Feature Extraction and Description

Outline

- **Feature Extraction Overview**
- Color Features
- Local Features
  - Edges
  - Corners
  - Interests Point
- Visual Bag of Words

Element of Image Analysis

Preprocess

Image Acquisition, Enhancement, and Restoration

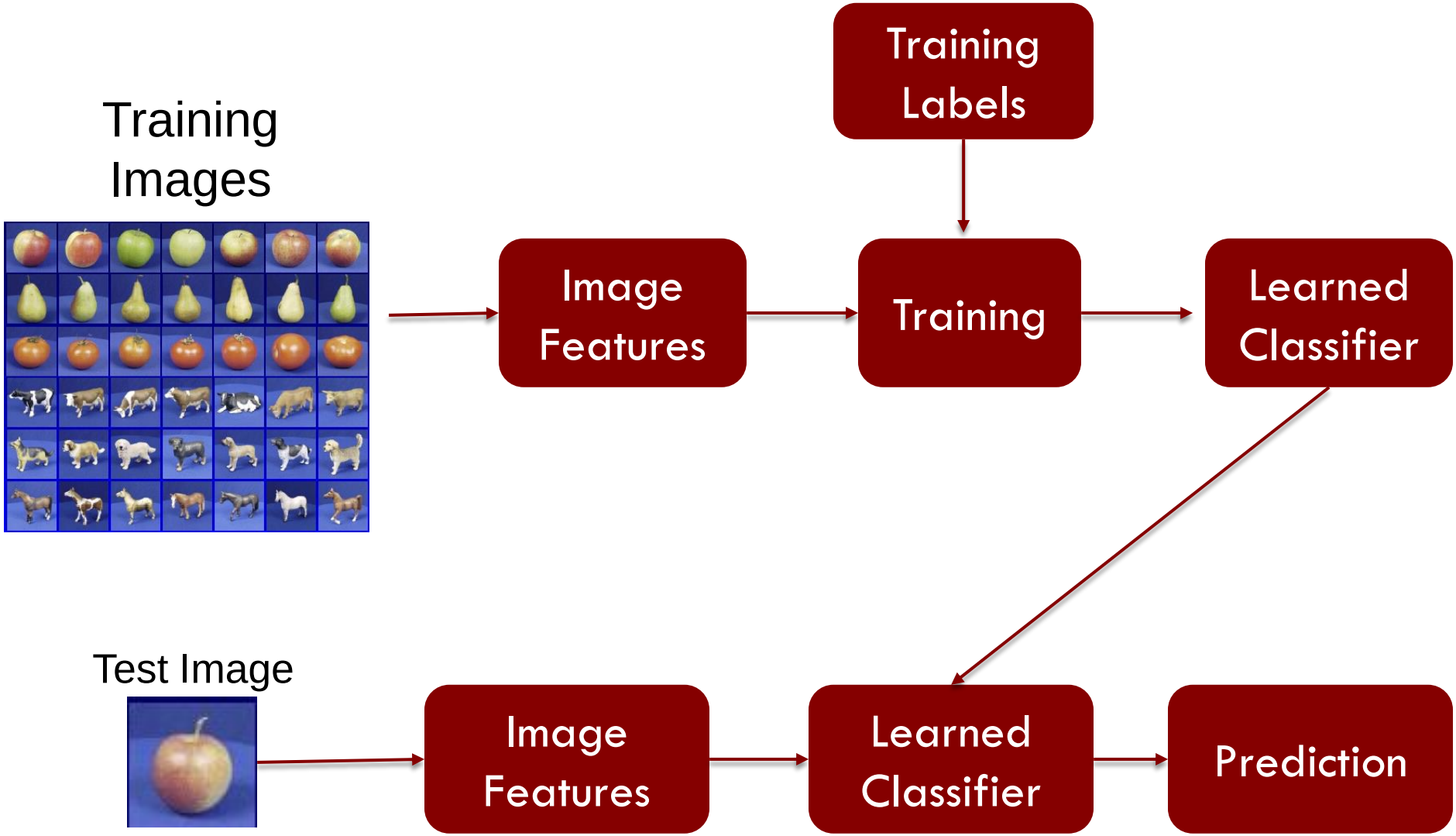
Intermediate process

Feature extraction & Image segmentation

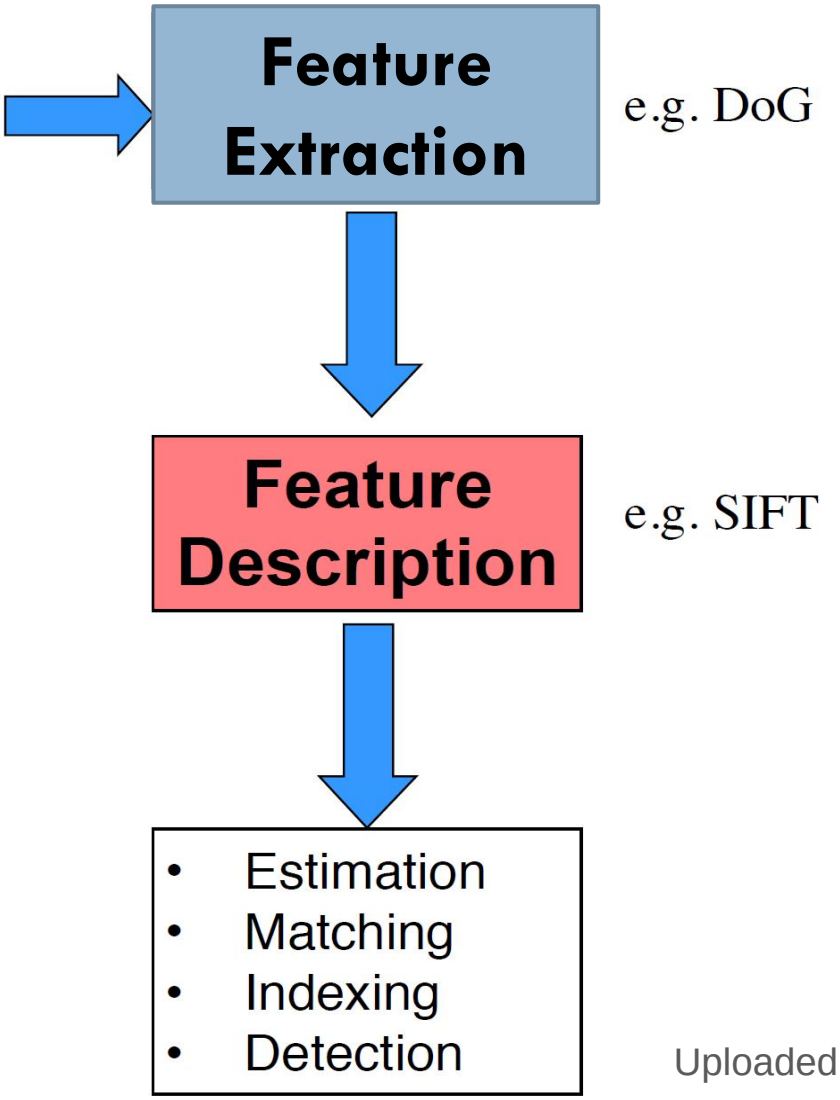
High level process

Image interpretation and recognition

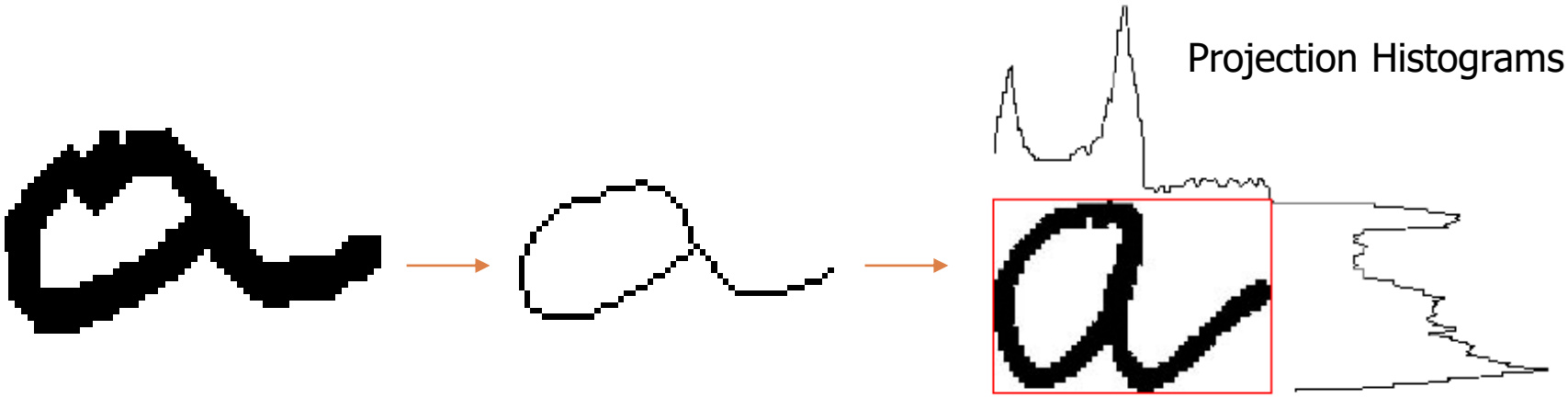
Image Classification Pipeline

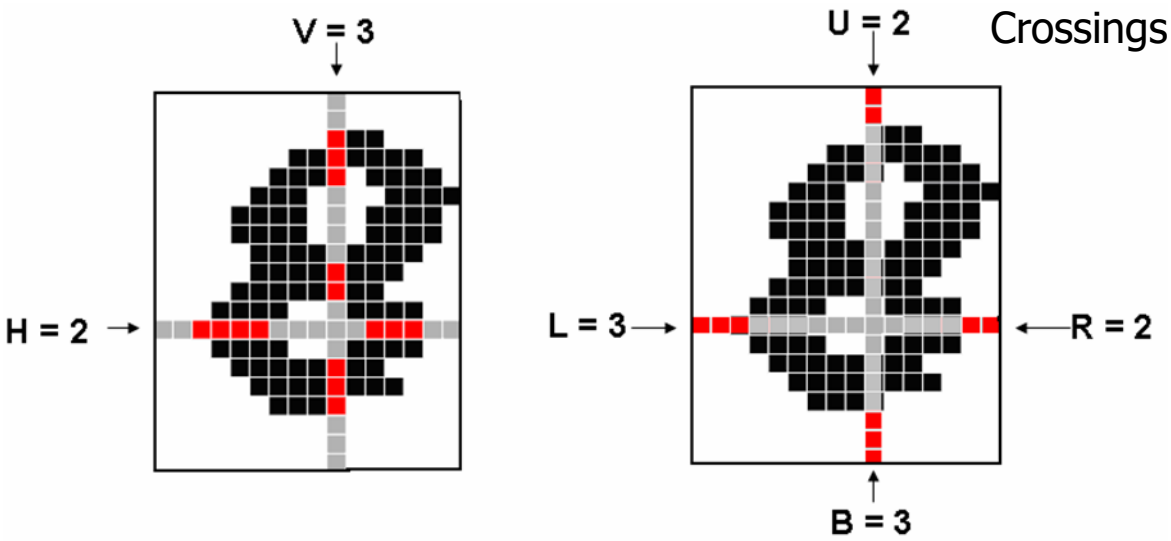


The big picture…



How to represent images?





