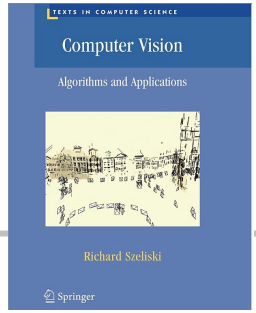
integrating
Vision & Language

Durham University

The bits the whirl-wind left out …..

Reading:

Szeliski



Edge Features

Reading:

Szeliski - 1.1 + 4.2.1

(Background: Ch 2 + 3.1 → 3.3)

Edges as Gradients

- **Edge detection** = **differential operators** to **detect gradients** of the **gray or colour levels** in the image

- If the image is $I(x, y)$ then the basic idea is to compute

$$\nabla I(x,y) = \hat{\mathbf{x}}\frac{\partial}{\partial x}I(x,y) + \hat{\mathbf{y}}\frac{\partial}{\partial y}I(x,y)$$

  - **i.e. Partial Derivatives** in x and y

How do we find gradients in an image ?

Image **Gradients**

- Let the image be f(x,y) thus with partial derivatives we have:

$$\frac{\partial}{\partial x} f(x,y) = \frac{f(x+h,y) - f(x,y)}{h}$$

and

$$\frac{\partial}{\partial y} f(x,y) = \frac{f(x,y+h) - f(x,y)}{h}$$

Image **Gradients**

- Image f(x,y) :

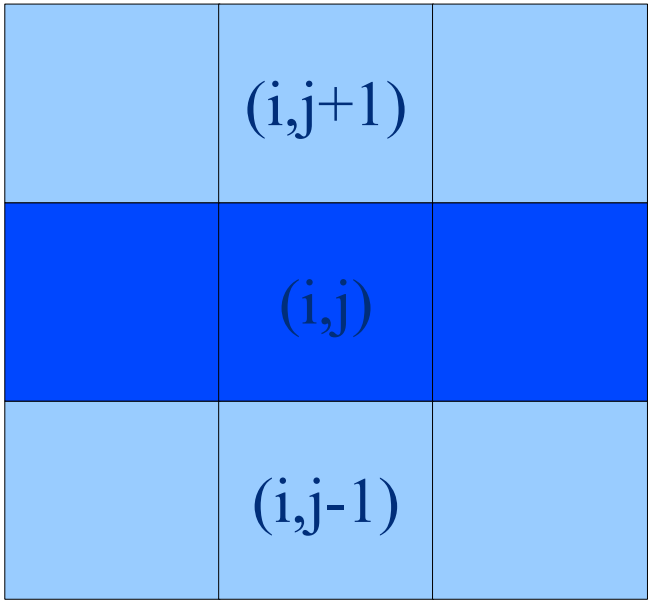
- With a digital image I_{ij}, we can replace the partial derivatives with differenes

$$\Delta_x I_{ij} = I_{i+1,j} - I_{ij}$$

and

$$\Delta_y I_{ij} = I_{i,j+1} - I_{ij}$$

- **N.B.** These operations are equivalent to convolving I_{ij} with the digital function $(-1, 1)$ in the x direction to give $\Delta_x I_{ij}$ and convolving I_{ij} with $(-1, 1)$ in the y direction to give $\Delta_y I_{ij}$.


(i,j+1)
(i,j)
(i,j-1)

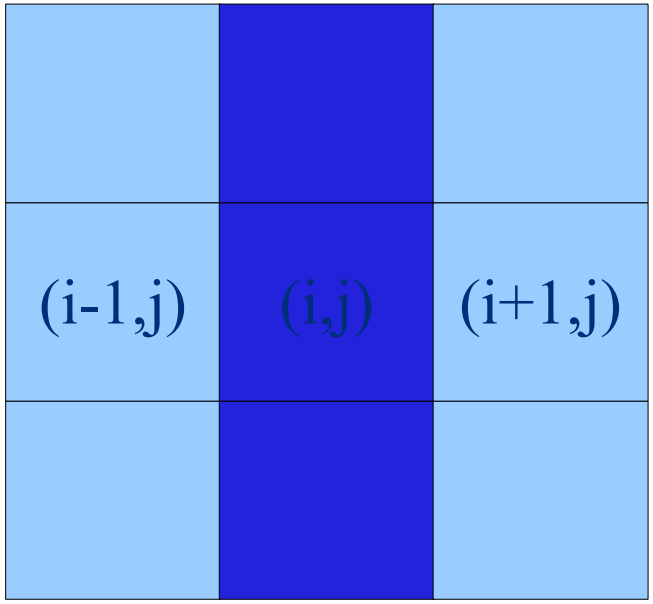

(i-1,j)
(i,j)
(i+1,j)

Image **Gradients = Convolution**

- **Example:** to obtain digital gradients centered at (i, j), we use the differencing scheme

$$\Delta_x I_{ij} = I_{i+1,j} - I_{i-1,j}$$

$$\Delta_y I_{ij} = I_{i,j+1} - I_{i,j-1}$$

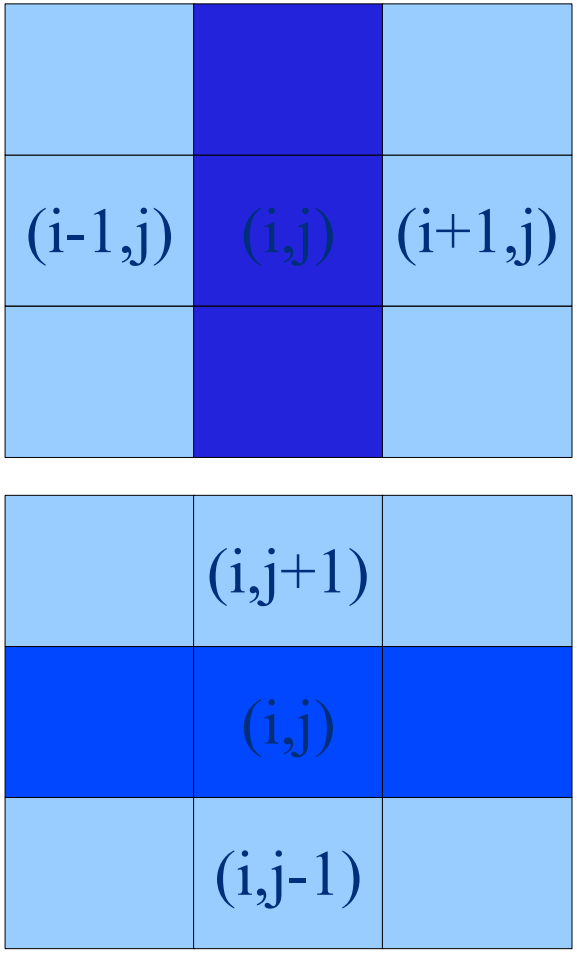
In this case, the convolution kernals are

$$\begin{pmatrix} -1 & 0 & 1 \end{pmatrix}$$

and

$$\begin{pmatrix} -1 \\ 0 \\ 1 \end{pmatrix}$$

respectively.



Aside: Image **Convolution**

- … is essentially the localised weighted sum of the image and the convolution kernel (mask) over a N x M pixel neighbourhood, at a given location within the image (x,y).

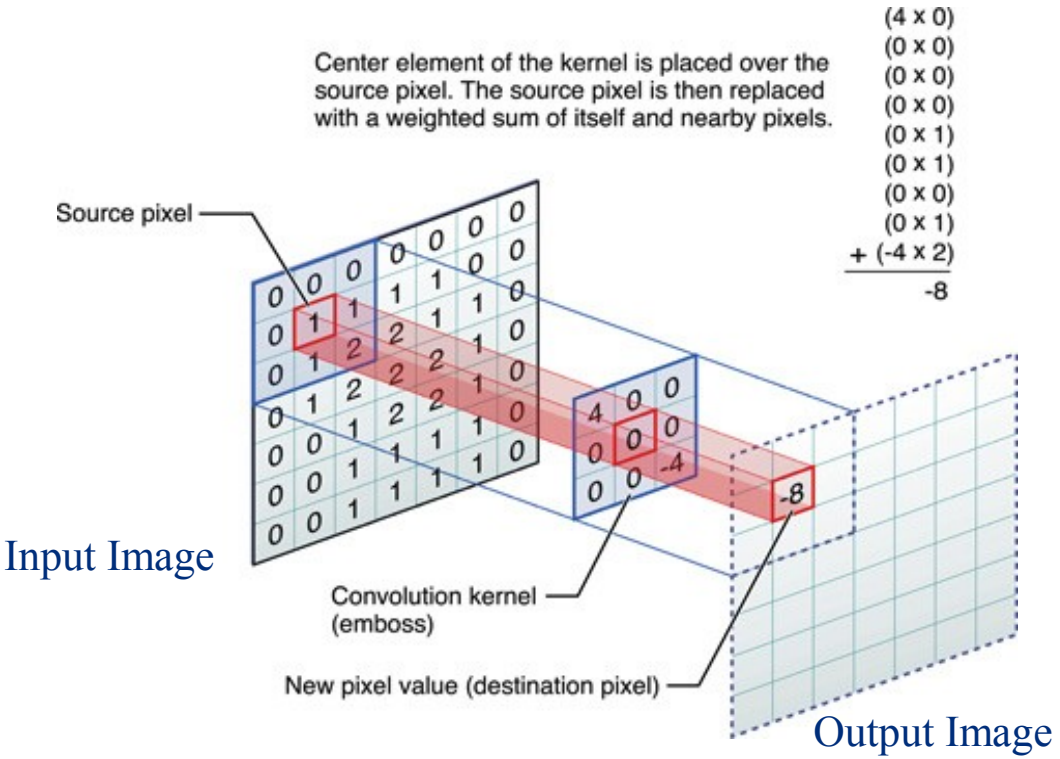


Image source: developer.apple.com

Sobel Edge Detection

source: HIPR2 © 2003/4

-1	0	+1
-2	0	+2
-1	0	+1

Gx

+1	+2	+1
0	0	0
-1	-2	-1

Gy

- **more computationally complex** convolution masks (convolve image with both, both masks sum to zero)
- edge **gradient** magnitude is given by:

$$|G| = \sqrt{Gx^2 + Gy^2}$$

- edge **orientation**: $\theta = \arctan(Gy/Gx)$

Sobel Edge Detection

source: HIPR2 © 2003/4

-1	0	+1
-2	0	+2
-1	0	+1

Gx

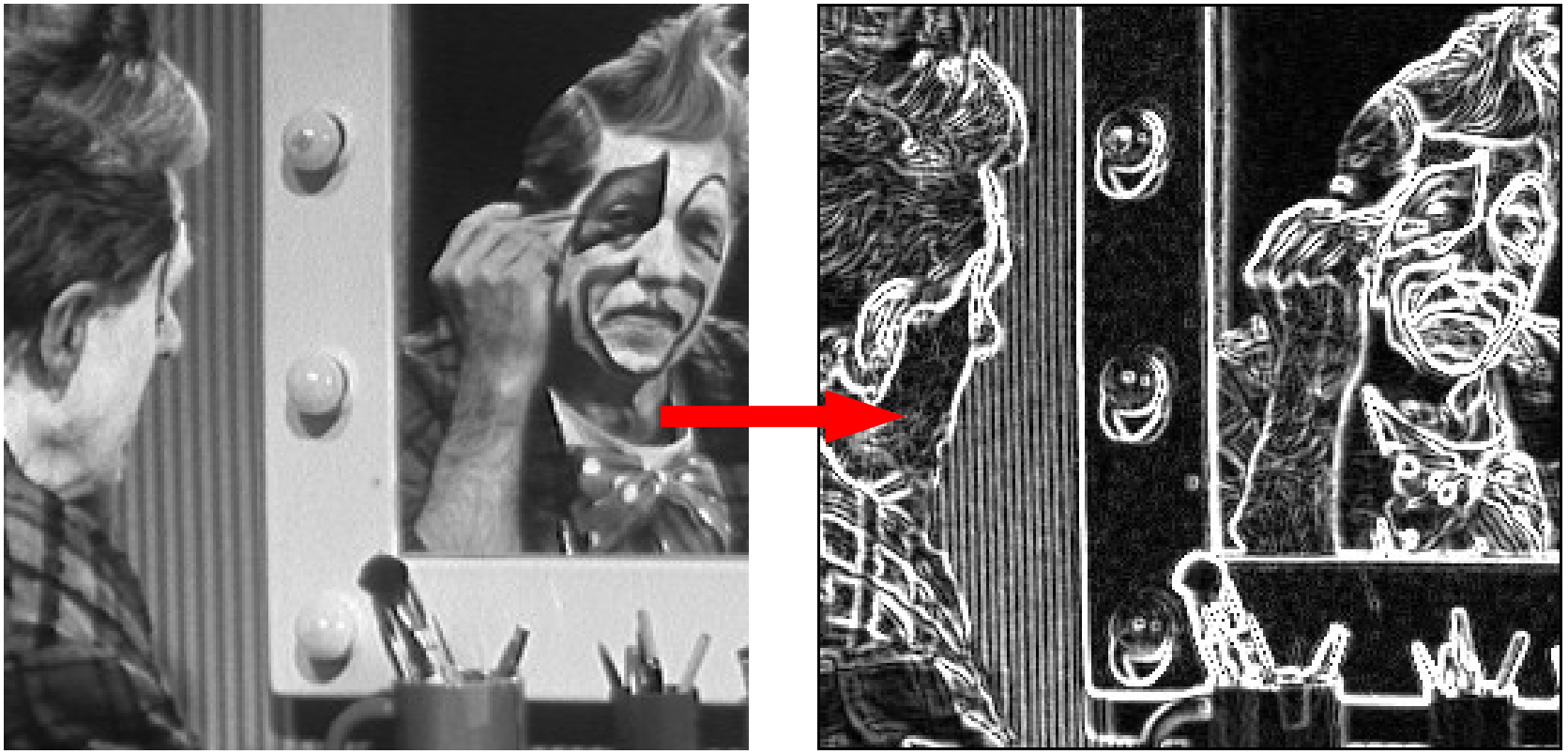
+1	+2	+1
0	0	0
-1	-2	-1

Gy

- designed to **respond maximally** to **edges** running v**ertically and horizontally**
  - one kernel for each of the two perpendicular orientations
- combined to get **magnitude** at each pixel and display this :

$$|G| = \sqrt{Gx^2 + Gy^2} \qquad |G| = |Gx| + |Gy|$$

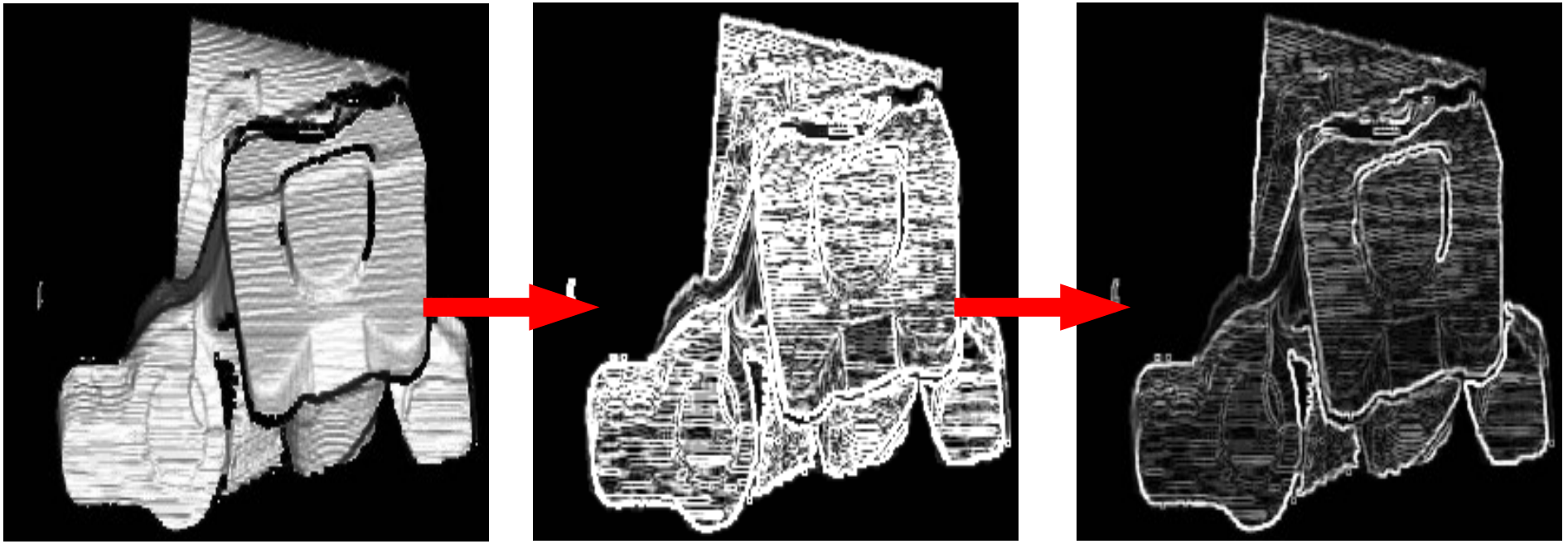
Example : **Sobel**



source: HIPR2 © 2003/4

- **Results** : *rough, noisy edges*
  - *true lines, variable response due to gradient strength*
  - *easy to **threshold***

Example : **Sobel**



source: HIPR2 © 2003/4

- **Results** :

  *But can **still** have **threshold difficulties***

Sobel Example

- Original

Sobel Example

- Sobel

Sobel on RGB

- Sobel on each RGB channel - *then recombined*

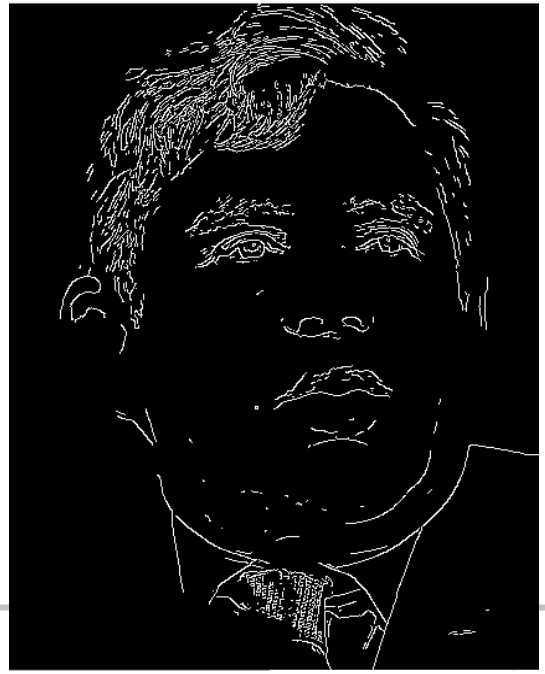
Sobel on RGB

- Sobel on each RGB channel - *then recombined*

Sobel Edge Detection - Summary

- **Advantages**
  - fast,basic edge detection
  - simple to implement using arithmetic operations
  - useful as a measure of localised image texture (i.e. structured pattern)

- **Disadvantages**
  - output is noisy
  - resulting edge width, strength varies



- **Importance:**
  - forms the basis for advanced edge detection, shape detection and object detection

Histograms of Orientated Gradients

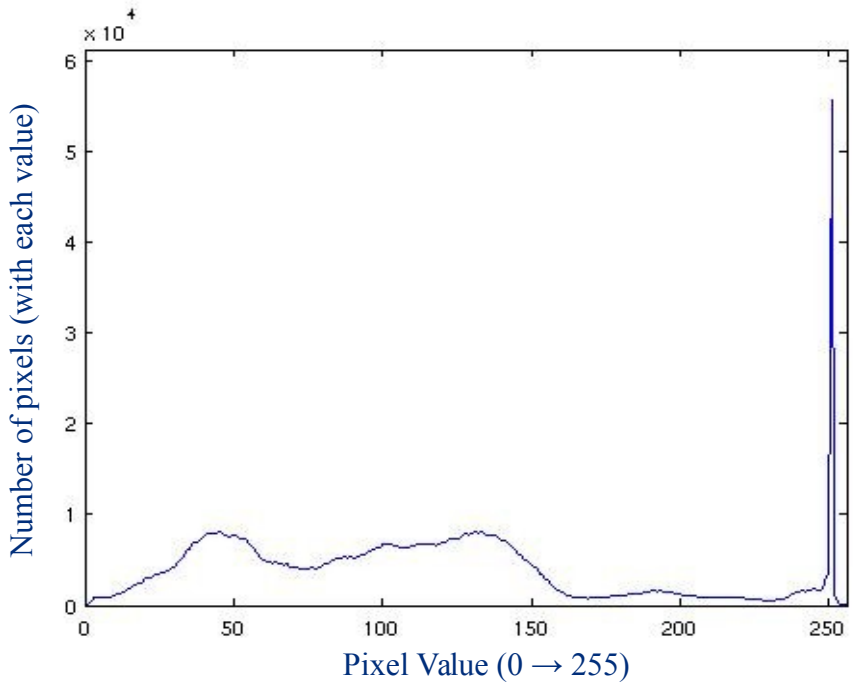
(localized gradient distributions)

Slide material acknowledgements (some portions): P. Barnum (CMU), N. Dalal and B. Triggs (INRIA), E. Seemann (UK).

