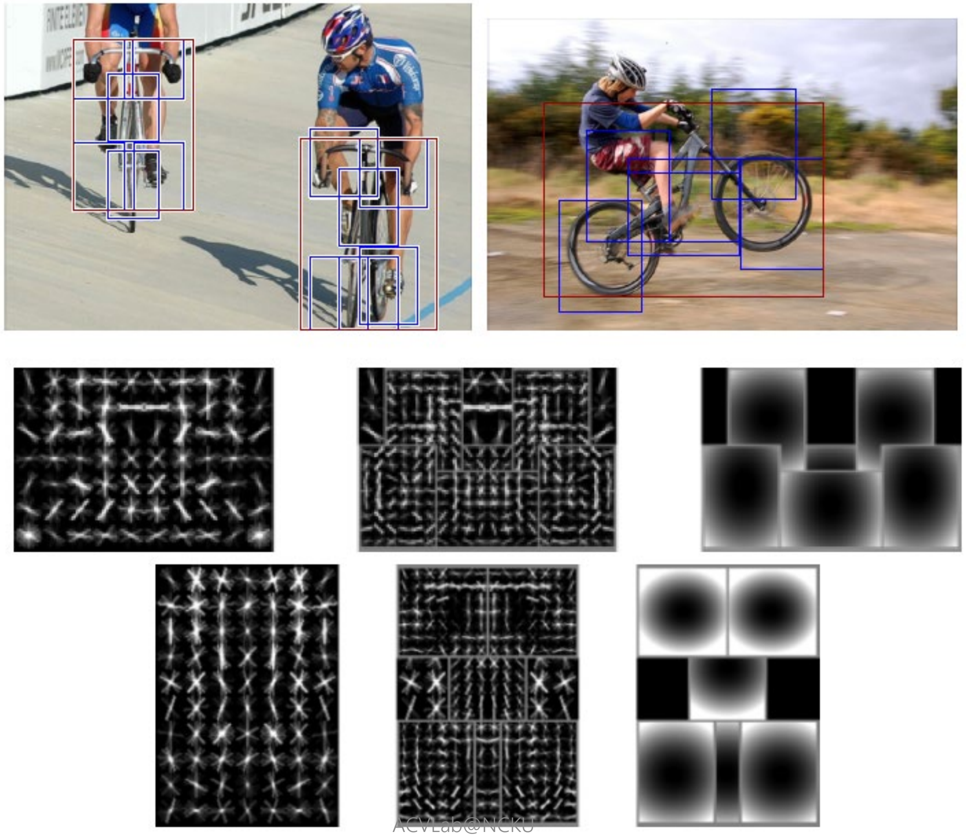
Representing people





Discriminatively trained part-based models

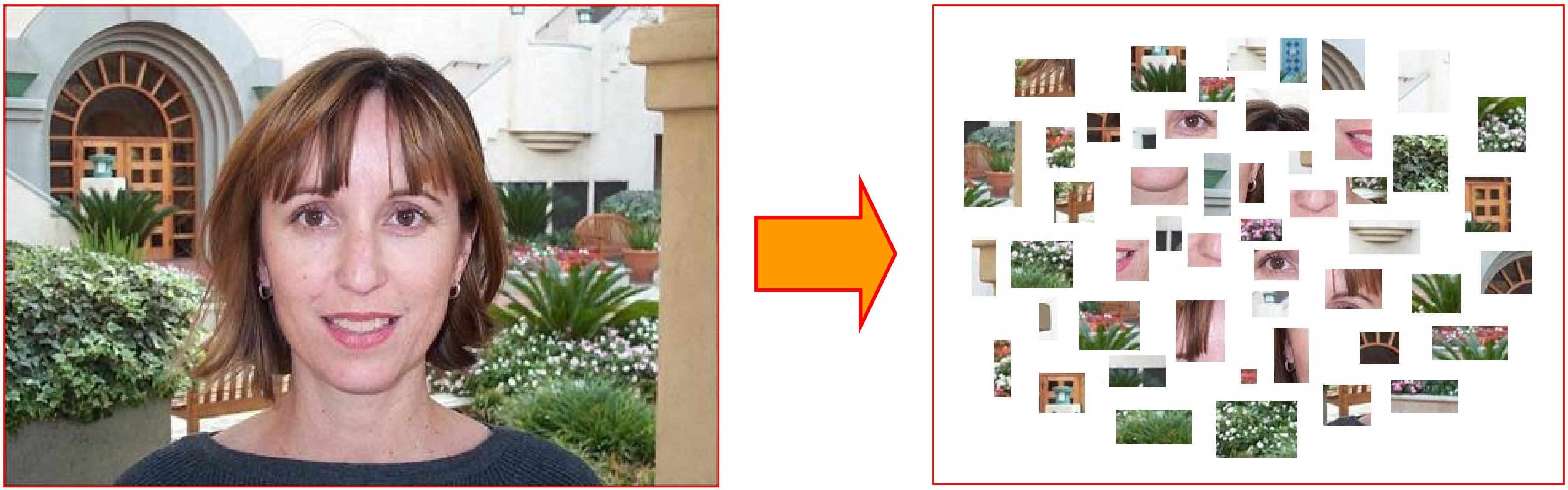
- P. Felzenszwalb, R. Girshick, D. McAllester, D. Ramanan, "Object Detection with Discriminatively Trained Part-Based Models," PAMI 2009



History of ideas in recognition

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

Bag-of-features models



Bag-of-features models

Object → **Bag of ‘words’**

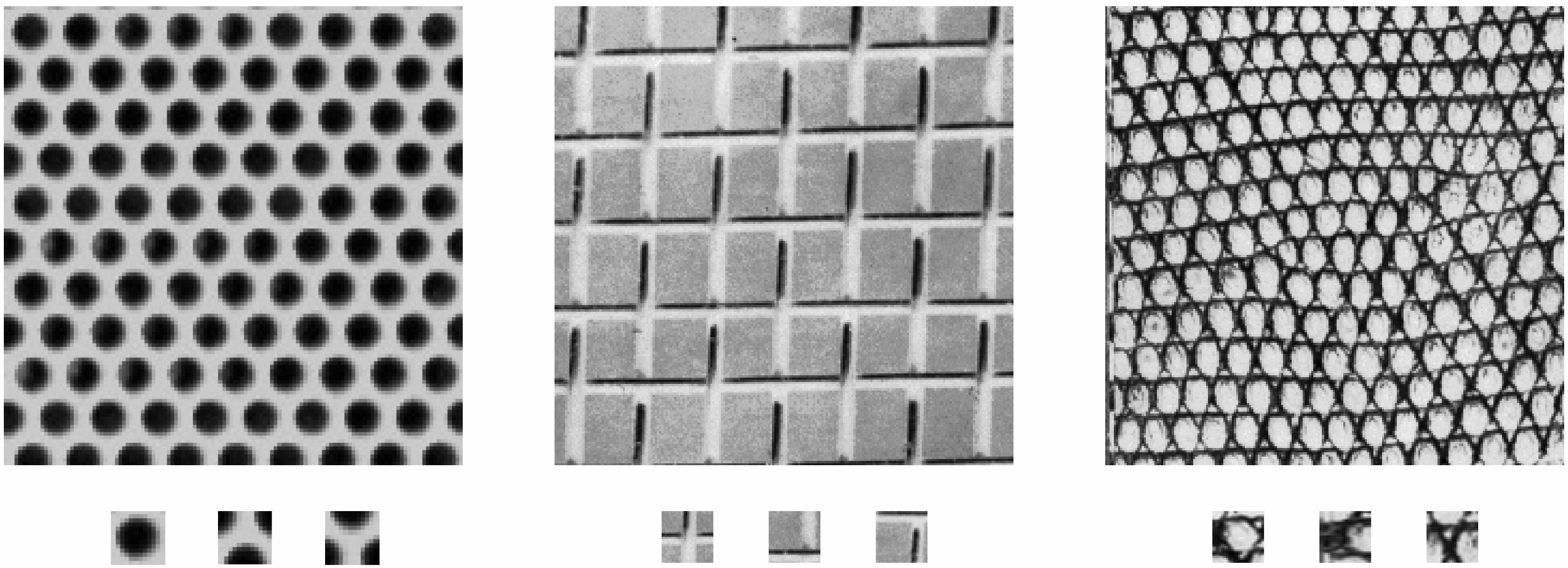
Objects as texture

- All of these are treated as being the same

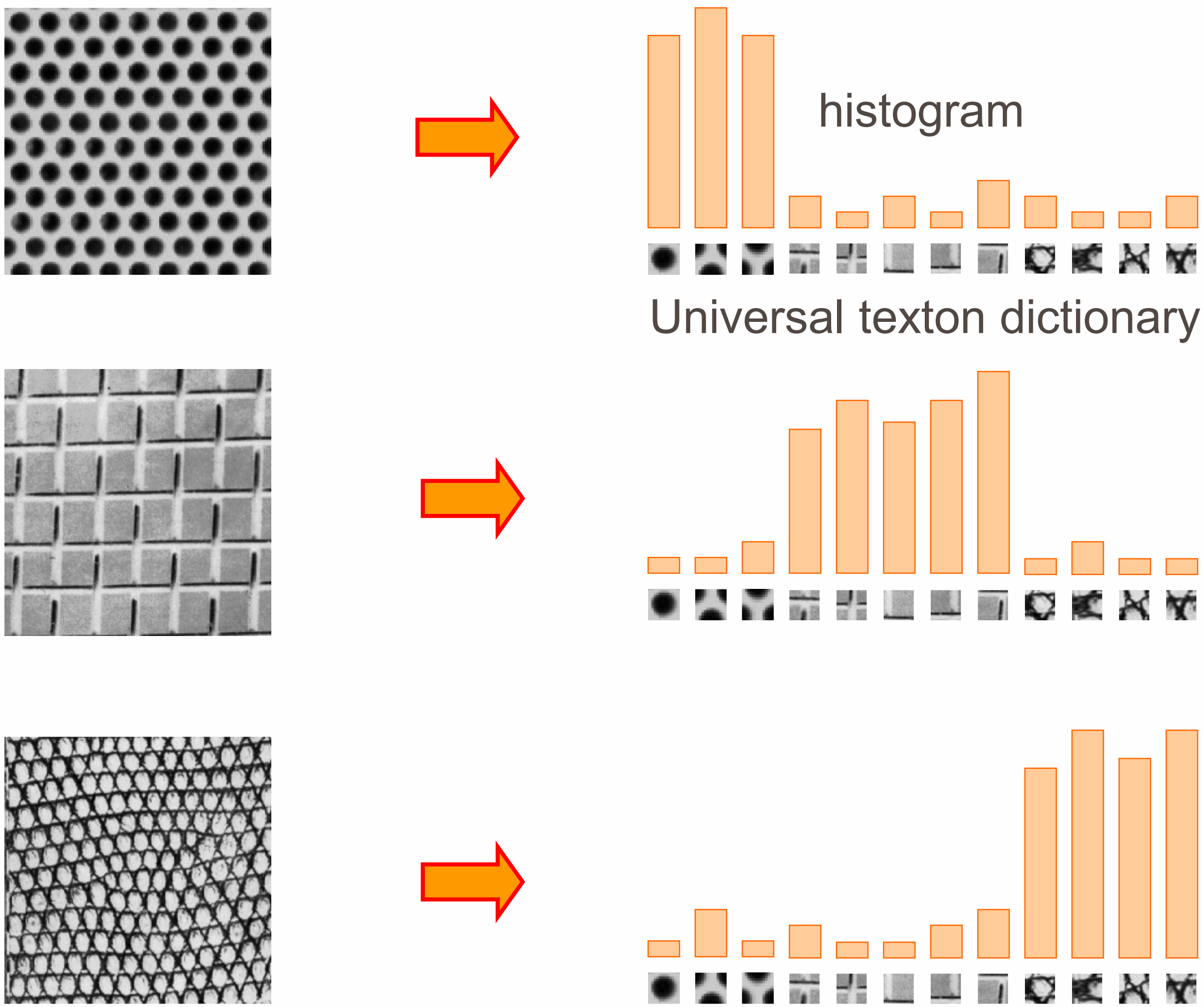
- No distinction between foreground and background: scene recognition?

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



Origin 1: Texture recognition



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

Origin 2: Bag-of-words models

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

Origin 2: Bag-of-words models

- Orderless document representation: frequencies of words from a

Origin 2: Bag-of-words models

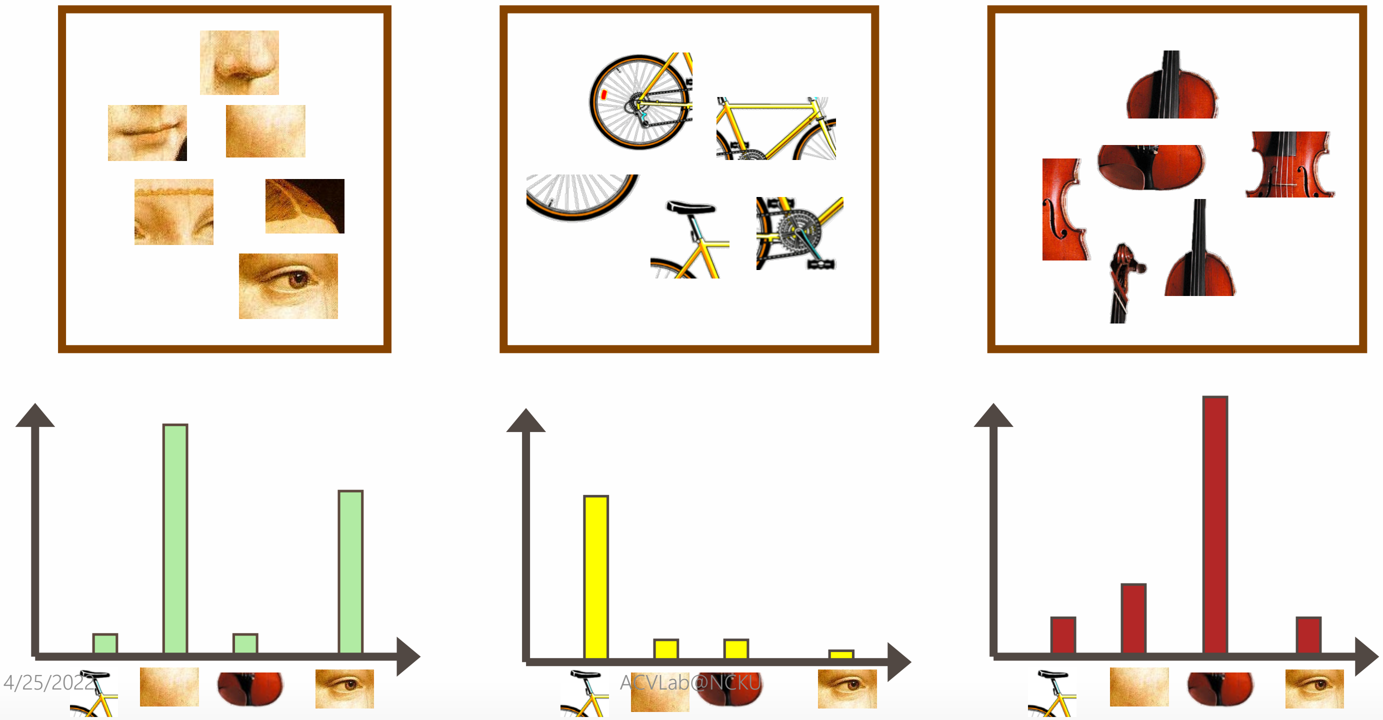
- Orderless document representation: frequencies of words from a

Origin 2: Bag-of-words models

- Orderless document representation: frequencies of words from a

Bag-of-features steps

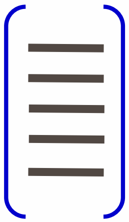
- Extract features
- Learn “visual vocabulary”
- Quantize features using visual vocabulary
- Represent images by frequencies of “visual words”