Features Extraction

- Feature = “point of interest” for image description
  - Features should contain information required to distinguish between classes and should be insensitive to irrelevant variability in the input

- Main goal of feature extraction
  - Obtain the most relevant information from the original data
  - Represent that information in a lower dimensionality space.

Features Extraction Classification

- **General features**: Application independent features such as **color, and texture**.
- **Application dependent features:** such as human faces, fingerprints, Characters, and conceptual features.

- On the other hand, features can be coarsely classified into:
  - **Low-level features:** features can be extracted directed from the original images such as Edges, Corners, and Interest points
  - **High-level features:** high-level feature extraction must be based on low level features such as Shape, Template Matching

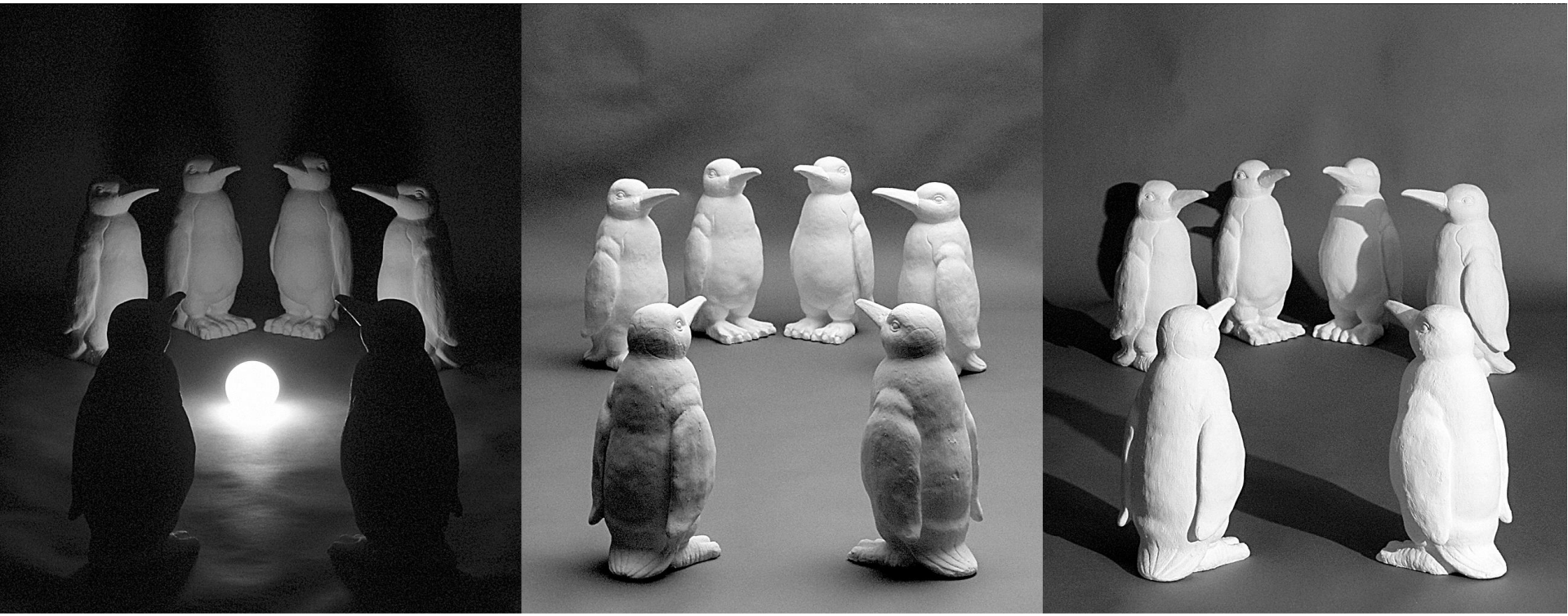
Features Extraction methodes

- **Hand-Crafted Features:**
  - Hand-crafted features involve the manual design of specific image descriptors by experts in the field.
  - These features are crafted based on domain knowledge and understanding of the problem at hand.
  - Examples include gradient-based features (e.g., Histogram of Oriented Gradients - HOG), texture descriptors, and color histograms.
- **Learning-Based Features:**
  - Learning-based features involve the automatic extraction of features from data using machine learning algorithms, such as neural networks.
  - Features are learned directly from the data, allowing the model to adapt to the task without the need for manual feature engineering

Features Extraction Challenges

- Variability in Scale and Orientation
- Occlusion
- Deformation
- Illumination and Lighting Conditions
- Noise and Distortions
- Complex Backgrounds
- Domain-Specific Challenges
- Interclass and Intraclass Variability
- Real-Time Constraints
- …

Challenges: illumination



Challenges: viewpoint variation

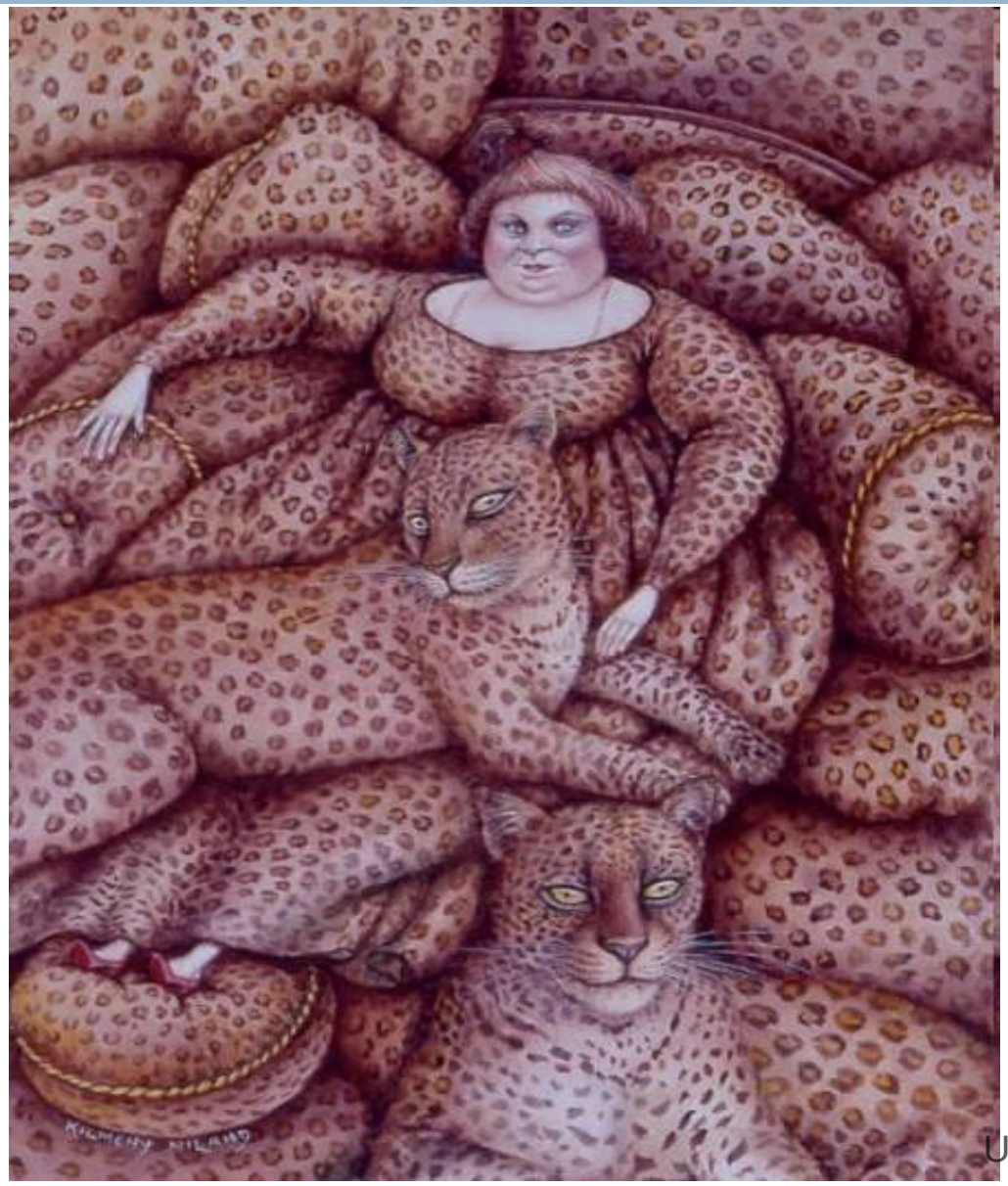


Challenges: scale

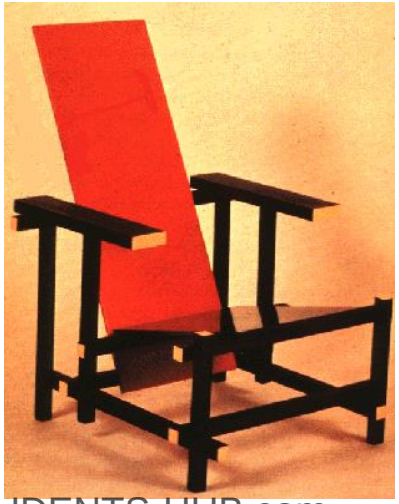
Challenges: deformation

Challenges: occlusion

Challenges: background clutter



Challenges: intra-class variation



Characteristics of good features

- **Identifiability**: shapes which are found perceptually similar by human have the same feature different from the others.
- **Repeatability**: The same feature can be found in several images despite geometric (**Translation, rotation and scale invariance)** and photometric **(Intensity)** transformations
- **Noise resistance**: features must be as robust as possible against noise, i.e., they must be the same whichever be the strength of the noise in a give range that affects the pattern.
- **Occultation invariance**: when some parts of a shape are occulted by other objects, the feature of the remaining part must not change compared to the original shape.
- **Statistically independent**: two features must be statistically independent. This represents compactness of the representation.
- **Reliable**: as long as one deals with the same pattern, the extracted features must remain the same.