Reading:

- **Szeliski – Section 14.1.2**

Reminder … **Histograms**



- **Histogram = statistical distribution** of image pixel values
  - no structural representation
  - **different images** have very **different histograms**
  - **similar images** have **similar histograms**

Edge Distribution Based Recognition

(using histograms of edge patterns for the recognition of objects)

Motivation

- Recognising **classes of object**:
  - e.g. people, vehicles
- **challenges**:
  - varying {scale, backgrounds, illumination, pose, colour appearance}
  - occlusion

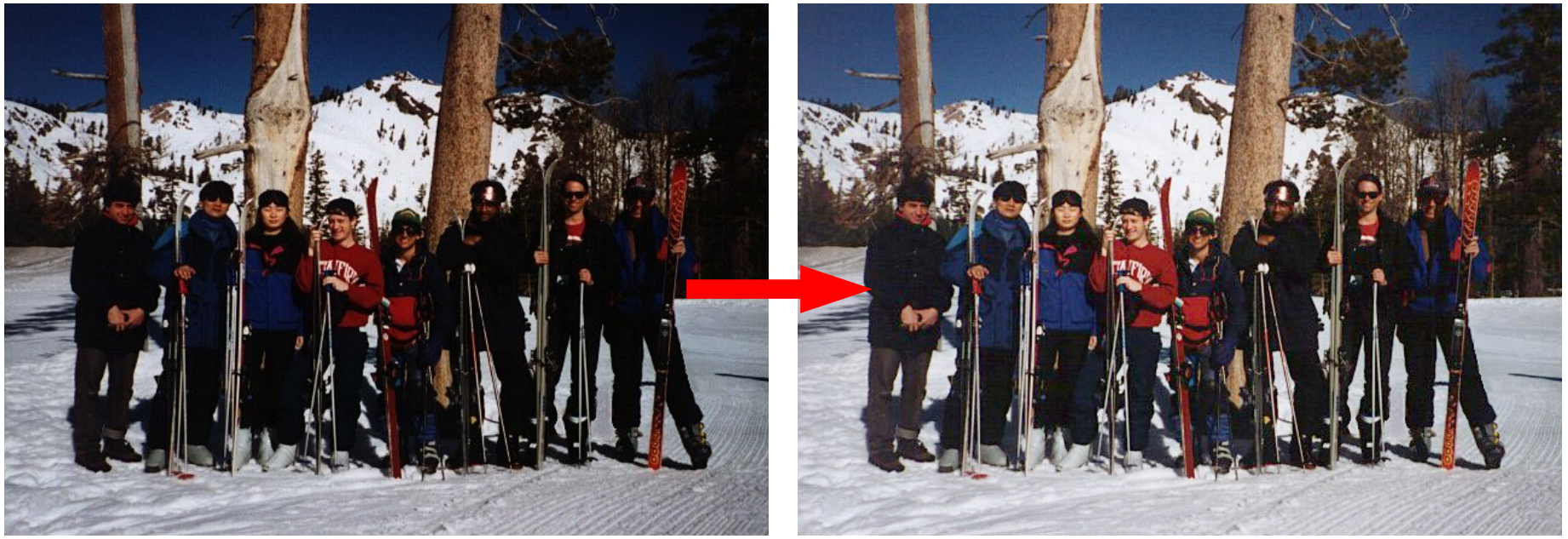
Approach: *overview*

- **Calculate Histograms of Gradient** (Edge) **Orientations**
- **Learn pattern** of **gradient orientations specific** to given **object class**
  - train **machine learning classifier** to differentiate

- **Search for pattern** in unseen image examples

Histogram of Orientated Gradients (HOG), *[Dalal/Triggs, 2005]*

Step 1: Image **Preprocessing**



source:http://graphics.stanford.edu/gamma.html

- Gamma Equalization / Correction – *optional step*
  - *See: http://www.cambridgeincolour.com/tutorials/gamma-correction.htm*

Step 2: **Compute Gradients**

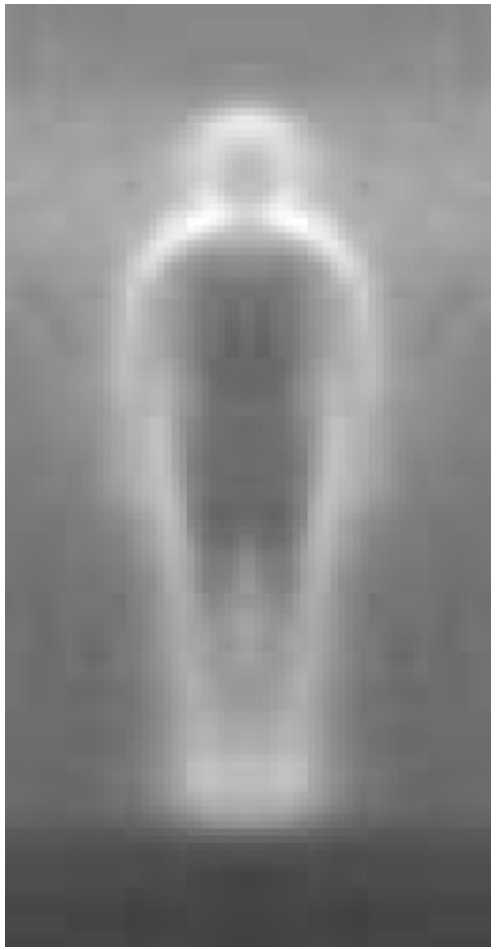
- Compute gradients over *NxM* image region
  - convolve with

    [-1 0 1] / $[-1\ 0\ 1]^T$ filters

    (i.e. Sobel edge response)
  - no smoothing
  - compute gradient magnitude + direction

    (as per Sobel – lecture 1)

Per pixel: use colour channel with greatest magnitude -> final gradient



-1	0	1

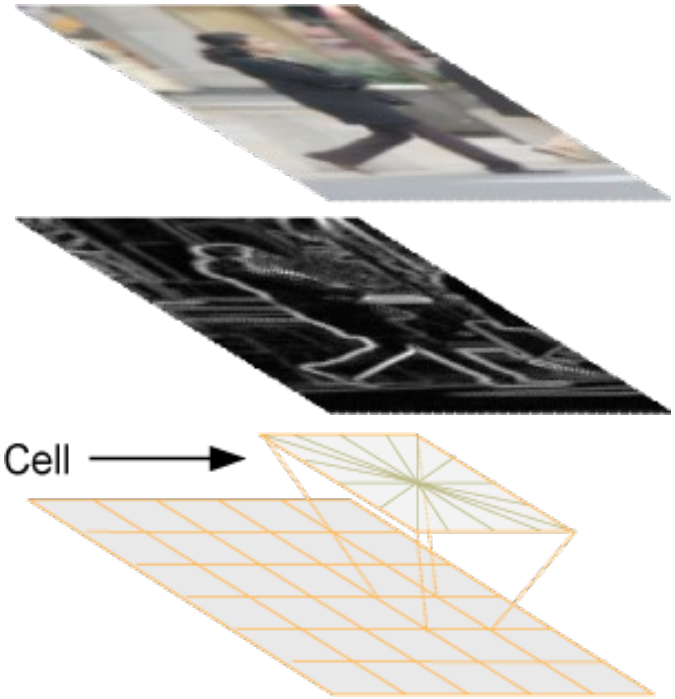
R

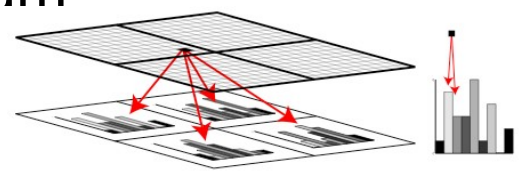
G

B

Step 3: Compute **Cell Histograms**

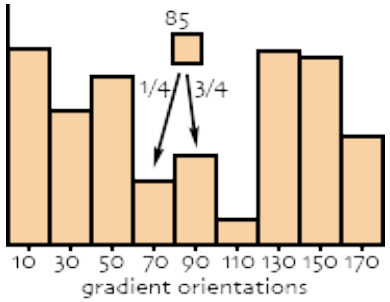
- Cell histograms computed over *CxC* (commonly 8x8) pixels
  - 9 histogram bins (per cell)
  - range 0 → 180 degrees
    - → 20 deg. range per bin
  - histogram entries filled with gradient magnitudes
    - **weighted assignment / vote** of gradient a pixel (x,y) to 4 adjacent cells **(spatial)** and to histogram bins **(orientation)**





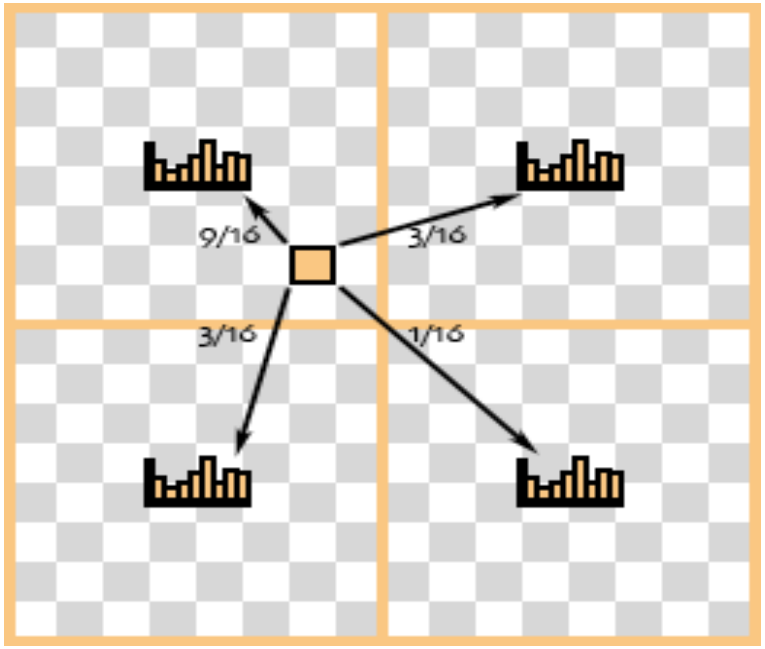
Weighted Assignment - *Example*

- **θ=85 degrees**
  Distance to histogram bin centers
    Bin 70 -> 15 degrees
    Bin 90 -> 5 degress
  Ratios: 5/20=**1/4**, 15/20=**3/4**



- Distance to **cell centers**
  - Left: 2, Right: 6
  - Top: 2, Bottom: 6

- Ratio Left-Right: 6/8, 2/8
- Ratio Top-Bottom: 6/8, 2/8
- Ratios:
  - 6/8*6/8 = 36/64 = 9/16
  - 6/8*2/8 = 12/64 = 3/16
  - 2/8*6/8 = 12/64 = 3/16
  - 2/8*2/8 = 4/64 = 1/16

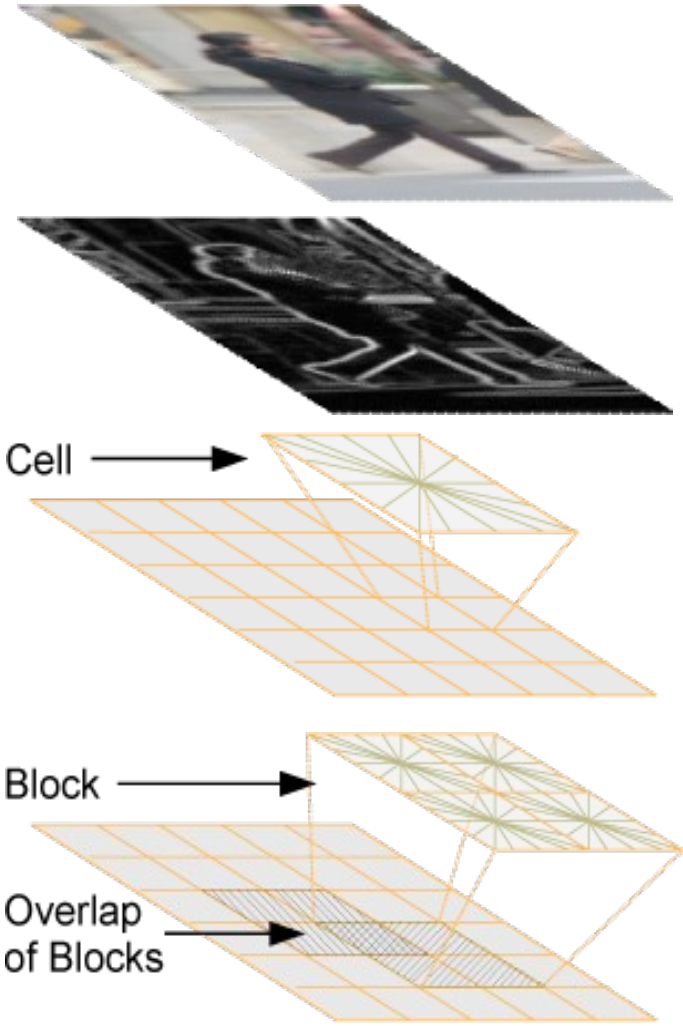


Example courtesy of E. Seemann

Step 4: Compute **Block Histograms**

- Blocks formed of ~2x2 cells
  - blocks overlap
  - each cell contributes to multiple blocks
- Block histogram = normalised sum of cell histograms
  - L2 Norm
- Finally … concatenate block histograms to give

  HOG descriptor
  - high dimensional vector



HOG descriptor (visualisation)