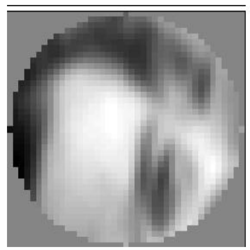
1. Feature extraction

- Regular grid or interest regions

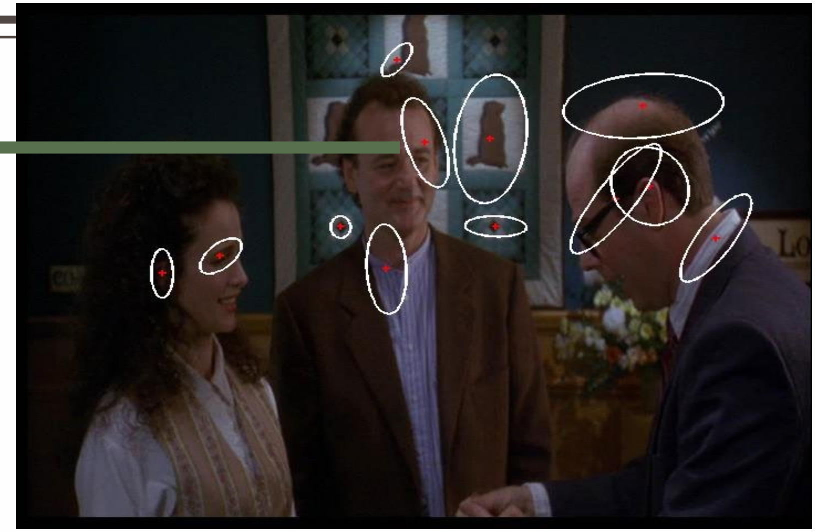
1. Feature extraction



Compute descriptor



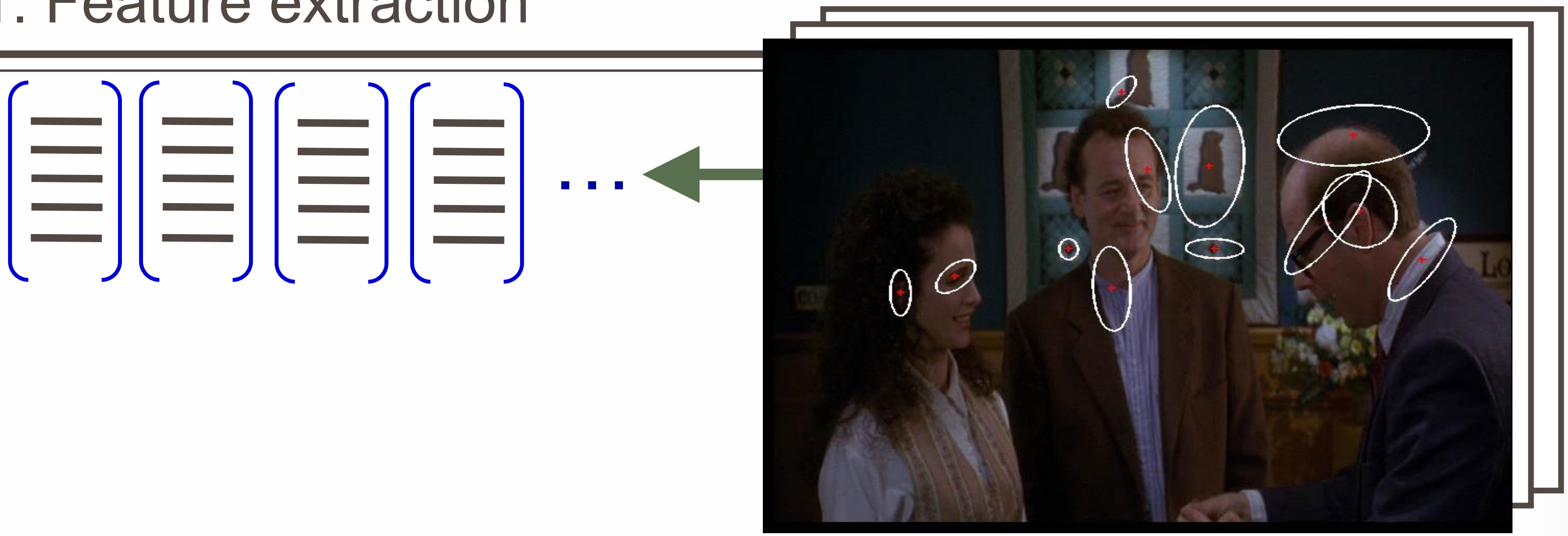
Normalize patch



Detect patches

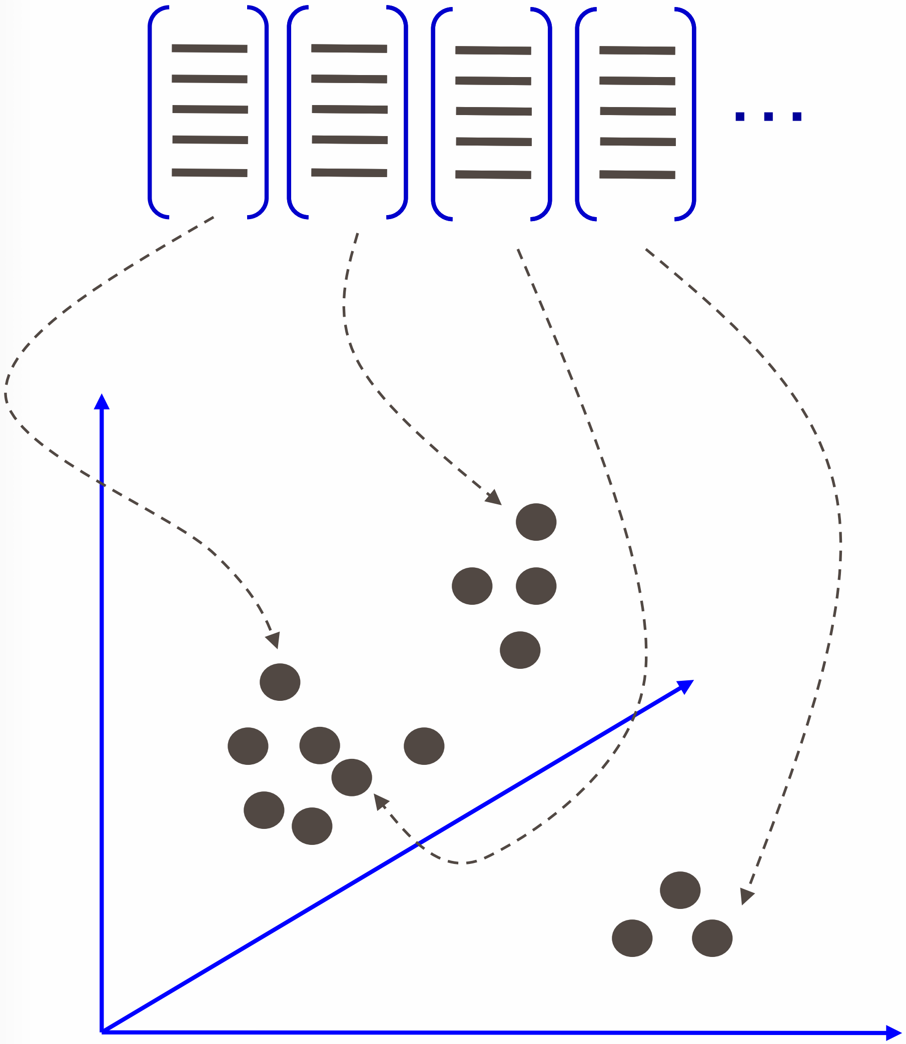
Slide credit: Josef Sivic

1. Feature extraction



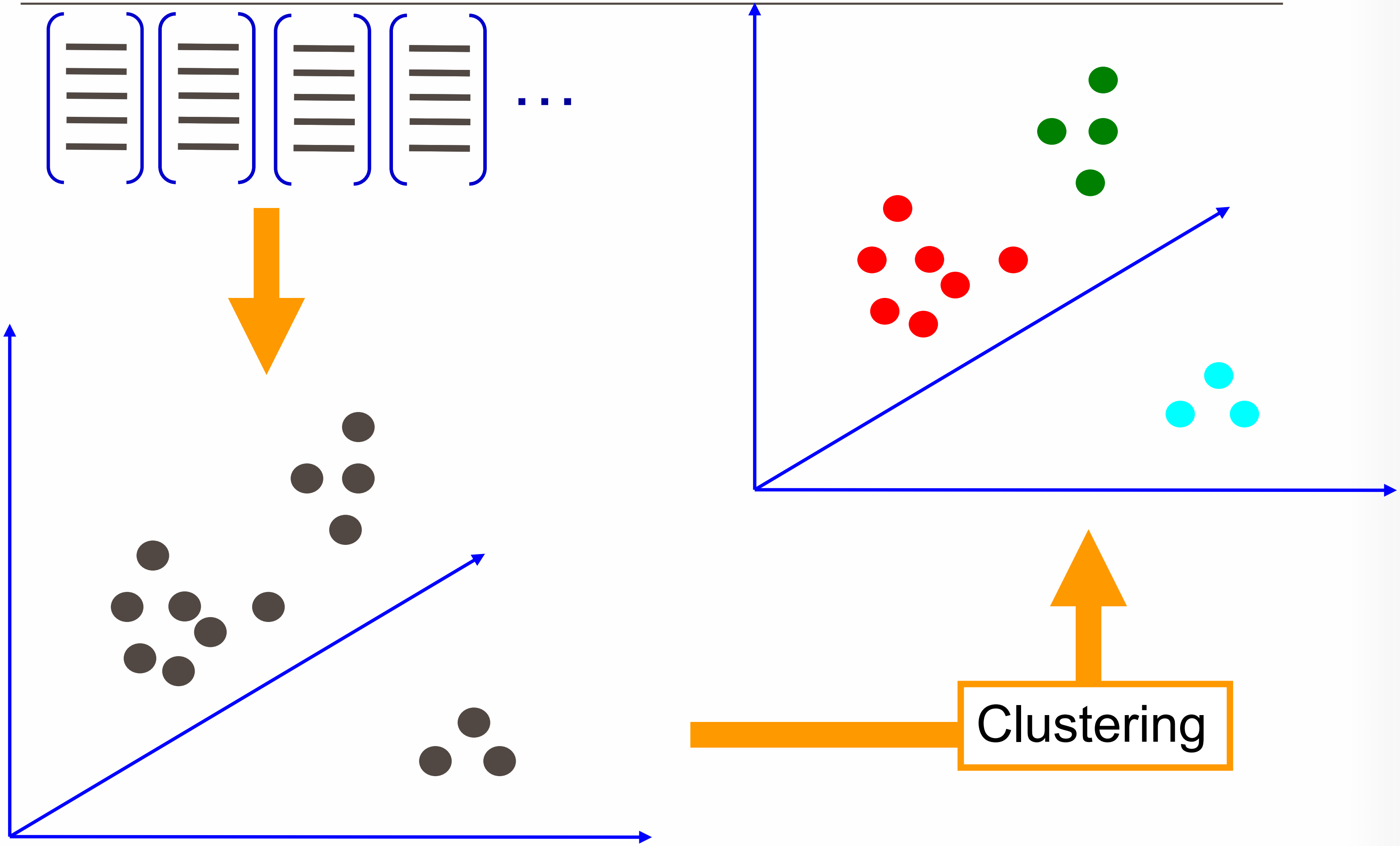
Slide credit: Josef Sivic

2. Learning the visual vocabulary



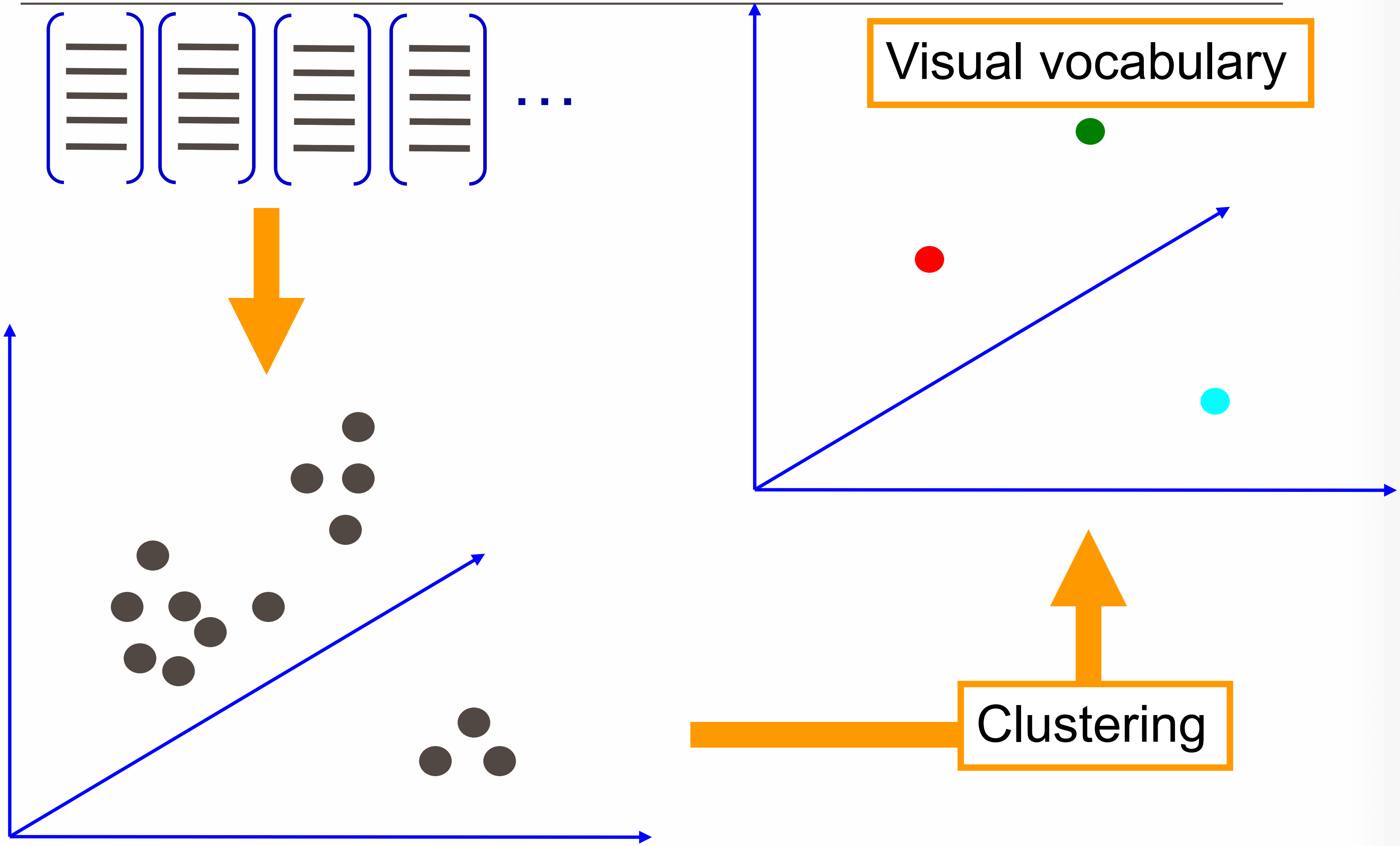
Slide credit: Josef Sivic

2. Learning the visual vocabulary



Slide credit: Josef Sivic

2. Learning the visual vocabulary



Slide credit: Josef Sivic

K-means clustering

- Want to minimize sum of squared Euclidean distances between points xi and their nearest cluster centers mk

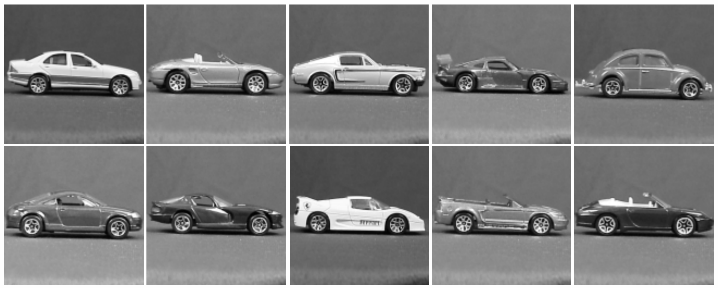
$$D(X,M) = \sum_{\text{cluster } k} \sum_{\substack{\text{point } i \text{ in} \\ \text{cluster } k}} (x_i - m_k)^2$$

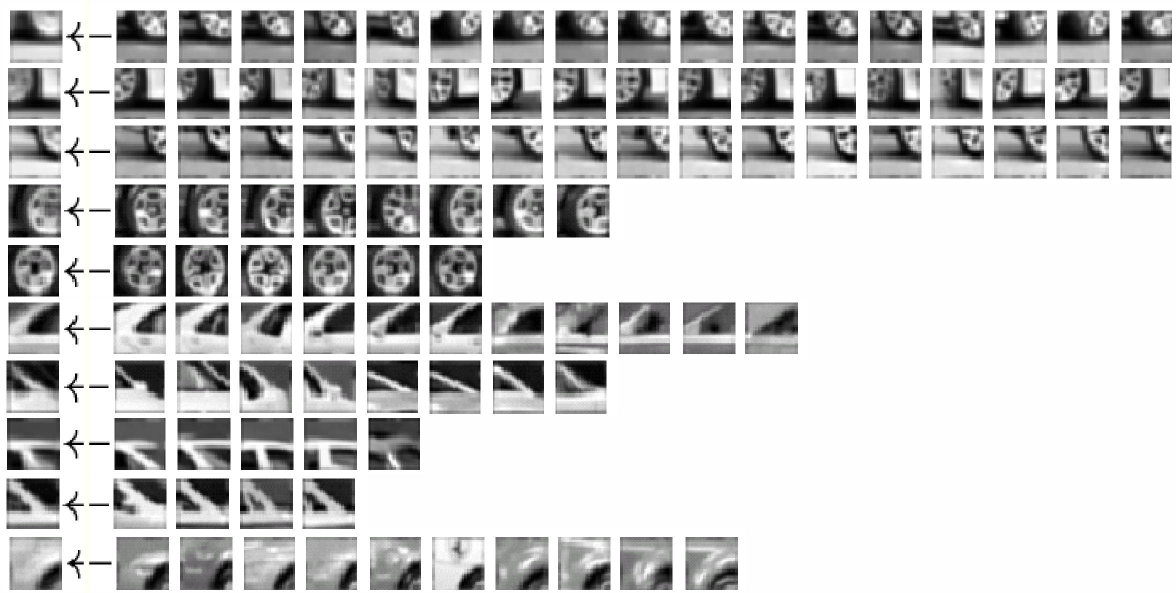
- Algorithm:
- Randomly initialize K cluster centers
- Iterate until convergence:
  - Assign each data point to the nearest center
  - Recompute each cluster center as the mean of all points assigned to it

Clustering and vector quantization

- Clustering is a common method for learning a visual vocabulary or codebook
  - Unsupervised learning process
  - Each cluster center produced by k-means becomes a 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 quantizing features
  - A vector quantizer takes a feature vector and maps it to the index of the nearest codevector in a codebook
  - Codebook = visual vocabulary
  - Codevector = visual word

Example codebook



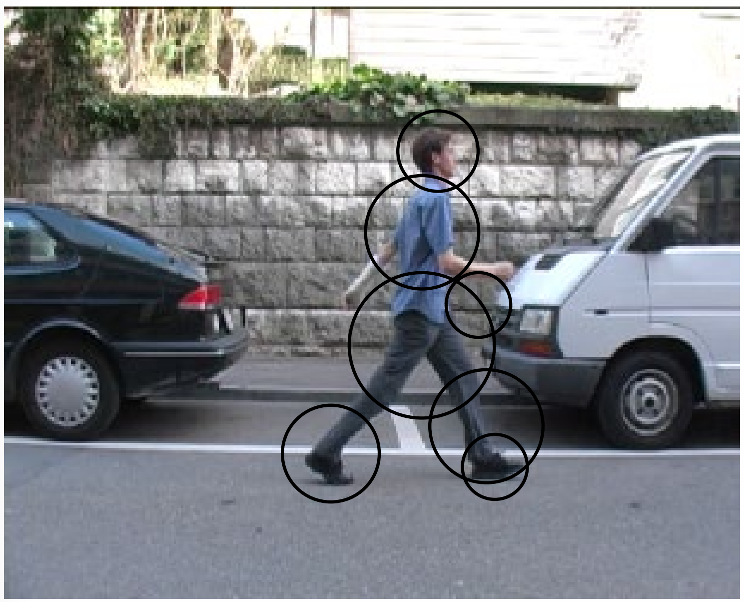


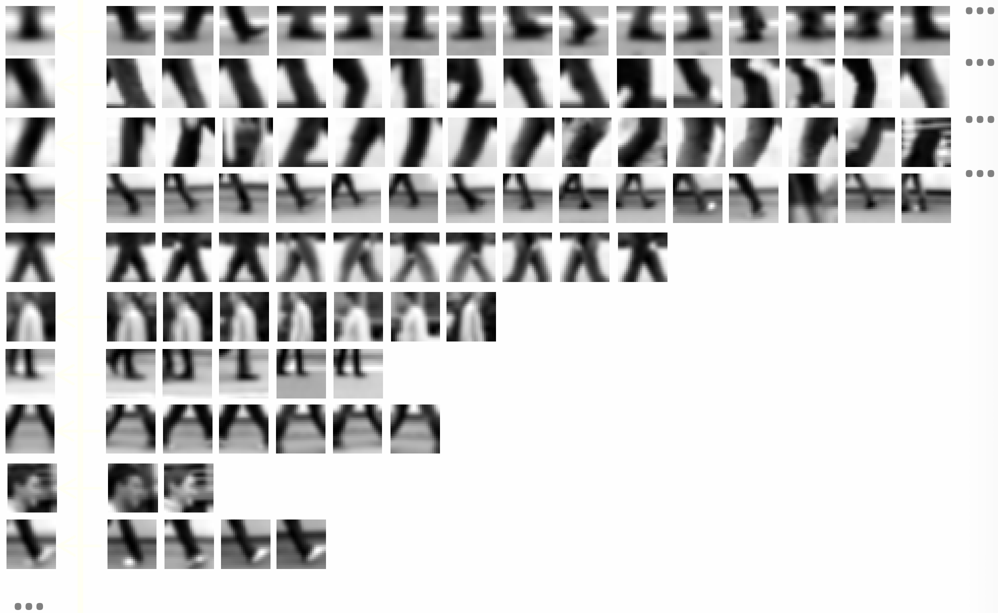
...

Appearance codebook

Source: B. Leibe

Another codebook



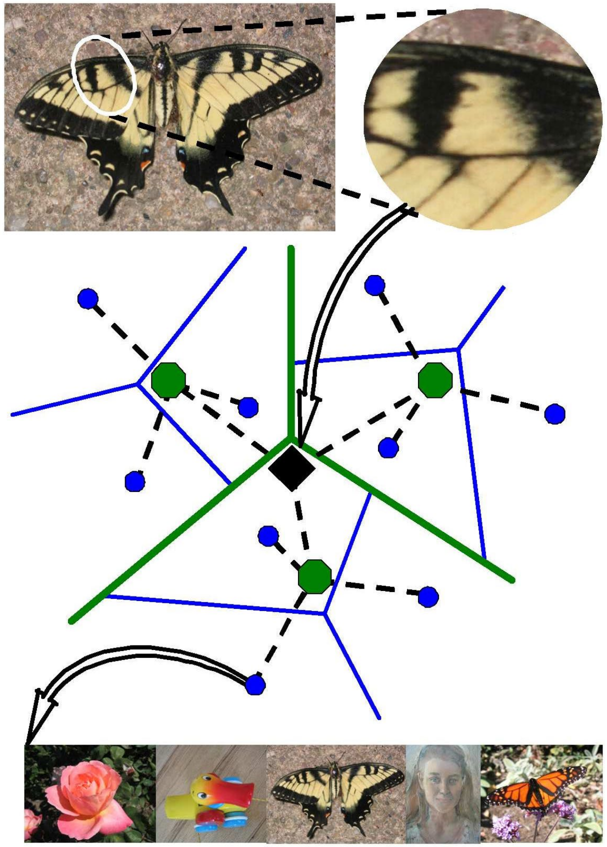


Appearance codebook

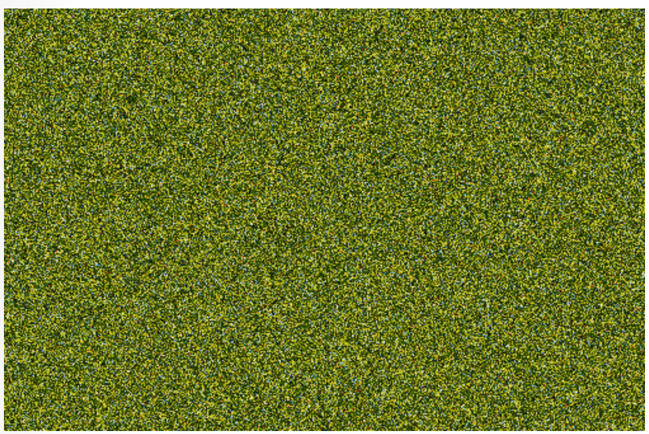
Source: B. Leibe

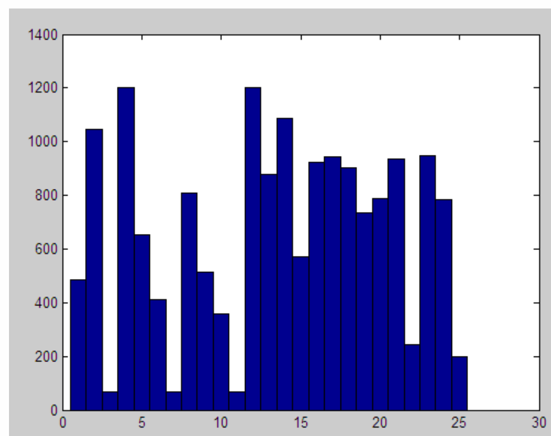
Visual vocabularies: Issues

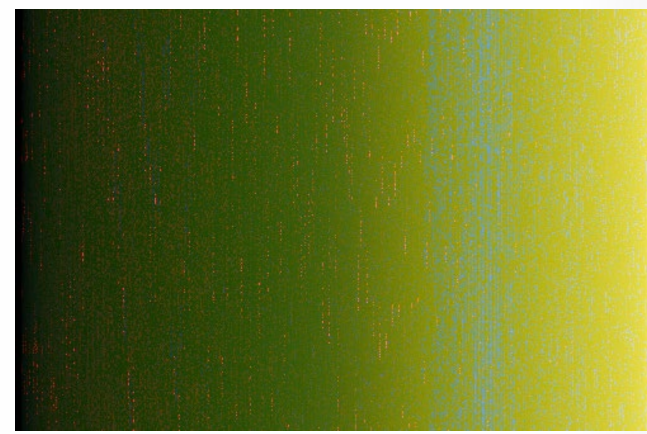
- How to choose vocabulary size?
  - Too small: visual words not representative of all patches
  - Too large: quantization artifacts, overfitting
- Computational efficiency
  - Vocabulary trees (Nister & Stewenius, 2006)



But what about layout?