What is the best method for feature extraction?

- It all depends on your application at hand.
- Few things you should keep in mind are:
  - Feature extraction is highly subjective in nature
  - There is no generic feature extraction scheme which works in all cases.
  - What kind of problem are you trying to solve? e.g. classification, detection, etc.
  - Do you have a lot of data?
  - Do your data have very high dimensionality?
  - Is your data labelled?
  - Do you want to use a very computationally intensive method or something rather inexpensive?

Outline

- Feature Extraction Overview
- **Color Features**
- Local Features
  - Edges
  - Corners
  - Interests Point
- Visual Bag of Words

Color Features

- The color feature is one of the most widely used visual features in image retrieval.
- Images characterized by color features have many advantages:
  - **Robustness**. The color histogram is invariant to rotation of the image on the view axis, and changes in small steps when rotated otherwise or scaled
  - **Effectiveness**. There is high percentage of relevance between the query image and the extracted matching images.
  - **Implementation simplicity**. The construction of the color histogram is a straightforward process
  - **Computational simplicity**. The histogram computation has O(X, Y) complexity for images of size X × Y .

Color Features

- Color features are defined subject to a particular color space or model.
- A number of color spaces have been used in literature, such as RGB, HSI, etc.
- Once the color space is specified, color feature can be extracted from images or regions.
- A number of important color features have been proposed in the literatures, including:
  - **Color histogram**
  - **Color moments(CM)**
  - Color coherence vector (CCV)
  - Color correlogram, *etc.*

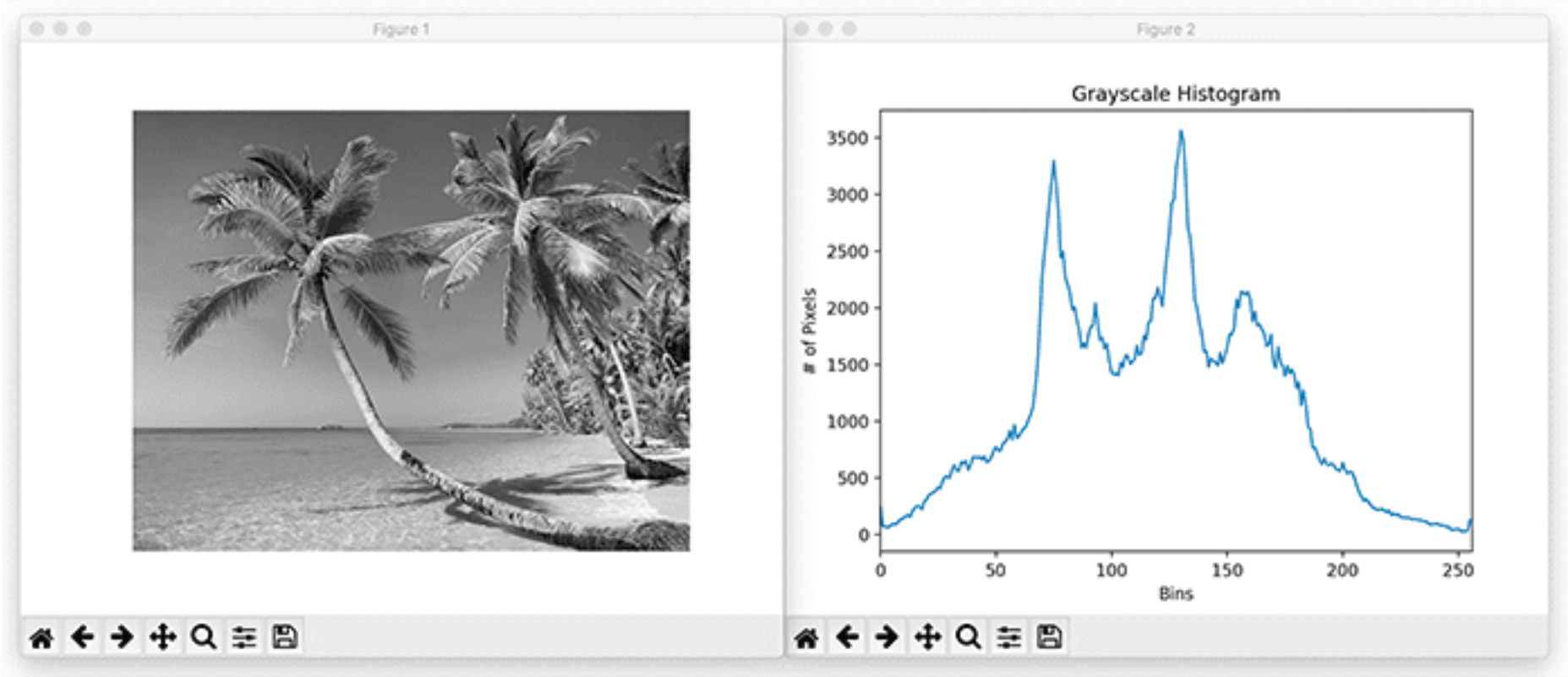
Color histogram

- A color histogram H for a given image is defined as a vector H = {h[1], h[2], . . . h[i], . . . , h[N]}
  - Where **i** represents a color in the color histogram,
  - **h[i]** is the number of pixels in color i in that image,
  - and **N** is the number of bins in the color histogram, i.e., the number of colors in the adopted color model.
- In order to compare images of different sizes, color histograms should be normalized.
- Can be used to Measures the similarity of images, speech, music, …
- Issue: how to capture perceptual similarity of an image

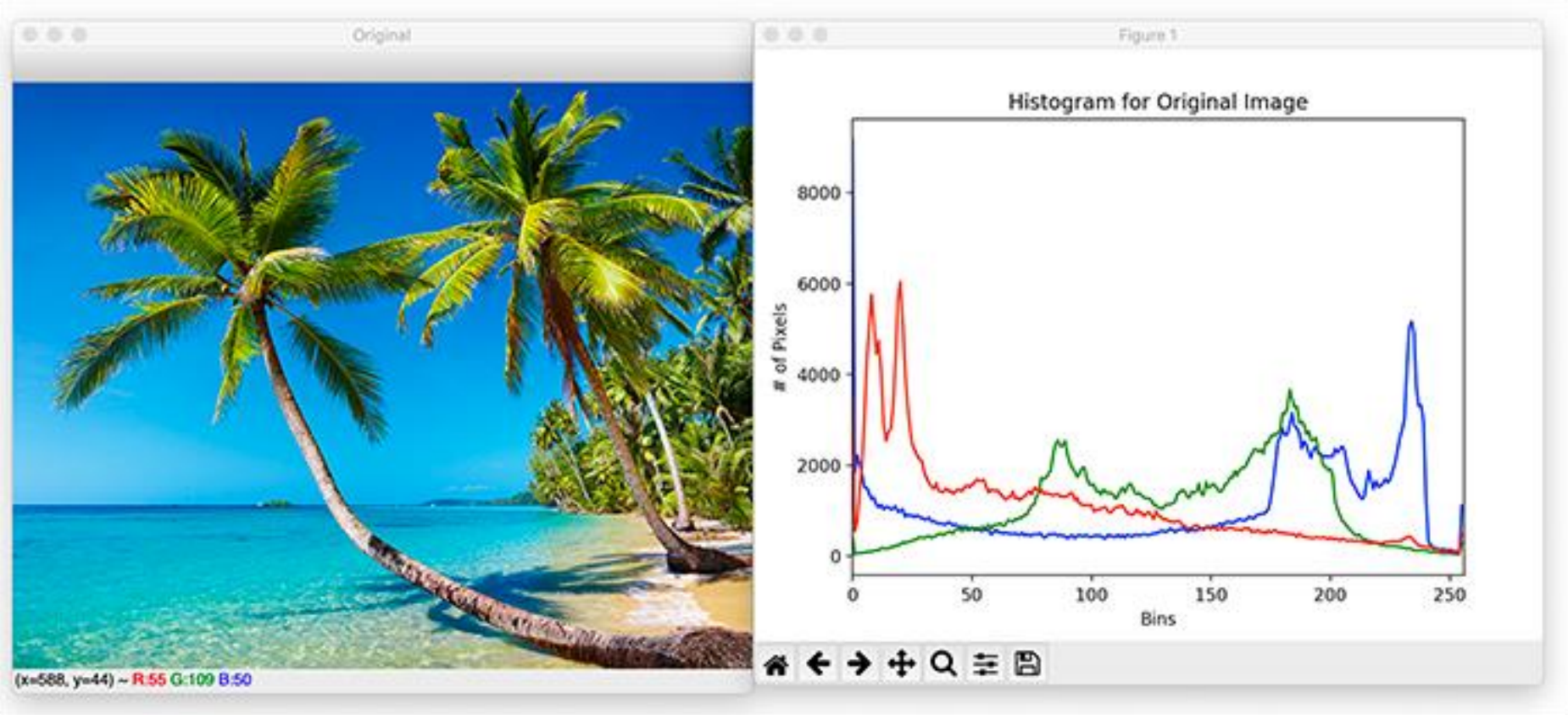
Color histogram

- The standard measure of similarity used for color histograms:
  - A color histogram H(i) is generated for each image h in the database (feature vector),
  - The histogram is *normalized so that its sum equals* unity (removes the size of the image),
  - The histogram is then stored in the database,
  - Now suppose we select a *model image (the new* image to match against all possible targets in the database).