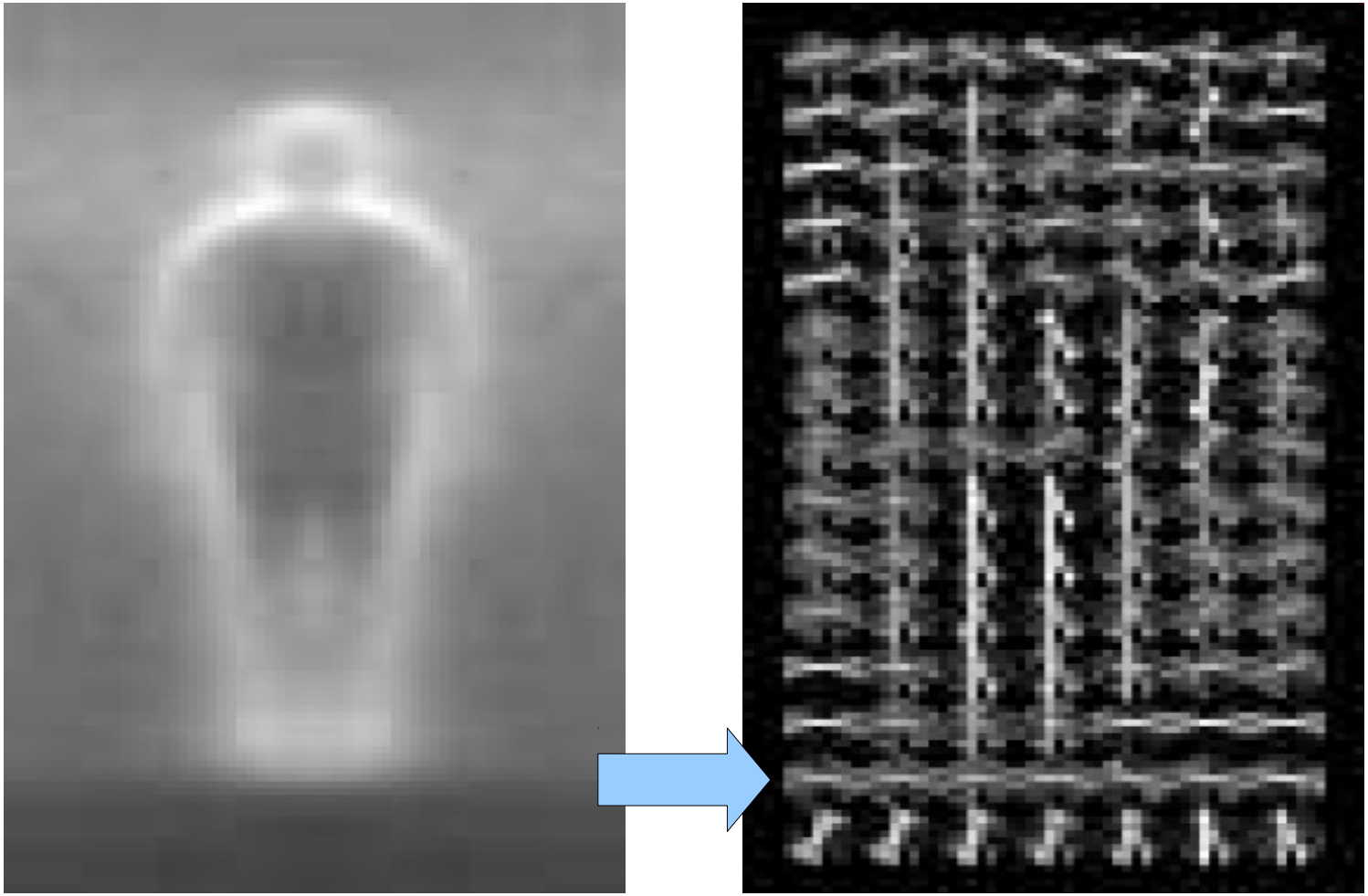
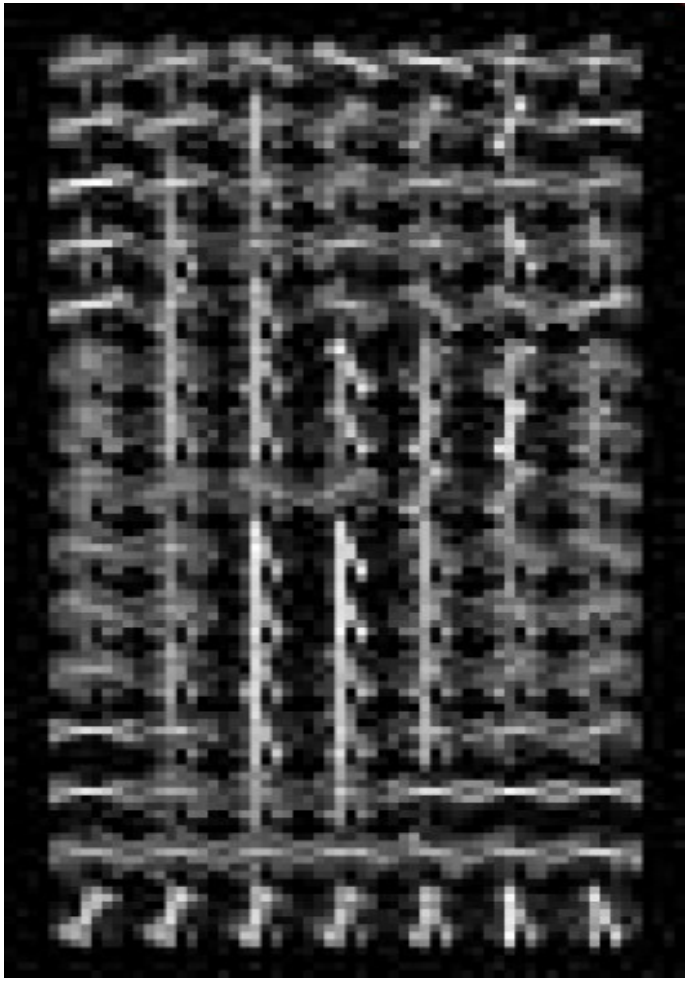


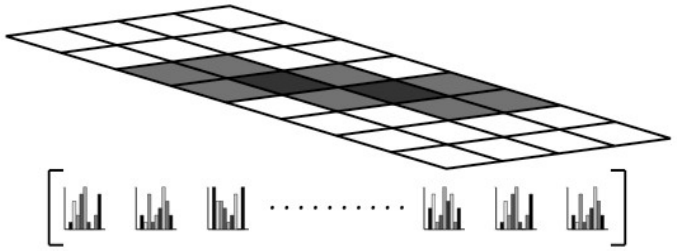
Image Region

Cell Histograms

HOG descriptor



Cell Histograms

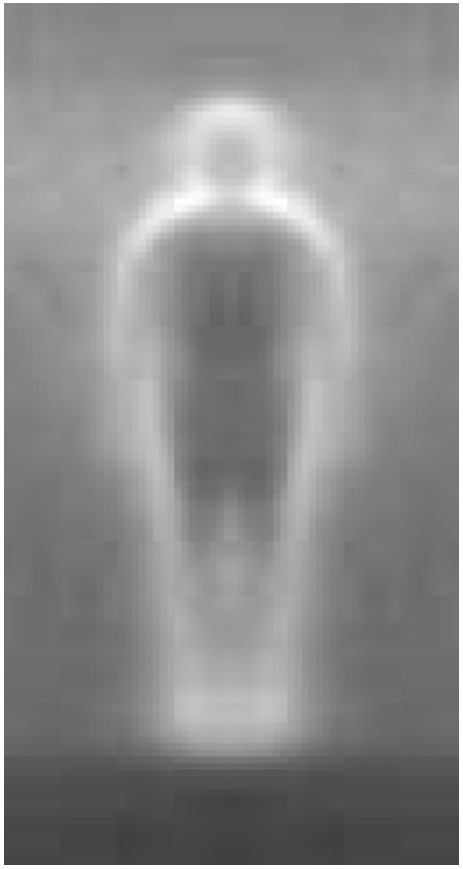


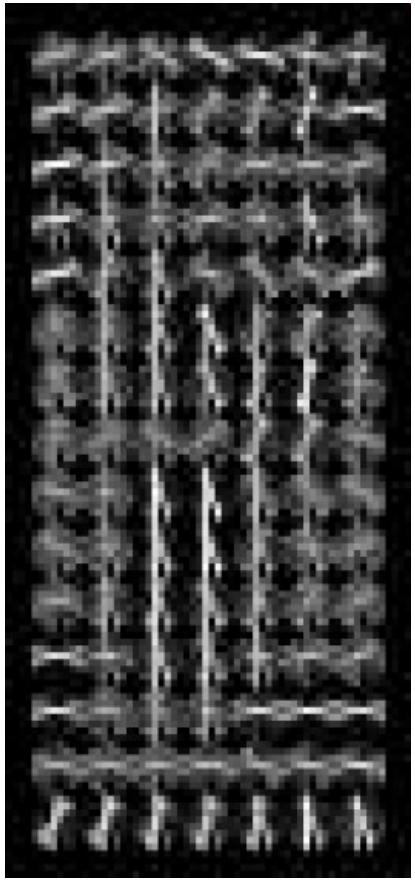
feature vector,

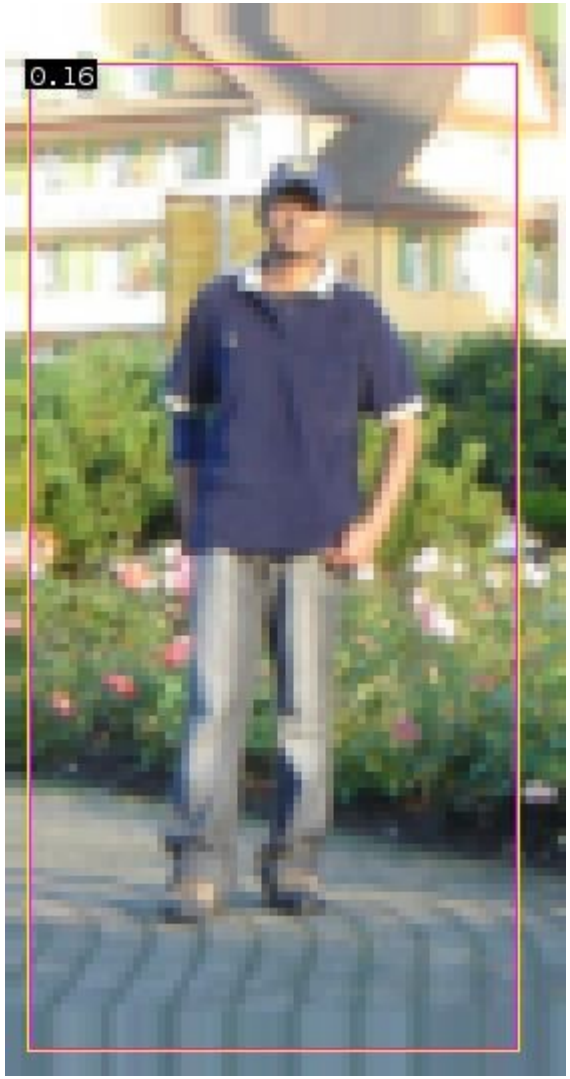
$V_{HOG} = \{\ldots, \ldots, \ldots.\}$

(dimension typically ~4000+)

HOG “Pattern” occurrence ...



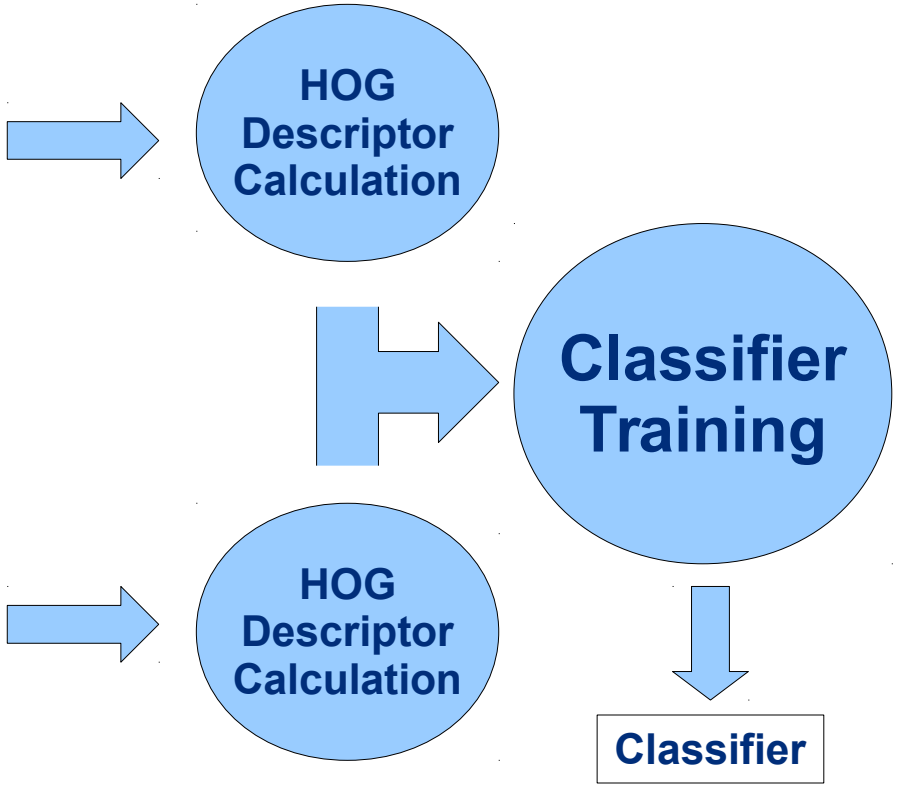




Step 5: Trained **Classifier Approach**

+ve class (people) examples
(all aligned so "subject in centre of image)

-ve class (not people) examples
(random, representative of backgrounds)



- A Machine Learning approach ….
  (typically **Support Vector Machine**)

Feature Points

(localized salient points within gradient space)

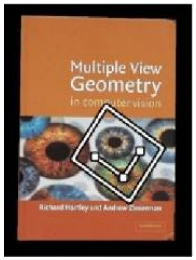
Reading:

Szeliski – Section 4.1 + 6.1 + 14.3 + 14.4

Slide material acknowledgements (some portions): Richard Szeliski, Microsoft Research - CVPR 2007 - Fei-Fei / Fergus / Torralba / Sivic

Outline of Approach

- Identify **generic features** within a sample object



Sample Features

- Identify **generic features** within a query scene image

Scene Features

- If a subset of **scene features and sample features match**

  → Sample Object Detected

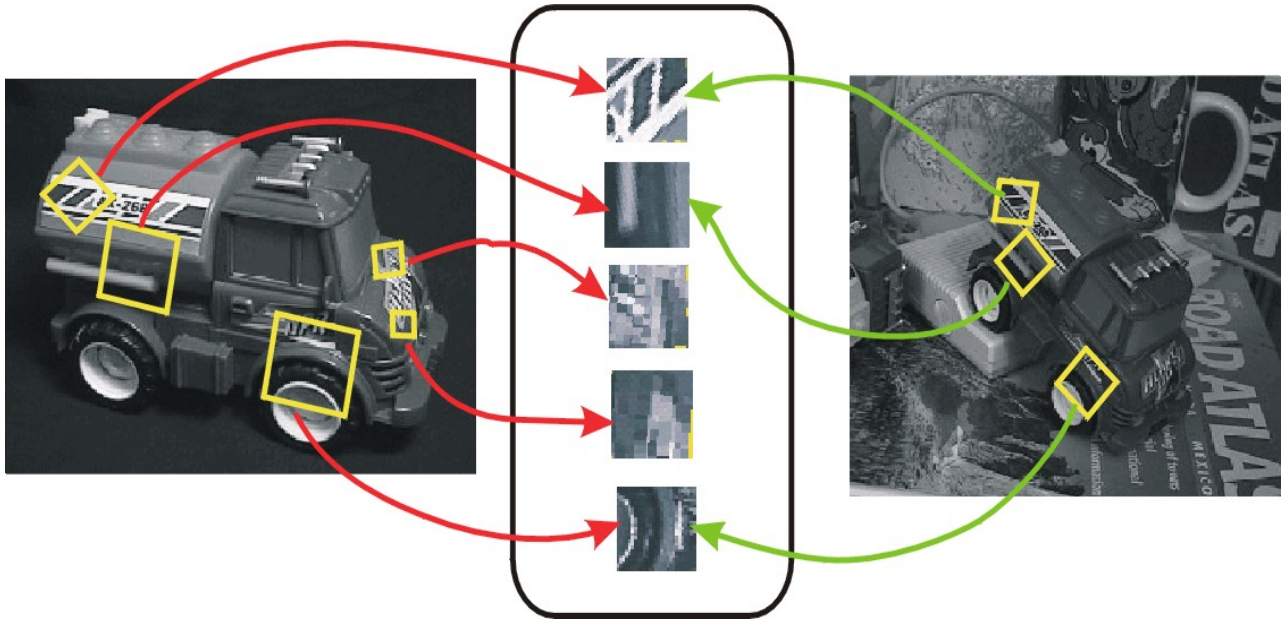
  (at a given pose)

Corresponding Features

- ***Generalize to object classes {people, car ....}***

Require: **Invariant local features**

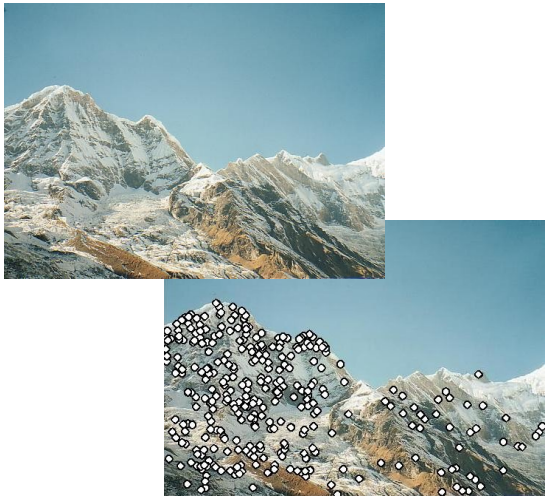
- Find features that are **invariant to transformations**
  - **geometric invariance**: translation, rotation, scale
  - **image intensity invariance**: brightness, exposure, *noise* ….



Feature Descriptors : an invariant method of describing localized image features

Why local features ?

- **Locality**
  - features are local, so **robust to occlusion and clutter**
- **Distinctiveness**
  - can **differentiate** a large database of objects
- **Quantity**
  - **hundreds or thousands** in a single image
- **Efficiency**
  - **real-time** performance achievable
- **Generality**
  - exploit **different types of features** in different situations



What makes a good feature?

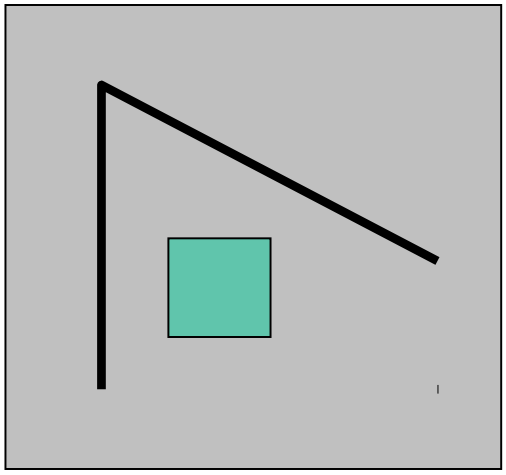
Look for **image regions that are unusual**
Lead to **unambiguous matches in other images**

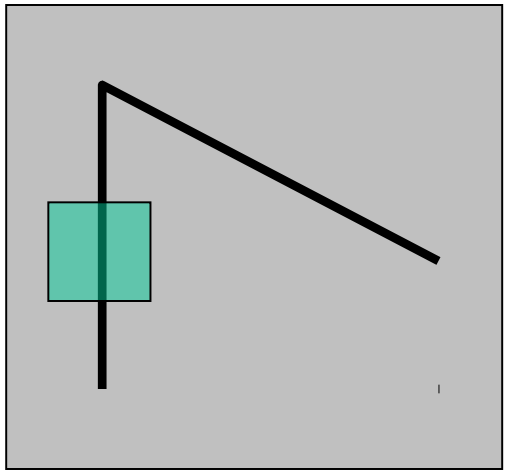
How to define "unusual"?

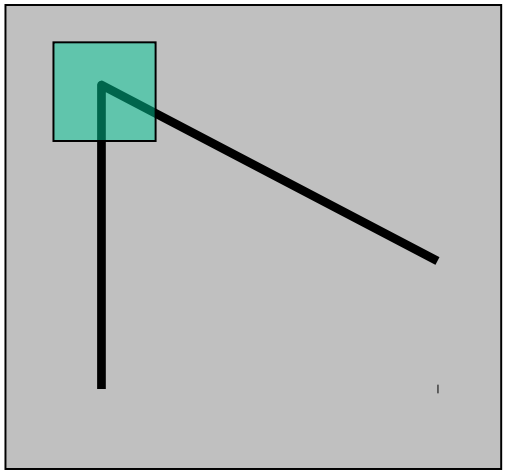
Local measures of **uniqueness**

Suppose we only consider a small window of pixels

- What defines whether a feature is a good or bad candidate?