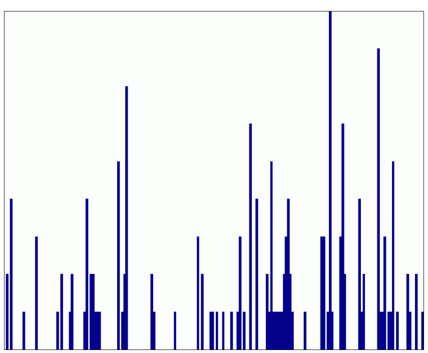
All of these images have the same color histogram

Spatial pyramid

Compute histogram in each spatial bin

Spatial pyramid representation

- Extension of a bag of features
- Locally orderless representation at several levels of resolution

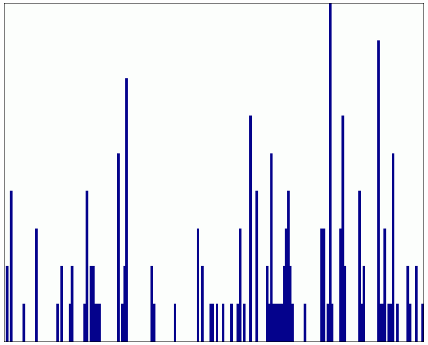


level 0

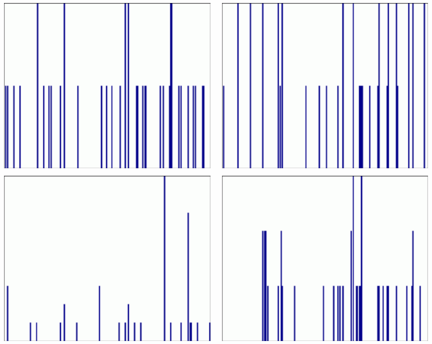
Lazebnik, Schmid & Ponce (CVPR 2006)

Spatial pyramid representation

- Extension of a bag of features
- Locally orderless representation at several levels of resolution



level 0

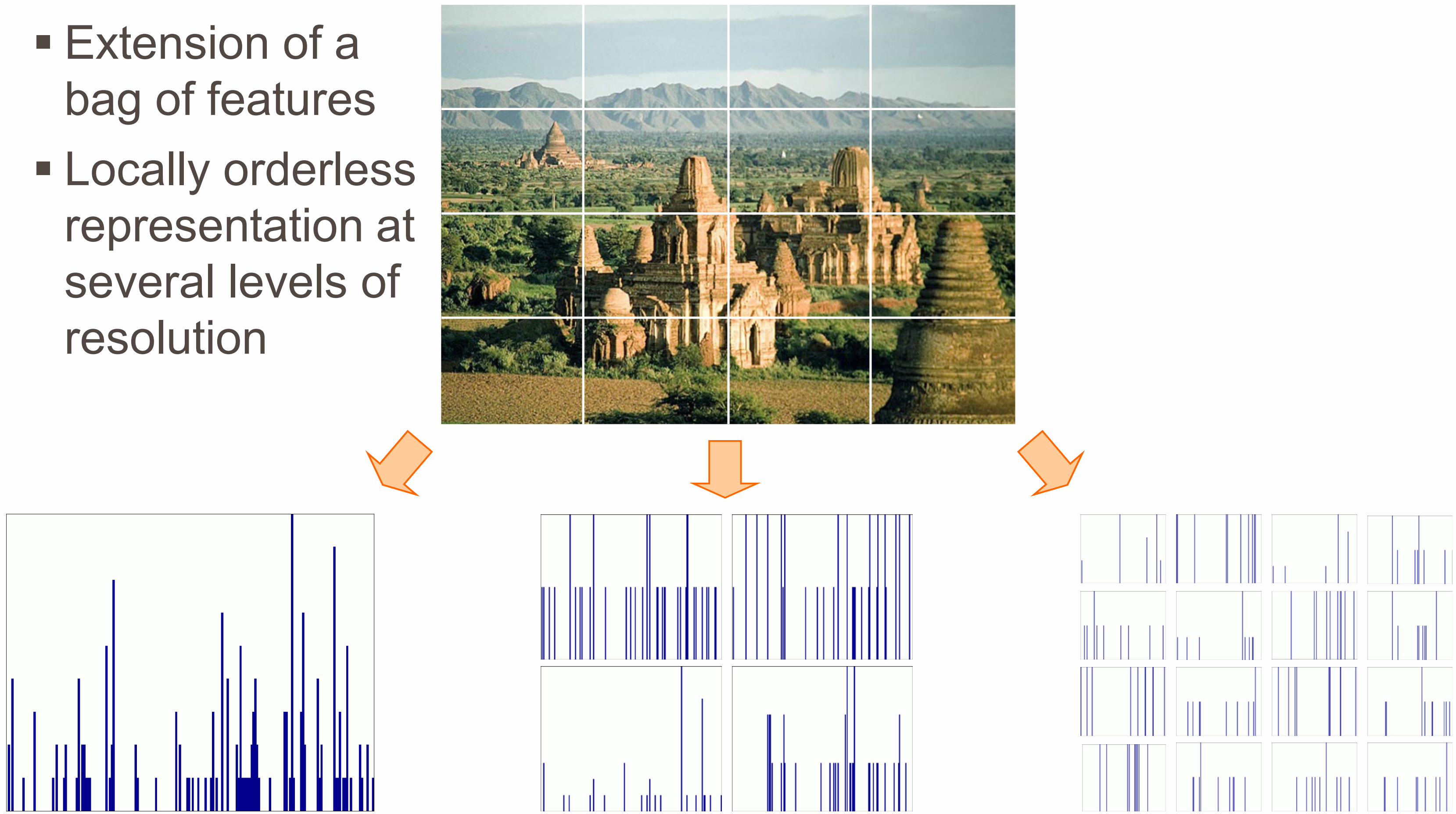


level 1

Lazebnik, Schmid & Ponce (CVPR 2006)

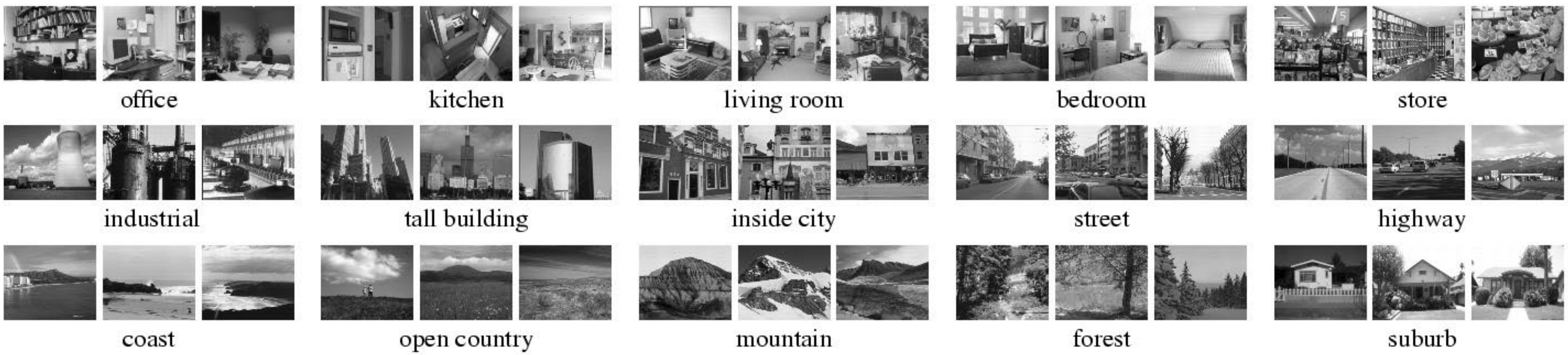
Spatial pyramid representation

- Extension of a bag of features
- Locally orderless representation at several levels of resolution



level 0 level 1 level 2

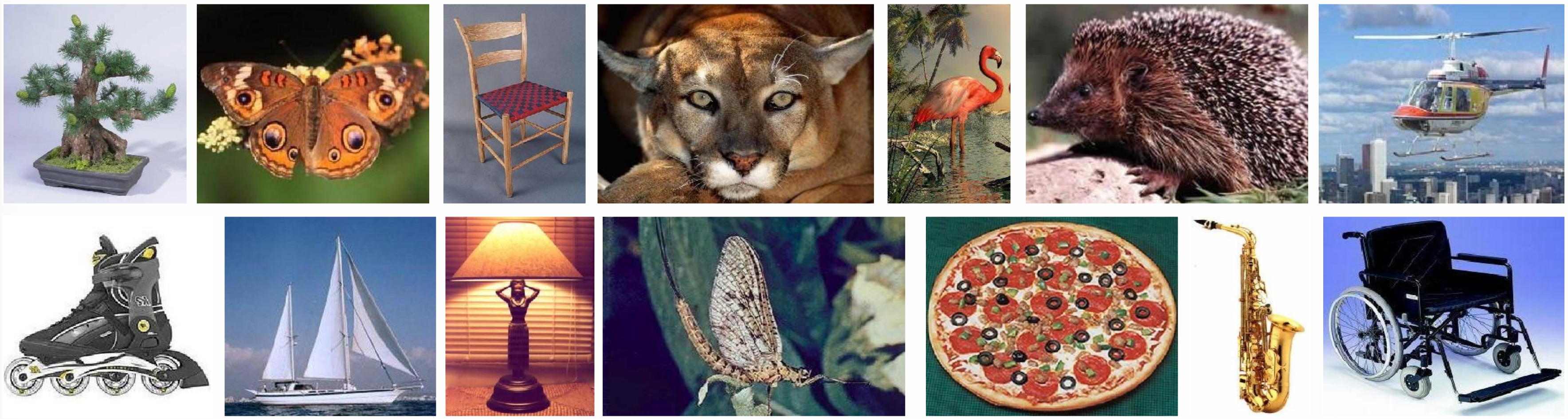
Lazebnik, Schmid & Ponce (CVPR 2006)



Multi-class classification results (100 training images per class)

	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

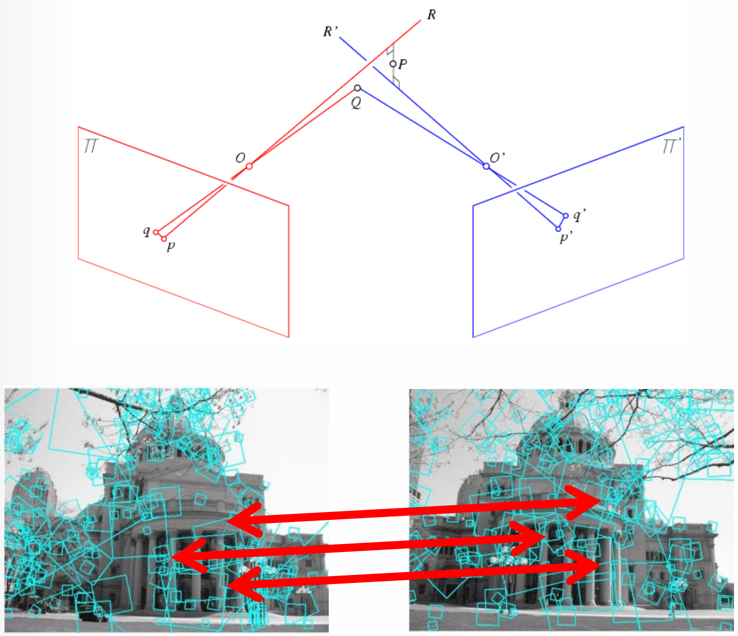
Caltech101 dataset



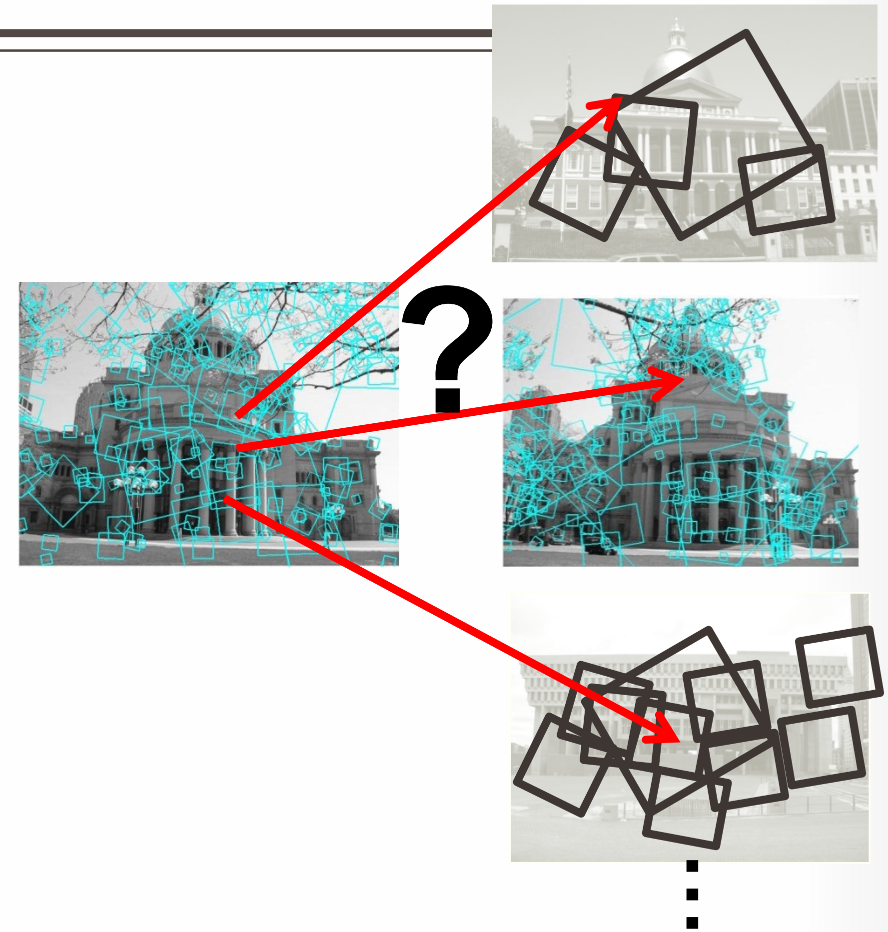
Multi-class classification results (30 training images per class)

	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

Multi-view matching



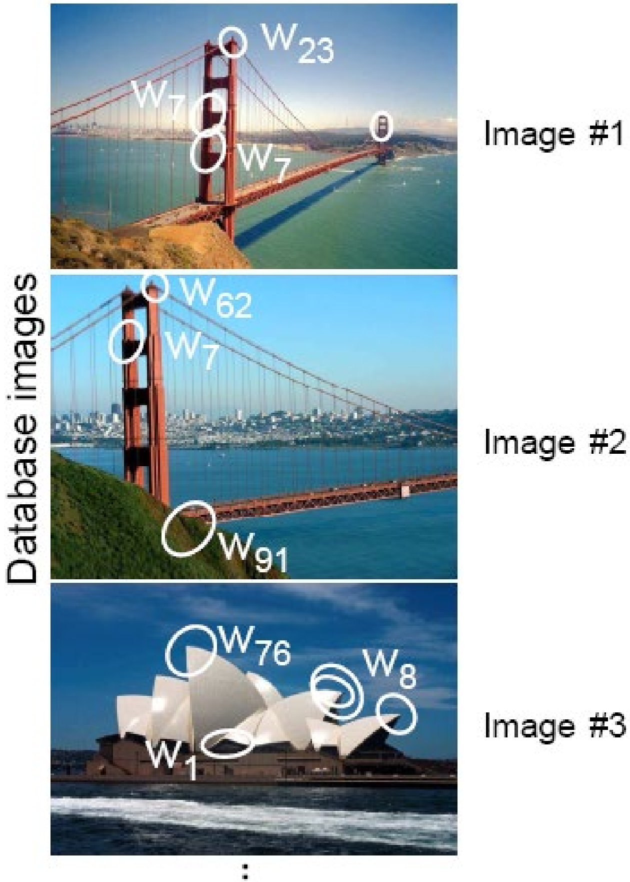
vs



Matching two given views for depth

Search for a matching view for recognition

Inverted file index



Word #	Image #
1	3
2 …	
7	1, 2
8	3
9	
10 …	
91	2
⋮	⋮

…

Database images are loaded into the index mapping words to image numbers

Inverted file index

- New query image is mapped to indices of database images that share a word.

New query image

Word #	Image #
1	3
2 ...	
7	1, 2
8	3
9	
10 ...	
91	2

...

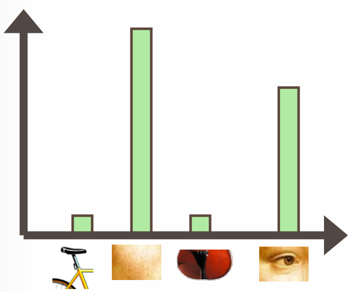
Inverted file index

- Key requirement for inverted file index to be efficient: sparsity
- If most pages/images contain most words then you're no better off than exhaustive search.
  - Exhaustive search would mean comparing the word distribution of a query versus every page.

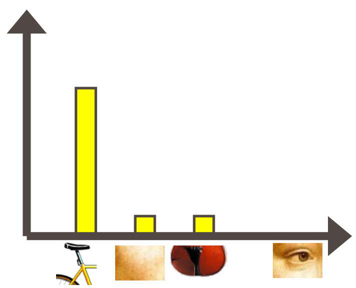
Comparing bags of words

- Rank frames by normalized scalar product between their (possibly weighted) occurrence counts---nearest neighbor search for similar images.

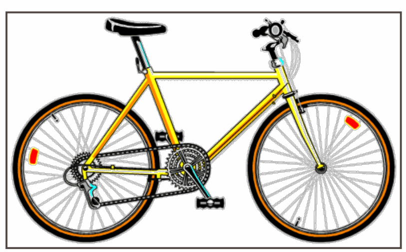
[1 8 1 4]



[5 1 1 0]



$\vec{d}_j$



$\vec{q}$

$$sim(d_j, q) = \frac{\langle d_j, q \rangle}{\|d_j\| \|q\|}$$

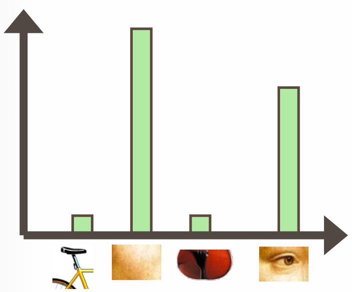
$$= \frac{\sum_{i=1}^{V} d_j(i) * q(i)}{\sqrt{\sum_{i=1}^{V} d_j(i)^2} * \sqrt{\sum_{i=1}^{V} q(i}}$$

for vocabulary of *V* words

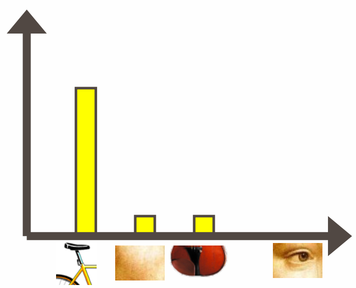
Comparing bags of words

- Other common histogram comparisons:

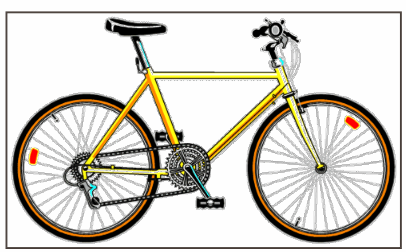
[1 8 1 4]



[5 1 1 0]



$\vec{d}_j$



$\vec{q}$

Histogram intersection

$$\frac{\sum_{j=1}^{n} min(I_j, M_j)}{\sum_{j=1}^{n} M_j}$$

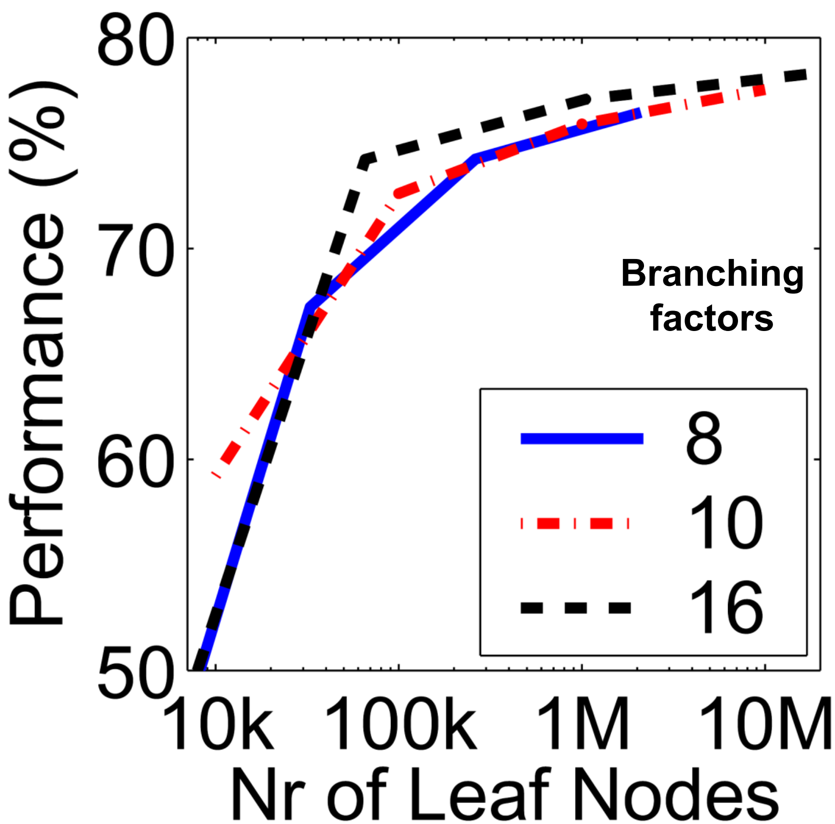
Chi squared

$$\sum_{i=1}^{n} \frac{(x_i - y_i)^2}{(x_i + y_i)}$$

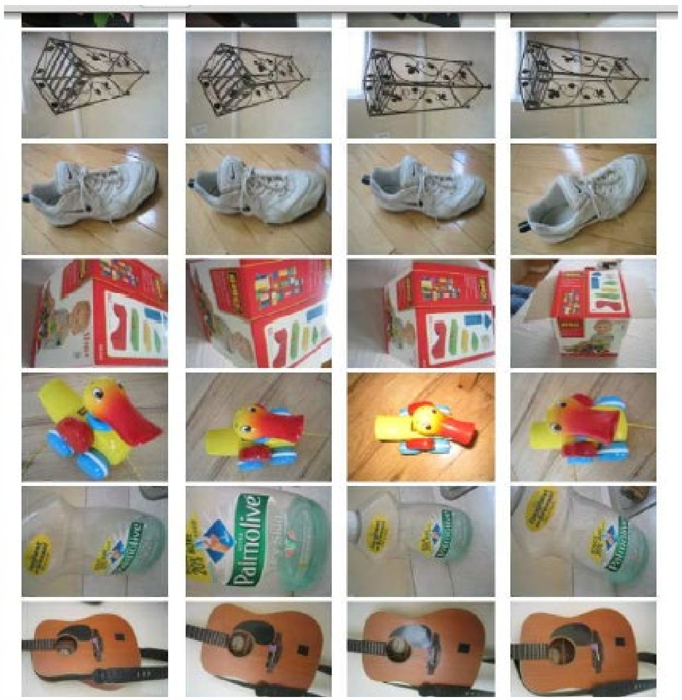
Instance recognition: remaining issues

- How to summarize the content of an entire image? And gauge overall similarity?
- How large should the vocabulary be? How to perform quantization efficiently?
- Is having the same set of visual words enough to identify the object/scene? How to verify spatial agreement?
- How to score the retrieval results?