Color histogram: an Example



Color histogram: an Example



Color Histogram: Pros

- **Simplicity:** Color histograms are simple and easy to compute, making them computationally efficient.
- **Interpretability:** Color histograms provide a straightforward representation of the color distribution in an image, making them interpretable and easy to understand.
- **Invariance to Rotation and Translation:** Color histograms are generally invariant to simple transformations like image rotation and translation.
- **Applicability to Image Retrieval:** Color histograms are commonly used in content-based image retrieval systems due to their simplicity and effectiveness in representing color information.

Building Image Retrieval with Color Histogram

1. **Image Preprocessing:** Convert images to a suitable color space, Normalize pixel values,
2. **Color Histogram Calculation:** Divide color space into bins, Count pixels in each bin for each channel, Normalize histogram values.
3. **Feature Representation:** Concatenate or flatten individual histograms into a feature vector.
4. **Database Construction:** Store feature vectors and image identifiers in a database.
5. **Query Image Processing:** Preprocess the query image.
6. **Color Histogram Calculation for Query Image:** Calculate the color histogram for the query image.
7. **Similarity Measurement:** Measure similarity between query histogram and database histograms.
8. **Ranking and Retrieval:** Rank images based on similarity scores. Retrieve top-ranked images as results.
9. **Presentation of Results:** Display retrieved images to the user, ranked by similarity.

Histogram distance measures

L_1 distance (Manhattan distance)

$$d_1(H,L) = \sum_{k=1}^{K} | h_k - l_k |$$

L_2 distance (euklidian distance)

$$d_2(H,L) = \sqrt{\sum_{k=1}^{K} | h_k - l_k |^2}$$

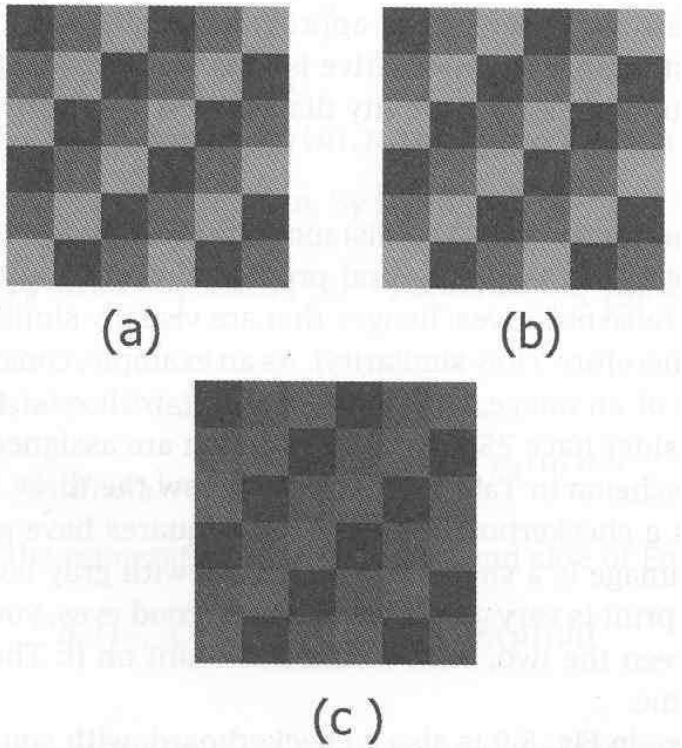
L_∞ distance (maximum distance)

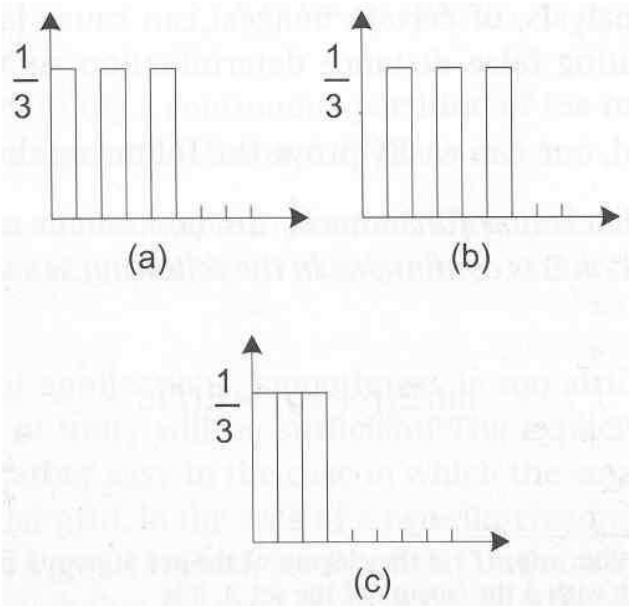
$$d_\infty(H,L) = \max_k (| h_k - l_k |)$$

Color Histogram: Cons

- **Lack of Spatial Information:** Color histograms do not consider spatial relationships between pixels, potentially limiting their performance in tasks requiring spatial information.
- **Insensitive to Local Changes:** They may not capture subtle local color changes or patterns, particularly in regions with small variations.
- **Vulnerability to Illumination Changes:** Color histograms are sensitive to changes in illumination, and variations in lighting conditions can significantly impact the feature.
- **Limited Discriminative Power:** In scenarios where color alone is not highly discriminative, such as when objects have similar color distributions, color histograms may struggle to distinguish between them.

Example for potential problem with histogram distance

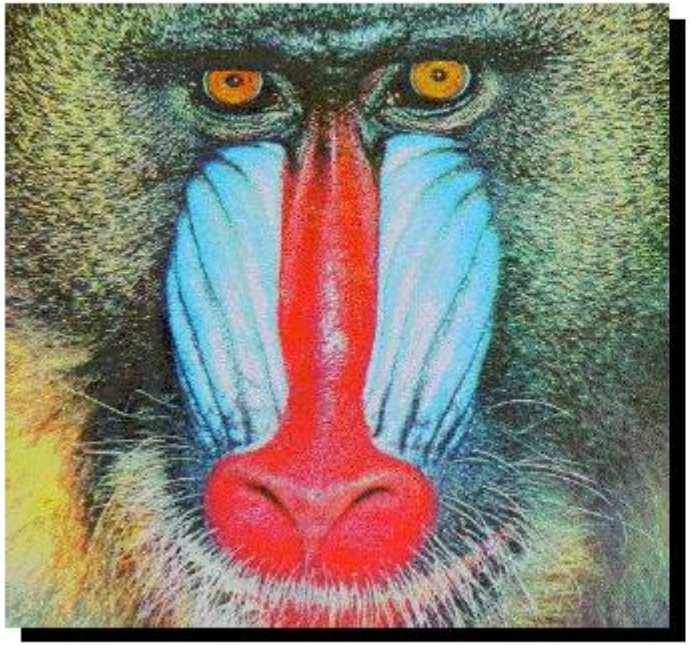




Distance type	d(a,b)	d(a,c)
L_1	2	~0.67
L_2	~0.82	~0.47
L_∞	~0.33	~0.33

None of the distances captures perceptual similarity

Another Issue: loss of regional information



Partition the image
One histogram per region

Color Moments

- Provide measurement for color similarity between images.
- These value of similarity can then be compared to the values of the images indexed in a database for tasks like image retrieval.
- The basis of color moments lays in the assumption that the distribution of color in an image can be interpreted as a probability distribution.
- Probability distributions are characterized by a number of unique moments (e.g. Normal distributions are differentiated by their mean and variance).
- It therefore follows that if the color in an image follows a certain probability distribution, the moments of that distribution can then be used as features to identify that image based on color.
- The first order (mean), the second (variance) and the third order (skewness) color moments have been proved to be efficient and effective in representing color distributions of images.

Color Moments

MOMENT 1 – Mean :

$$E_i = \sum_{N}^{j=1} \frac{1}{N} p_{ij}$$

Mean can be understood as the average color value in the image.

MOMENT 2 - Standard Deviation :

$$\sigma_i = \sqrt{\left(\frac{1}{N} \sum_{N}^{j=1} (p_{ij} - E_i)^2\right)}$$

The standard deviation is the square root of the variance of the distribution.

MOMENT 3 – Skewness :

$$s_i = \sqrt[3]{\left(\frac{1}{N} \sum_{N}^{j=1} (p_{ij} - E_i)^3\right)}$$

Skewness can be understood as a measure of the degree of asymmetry in the distribution.

Similarity Between Images using Color Moments

A function of the similarity between two image distributions is defined as the sum of the weighted differences between the moments of the two distributions. Formally this is:

$$d_{mom}(H, I) = \sum_{i=1}^{r} w_{i1}\left|E_i^1 - E_i^2\right| + w_{i2}\left|\sigma_i^1 - \sigma_i^2\right| + w_{i3}\left|s_i^1 - s_i^2\right|$$

Where:

(H, I)	: are the two image distributions being compared
i	: is the current channel index (e.g. 1 = H, 2 = S, 3 = V)
r	: is the number of channels (e.g. 3)
E_i^1, E_i^2	: are the first moments (mean) of the two image distributions
σ_i^1, σ_i^2	: are the second moments (std) of the two image distributions
s_i^1, s_i^2	: are the third moments (skewness) of the two image distributions
w_i	: are the weights for each moment (described below)

Pairs of images can be ranked based on their d_{mom} values. Those with greater values are ranked lower and considered less similar than those with a higher rank and lower d_{mom} values.