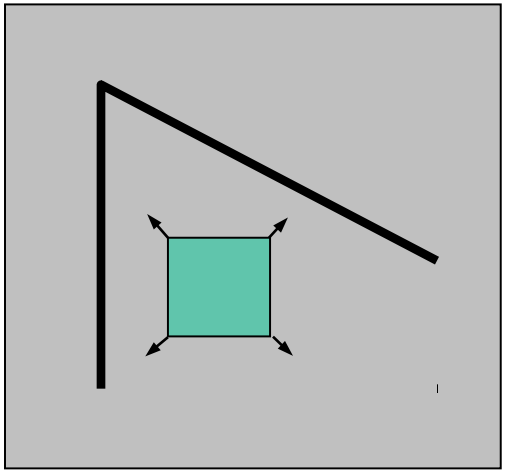


Slide adapted from Darya Frolova, Denis Simakov, Weizmann Institute.

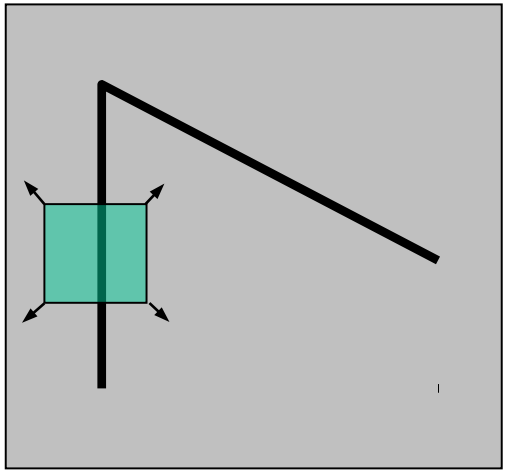
Feature **detection**

Local measure of feature uniqueness

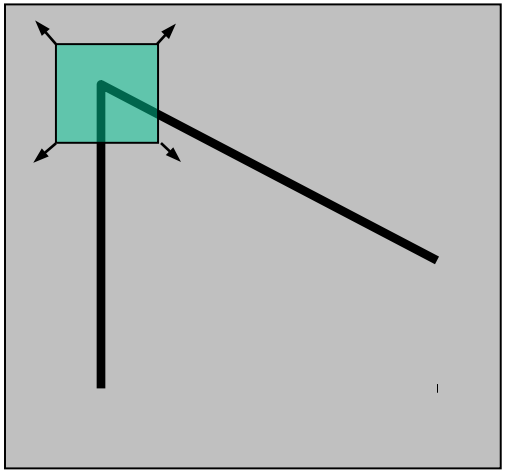
- How does the window change when you shift it?
- Shifting the window in *any direction* causes a *big change*



"flat" region:
no change in all directions



"edge":
no change along the edge direction



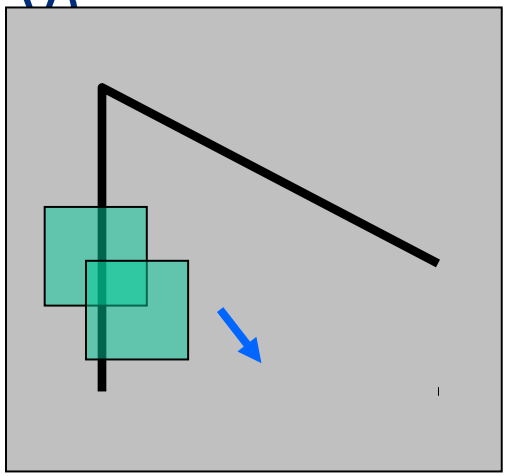
"corner":
significant change in all directions

Slide adapted from Darya Frolova, Denis Simakov, Weizmann Institute.

Feature detection: *the maths*

Consider shifting the window W by (u,v)

- how do the pixels in W change?
- compare each pixel before and after by Sum of Squared (pixel) Differences (SSD)
- this defines an SSD "error" of *E(u,v)*:



$$E(u,v) = \sum_{(x,y)\in W} [I(x+u, y+v) - I(x,y)]^2$$

Small motion assumption

Taylor Series expansion of I:

$$I(x+u, y+v) = I(x,y) + \frac{\partial I}{\partial x}u + \frac{\partial I}{\partial y}v + \text{higher order terms}$$

If the motion (u,v) is small, then first order approx is good

$$I(x+u, y+v) \approx I(x,y) + \frac{\partial I}{\partial x}u + \frac{\partial I}{\partial y}v$$

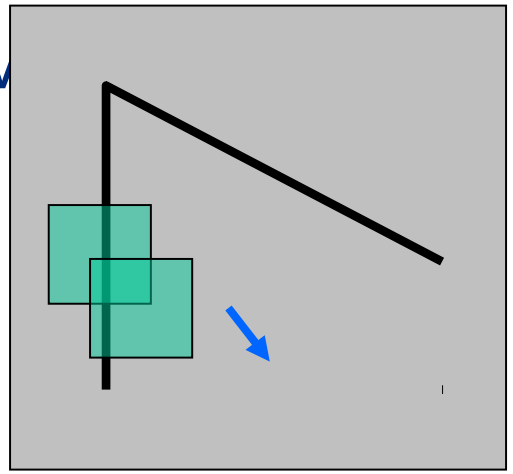
$$\approx I(x,y) + [I_x \;\; I_y] \begin{bmatrix} u \\ v \end{bmatrix}$$

shorthand: $I_x = \frac{\partial I}{\partial x}$

Feature detection: *the maths*

Consider shifting the window W by (u,v

- how do the pixels in W change?
- compare each pixel before and after by Sum of Squared Differences (SSD)
- this defines an SSD "error" of *E(u,v)*:



$$E(u,v) = \sum_{(x,y)\in W} [I(x+u,y+v) - I(x,y)]^2$$

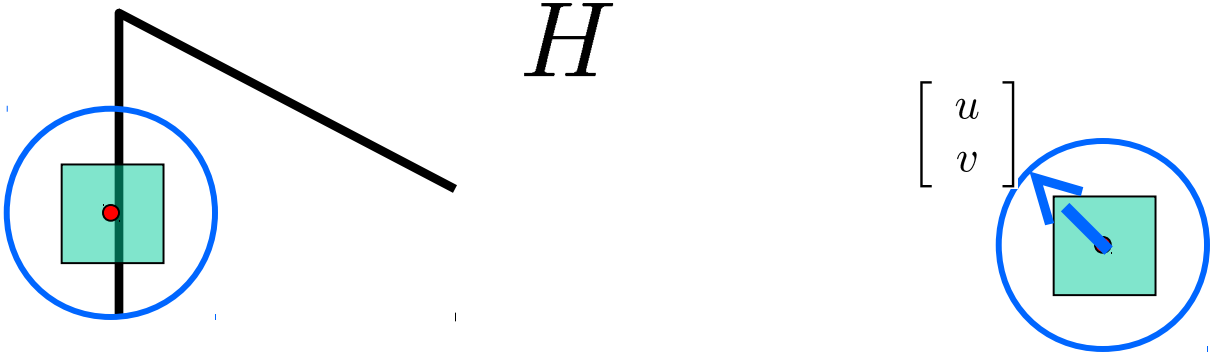
$$\approx \sum_{(x,y)\in W} [I(x,y) + [I_x \; I_y] \begin{bmatrix} u \\ v \end{bmatrix} - I(x,y)]^2$$

$$\approx \sum_{(x,y)\in W} \left[[I_x \; I_y] \begin{bmatrix} u \\ v \end{bmatrix} \right]^2$$

Feature detection: *the maths*

This can be rewritten:

$$E(u,v) = \sum_{(x,y)\in W} [u\ v] \underbrace{\begin{bmatrix} I_x^2 & I_x I_y \\ I_y I_x & I_y^2 \end{bmatrix}}_{H} \begin{bmatrix} u \\ v \end{bmatrix}$$



For the example above

- You can move the centre of the green window to anywhere on the blue unit circle
- Which directions will result in the largest and smallest E values?
- We can find these directions by looking at the eigenvectors of *H*

Quick eigenvalue/eigenvector review:

The **eigenvectors** of a matrix **A** are the vectors **x** that satisfy:

$$Ax = \lambda x$$

The scalar λ is the **eigenvalue** corresponding to **x**

- The eigenvalues are found by solving:

$$det(A - \lambda I) = 0$$

- In our case, ***A*** = ***H*** is a 2*x*2 matrix, so we have

$$det \begin{bmatrix} h_{11} - \lambda & h_{12} \\ h_{21} & h_{22} - \lambda \end{bmatrix} = 0$$

- The sc[illegible]

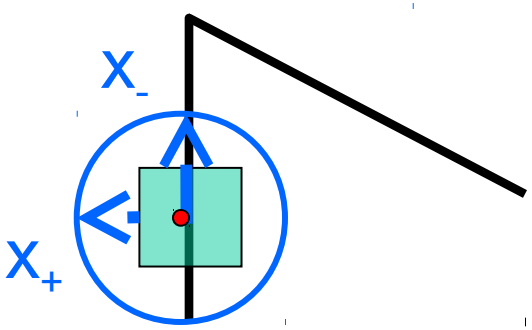
$$\lambda_{\pm} = \frac{1}{2} \left[(h_{11} + h_{22}) \pm \sqrt{4h_{12}h_{21} + (h_{11} - h_{22})^2} \right]$$

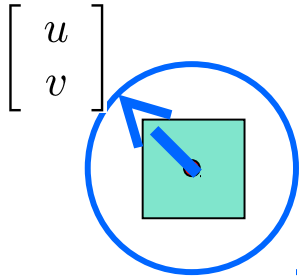
Once you know λ, you find **x** by solving $\begin{bmatrix} h_{11} - \lambda & h_{12} \\ h_{21} & h_{22} - \lambda \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = 0$

Feature detection: *the maths*

This can be rewritten:

$$E(u,v) = \sum_{(x,y)\in W} [u\ v] \underbrace{\begin{bmatrix} I_x^2 & I_x I_y \\ I_y I_x & I_y^2 \end{bmatrix}}_{H} \begin{bmatrix} u \\ v \end{bmatrix}$$





Eigenvalues and eigenvectors of H

- Define shifts with the smallest and largest change (E value)
- x_+ = direction of largest increase in E.
- ∀ λ_+ = amount of increase in direction x_+
- x_- = direction of smallest increase in E.
- ∀ λ_- = amount of increase in direction x_-

$$Hx_+ = \lambda_+ x_+$$
$$Hx_- = \lambda_- x_-$$

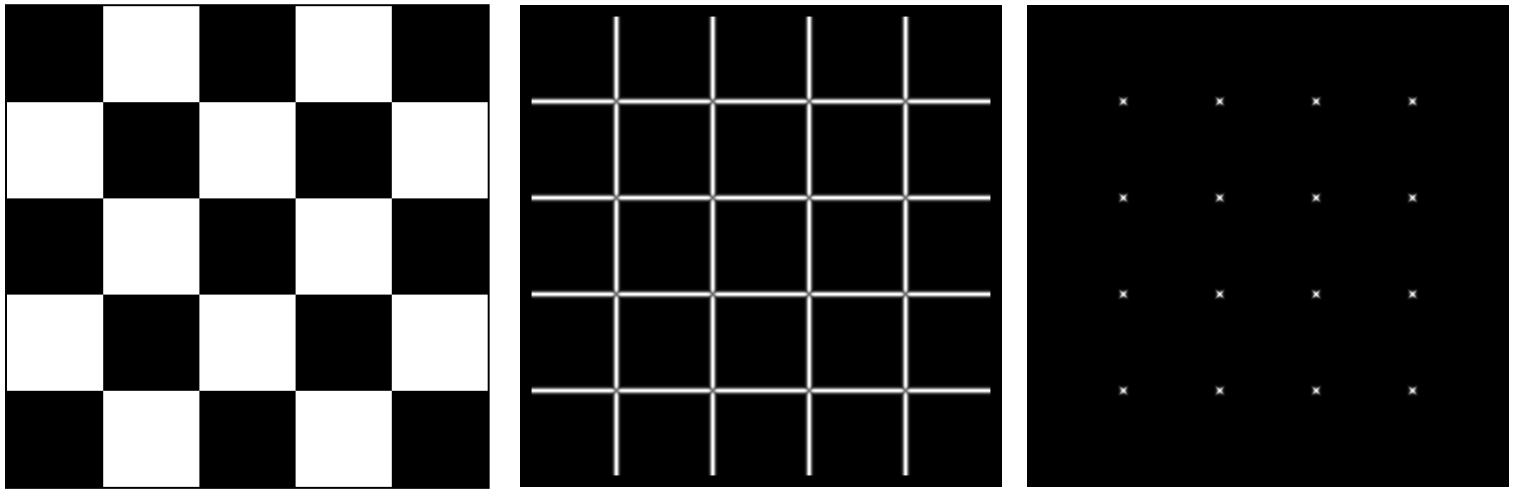
How are λ_+, x_+, λ_-, and x_- relevant for feature detection?

What's our feature scoring function?

Feature detection: *the maths*

Want *E(u,v)* to be *large* for small shifts in *all* directions

- the *minimum* of *E(u,v)* should be large, over all unit vectors [u v]
- this minimum is given by the smaller eigenvalue (λ_-) of *H*
  - *corresponding eigenvector*

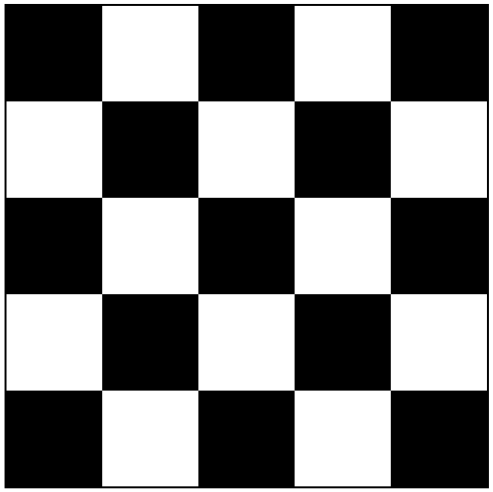


I λ_+ λ_-

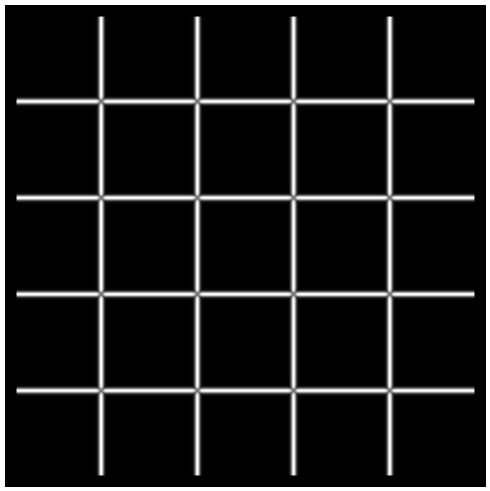
Feature detection summary

Approach Summary:

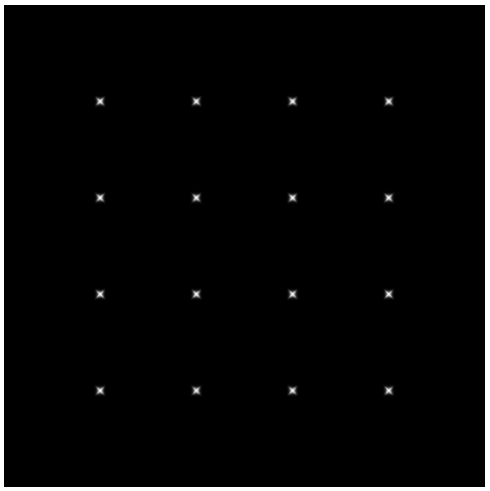
- Compute the **gradient at each point in the image** (e.g. Sobel)
- **Create the *H* matrix** from the entries in the gradient
- Compute the **Eigenvalues**
- Find points with large response (λ_+ > threshold)
- Choose those points where λ_- is a local maximum as features



I



λ_+



λ_-

Feature Points - *Harris*

λ_- is a variant of the "Harris operator" for feature detection

$$f = \frac{\lambda_1 \lambda_2}{\lambda_1 + \lambda_2}$$

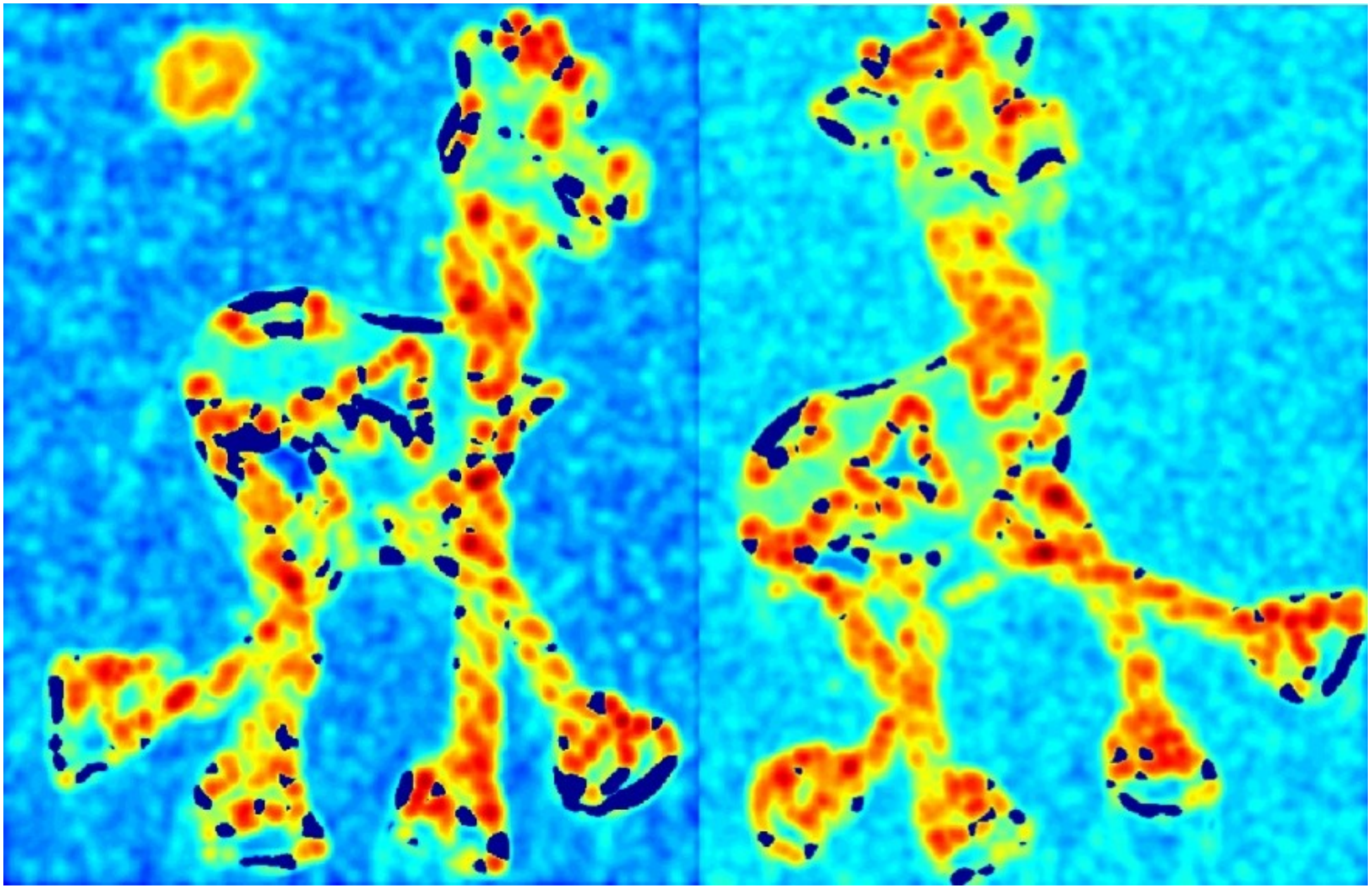
$$= \frac{determinant(H)}{trace(H)}$$

- The *trace* is the sum of the diagonals, i.e., *trace(H)* = $h_{11} + h_{22}$
- Very similar to λ_- but less expensive (no square root)
- Called the "Harris Corner Detector" or "Harris Operator"
- Lots of other detectors, this is one of the most popular and earliest