Vocabulary size



Results for recognition task with 6347 images

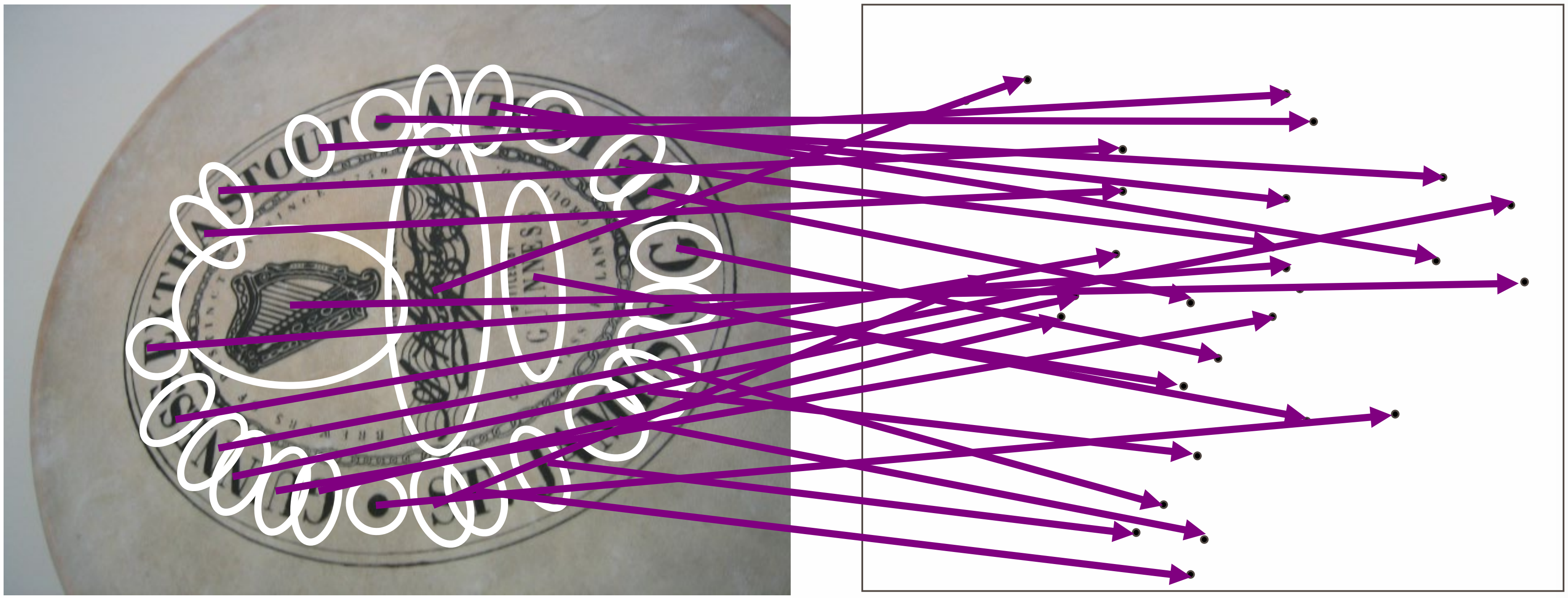


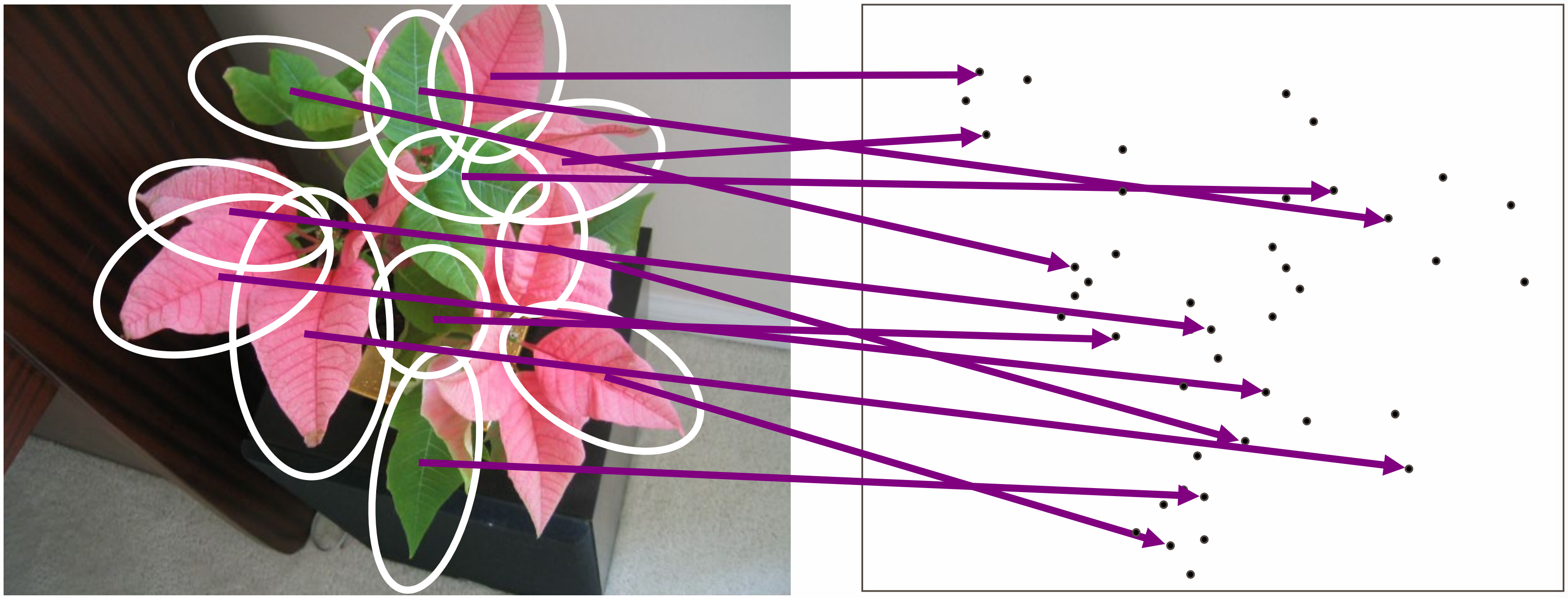
Influence on performance, sparsity

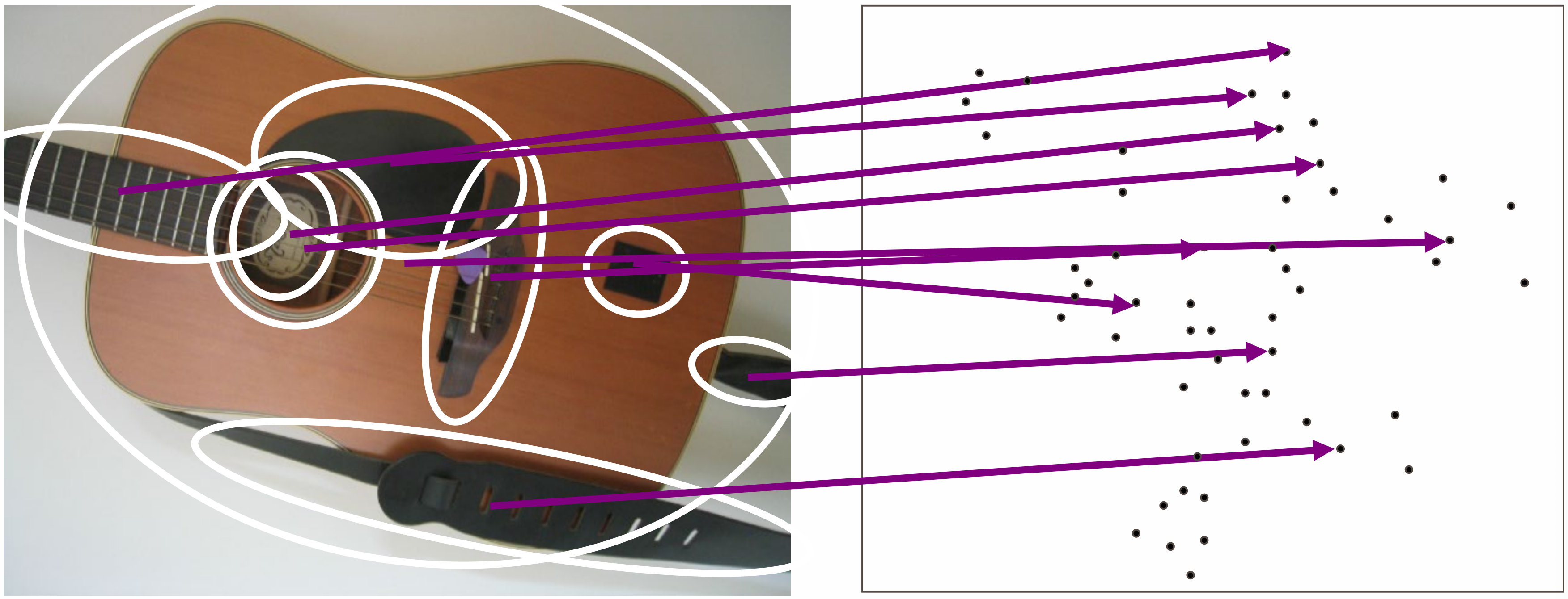
Nister & Stewenius, CVPR 2006

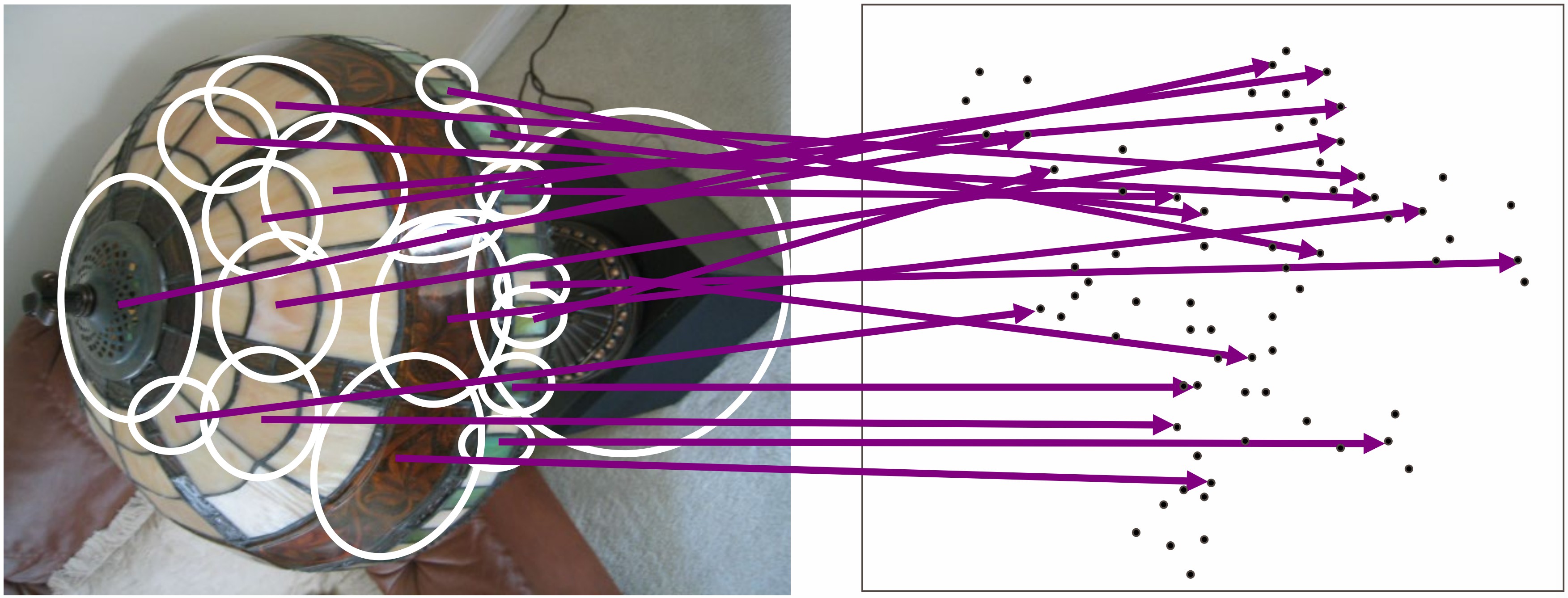
BOOSTED IMAGE MATCHING

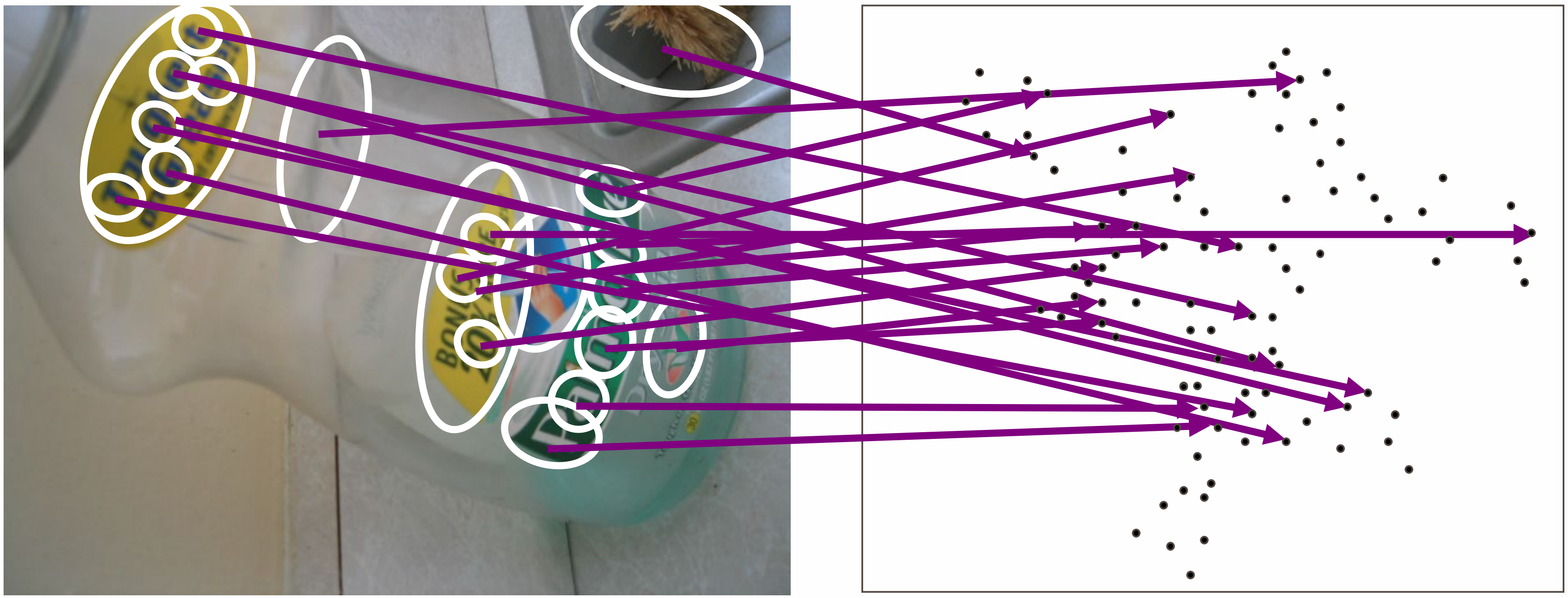
Following slides by David Nister (CVPR 2006)