Color Moments example

Index Image

Test Image 1

Test Image 2

$$d_{mom}(Index, Test1) = 0.5878$$

$$d_{mom}(Index, Test2) = 1.5585$$

Color Features Technique Summery

Color method	Pros.	Cons.
Histogram	Simple to compute, intuitive	High dimension, no spatial info, sensitive to noise
CM	Compact, robust	Not enough to describe all colors, no spatial info
CCV	Spatial info	High dimension, high computation cost
Correlogram	Spatial info	Very high computation cost, sensitive to noise, rotation and scale
DCD	Compact, robust, perceptual meaning	Need post-processing for spatial info
CSD	Spatial info	Sensitive to noise, rotation and scale
SCD	Compact on need, scalability	No spatial info, less accurate if compact

CCV: color coherence vector
DCD: dominant color descriptor
CSD: color structure descriptor
SCD: scalable color descriptor respectively

Outline

- Feature Extraction Overview
- Color Features
- **Local Features**
  - Edges
  - Corners
  - Interests Point
- Visual Bag of Words

Local Features

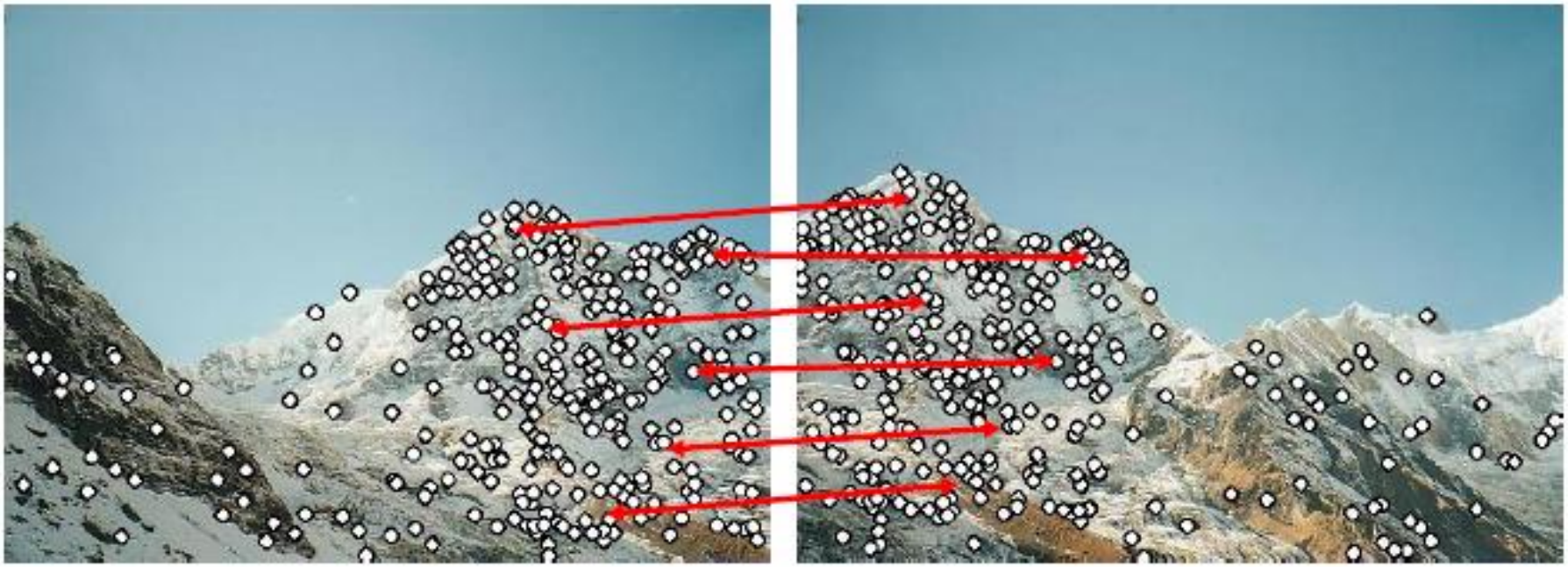
- Features that can be extracted automatically from an image without any shape information (information about *spatial* relationships)
- Can be used in high-level feature extraction, where we find shapes in Images.
- Types
  - Edges
  - Texture: Corners, Interest points

Local Features – Motivation

- Panorama stitching
  - We have two images – how do we combine them?

Local Features – Motivation

- Panorama stitching
  - We have two images – how do we combine them?



Extract and match features

Why extract features?

- Panorama stitching
  - We have two images – how do we combine them?

Align images

Advantages of 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

Outline

- Feature Extraction Overview
- Color Features
- Local Features
  - **Edges**
  - Corners
  - Interests Point
- Visual Bag of Words

Edges

- Edge refers to a significant change or discontinuity in intensity or color in an image.
- Edges often correspond to boundaries between different objects or regions within the image.
- Various algorithms are used for edge detection, ranging from simple methods like Sobel and Prewitt operators to more advanced techniques such as the Canny edge detector.
- The gradient of an image at a particular point indicates the direction and magnitude of the steepest increase in intensity.
- The orientation of an edge refers to the direction in which the intensity changes most rapidly. Edge orientation information is valuable for tasks like object recognition and shape analysis.
- The strength or magnitude of an edge is a measure of how rapidly the intensity changes at a particular point. It is often computed based on the gradient magnitude.
- Edges can be represented using **Histogram of Oriented Gradients (HOG)**

Histogram of Oriented Gradients (HOG)

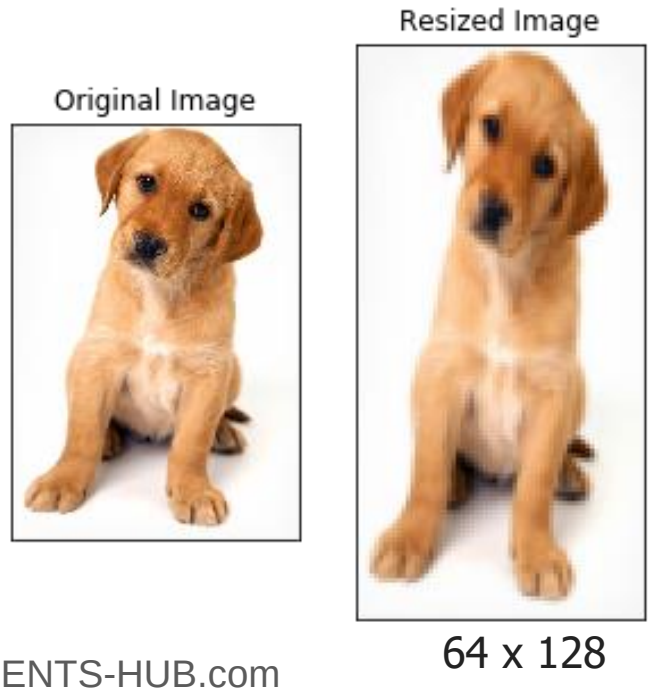
- The Histogram of Oriented Gradients (HOG) is a feature descriptor widely used in computer vision and image processing for object detection and recognition.
- The technique counts occurrences of gradient orientation in localized portions of an image.
- HOGs are calculated by dividing an image into a grid of cells and then computing a histogram of the gradient orientations for each cell.
- HOG features are then extracted from the histograms by concatenating the histograms of all of the cells in the image.
- The HOG descriptor focuses on the structure or the shape of an object. It is better than any edge descriptor as it uses magnitude as well as angle of the gradient to compute the features.

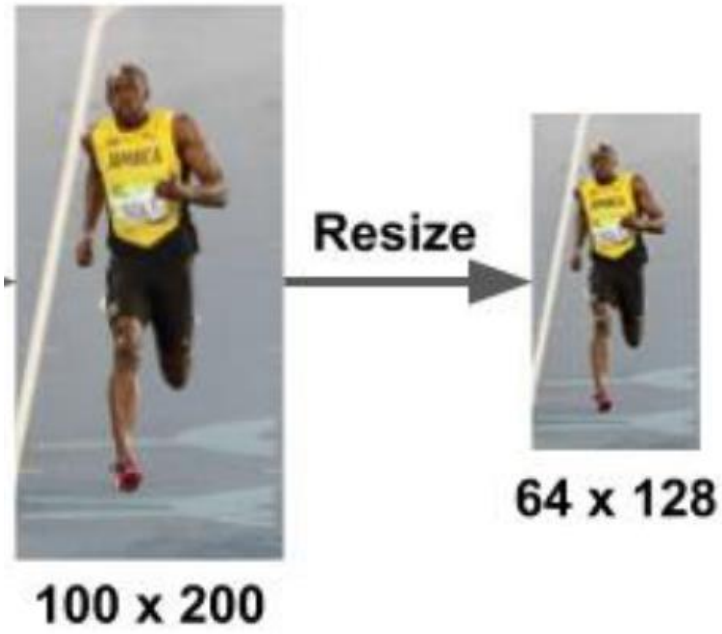
How HOG works?

1) Preprocess the image, including contrast enhancement, noise reduction, resizing and color normalization.
2) Compute the gradient vector of every pixel, as well as its magnitude and direction.
3) Divide the image into many 8x8 pixel cells. In each cell, the magnitude values of these 64 cells are binned and cumulatively added into 9 buckets of unsigned direction (no sign, so 0-180 degree rather than 0-360 degree; this is a practical choice based on empirical experiments).
4) Then we slide a 2x2 cells (thus 16x16 pixels) block across the image. In each block region, 4 histograms of 4 cells are concatenated into one-dimensional vector of 36 values and then normalized to have an unit weight. The final HOG feature vector is the concatenation of all the block vectors. It can be fed into a classifier like SVM for learning object recognition tasks.