[Harris / Stephens, 1988]

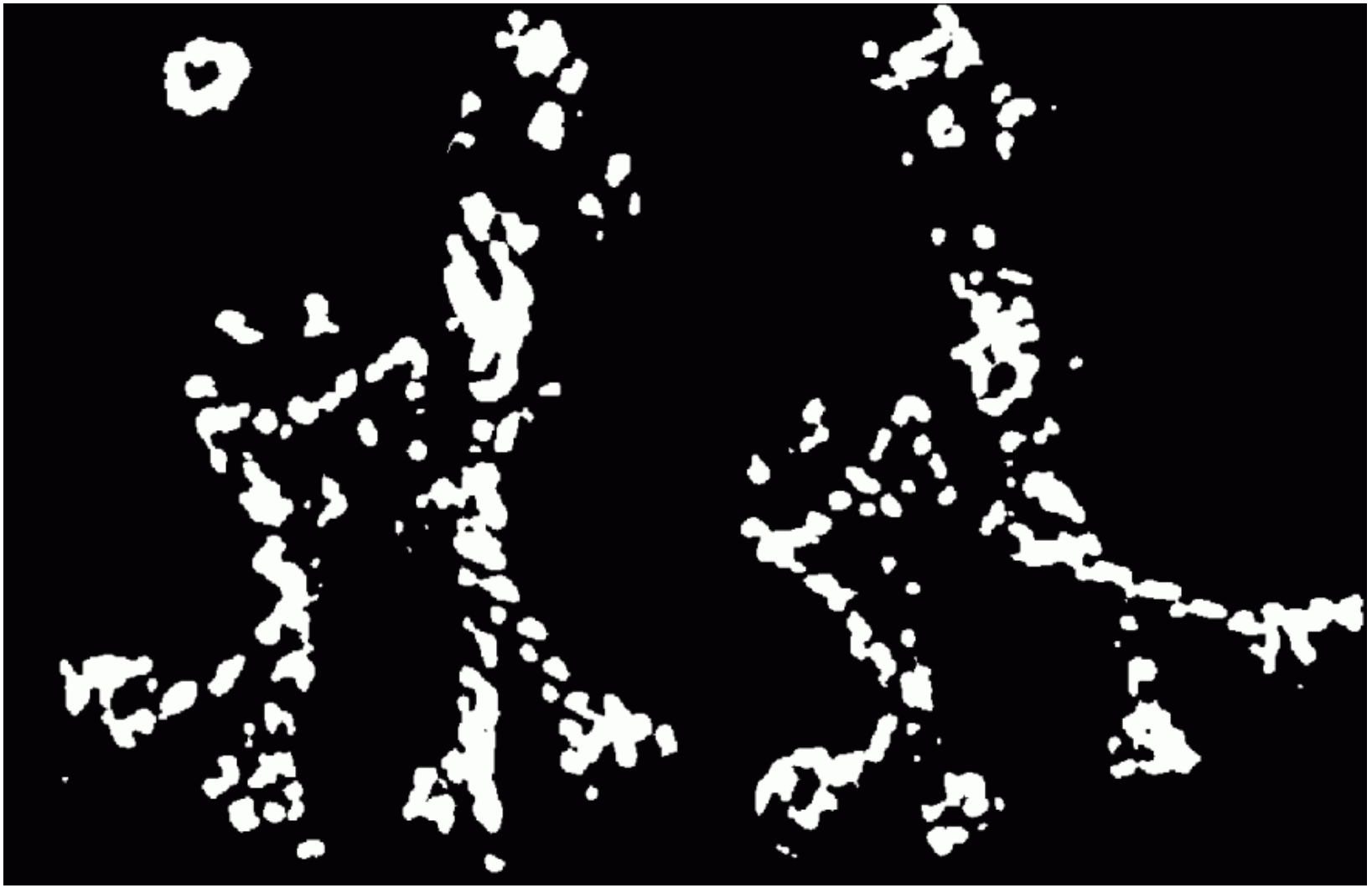
Feature Points - *Harris*

Feature Points - *Harris*



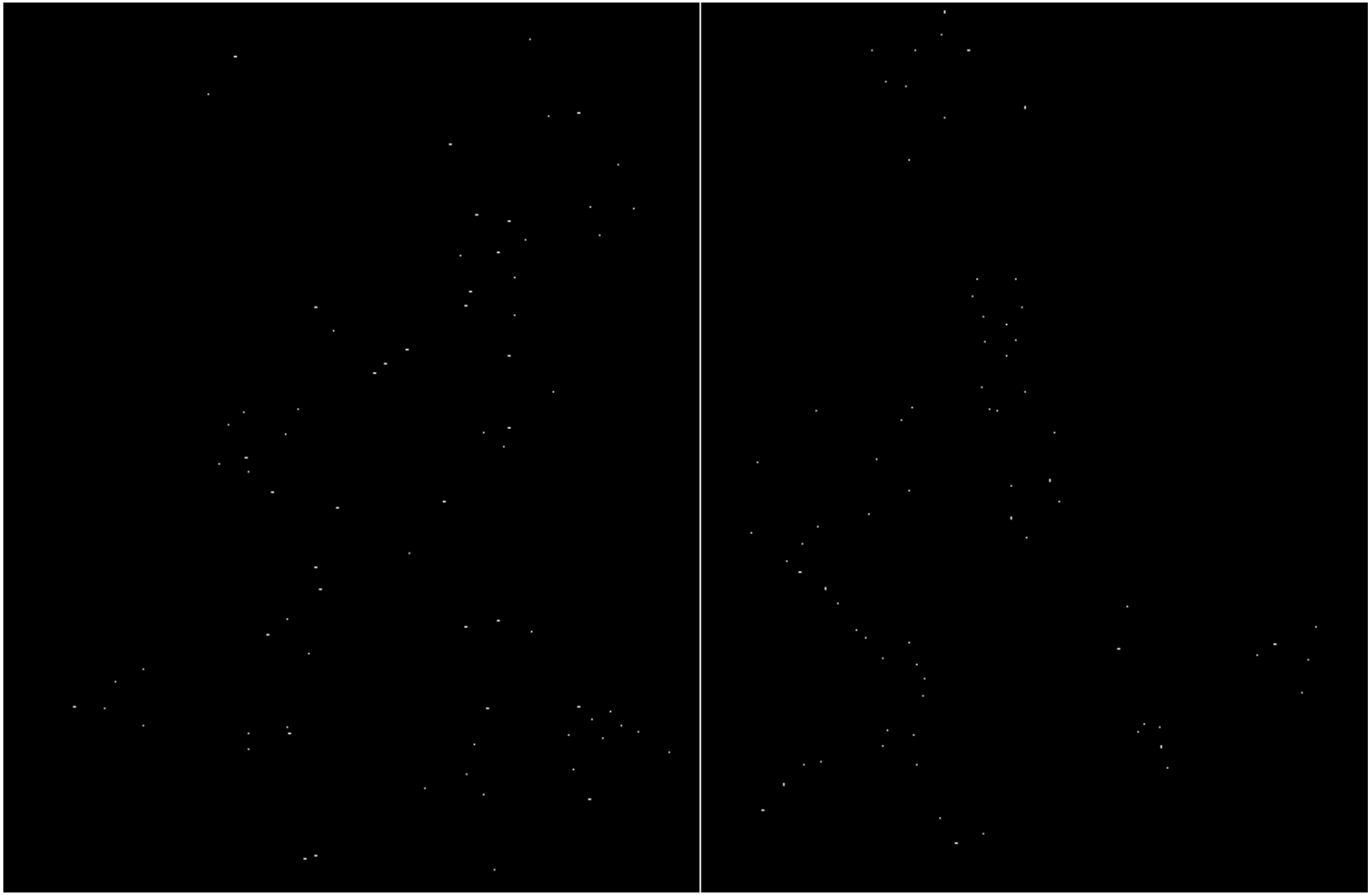
f value (red high, blue low)

Feature Points - *Harris*



f thresholded > value

Feature Points - *Harris*



f local maxima

Feature Points - *Harris*

Harris Feature Points (red)

What about Feature Invariance ?

Suppose you **rotate** the image by some angle

- Will you still pick up the same features?

What if you change the image brightness?

(i.e. ***lighting***)

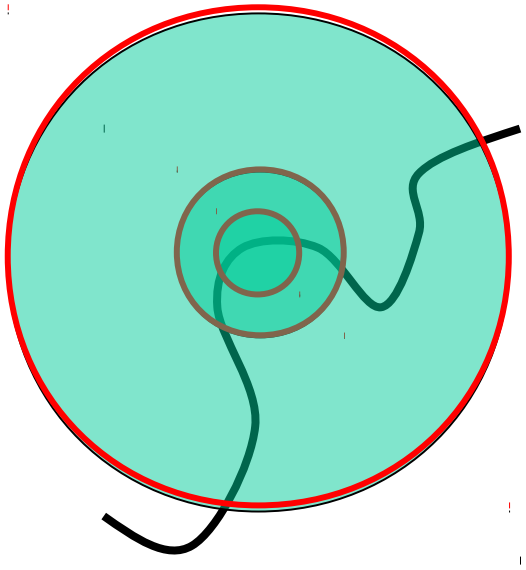
LOOK BACK AT GIRAFFE EXAMPLE

But what about Scale?

[Ideally (for recognition) we want invariance to all

Scale invariant feature detection (*SIFT*)

Suppose you're looking for corners



Key idea: find scale that gives local maximum of f

- f is a local maximum in both position and scale
- *Common definition of f:* difference of two Gaussian filtered images with different σ

Difference of Gaussian (DoG)

1. Compute Difference of Gaussian

(i.e. same image, two diff. levels of Gauss. filtering, subtract one from other)

σ=1.0;σ=1.4;

σ=5;σ=7;

σ=10;σ=14;

Difference of Gaussian (DoG)

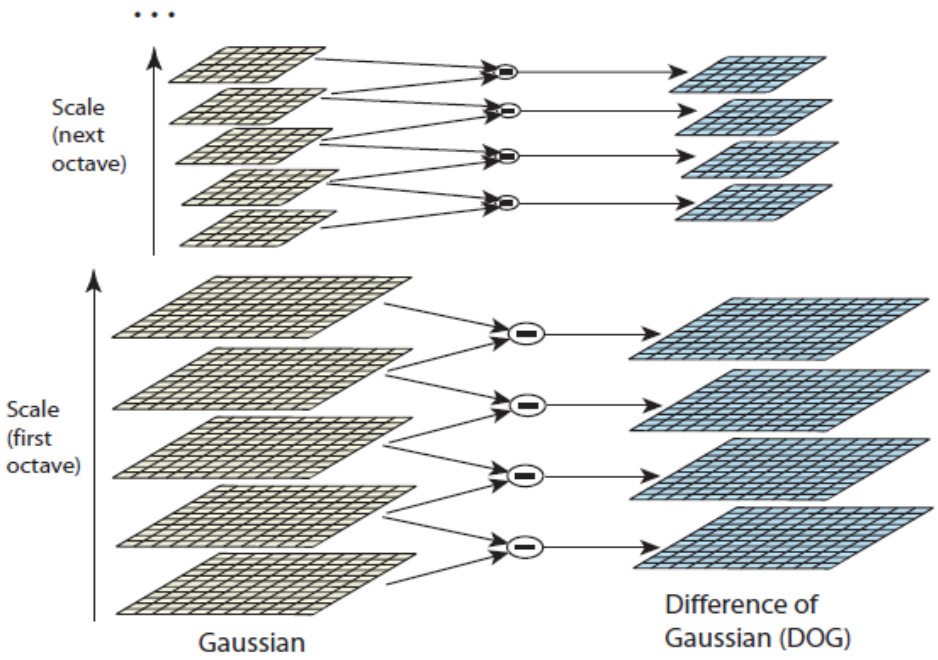
σ=1.0

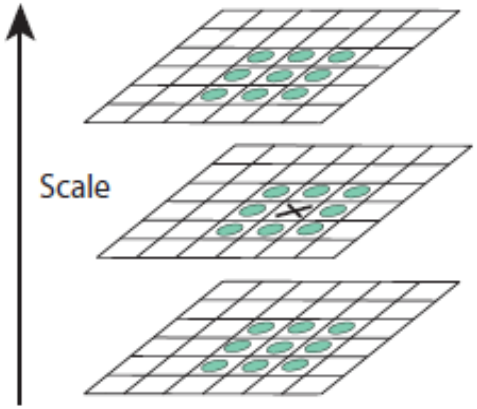
-

σ=1.4

=

Perform over **multi-scales** (octaves)





- DoG performed over multiple scales
- Consistent feature points identified
  as those present over multiple scales (from DoG) *[Lowe '04]*

Feature Point *Filtering*

*2. Find the **local maximal pixels** in space and scale*

(i.e. over σ, max of 3x3 neighbourhood)

*3. **Interpolate intermediate values***

(i.e. to get point location accurately)

*4. **Discard** feature points in **regions of low-contrast***

*5. **Compute Hessia**[illegible]**efore)***

*Compute **ratio R**:* $R = \frac{(trace(H))^2}{determinant(H)}$

(equiv. to ratio of eigenvalues of H)

***Reject** if R > (r th + 1) / r th) (r th = 10)*

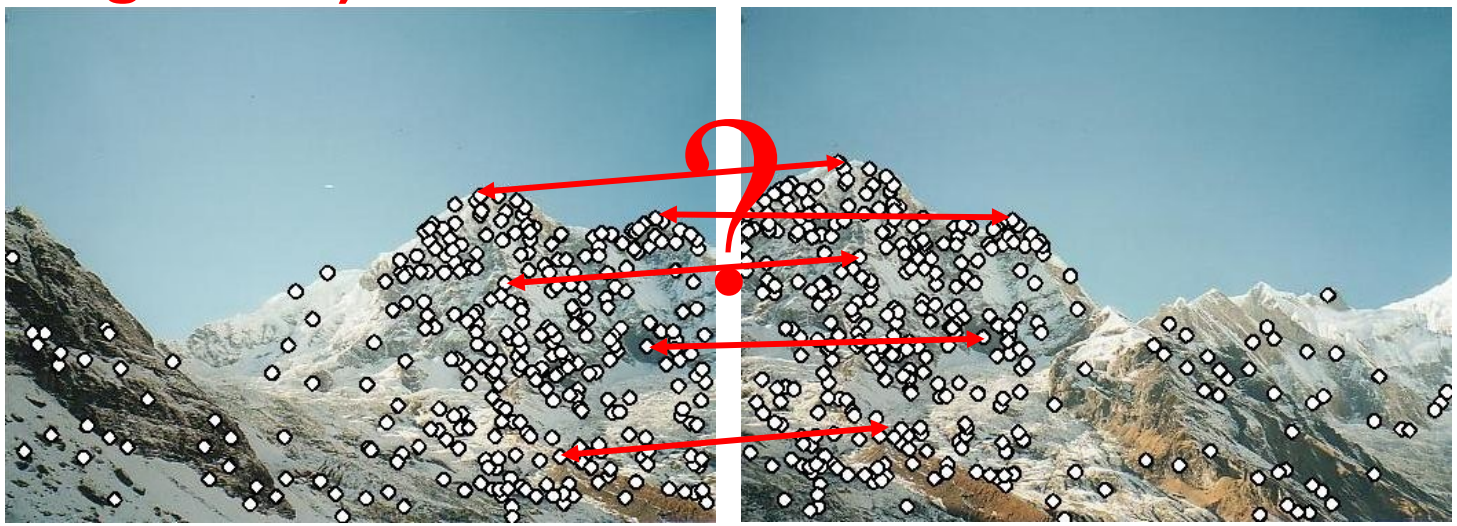
*Removes **poorly localised points** (varying "along/on edge" etc.)*

Image source:wikipedia

Feature Point ***Descriptors***

We know how to detect good points

Next question: **How to match them? (i.e. recognition)**



Lots of possibilities (this is a popular research area)

- Simple option: match square windows around the point
- State of the art approach: SIFT
  - David Lowe, UBC http://www.cs.ubc.ca/~lowe/keypoints/

Key Requirement - Invariance

Suppose we are comparing two images I_1 and I_2

- I_2 may be a transformed version of I_1
- What **kinds of transformations** are we likely to encounter in practice?

Want to find the same features regardless of the transformation

- This is called transformational ***invariance***
- Most feature methods are designed to be invariant to
  - Translation, 2D rotation, scale
- They can usually also handle
  - Limited 3D rotations (SIFT works up to about 60 degrees)
  - Limited affine transformations (some are fully affine invariant)
  - Limited illumination/contrast changes

Achieving **invariance** ....