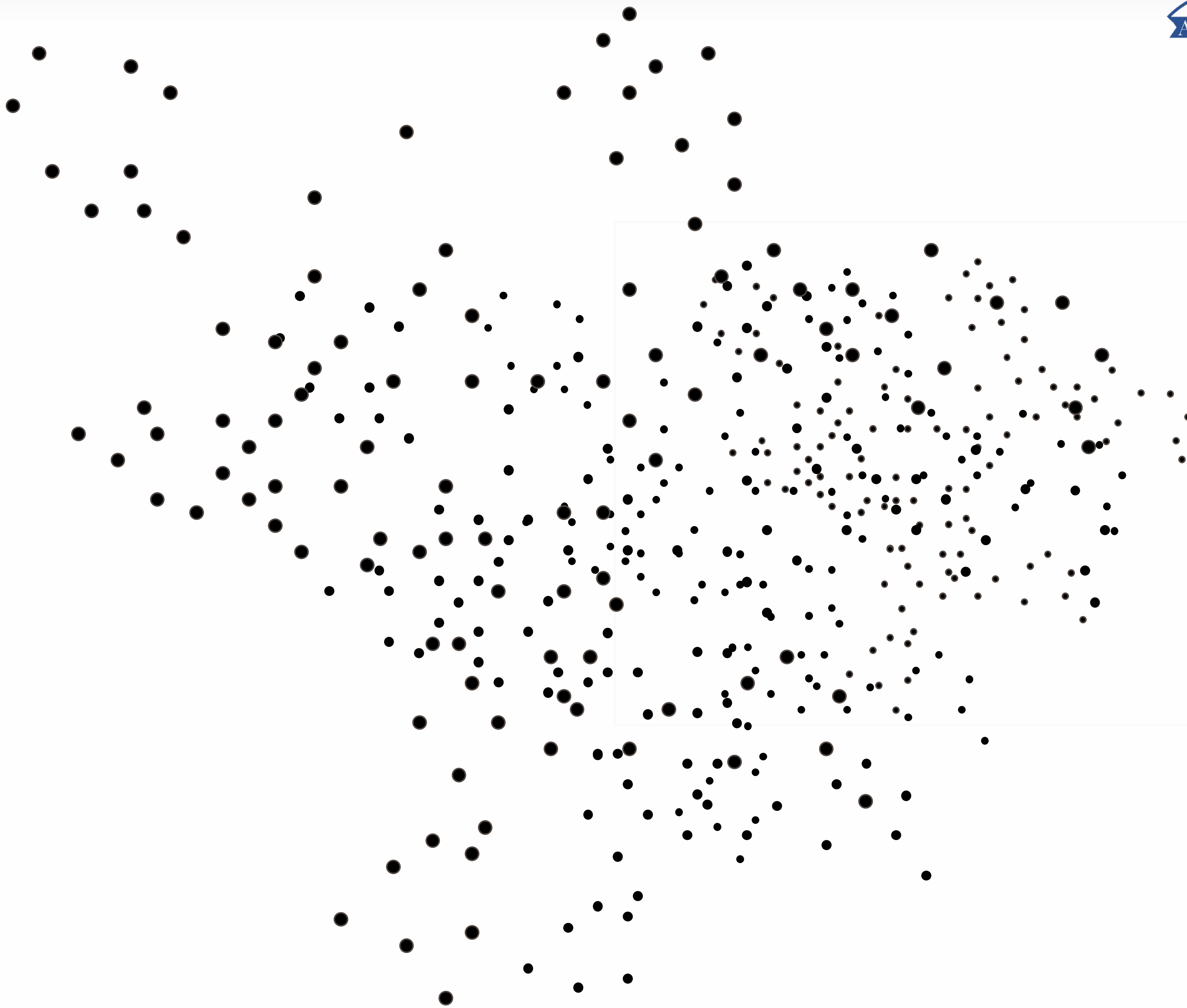


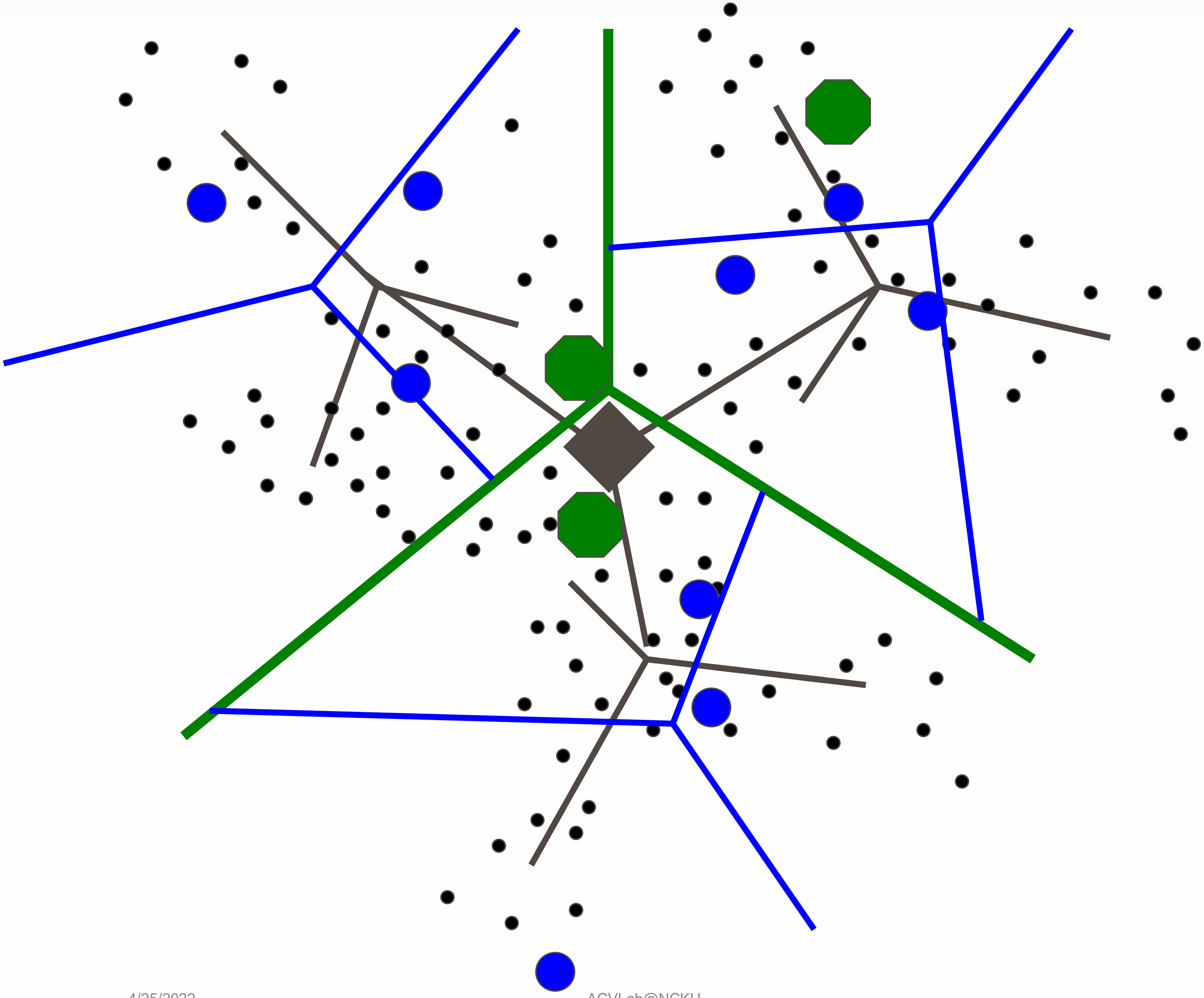


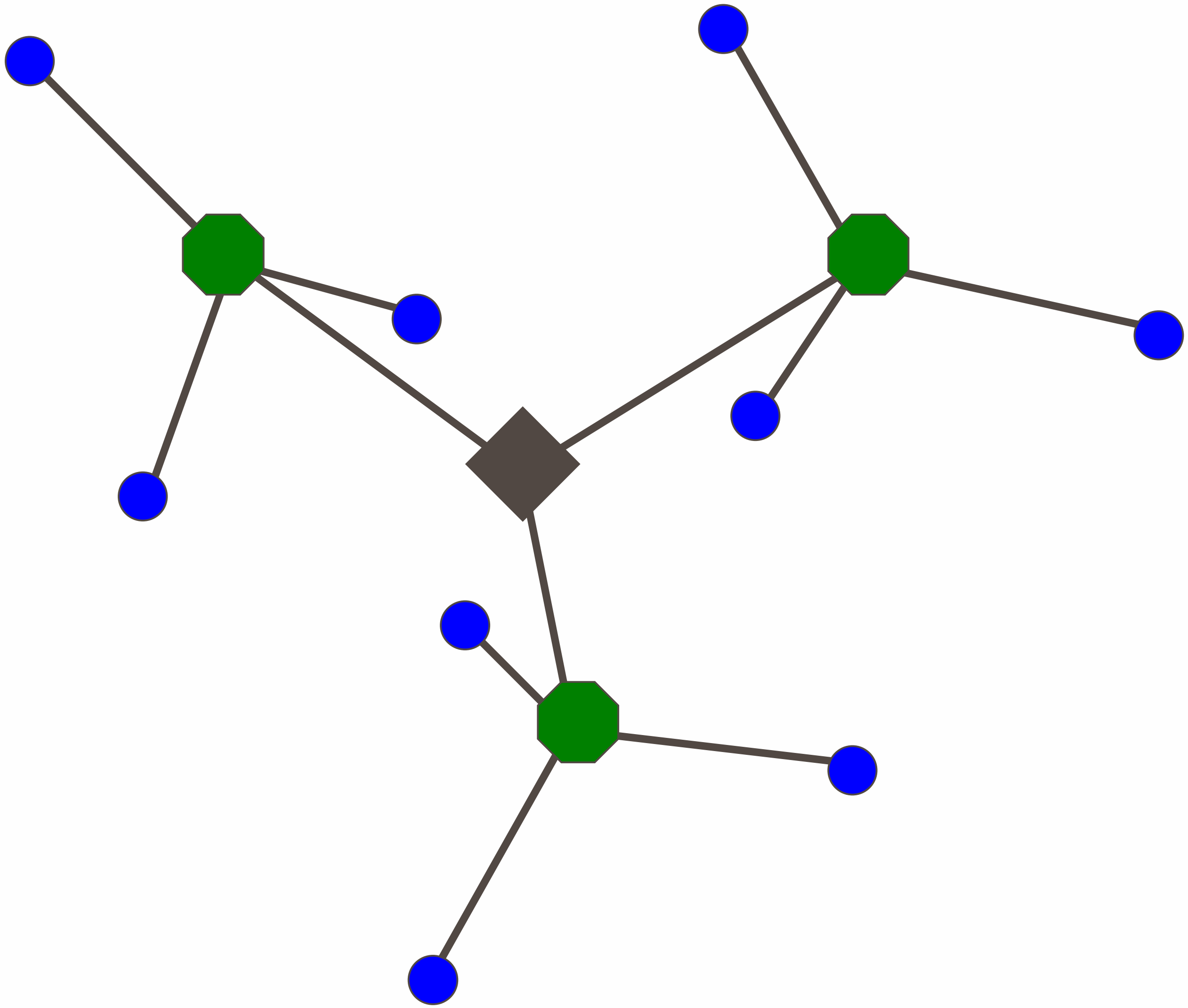


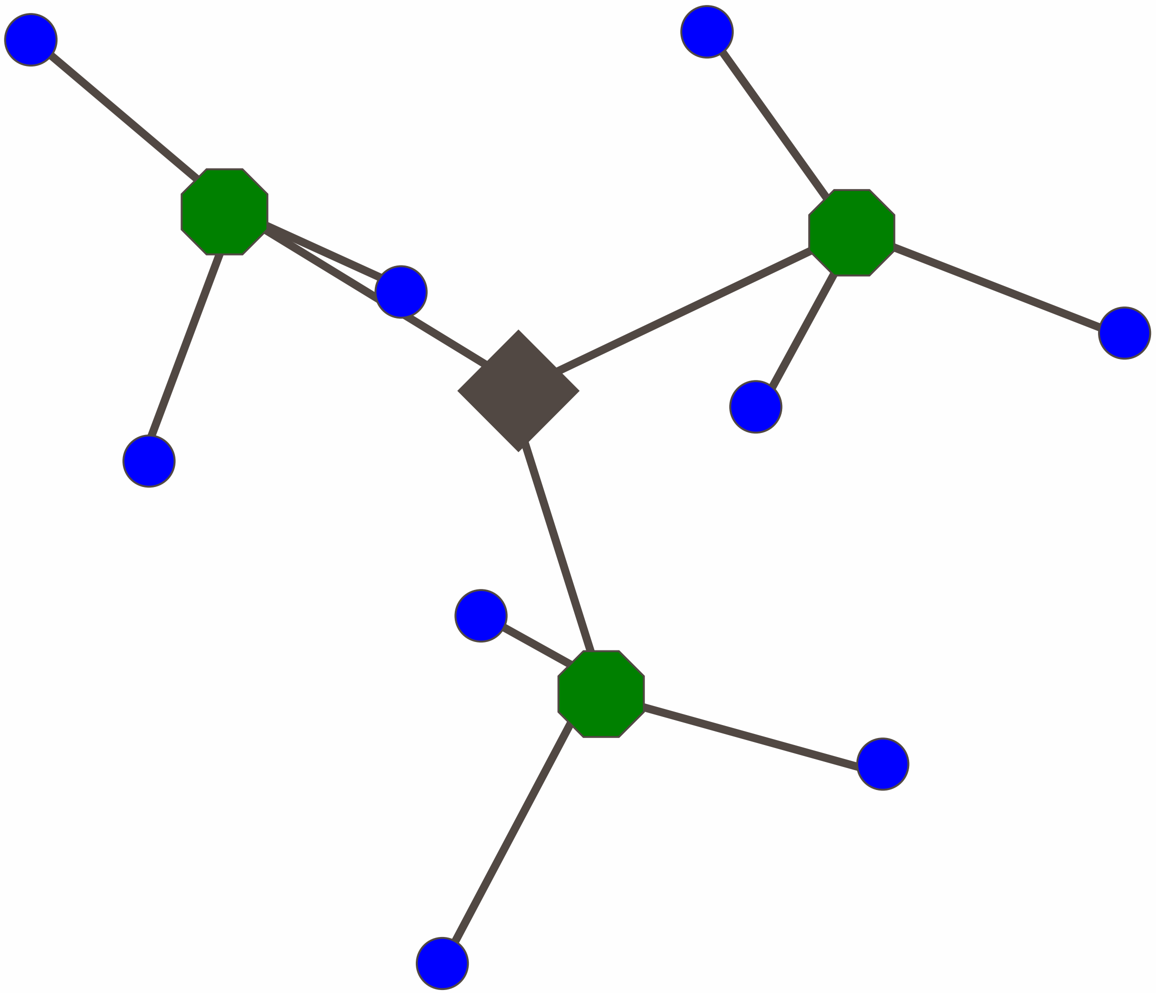


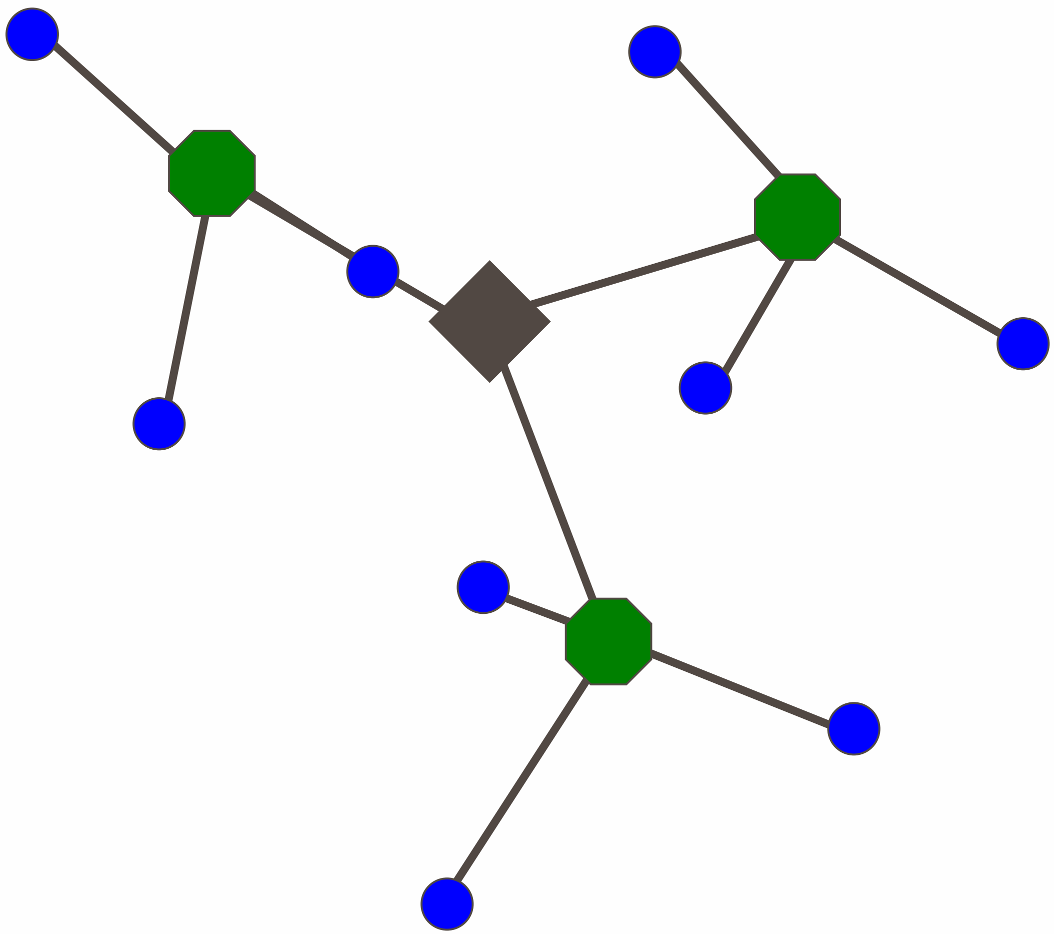


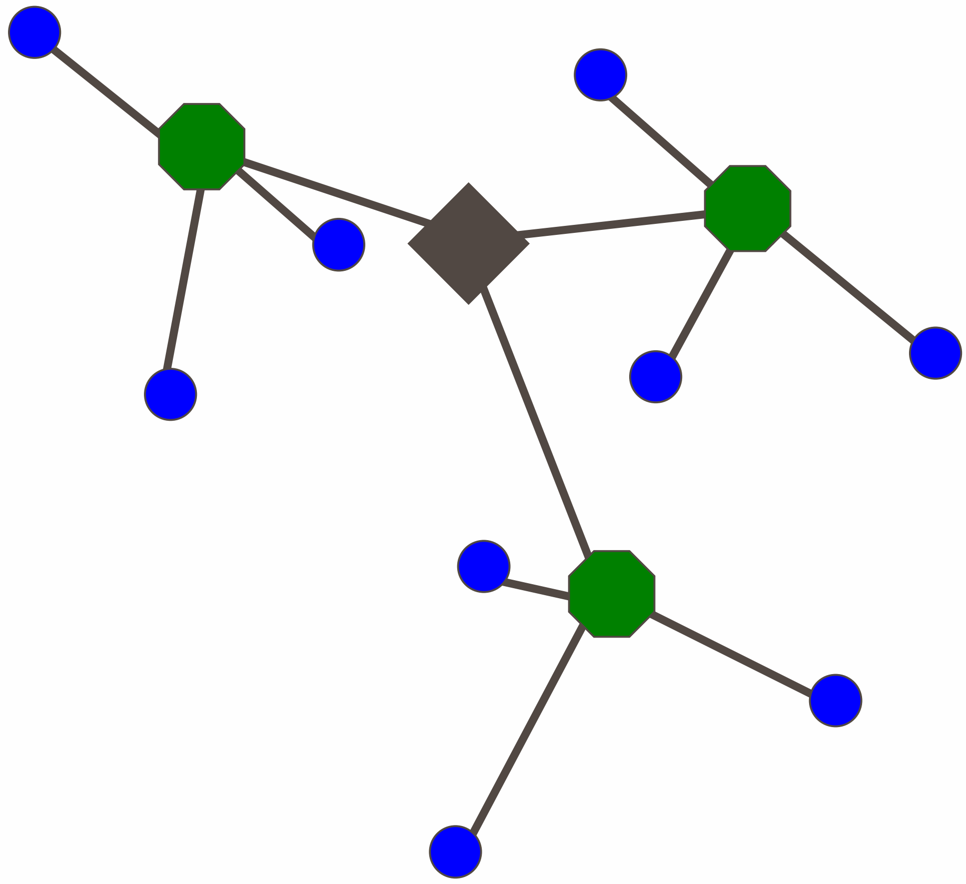


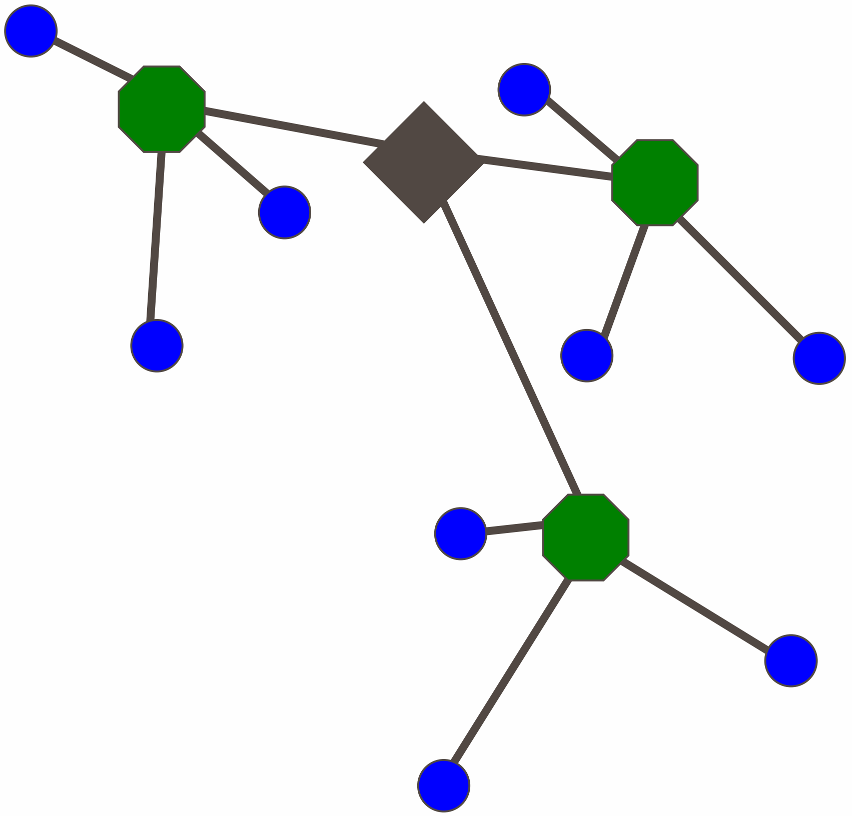


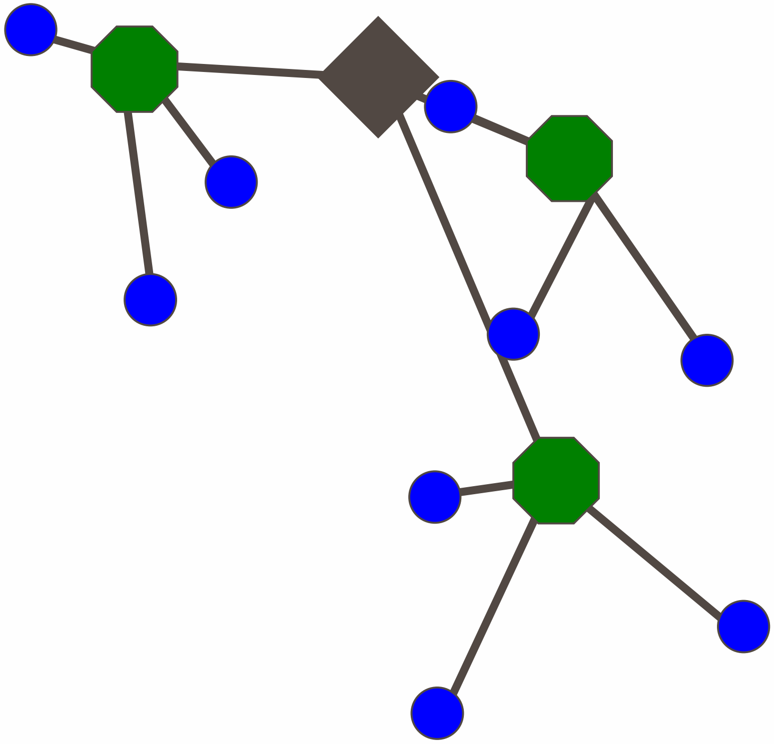


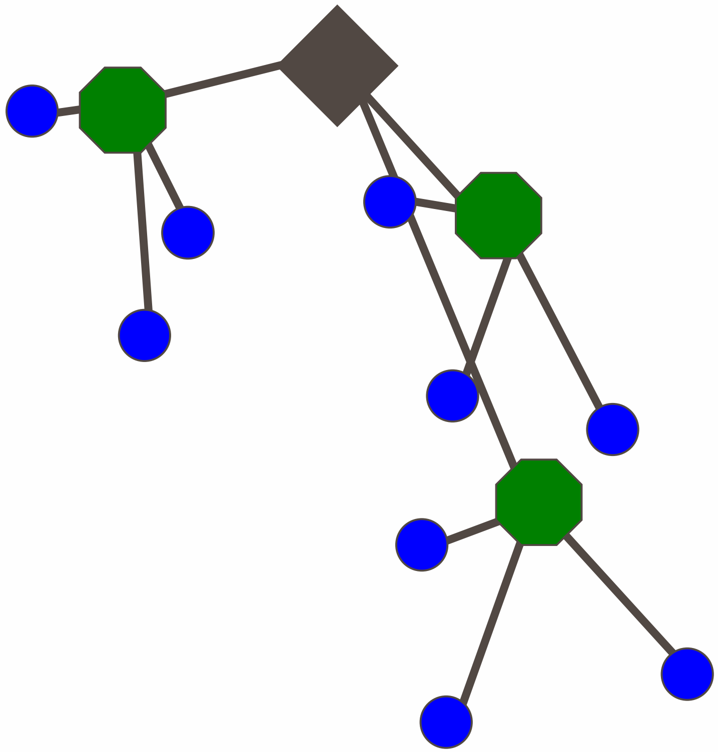


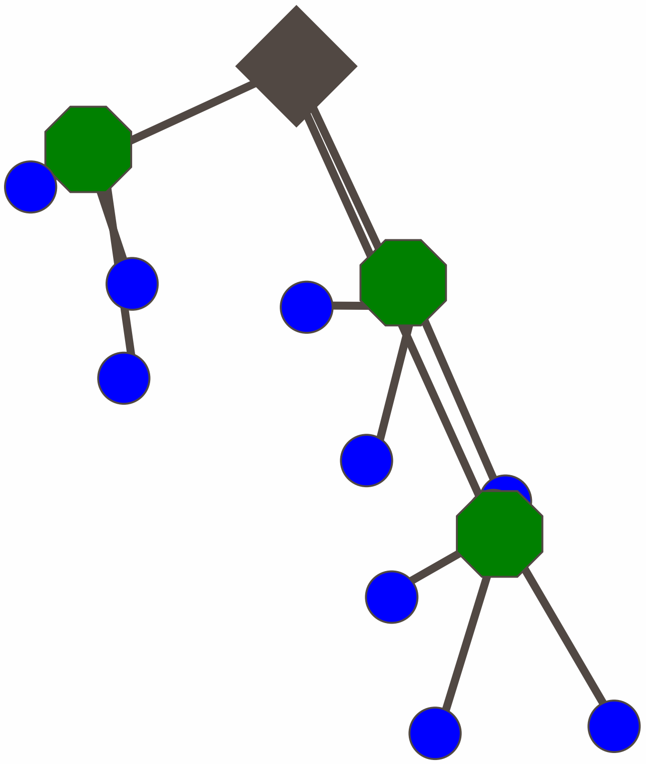


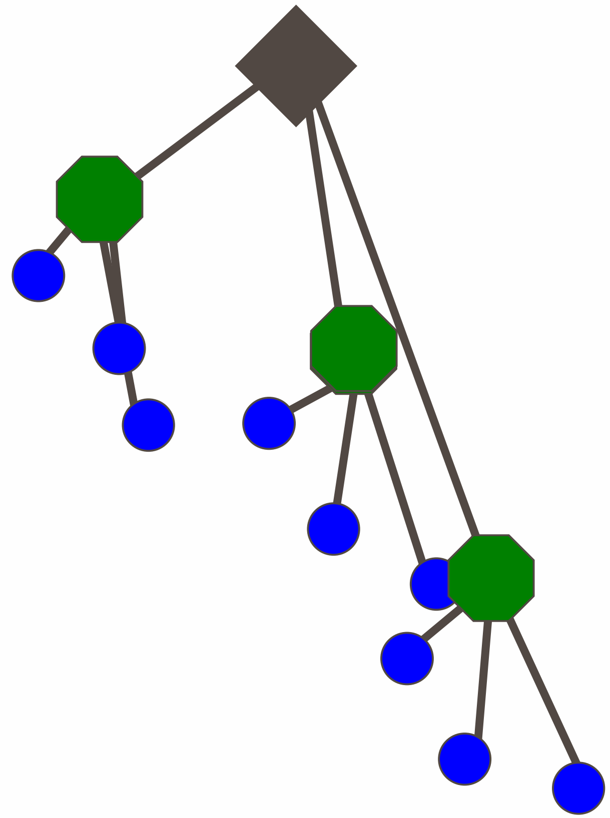


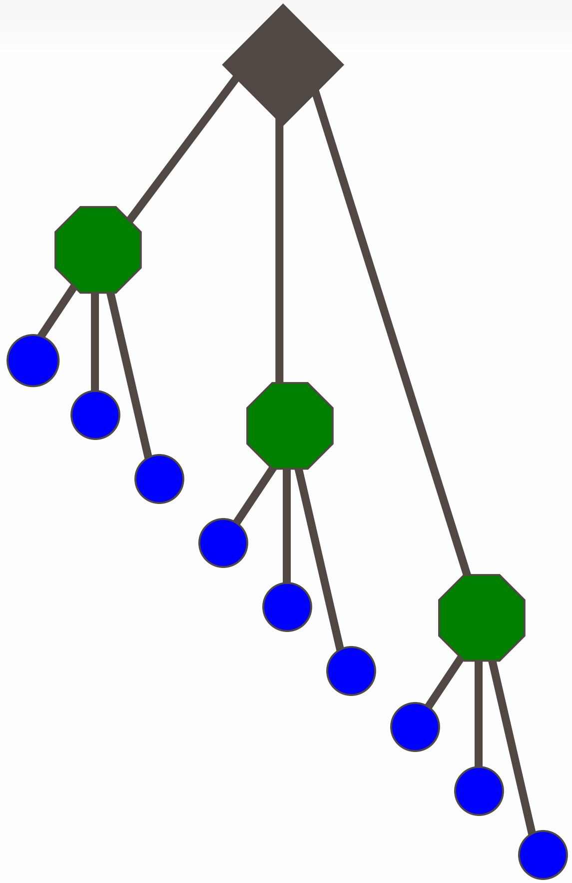


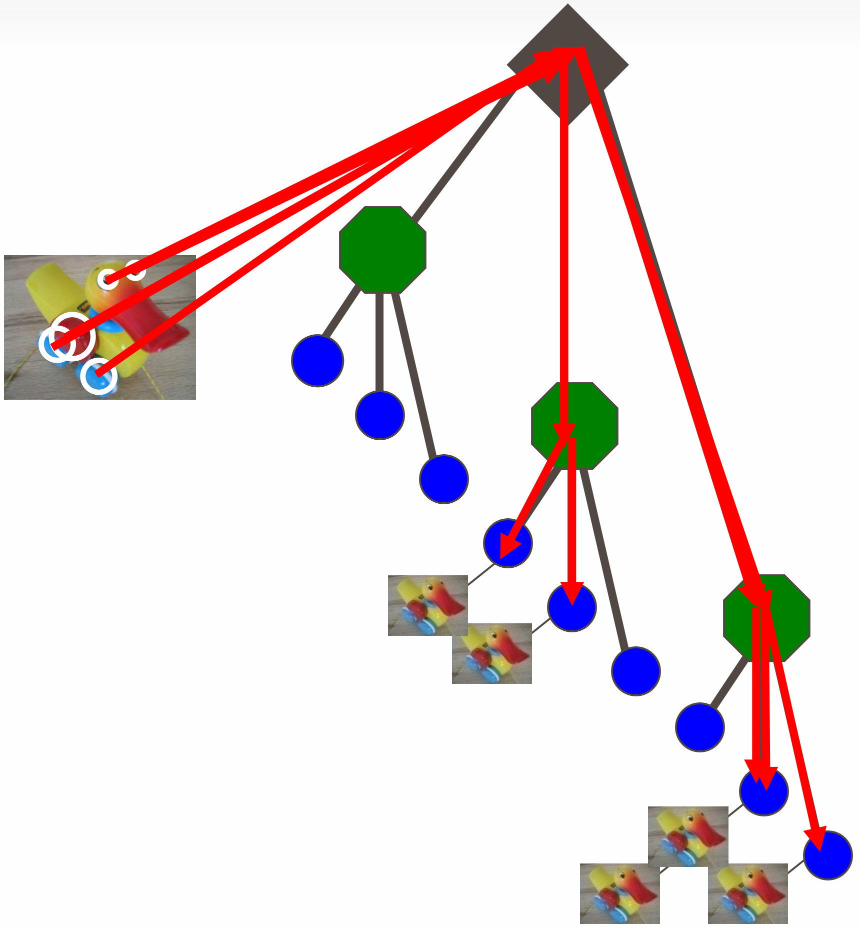


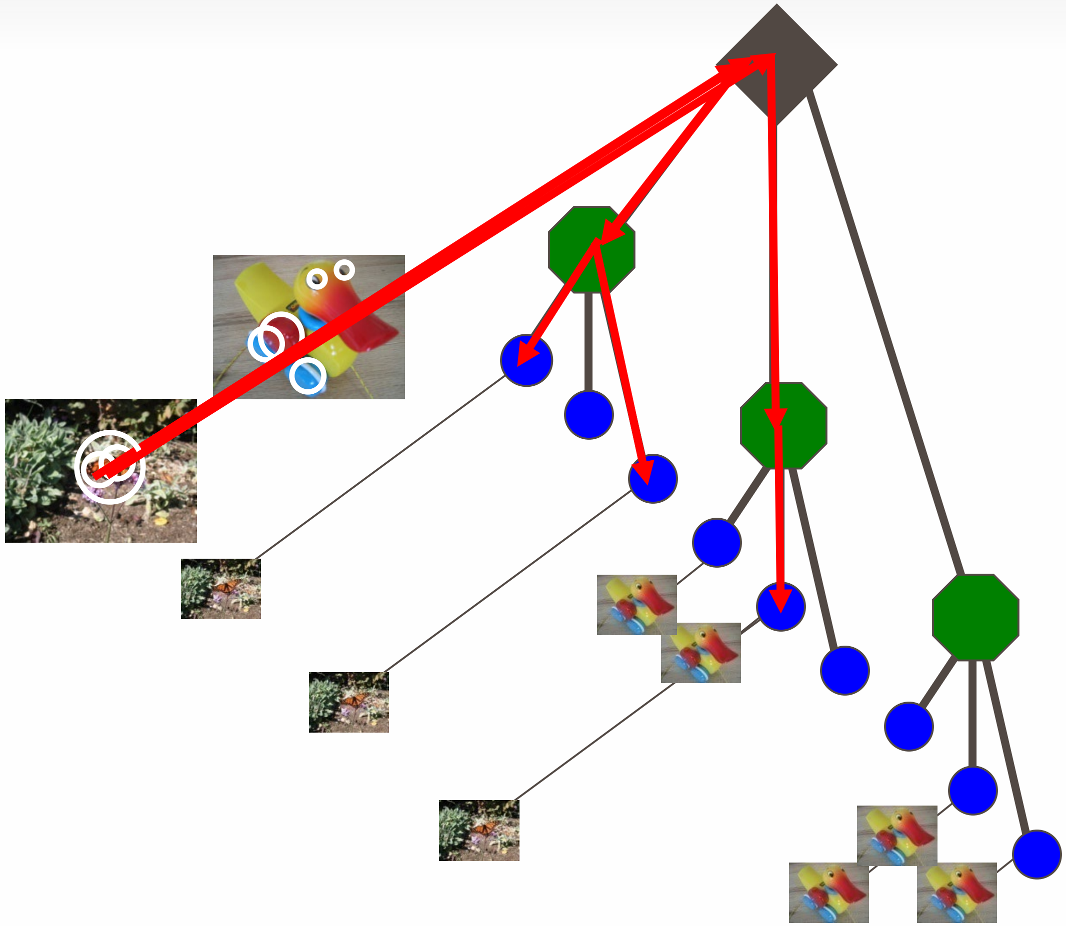


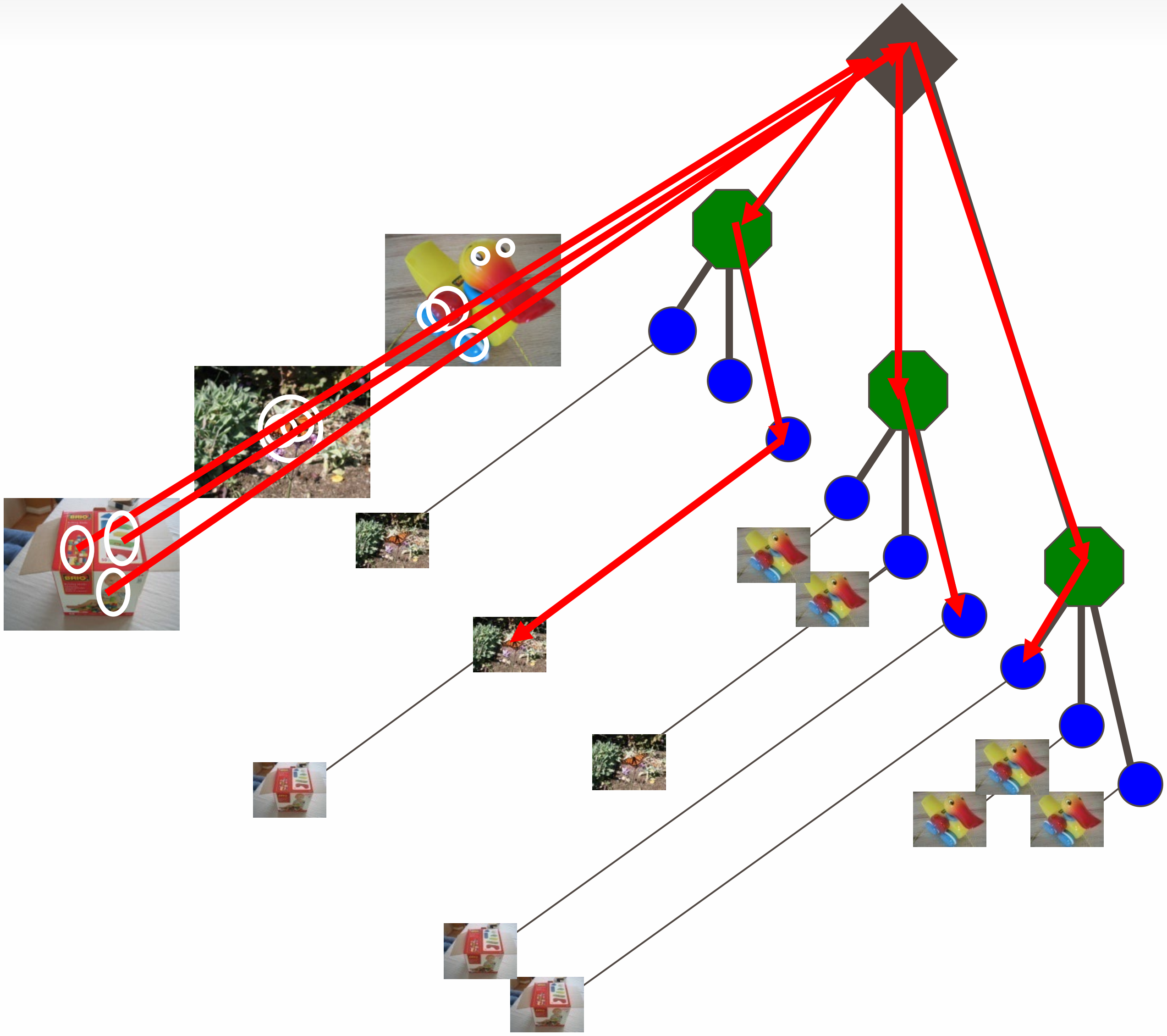


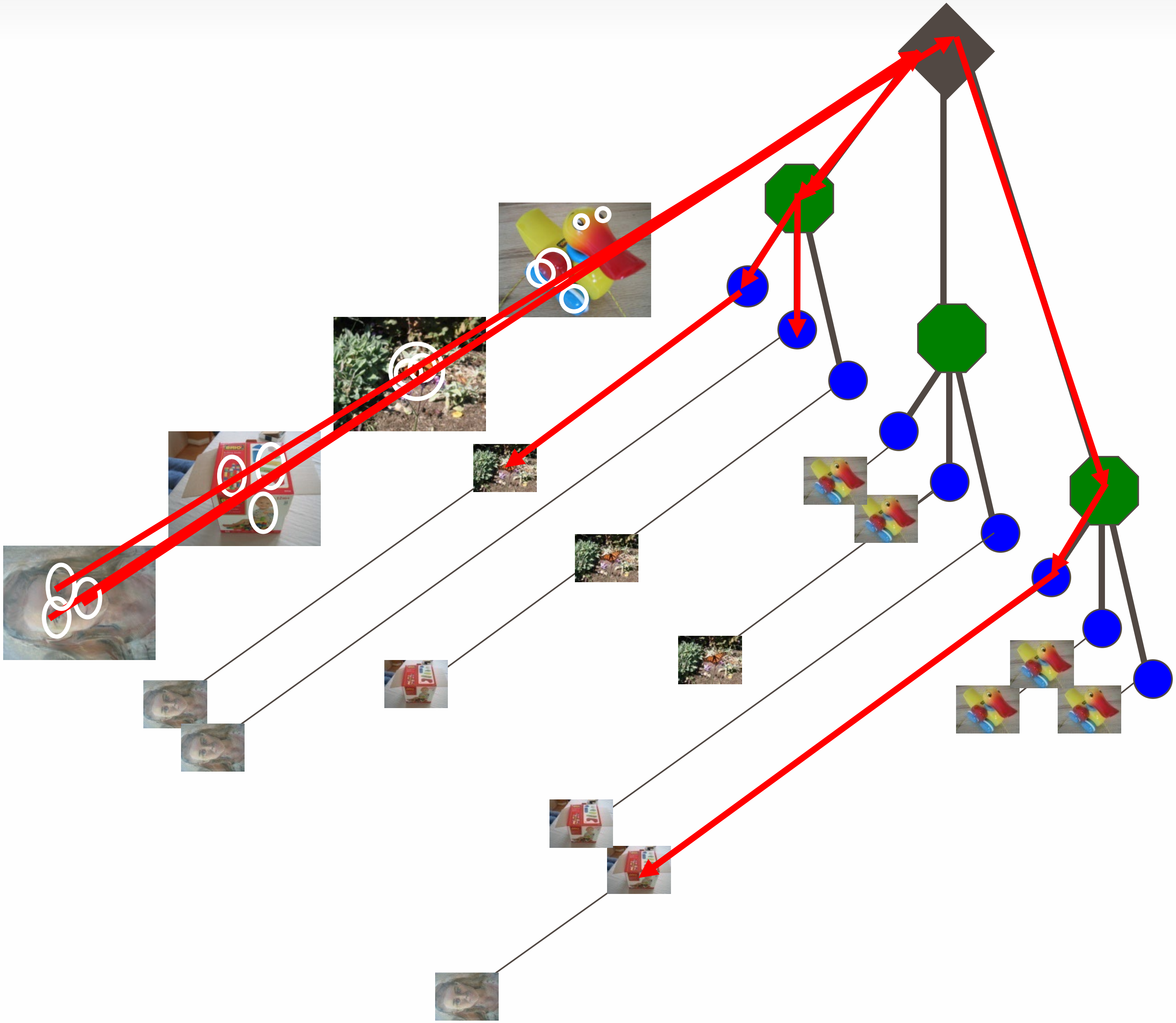


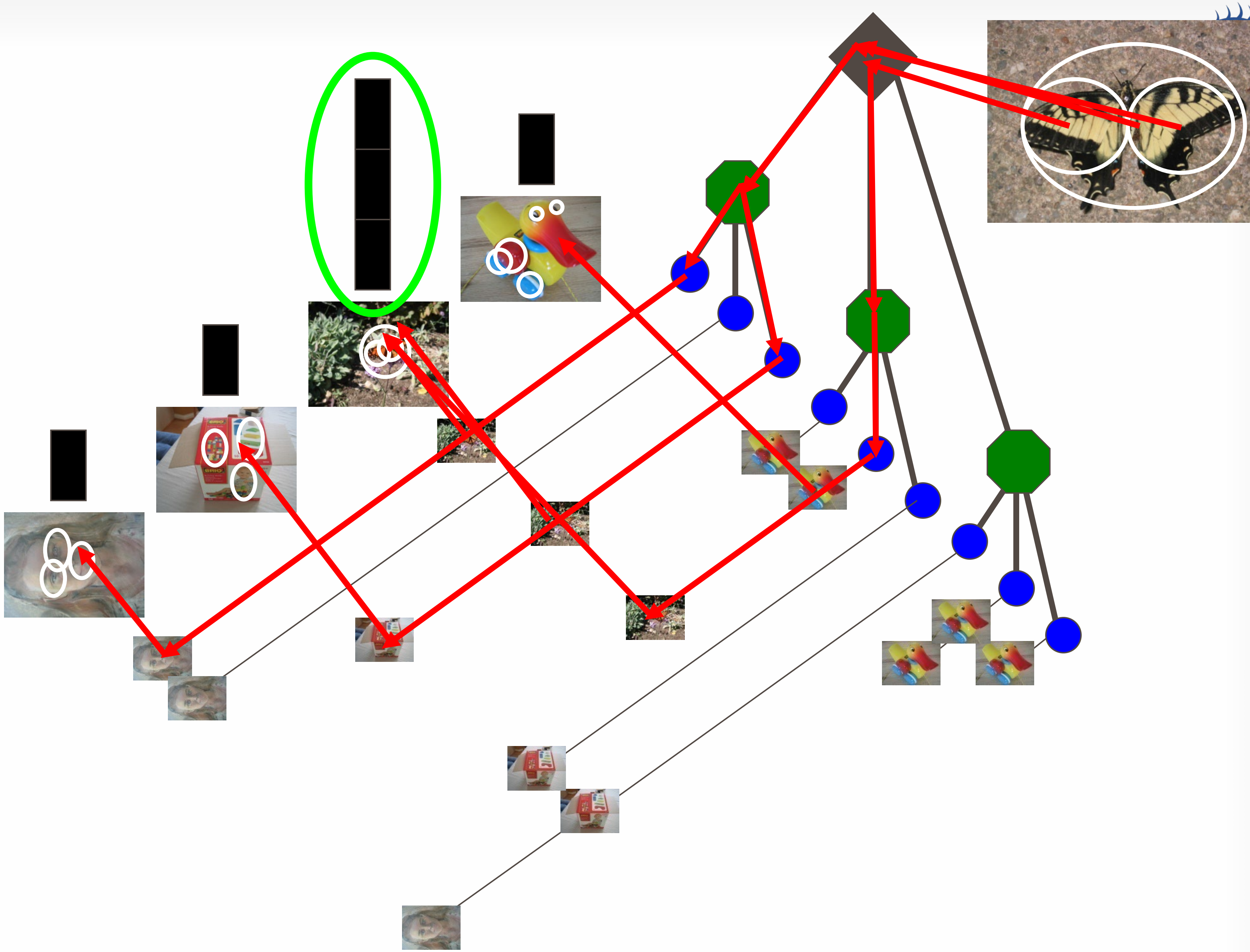












Vocabulary trees: complexity

Number of words given tree parameters: branching factor and number of levels

branching_factor^number_of_levels

Word assignment cost vs. flat vocabulary

O(k) for flat

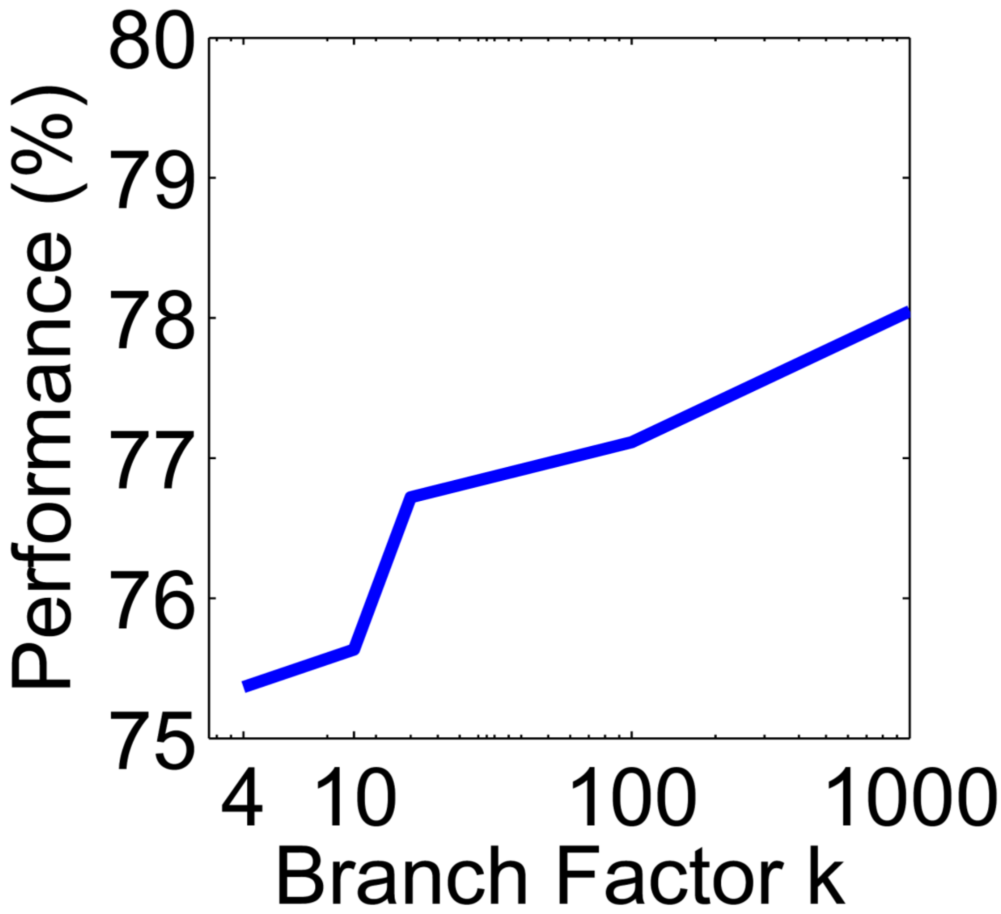
$O(\log_{branching\_factor}(k) * branching\_factor)$

Is this like a kd-tree?

Yes, but with better partitioning and defeatist search.

This hierarchical data structure is lossy – you might not find your true nearest cluster.

Higher branch factor works better (but slower)