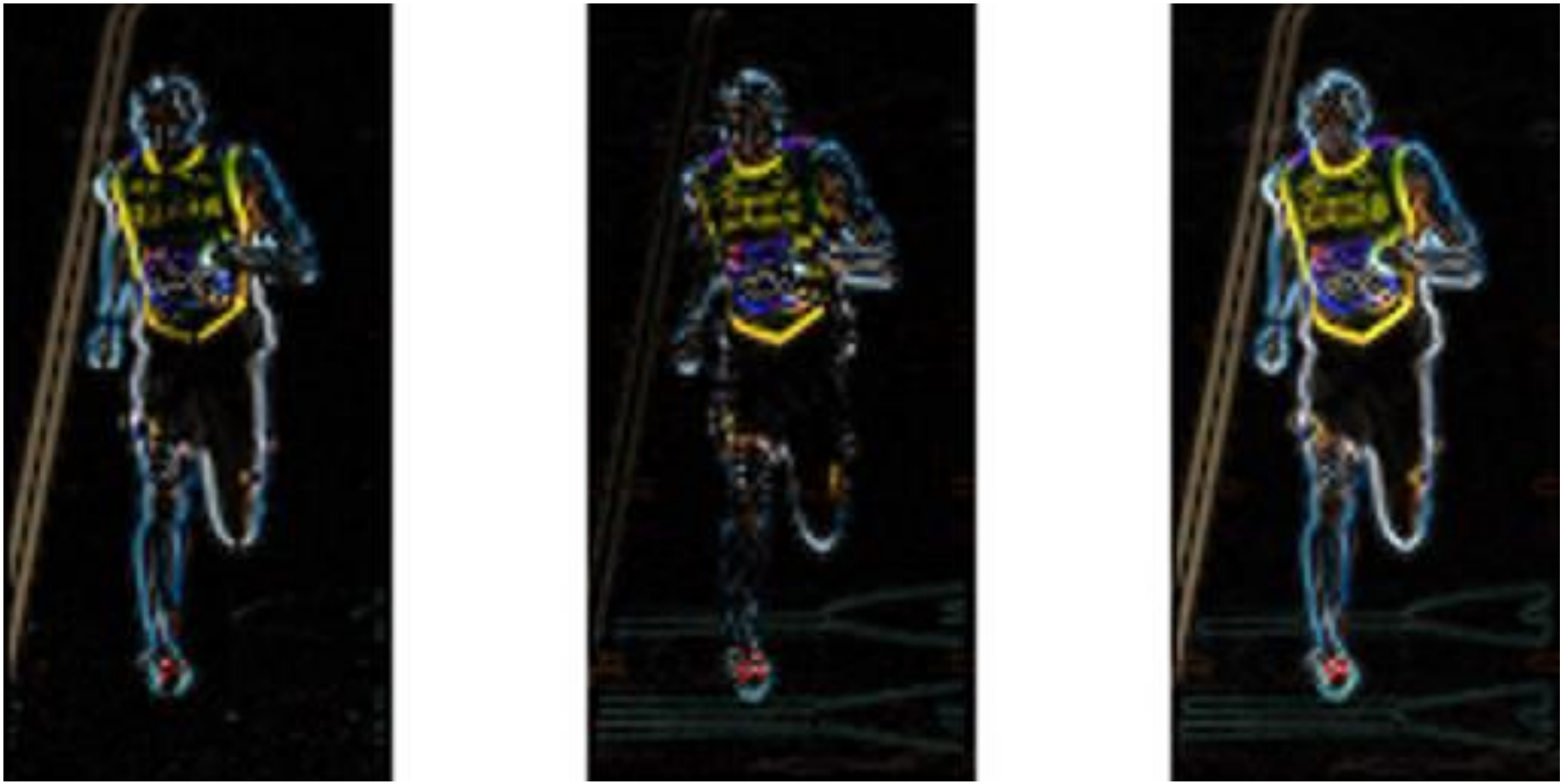
Step 1: Preprocessing

- HoG feature descriptor used for pedestrian detection is calculated on patches having a fixed aspect ratio of an image.
  - In HoG, the patches need to have an aspect ratio of 1:2
  - The image size should preferably be 64 x 128. This is because we will be dividing the image into 8*8 and 16*16 patches to extract the features. Having the specified size (64 x 128) will make all our calculations pretty simple.



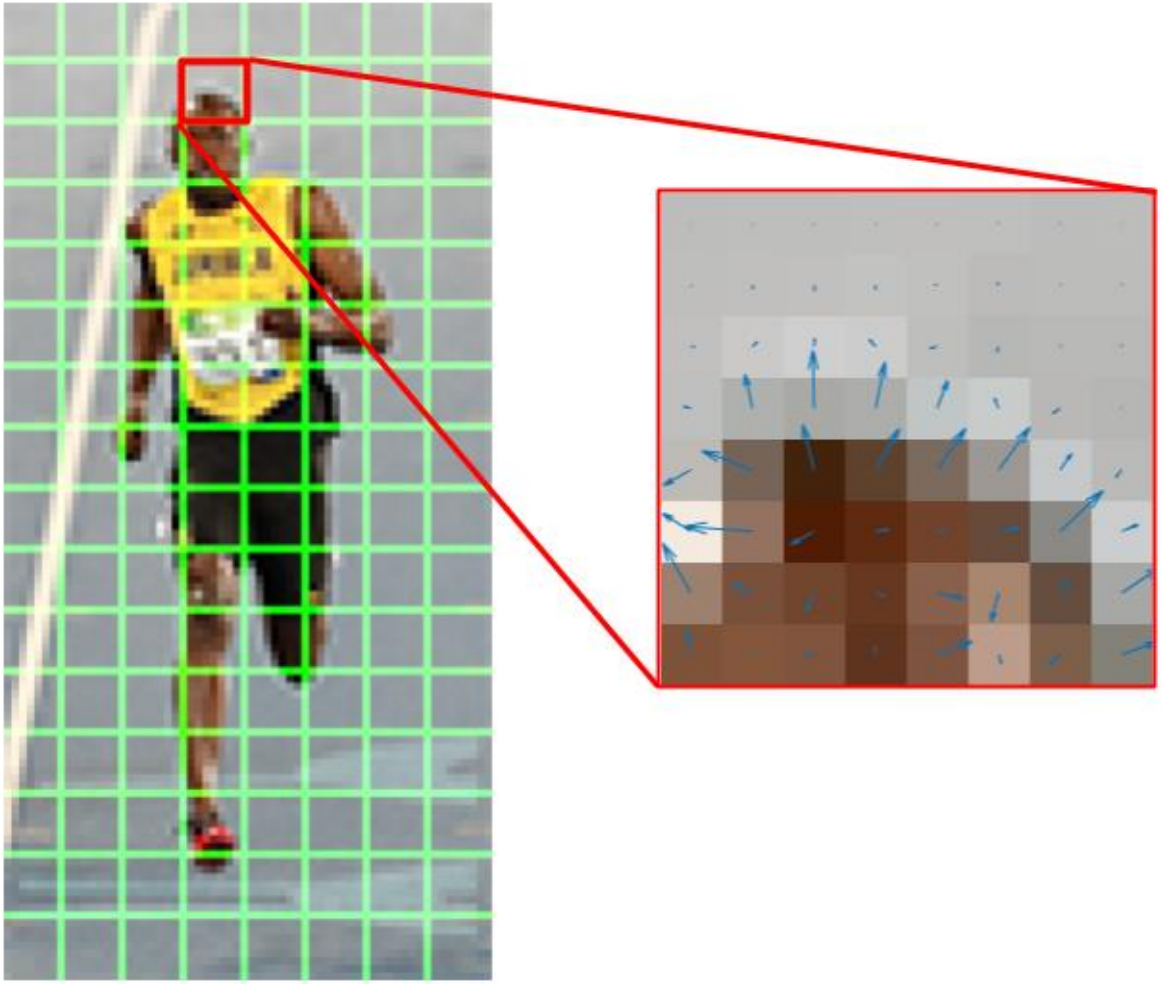


Step 2: Compute the gradient



Left : Absolute value of x-gradient. Center : Absolute value of y-gradient. Right : Magnitude of gradient.

Step 3: Calculate Histogram of Gradients in 8×8 cells



2	3	4	4	3	4	2	2
5	11	17	13	7	9	3	4
11	21	23	27	22	17	4	6
23	99	165	135	85	32	26	2
91	155	133	136	144	152	57	28
98	196	76	38	26	60	170	51
165	60	60	27	77	85	43	136
71	13	34	23	108	27	48	110

Gradient Magnitude

80	36	5	10	0	64	90	73
37	9	9	179	78	27	169	166
87	136	173	39	102	163	152	176
76	13	1	168	159	22	125	143
120	70	14	150	145	144	145	143
58	86	119	98	100	101	133	113
30	65	157	75	78	165	145	124
11	170	91	4	110	17	133	110

Gradient Direction

Step 3: Calculate Histogram of Gradients in 8×8 cells

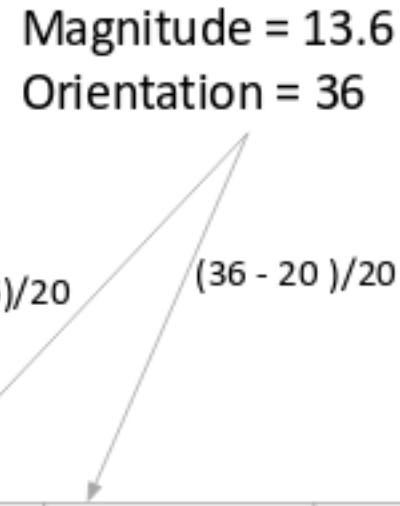
- **Method 1:** add the pixel's gradient to the bin which is closer to the orientation.