Need both of the following:

1. Make sure your **detector is invariant**

   - Harris is invariant to translation and rotation
   - Scale is trickier .... use SIFT
     - common approach is to detect features at many scales using a Gaussian pyramid (e.g., MOPS)
     - more sophisticated methods find “the best scale” to represent each feature (e.g., SIFT)

2. Design an **invariant feature *descriptor***

   - A descriptor captures the information in a region around the detected feature point
   - The simplest descriptor: a square window of pixels
     - What’s this invariant to?
   - Let’s look at some better approaches…

Rotation invariance for feature descriptors

Find dominant orientation of the image patch

- This is given by $\mathbf{x}_+$, the eigenvector of **H** corresponding to

  λ_+ is the *larger* eigenvalue

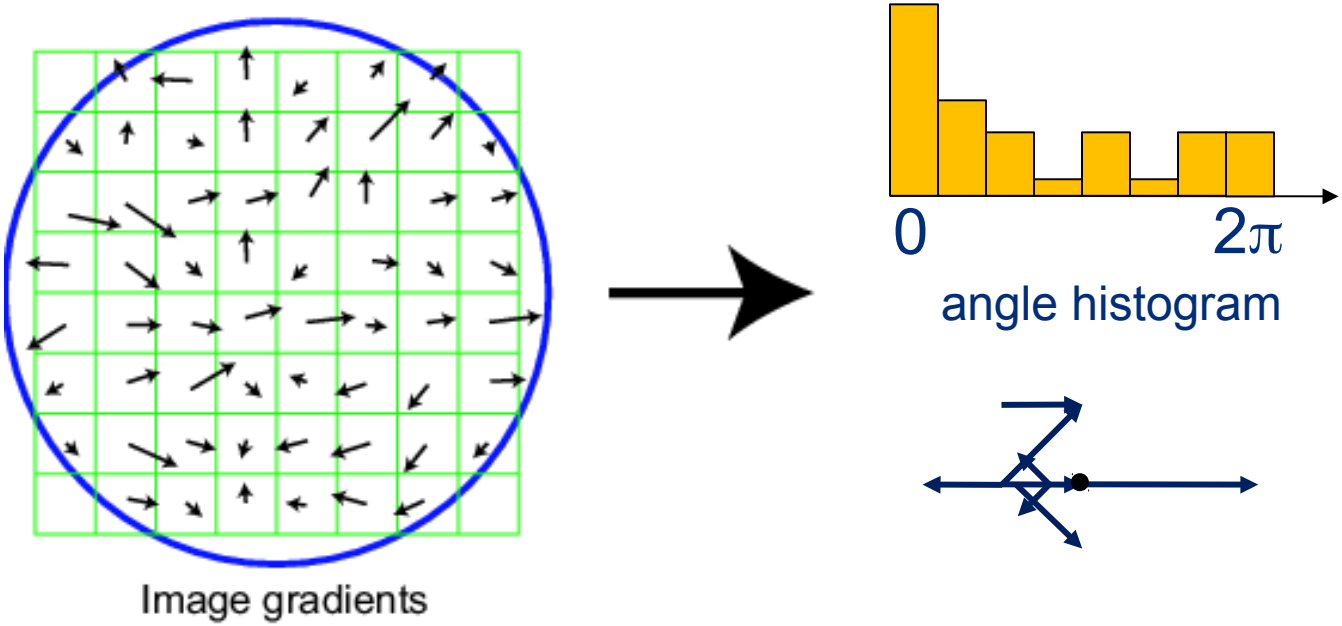
- Can rotate / align the descriptor image patch according to this angle

Image: Matthew Brown, UBC

Scale Invariant Feature Transform (SIFT)

Basic idea:

- Take 16x16 square window around detected feature
- Compute edge orientation (angle of the gradient - 90°) for each pixel
- Throw out weak edges (threshold gradient magnitude)
- Create histogram of surviving edge orientations



Adapted from slide by David Lowe, UBC.

SIFT descriptor

Full version

- Divide the 16x16 window into a 4x4 grid of cells (2x2 case shown below)
- Compute an orientation histogram for each cell
- 16 cells * 8 orientations = 128 dimensional descriptor
- Determine descriptor orientation as peak of histogram
  - (+ secondary peaks within 80% of primary peak)

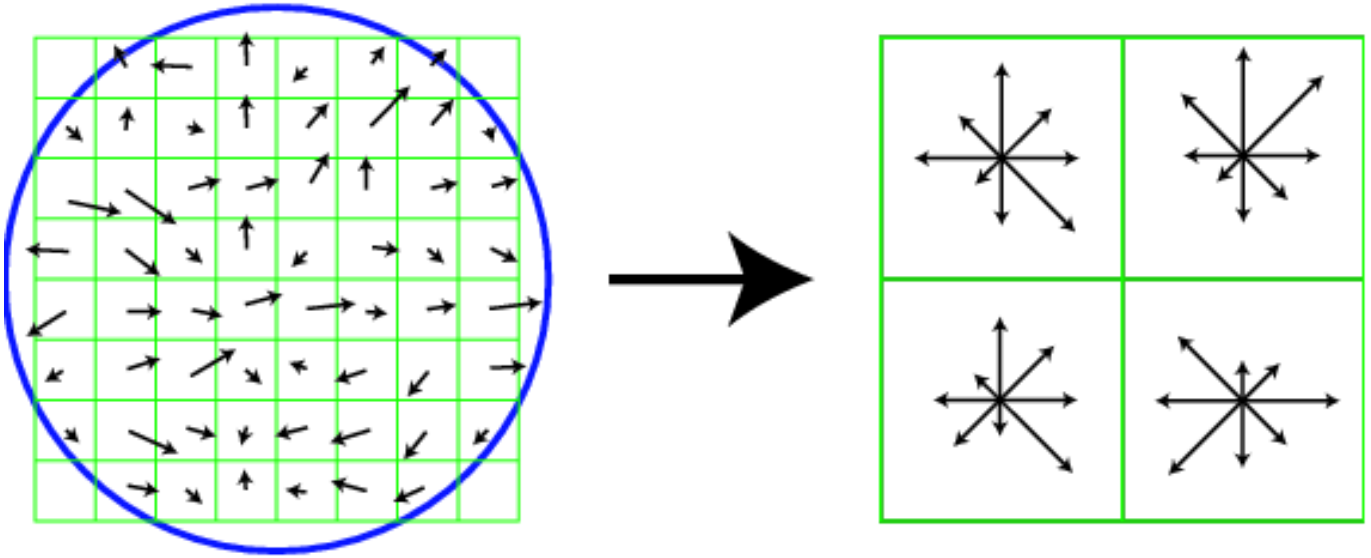


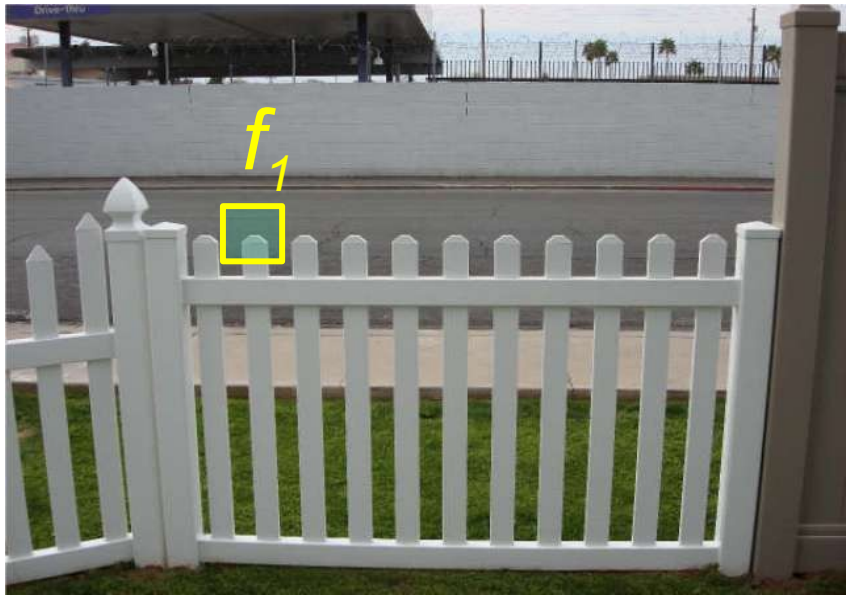
Image gradients

Keypoint descriptor

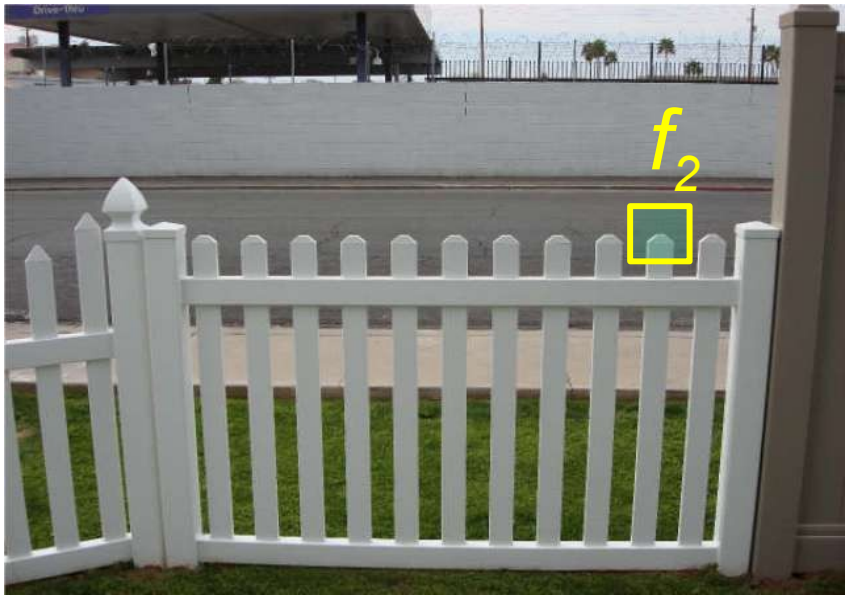
Adapted from slide by David Lowe, UBC.

SIFT **Feature Matching**

- Given a feature in I_1, how to find the best match in I_2?
  - Define **distance function** that compares **two descriptors**
  - **Test all the features** in I_2, **find the one with min distance**



I_1

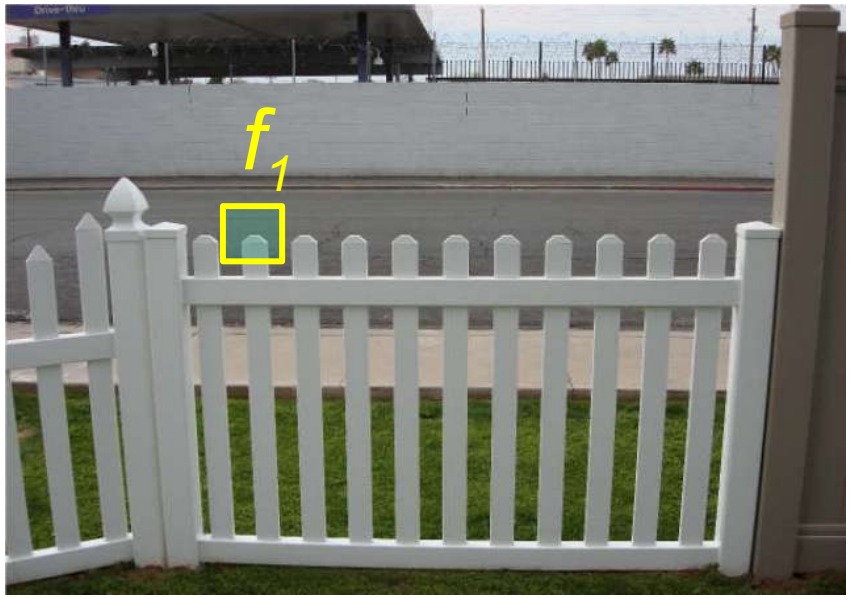


I_2

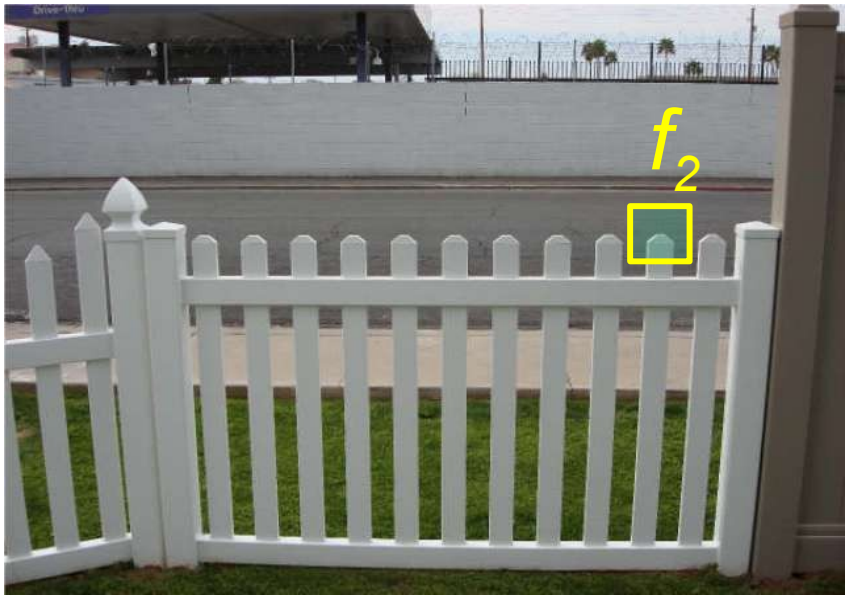
Feature Distance

How to define the difference between two features f_1, f_2?

- **Better approach:** ratio distance = $SSD(f_1, f_2) / SSD(f_1, f_2')$
  - f_2 is best SSD match to f_1 in I_2
  - f_2' is 2nd best SSD match to f_1 in I_2
  - gives small values for ambiguous matches



I_1



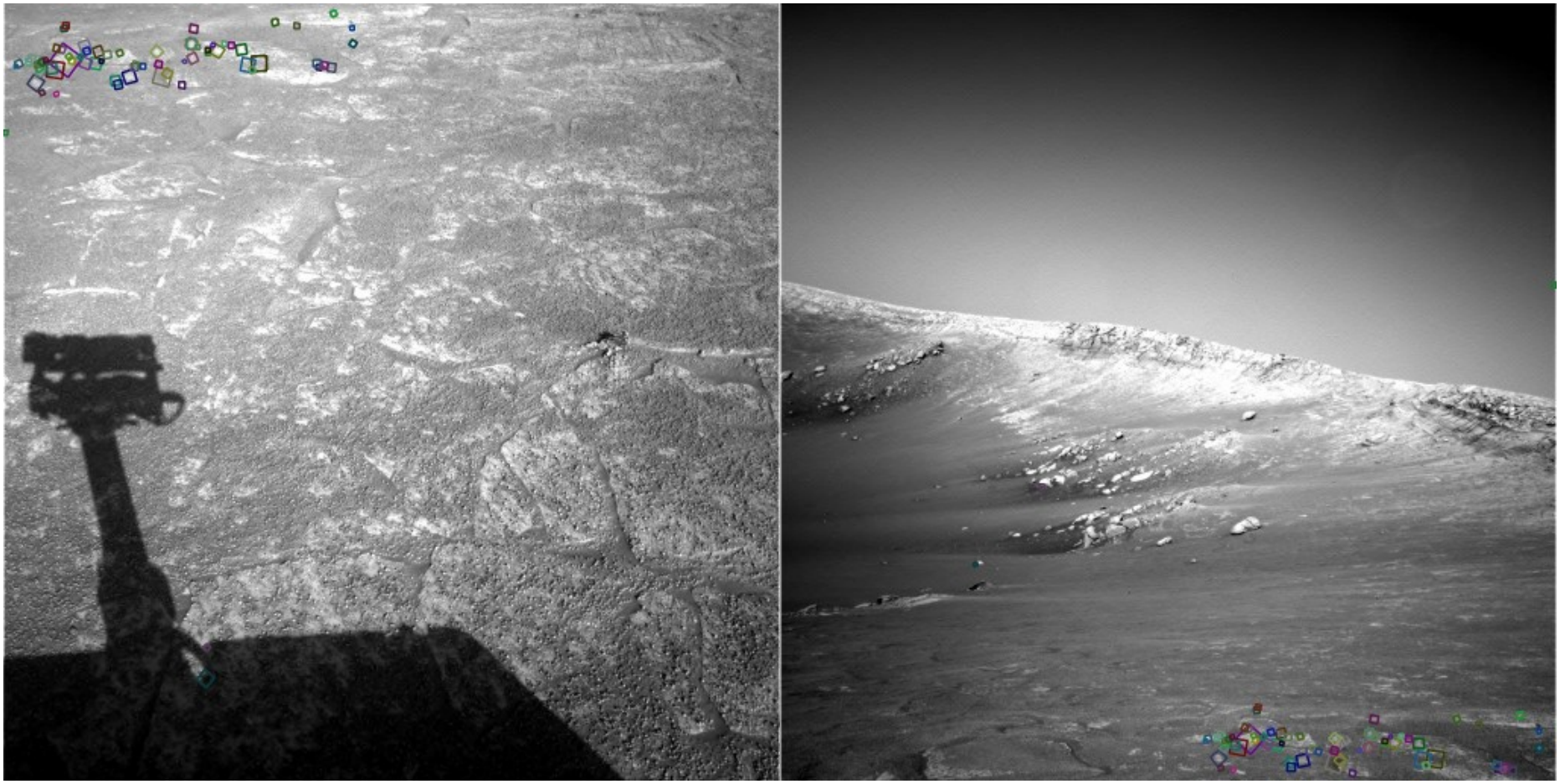
I_2

SIFT **Feature Distance**

- **Efficiency**
  - can compare features as N-Dimensional vectors in R^N using k-D trees (**nearest neighbour search**)
    - Query f_1 to f_j reponse in linear time
  - several optimisations on this approach

- **Original SIFT approach** *[Lowe 2004]*
  - variation on k-D tree approach
  - **probability of match correct = ratio of 1st nearest match to 2nd nearest match**
    - Reject all matches with ratio > 0.8
    - Effect = eliminates ~90% of false matches, discards ~5% of correct matches, only very "unique" matches are kept

SIFT *Feature Match Examples*



NASA Mars Rover images with SIFT feature matches
(Figure by Noah Snavely)