Visual words/bags of words

- \+ flexible to geometry / deformations / viewpoint
- \+ compact summary of image content
- \+ provides fixed dimensional vector representation for sets
- \+ very good results in practice

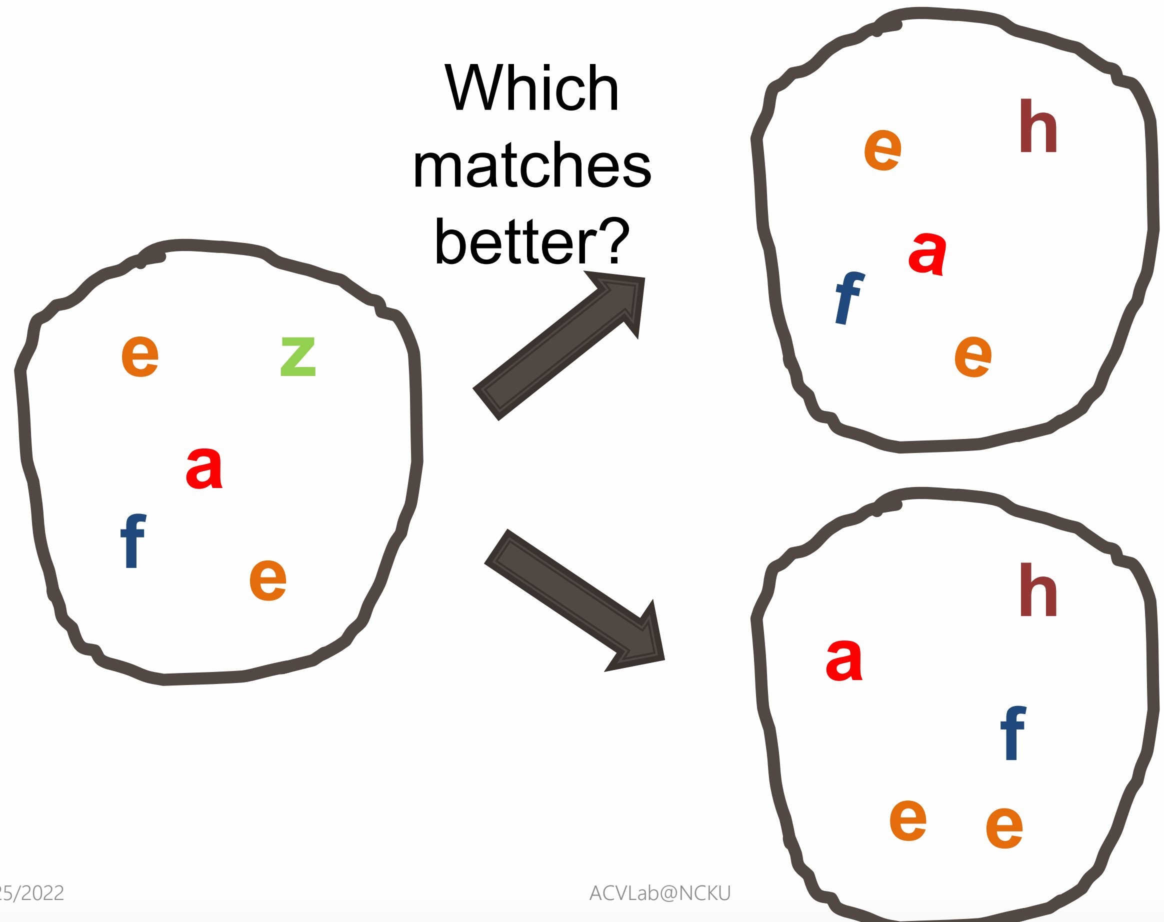
- background and foreground mixed when bag covers whole image
- optimal vocabulary formation remains unclear
- basic model ignores geometry – must verify afterwards, or encode via features

Instance recognition: remaining issues

- How to summarize the content of an entire image? And gauge overall similarity?
- How large should the vocabulary be? How to perform quantization efficiently?
- Is having the same set of visual words enough to identify the object/scene? How to verify spatial agreement?
- How to score the retrieval results?

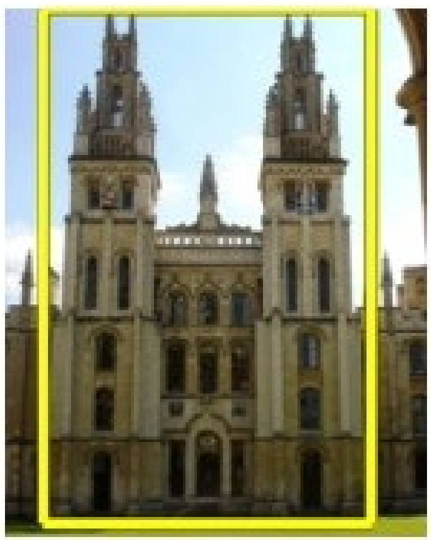
Can we be more accurate?

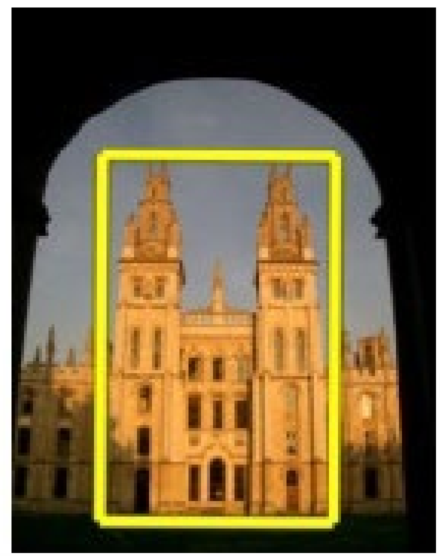
- So far, we treat each image as containing a “bag of words”, with no spatial information



Can we be more accurate?

- So far, we treat each image as containing a “bag of words”, with no spatial information





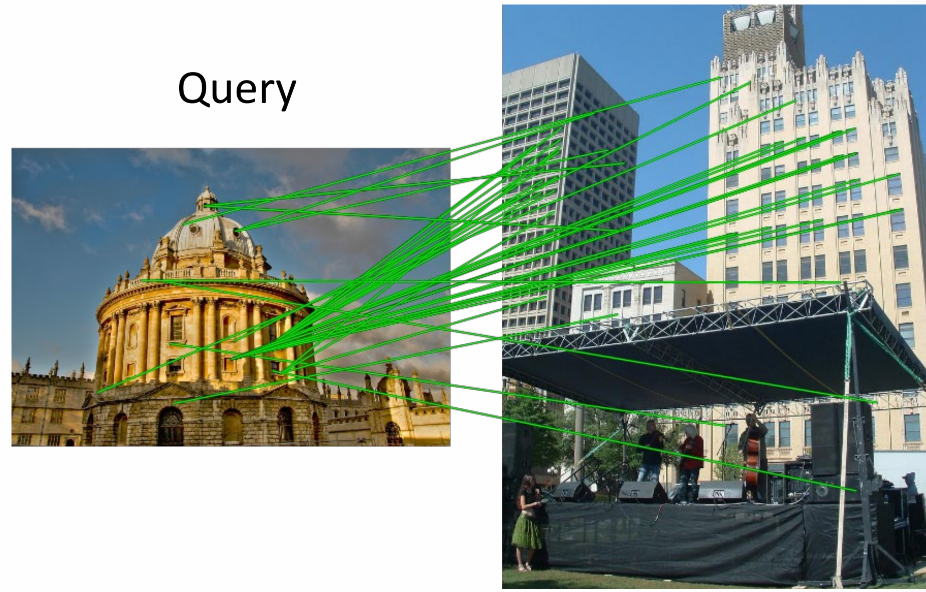
Real objects have consistent geometry