Magnitude = 13.6
Orientation = 36

Magnitude		13.6							
Bin	0	20	40	60	80	100	120	140	160

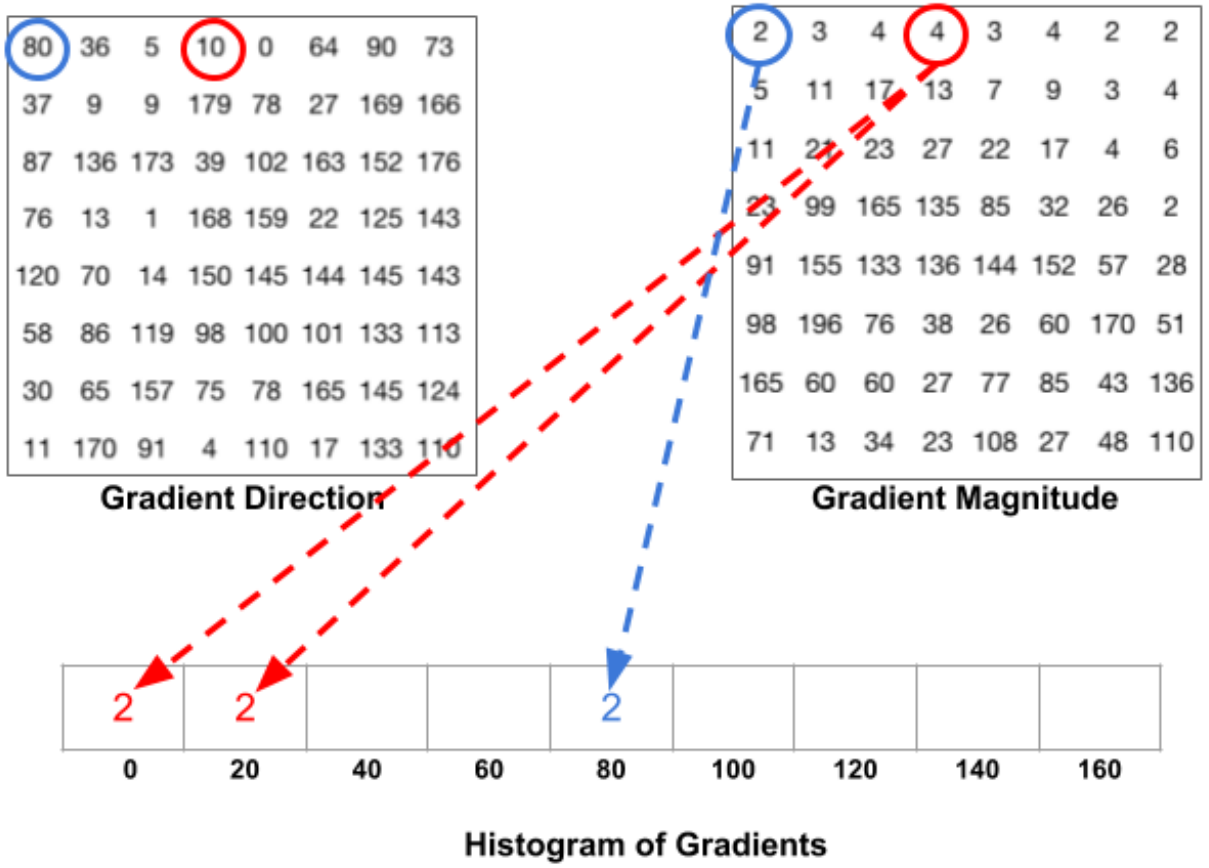
Step 3: Calculate Histogram of Gradients in 8×8 cells

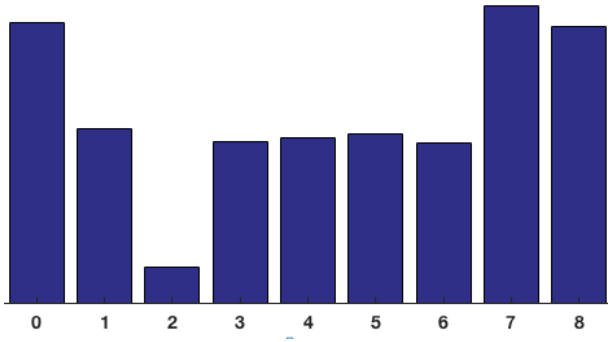
- **Method 2:** add the contribution of a pixel's gradient to the bins on either side of the pixel gradient. The higher contribution should be to the bin value which is closer to the orientation.



Magnitude		(4/20)*13.6	(16/20)*13.6						
Bin	0	20	40	60	80	100	120	140	160

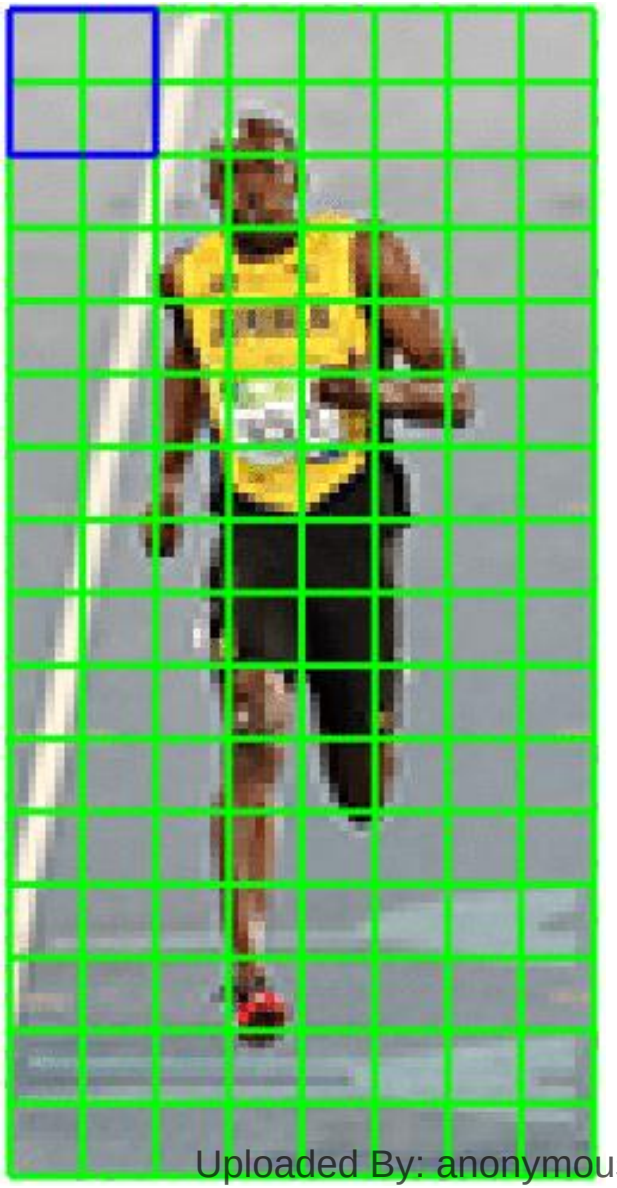
Step 3: Calculate Histogram of Gradients in 8×8 cells





Step 4 : Block Normalization

- Gradients of an image are sensitive to overall lighting. If you make the image darker by dividing all pixel values by 2, the gradient magnitude will change by half, and therefore the histogram values will change by half.
- Ideally, we want our descriptor to be independent of lighting variations. In other words, we would like to "normalize" the histogram so they are not affected by lighting variations.



Aside: Vector Normalization

- Vector normalization is a process of scaling a vector to have a length of 1 while preserving its direction.
- There are various normalization techniques, and one common method is L2 normalization (Euclidean normalization).
- **L2 Normalization (Euclidean Normalization):**
  1. **Calculate L2 Norm:** Compute the L2 norm of the vector **v**, denoted as $\|\mathbf{v}\|_2$. The L2 norm is the square root of the sum of the squared elements of the vector.

$$\|\mathbf{v}\|_2 = \sqrt{v_1^2 + v_2^2 + \ldots + v_n^2}$$

  1. **Normalize Vector:** Divide each element of the vector by its L2 norm to obtain the normalized vector.

$$\text{Normalized vector} = \frac{\mathbf{v}}{\|\mathbf{v}\|_2}$$

Aside: Vector Normalization

- Let's say we have a color vector [128, 64, 32].
  - The length of this vector is sqr(128^2 + 64^2 + 32^2) .
  - This is also called the L2 norm of the vector.
  - Dividing each element of this vector by 146.64 gives us a normalized vector [0.87, 0.43, 0.22].
  - Now consider another vector in which the elements are twice the value of the first vector 2 x [128, 64, 32] = [256, 128, 64].
  - Normalizing [256, 128, 64] will result in [0.87, 0.43, 0.22], which is the same as the normalized version of the original vector.
  - You can see that normalizing a vector removes the intensity transformation.

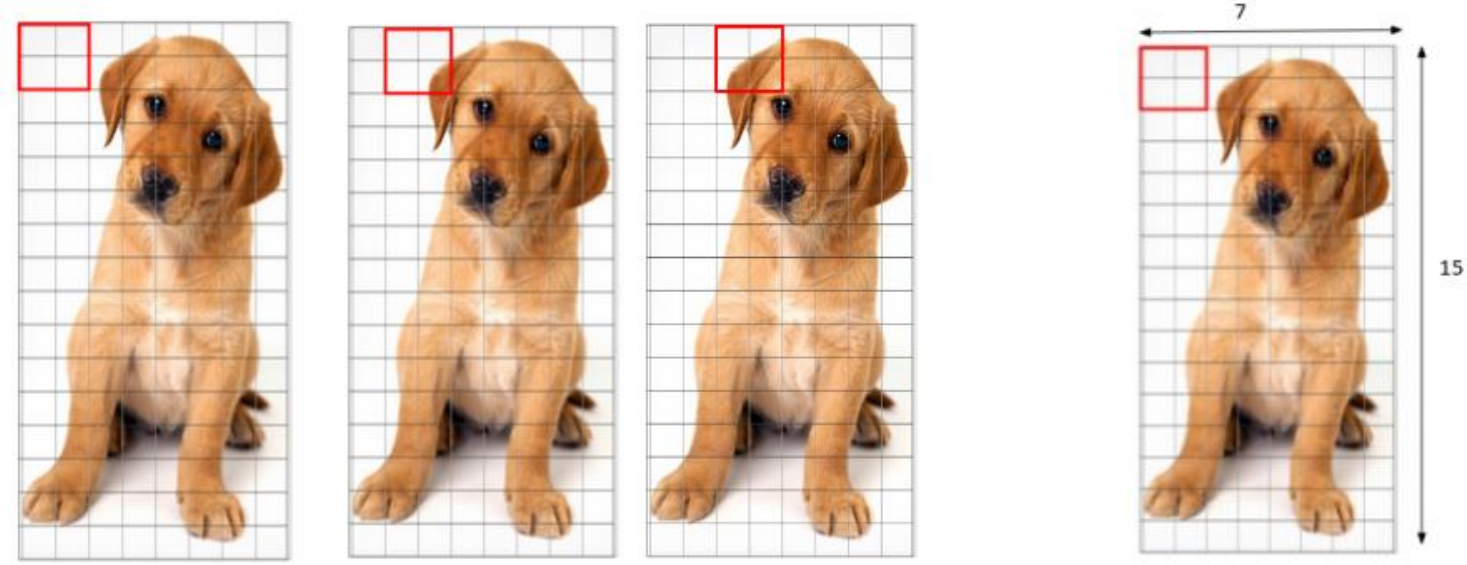
Step 4 : 16×16 Block Normalization

- A better idea is to normalize over a bigger sized block of 16×16.
  - Normalizing over a larger block helps in making the feature descriptor more robust to local variations in illumination and contrast within the image.
  - A larger block size enables the descriptor to capture more significant spatial relationships between different parts of the object.
  - Normalizing over larger blocks provides some level of invariance to deformations or distortions within the object.