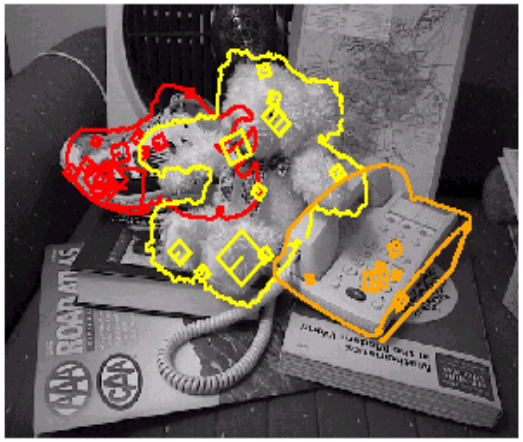
SIFT *Feature Match Examples*

Extraordinarily robust matching technique

- **changes** in **viewpoint**
  - Up to about 60 degree out of plane rotation
- significant **changes in illumination**
  - Sometimes even day vs. night (below)
- **Fast and efficient**—can run in (near) real time

Detecting **an** Object – Method 1

- Select a subset of 3 matches
  - RANSAC on matches
- Estimate Object Pose
  - Least Squares Method of lecture 1 between 2D points matches (instead of 2D edge matches)
- Verification of Pose
  - Apply transformation of object to image and test number of SIFT point matches within pixel distance $< d_t$



(Images: David Lowe, UBC)

Detecting **an** Object – Method 2

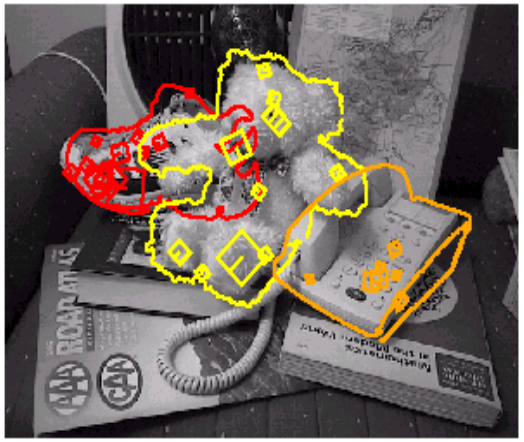
- Each SIFT feature match specifies a potential

  {position | rotation | scale}

- Use a *Hough Transform* based voting method to identify clusters of feature matches pointing to consistent

  {position | rotation | scale}

- Estimate and verify pose of top N clusters as per Method 1



(Images: David Lowe, UBC)

Detecting **Classes of** Objects

- **Cluster Features** in R^n space
  - over *(lots of)* example *(training)* images
  - K-means clustering (need to pick k!)
- Cluster *"membership"* for a given object example creates a **histogram of feature occurrence**
- Use **histogram of feature occurrence** as inputs to a machine learning classification algorithm
  - *(Support Vector Machines)*

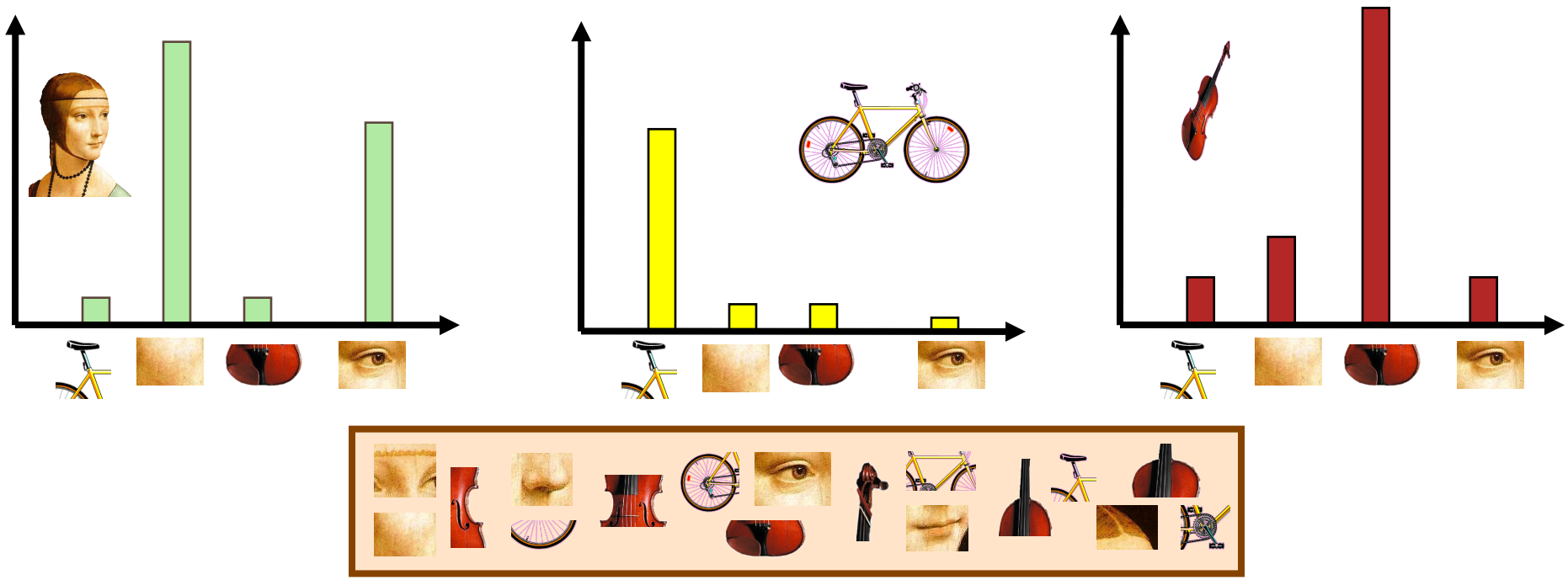
“Bag of (Visual) Words” (BoVW / BOW) Recognition

Object → **Bag of ‘words’**

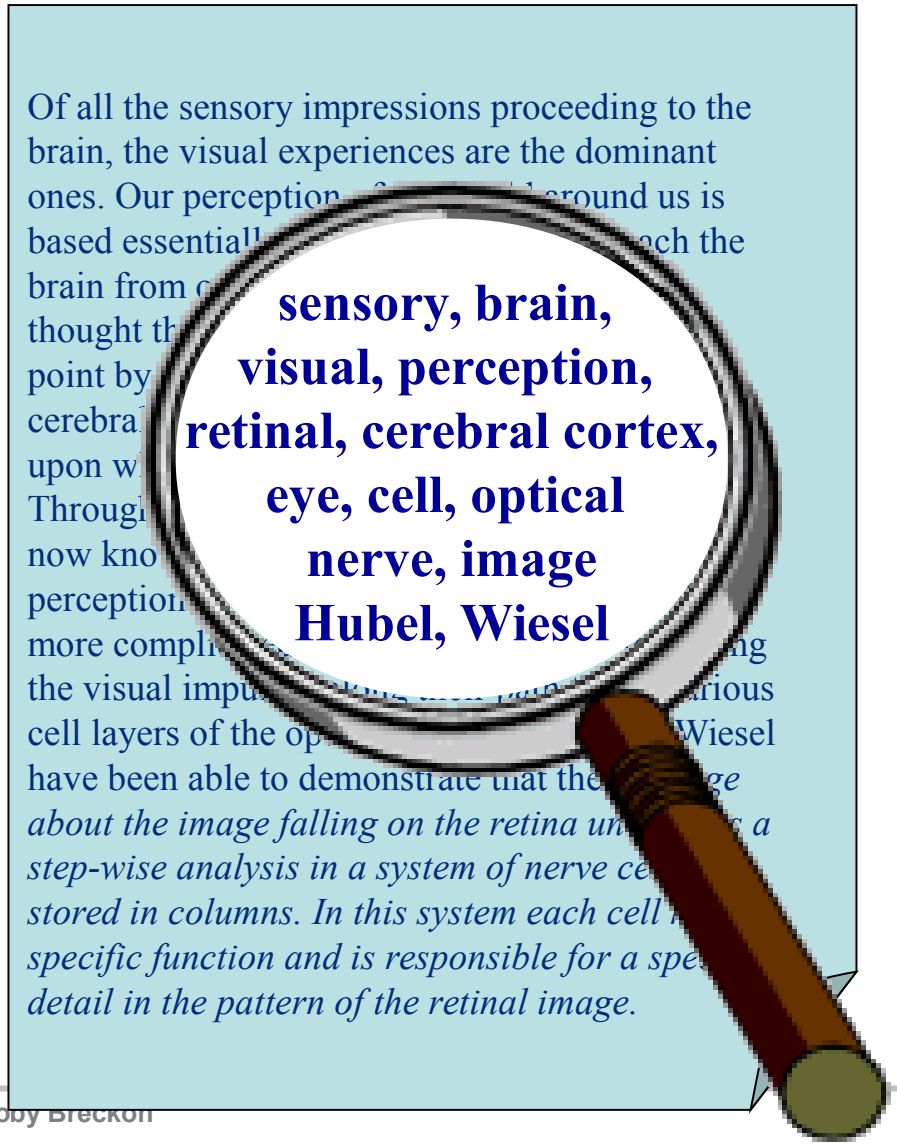
A clarification: definition of “BoVW”

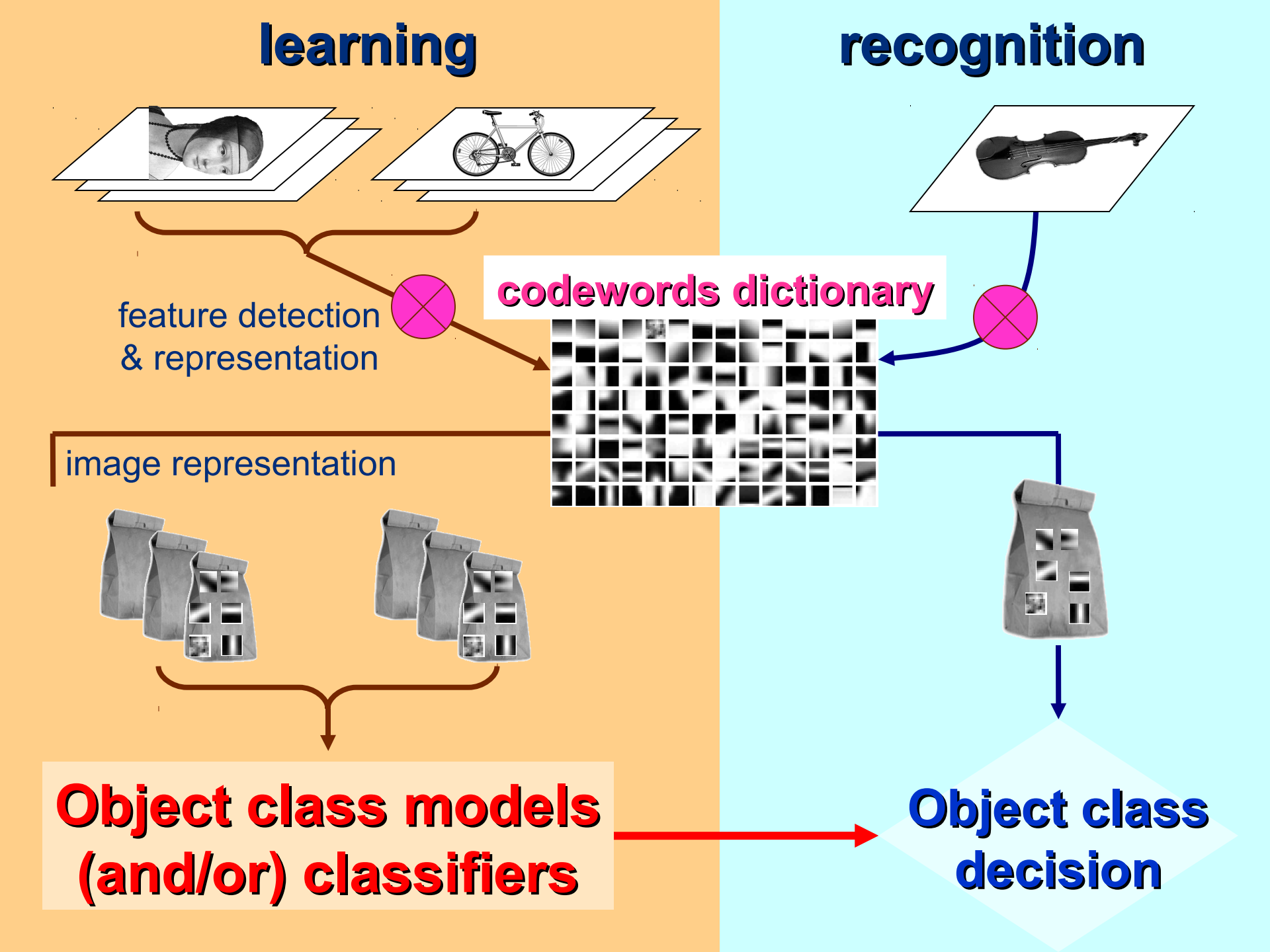
Strict definition ***(a.k.a. “bag of features)***

- **Independent features**
- **histogram representation**

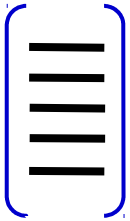


Analogy to documents



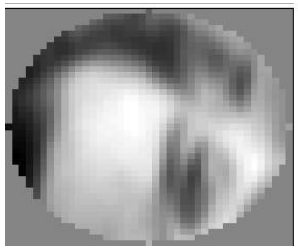

learning
recognition
feature detection
& representation
codewords dictionary
image representation
Object class models
(and/or) classifiers
Object class
decision

BoVW : Stage 1 - Feature detection and representation



Compute descriptor

e.g. **SIFT** [Lowe'99]



Normalize patch

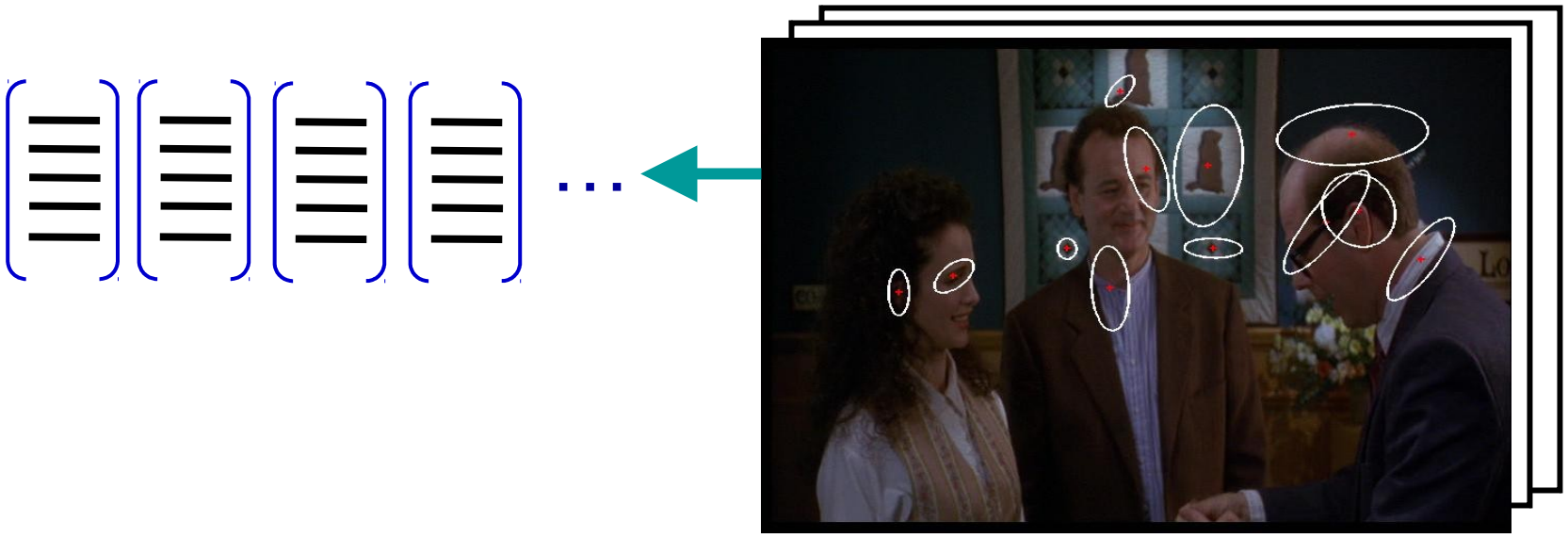
Detect patches

[Harris Feature Points / SIFT]

Local interest operator (Harris)
or
Regular grid (every N pixels in X and Y)

Slide credit: Josef Sivic

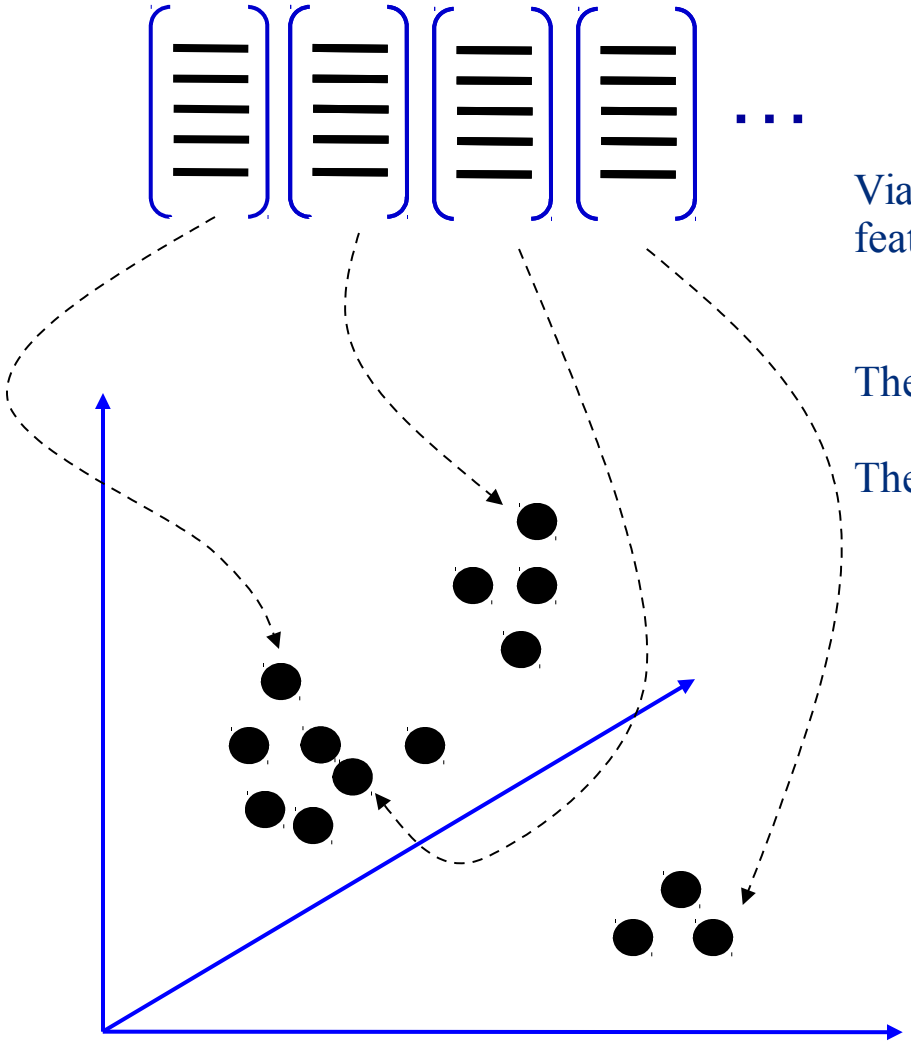
BoVW : Stage 1 - Feature detection and representation