Spatial Verification

Query

DB image with high BoW similarity

Query



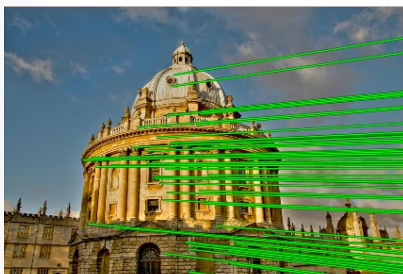
DB image with high BoW similarity

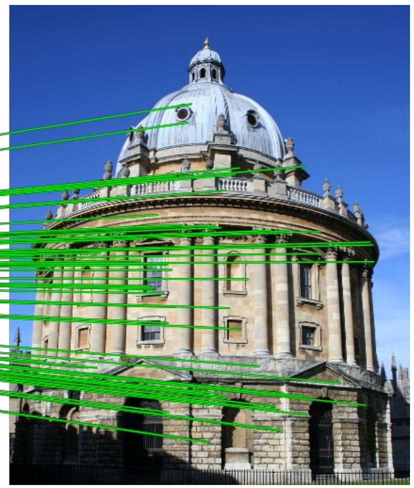
Both image pairs have many visual words in common.

Slide credit: Ondrej Chum

Spatial Verification

Query





DB image with high BoW similarity

Query

DB image with high BoW similarity

Only some of the matches are mutually consistent

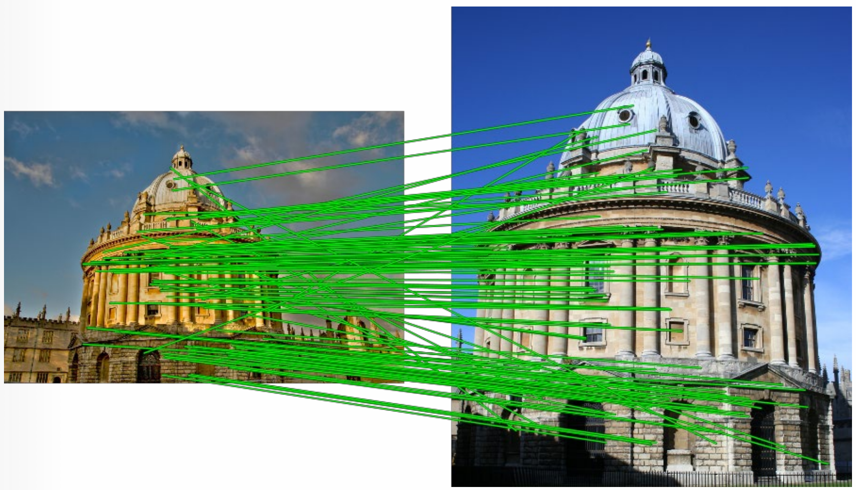
Slide credit: Ondrej Chum

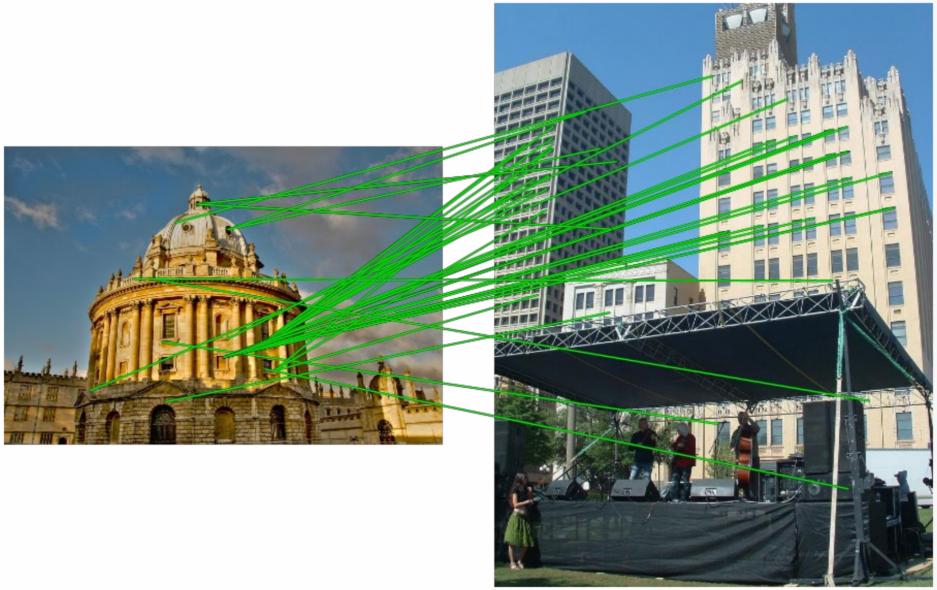
Spatial Verification: two basic strategies

- RANSAC
  - Typically sort by BoW similarity as initial filter
  - Verify by checking support (inliers) for possible transformations
    - e.g., “success” if find a transformation with > N inlier correspondences

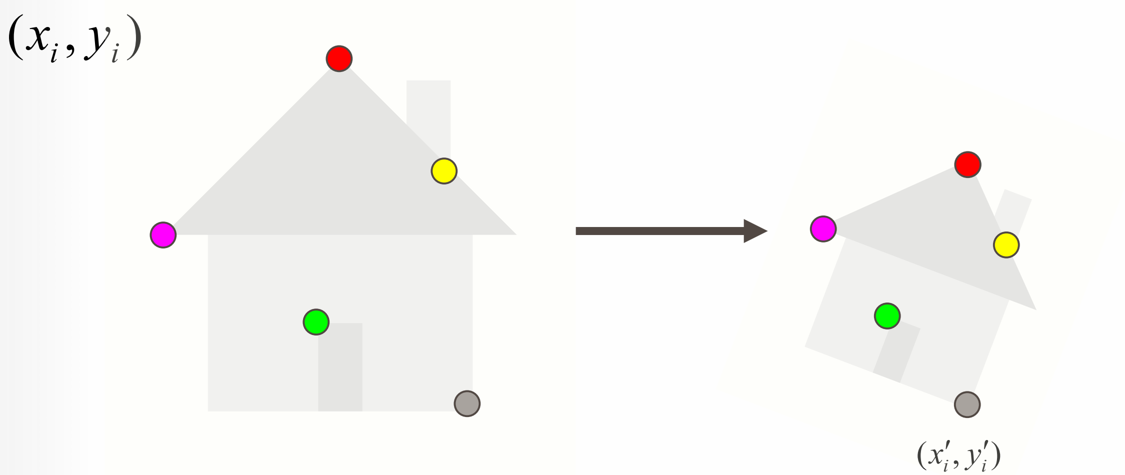
- Generalized Hough Transform
  - Let each matched feature cast a vote on location, scale, orientation of the model object
  - Verify parameters with enough votes

RANSAC verification





Recall: Fitting an affine transformation

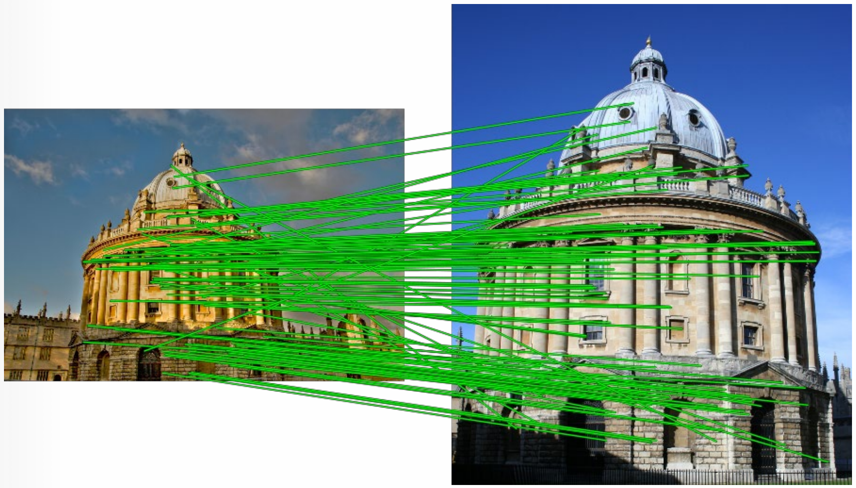


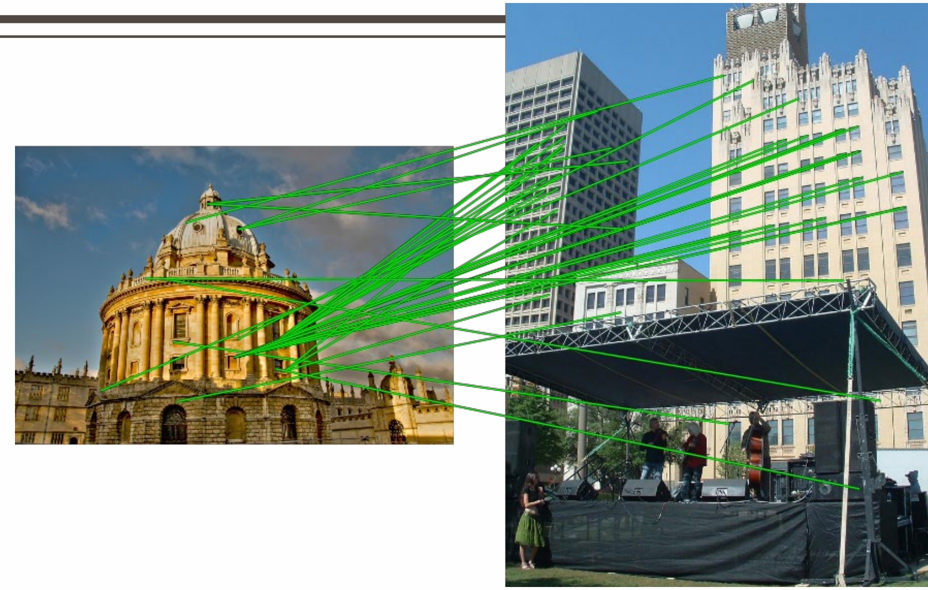
Approximates viewpoint changes for roughly planar objects and roughly orthographic cameras.