Over multiple image ….. (100s / 1000s +)

Slide credit: Josef Sivic

BoVW : Stage 2 - Codewords dictionary formation



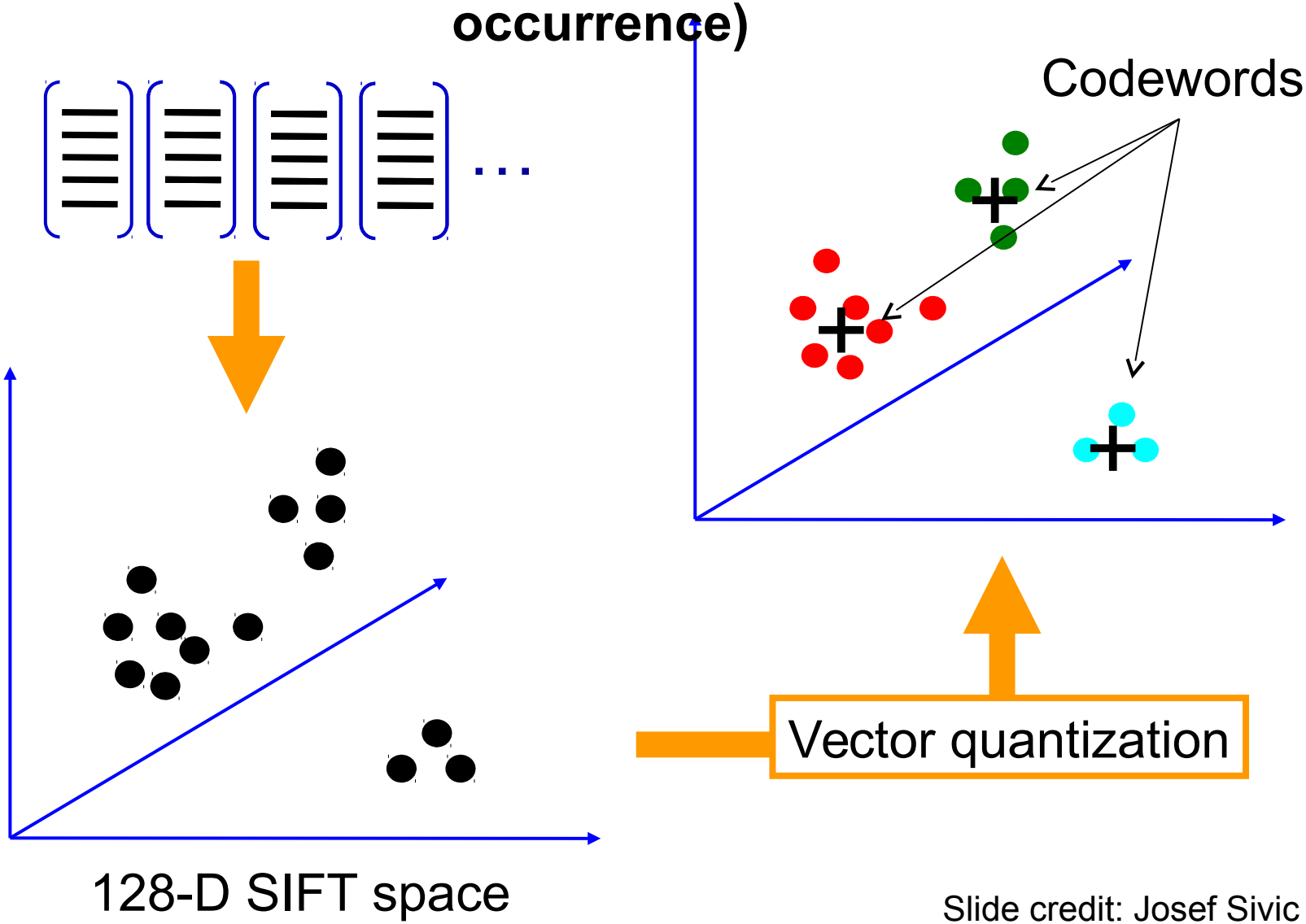
Via *k*-means clustering to find *k* clusters of feature descriptors
(user defines/choose – k)

The **centre** of each cluster **forms the codewords.**

The **set** of codewords is **known as the dictionary.**

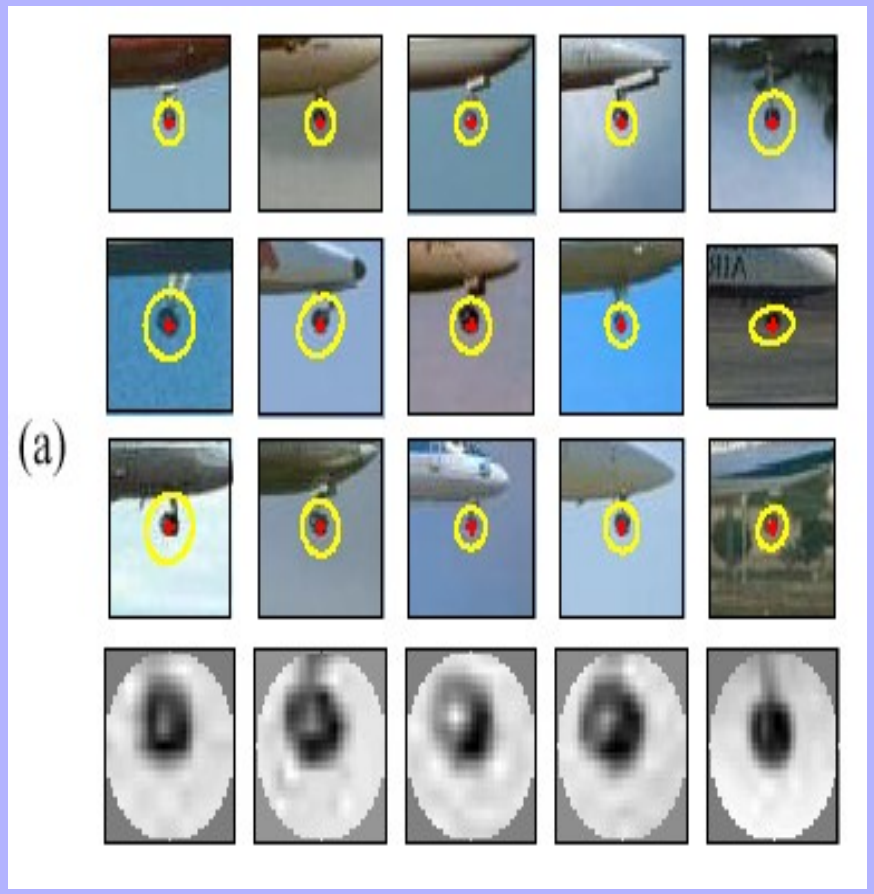
Slide credit: Josef Sivic

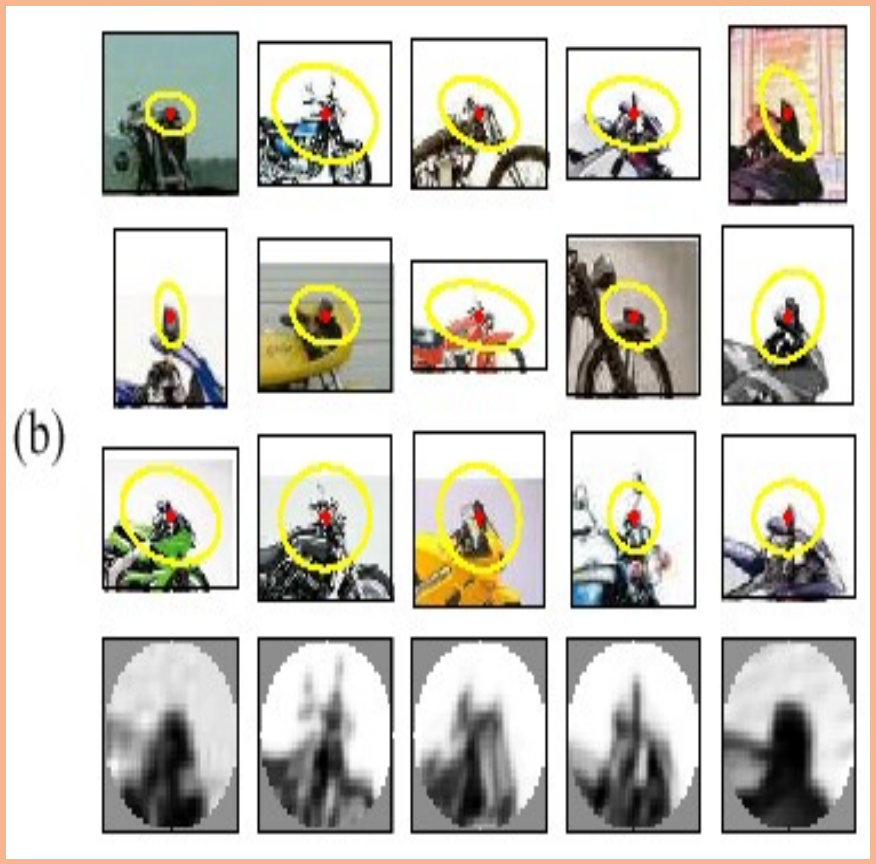
BoVW : Stage 3 - Encode images as occurrences of codewords (forming histogram of feature occurrence)



Slide credit: Josef Sivic

Can visualize codewords as image patches in examples

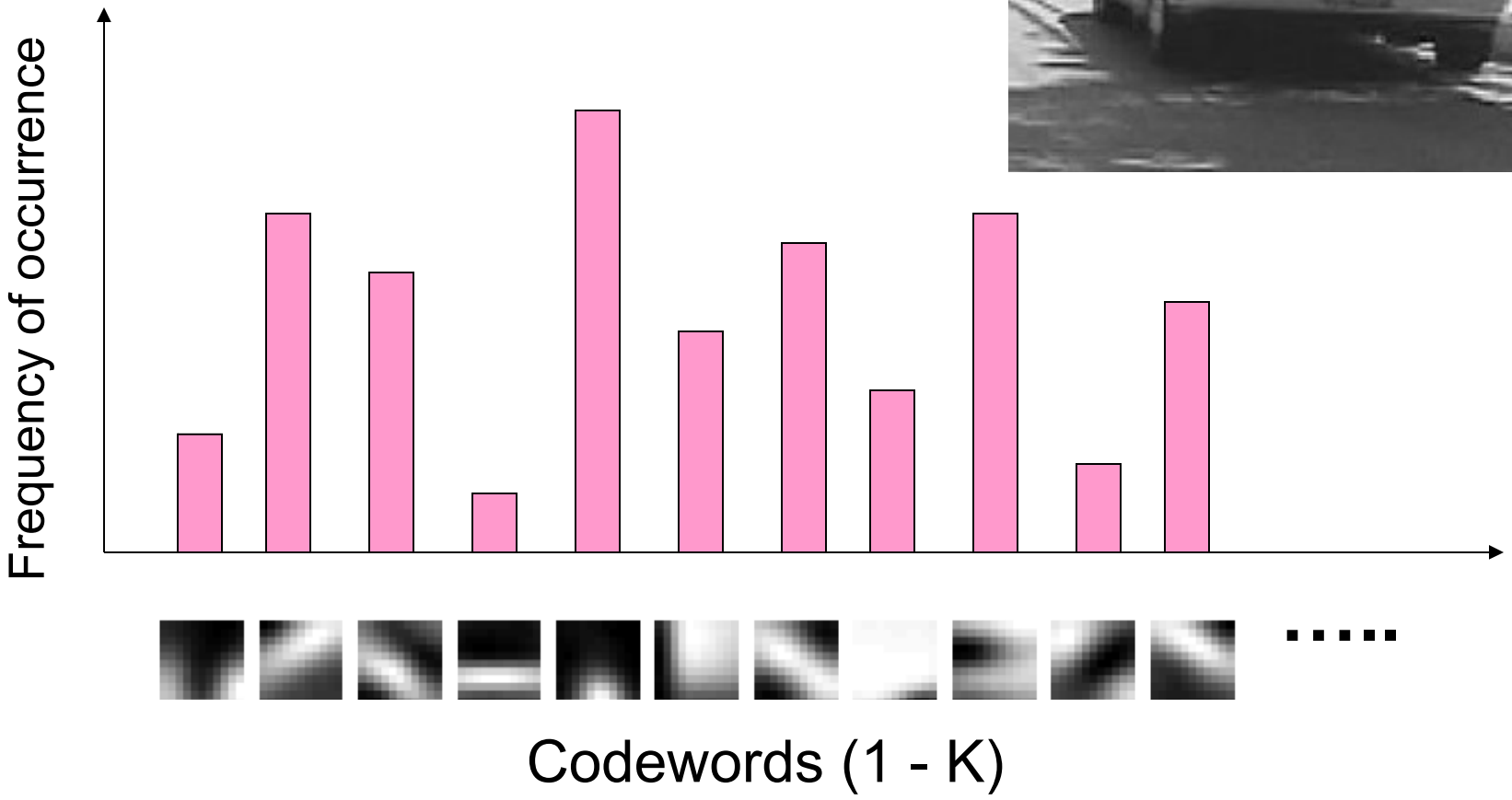




Sivic et al. 2005

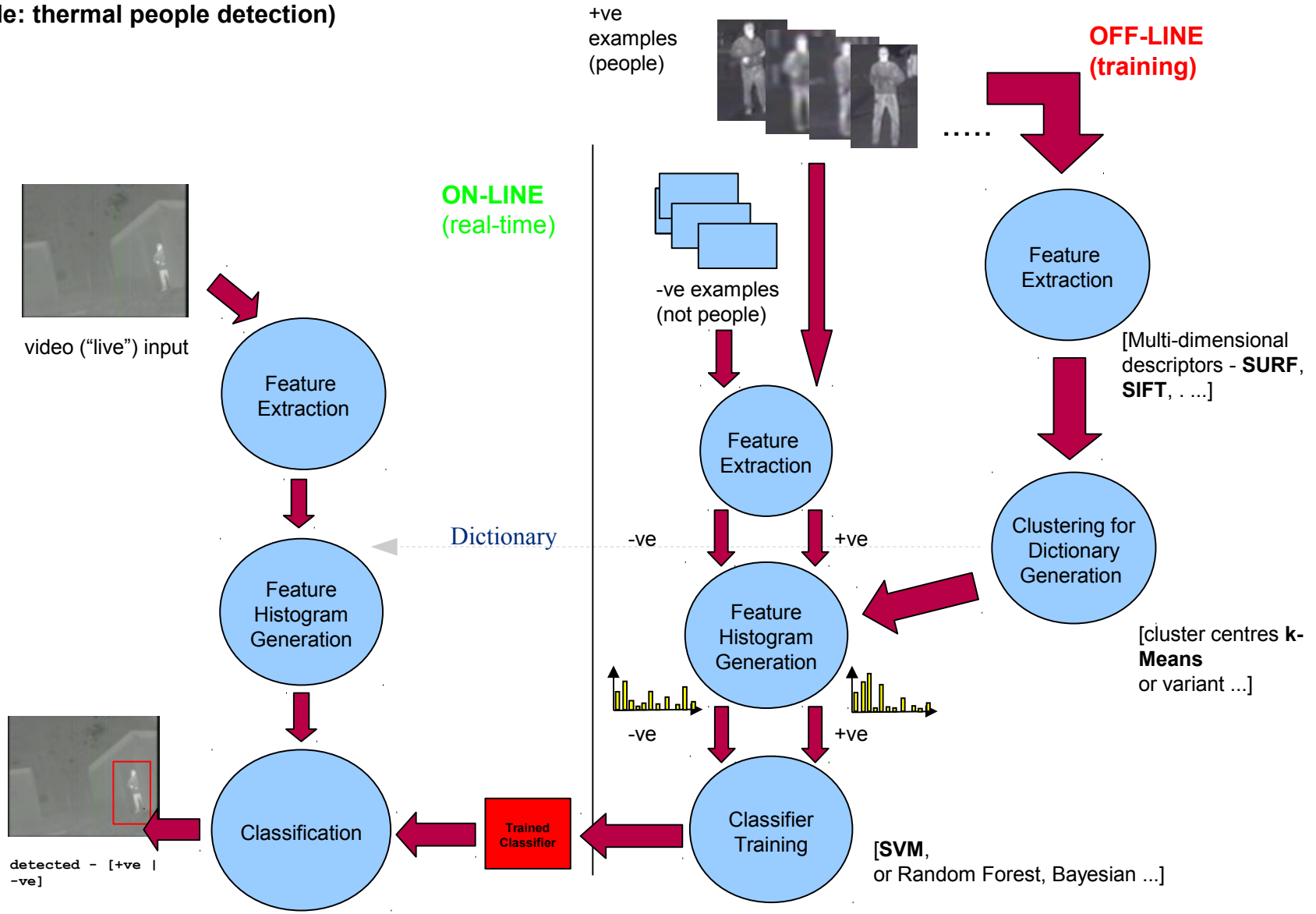
BoW : Stage 4 BoVW Image representation

Histogram of features
assigned to each cluster



BoVW : Complete Architecture

(example: thermal people detection)



Uses of BoVW representation

- Treat as feature vector for standard classifier
  - e.g SVM
- Cluster BoW vectors over image collection
  - Discover visual themes (via clustering)
- Hierarchical models
  - Decompose scene/object into parts