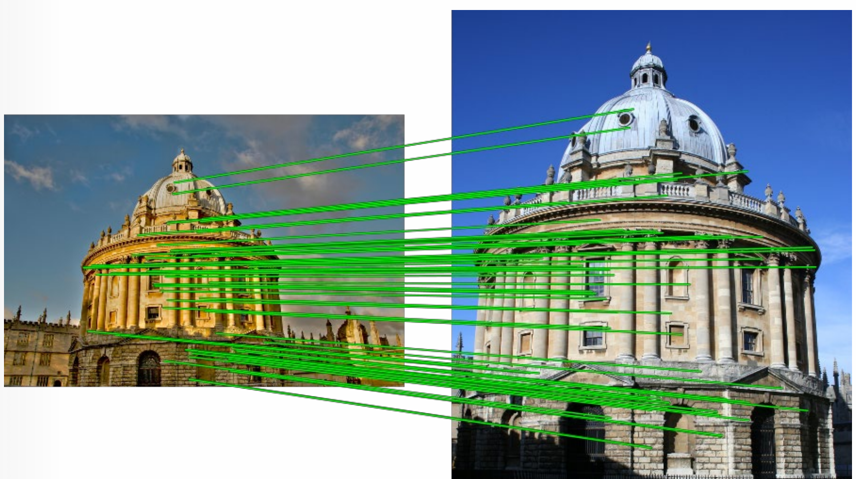
$$\begin{bmatrix} x_i' \\ y_i' \end{bmatrix} = \begin{bmatrix} m_1 & m_2 \\ m_3 & m_4 \end{bmatrix} \begin{bmatrix} x_i \\ y_i \end{bmatrix} + \begin{bmatrix} t_1 \\ t_2 \end{bmatrix}$$

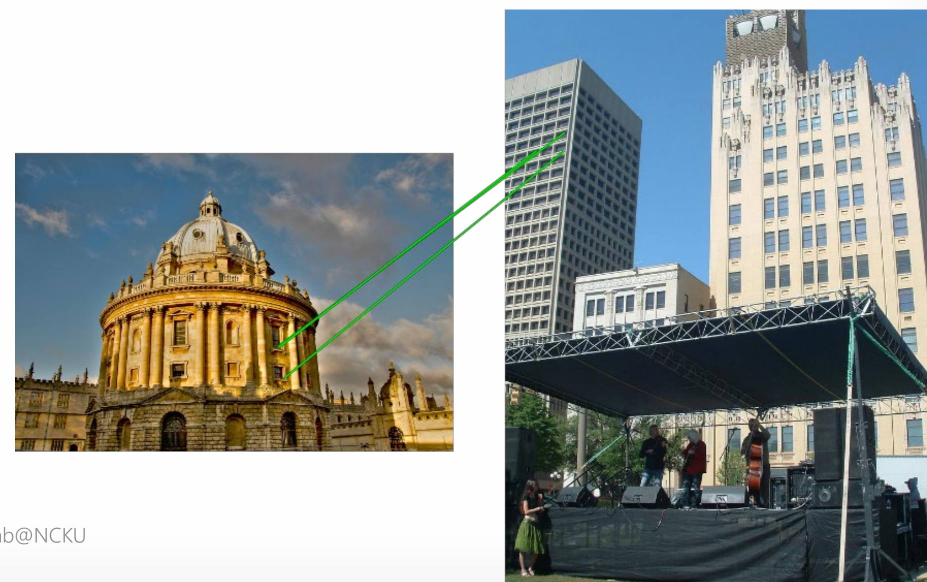
$$\begin{bmatrix} & & \cdots & & & \\ x_i & y_i & 0 & 0 & 1 & 0 \\ 0 & 0 & x_i & y_i & 0 & 1 \\ & & \cdots & & & \end{bmatrix} \begin{bmatrix} m_1 \\ m_2 \\ m_3 \\ m_4 \\ t_1 \\ t_2 \end{bmatrix} = \begin{bmatrix} \cdots \\ x_i' \\ y_i' \\ \cdots \end{bmatrix}$$

RANSAC verification









Instance recognition: remaining issues

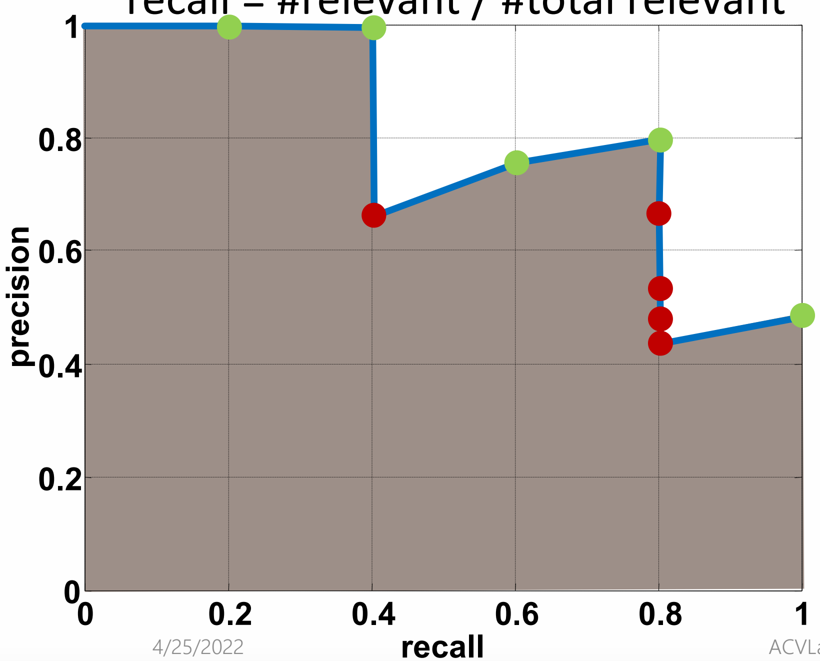
- How to summarize the content of an entire image? And gauge overall similarity?
- How large should the vocabulary be? How to perform quantization efficiently?
- Is having the same set of visual words enough to identify the object/scene? How to verify spatial agreement?
- How to score the retrieval results?

Scoring retrieval quality

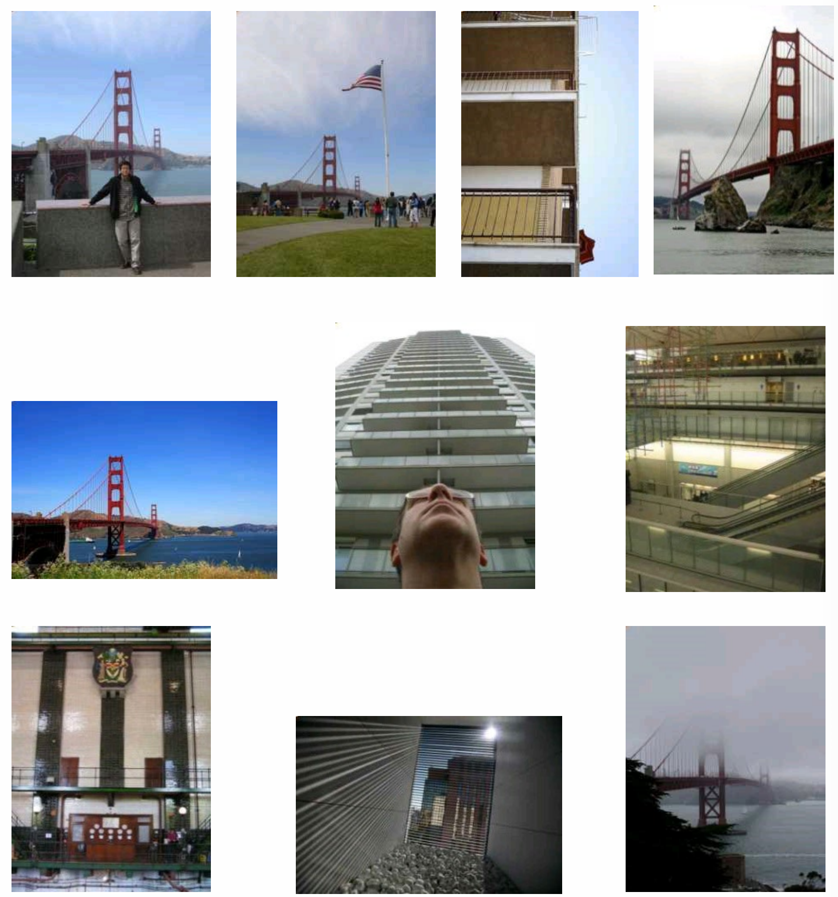
Database size: 10 images
Relevant (total): 5 images

Query

precision = #relevant / #returned
recall = #relevant / #total relevant



Results (ordered):



Slide credit: Ondrej Chum

Query Expansion

Chum, Philbin, Sivic, Isard, Zisserman: Total Recall..., ICCV 2007

Slide credit: Ondrej Chum

Recognition via alignment

- Pros:
  - Effective when we are able to find reliable features within clutter
  - Great results for matching specific instances
- Cons:
  - Scaling with number of models
  - Spatial verification as post-processing – not seamless, expensive for large-scale problems
  - Not suited for category recognition.

Summary

- Matching local invariant features
  - Useful not only to provide matches for multi-view geometry, but also to find objects and scenes.
- Bag of words representation: quantize feature space to make discrete set of visual words
  - Summarize image by distribution of words
  - Index individual words
- Inverted index: pre-compute index to enable faster search at query time
- Recognition of instances via alignment: matching local features followed by spatial verification
  - Robust fitting : RANSAC, GHT

INTRODUCTION TO IMAGE PROCESSING

Chih-Chung Hsu (許志仲)
Assistant Professor
ACVLab, Institute of Data Science
National Cheng Kung University
https://cchsu.info

Applications- Why/ Who

- Useful in a variety of fields from science to design
- Adjust images and run analysis
  - Add filters and text
  - Make images easier to view
  - Count cells or items in an image
  - Track movement
  - Facial Recognition
  - Create movies/gifs

Basics- What is an Image?

- Image Type
- Vector
  - Made of independent and editable lines/shapes
- Pixel
  - Also called bitmap or raster
  - Made of uniform grid of colored dots (pixels)

Format

Extension	Colour	Compression	Common Uses
JPG, JPEG	24-bit	Lossy	Photos, web pics
GIF	8-bit	Lossless	Web graphics – buttons, icons, etc
PNG	up to 24-bit	Lossless	Web – replacement for GIF
TIF, TIFF	24-bit	Lossless	Professional Photos etc

https://www.slideshare.net/bobwatson/image-file-formats

- Compression
  - JPEG ➔ Common
  - Loss vs Lossless

123

Opening an image in Python

- Need Numpy and Scipy:
- Numpy: basic array manipulation
  - Pixel images are stored as arrays
    - Each pixel has (x,y,rgb)
    - Means you can loop through images and has array functions
- Scipy: dedicated to image processing

124

Opening an image in Python

Here are the imports for this first section:

```python
import os
import numpy as np
from scipy import ndimage, misc
import matplotlib.pyplot as plt
from PIL import Image, ImageEnhance
```

This is how I import an image using numpy and PIL.
Make sure you have navigated to the correct file folder.

```python
##Open and view an image ########################
im = misc.imread('cricket.jpg')
type(im) #should show as np.array
im.shape, im.dtype #shows current shape and type
plt.imshow(im) #view image

imPIL = Image.open('cricket.jpg')
imPIL.show()
np.array(imPIL).shape
```

Image Properties

Resolution

- Size of each pixel expressed as number of pixels per unit
  - Dots per inch (DPI)
  - Pixels per inch (PPI)
- Dimensions
- Number of pixels along X and Y axes
- Can be:
  - Pixels (800 X 600 pixels)
  - Physical (89 mm X 66 mm)

Color

- Models:
  - RGB
  - CMYK
- Format:
  - RGB Values
  - Hex Values

Code

- The following sections reference python code
  - The title is a description of the section
    - Information about the images needed is in notes
    - Most of it works with your image of choice

Basics of Numpy/Scipy and PIL

```python
##crop dog image
im = im[100: 500, 0: 360] #YAxis, XAxis
plt.imshow(im) #view image

imPIL = imPIL.crop((0, 100, 360, 500)) #ULX, ULY, LRX, LRY
imPIL.show()

##Rotate and Flip dog
flipped_im = np.flipud(im)
plt.imshow(flipped_im)

rotate_im = ndimage.rotate(im,30)
plt.imshow(rotate_im)

rotate_imPIL = imPIL.rotate(30)
rotate_imPIL.show()

##Change to GreyScale
from skimage.color import rgb2gray
grayIm = rgb2gray(im)
plt.imshow(grayIm, cmap = plt.get_cmap('gray'))
plt.imshow(grayIm) #EXAMPLE: Does not work

grayImPIL = imPIL.convert('LA') #using PIL
grayImPIL.show()

##Change Contrast of Image
from skimage import util
inverted_im = util.invert(im)
plt.imshow(inverted_im)

from skimage import exposure
vmin, vmax = np.percentile(im, (0.2,99.8))
vmin, vmax
contrast_im = exposure.rescale_intensity(im, in_range=(vmin,vmax))
plt.imshow(contrast_im) #FIX: Not working right now

imPIL2 = ImageEnhance.Contrast(imPIL)
imPIL2.enhance(2).show()
```

Converting between np.array and PIL

```
#Convert between numpy array and PIL
Array2PIL = Image.fromarray(im) #converts to PIL from array
Array2PIL.show()

PIL2Array = np.array(imPIL) #change PIL to a numpy array
plt.imshow(PIL2Array)
```

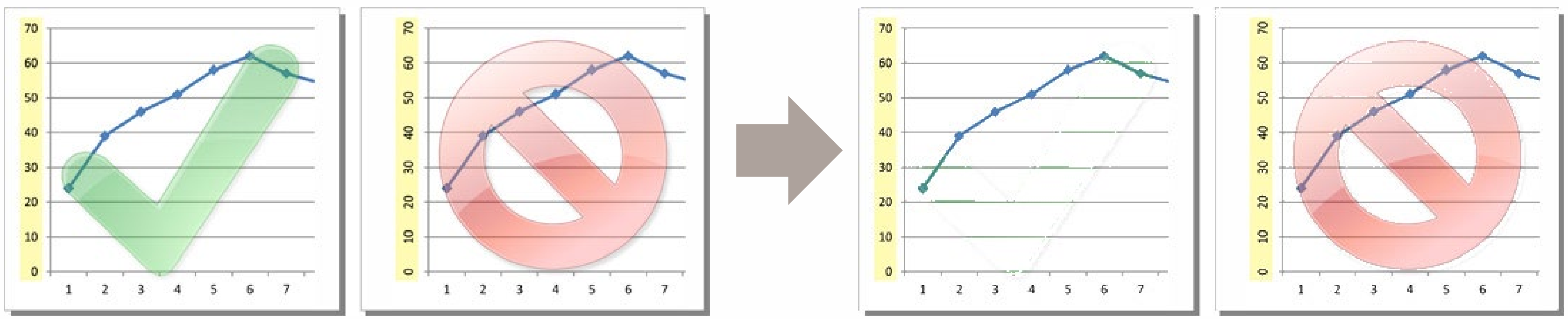
Manually Working with np.array

```python
from PIL import Image
from numpy import *
import matplotlib.pyplot as plt
import sys

#read in an image
pil_im = Image.open('picToFix.jpg') #change to sys.argv[1] if calling from cmd
pil_im.show()
imArray = array(pil_im)

#find shape of array and get number of rows and columns
arrayShape = imArray.shape
rows = arrayShape[0]
rows
cols = arrayShape[1]
cols

#create a negative image
negIm = []
negIm = 255-imArray
plt.imshow(negIm)

#create a greyscale image
greyIm = 0 + imArray
for row in range(rows):
    for col in range(cols):
        grey = (float(imArray[row,col,0]) + float(imArray[row,col,1]) + float(imArray[row,col,2]))/3
        for color in range(3):
            grey = int(grey)
            greyIm[row][col][color] = grey
plt.imshow(greyIm)
```

Using Numpy allows you to iterate through pixels and make very specific changes. Most things are best done using PIL or a pre-made function, but all things can be done “manually”.

Manually Working with np.array

- You can get very (overly?) specific like in the following case where I have removed the green check from the image. I used PixelMath to find the exact range of green in the image.

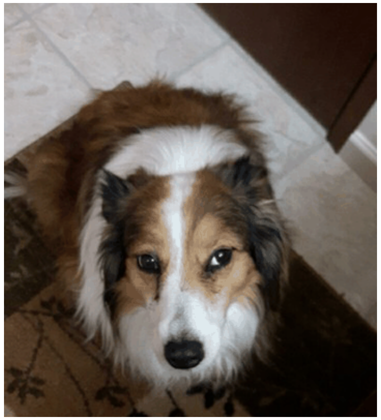



```python
#create an image without the green check
no_g = 0 + imArray
for row in range(rows):
    for col in range(cols):
        if imArray[row,col,0] < 240 and imArray[row,col,1] > 200 and imArray[row,col,1] < 250:
            for color in range(3):
                no_g[row][col][color] = 255
```

Creating a Gif or Movie from a Series of Images

- First, I need to navigate to an empty folder and fill it with numbered images that I want to turn into a movie/gif.

```
## Create a gif from images in file ###########################################
import glob
import moviepy.editor as mpy

os.chdir(r'C:\Users\Stefanie\Documents\WSU\CougPy\gif')

angle = 0
while(angle<360): #creates images that rotate in a circle
    rotate_imPIL = imPIL.rotate(angle)
    #you will need to change your filepath as needed
    fileNameTemplate = r'C:\Users\Stefanie\Documents\WSU\CougPy\gif\dog_{0:02d}.png'
    rotate_imPIL.save(fileNameTemplate.format(angle), format='png')
    angle = angle + 10

gif_name = 'godog'
fps = 50 #set the frame rate
file_list = glob.glob('*.png') # Get all the pngs in the current directory
list.sort(file_list, key=lambda x: int(x.split('_')[1].split('.png')[0])) # Sort the images by #,
clip = mpy.ImageSequenceClip(file_list, fps=fps) #creates the image sequence
clip.write_gif('{}.gif'.format(gif_name), fps=fps) #this will create a gif
#clip.write_videofile('{}.mp4'.format(gif_name), fps=fps) #this will create a video
```



Denoising and Feature Extraction

- This code is from the SciKit tutorial on plot boundaries listed in references

```python
##Removing noise with filters
from skimage.morphology import disk
from skimage import data
from skimage import filters
from skimage import restoration
coins = data.coins()
coins_zoom = coins[10:80, 300:370]
plt.imshow(coins_zoom, cmap='gray', interpolation='nearest')
median_coins = filters.median(coins_zoom, disk(1))
plt.imshow(median_coins, cmap='gray',
           interpolation='nearest')
tv_coins = restoration.denoise_tv_chambolle(coins_zoom, weight=0.1)
plt.imshow(tv_coins, cmap='gray',
           interpolation='nearest')
gaussian_coins = filters.gaussian(coins_zoom, sigma=2)
plt.imshow(gaussian_coins, cmap='gray',
           interpolation='nearest')
```

Denoising and Feature Extraction

- This code is from the SciPy tutorial listed in references

```
##Fixing Mathematical Morphology
square = np.zeros((32, 32))
square[10:-10, 10:-10] = 1
np.random.seed(2)
x, y = (32*np.random.random((2, 20))).astype(np.int)
square[x, y] = 1
plt.imshow(square, cmap='gray')

open_square = ndimage.binary_opening(square)
plt.imshow(open_square, cmap='gray')

eroded_square = ndimage.binary_erosion(square)
plt.imshow(eroded_square, cmap='gray')

reconstruction = ndimage.binary_propagation(eroded_square, mask=square)
plt.imshow(reconstruction, cmap='gray')
```

Denoising and Feature Extraction

```python
##Feature Extraction: Counting "Red Blood Cells" #############
##Create synthetic data
n = 10
l = 256
im = np.zeros((l, l))
points = l*np.random.random((2, n**2))
im[(points[0]).astype(np.int), (points[1]).astype(np.int)] = 1
im = ndimage.gaussian_filter(im, sigma=l/(4.*n))
plt.imshow(im)

mask = im > im.mean() #threshold the image into binary colors by mean
plt.imshow(mask, cmap='gray')

label_im, nb_labels = ndimage.label(mask) #count the 'red blood cells'
nb_labels #will give int count
plt.imshow(label_im) #colors our individual counts
#clearly some are touching and it is counting that as one large cell
#can be adjusted with threshold

##compute the size and mean volume of each region
sizes = ndimage.sum(mask, label_im, range(nb_labels + 1))
sizes
mean_vals = ndimage.sum(im, label_im, range(1, nb_labels + 1))
mean_vals
```

```python
##Remove anoything that is too small for us to care about
mask_size = sizes < 1000
remove_pixel = mask_size[label_im]
remove_pixel.shape
label_im[remove_pixel] = 0
plt.imshow(label_im)
```

This code is from the SciPy tutorial listed in references

More Techniques

- Plenty of other things possible:
- Segmentation:
  - Mark edges of features in image
- More Feature Extraction
  - Computer vision can detect features (ie corners)
- Facial Recognition
  - Using OpenCV or others

Resources

- Practical Computing for Biologists by Haddock & Dunn
- https://realpython.com/blog/python/face-recognition-with-python/
- https://pillow.readthedocs.io/en/3.1.x/reference/Image.html
- http://www.scipy-lectures.org/advanced/image_processing/#geometrical-transformations
- http://scikit-image.org/docs/dev/user_guide/transforming_image_data.html
- http://www.scipy-lectures.org/packages/scikit-image/auto_examples/plot_boundaries.html
- https://www.safaribooksonline.com/library/view/programming-computer-vision/9781449341916/ch01.html