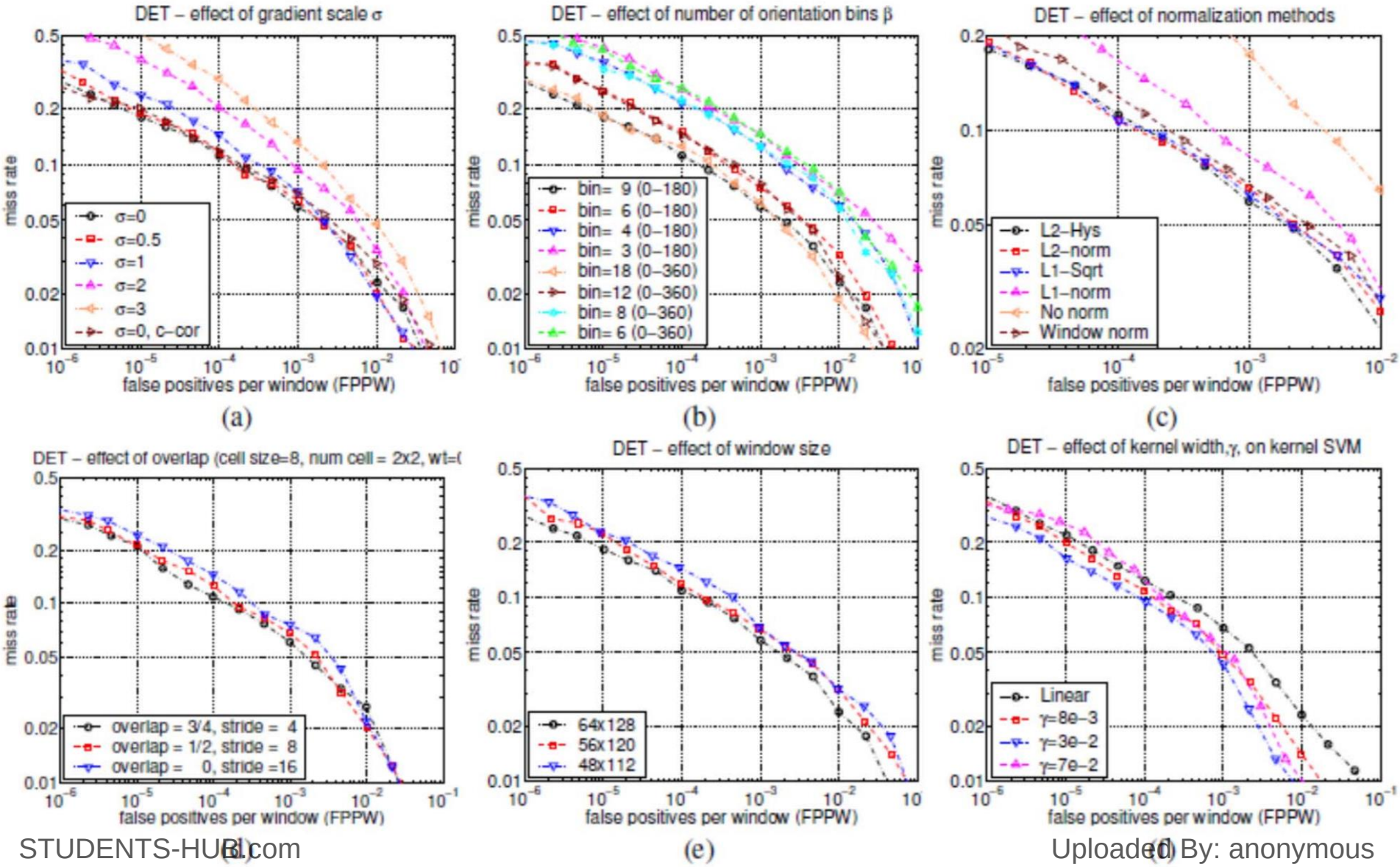
- A 16×16 block has 4 histograms which can be concatenated to form a 36 x 1 element vector and it can be normalized just the way a 3×1 vector is normalized.
- The window is then moved by 8 pixels and a normalized 36×1 vector is calculated over this window and the process is repeated.

Step 5: Calculate the HOG feature vector

- To calculate the final feature vector for the entire image patch, the 36×1 vectors are concatenated into one giant vector.
- What is the size of this vector ? Let us calculate
  - How many positions of the 16×16 blocks do we have ? There are 7 horizontal and 15 vertical positions making a total of 7 x 15 = 105 positions.
  - Each 16×16 block is represented by a 36×1 vector. So when we concatenate them all into one giant vector we obtain a 36×105 = **3780** dimensional vector.



How to Select Different Parameters?



HOG Summary

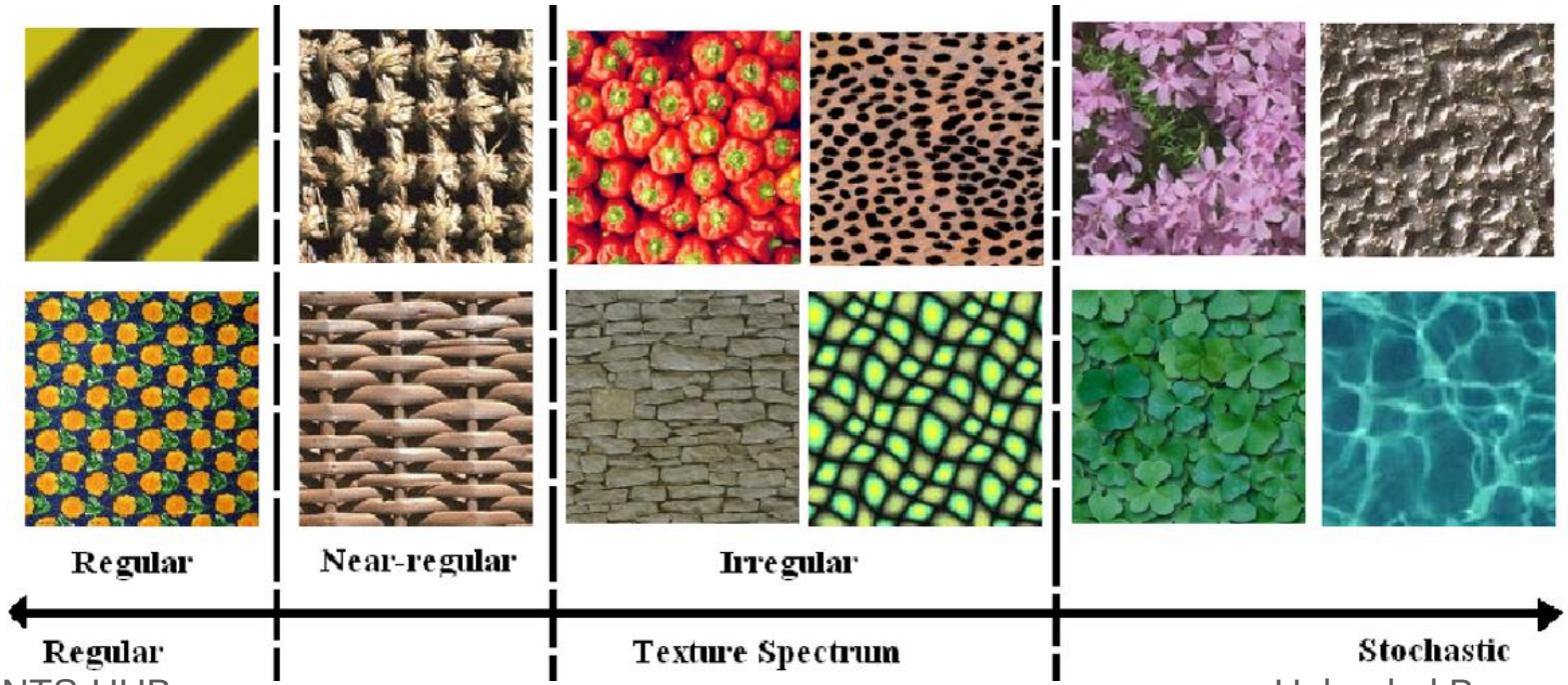
- HoGs are invariant to illumination: HoGs are calculated based on the orientations of the edges in an image, which are not affected by changes in illumination.
- HOGs are computationally efficient: HOGs can be calculated efficiently using a variety of algorithms.
- HOG has been successfully applied in various domains such as object detection
- **However:**
  - HOGs may not be discriminative enough for some tasks, such as classifying fine-grained objects.
  - HOGs are not scale invariant.
  - HOG is not rotation invariant feature.
- HOG is sometimes used in conjunction with other descriptors or feature extraction methods to create a more comprehensive representation of images.

Outline

- Feature Extraction Overview
- Color Features
- Shape Features
- Local Features
  - Edges
  - **Corners**
  - Interests Point
- Visual Bag of Words

Texture Features: What´s in the image?

- Texture is a tactile or visual characteristic of a surface.
- In general, color is usually a pixel property **while texture can only be measured from a group of pixels.**
- Texture gives us information about the spatial arrangement of the colors or intensities in an image.
- Types of texture:



Extraction of Texture Features

- Aim: to find a unique way of representing the underlying characteristics of textures and represent them in some simpler but unique form, so then they can be used to accurately and robustly classify and segment objects.

- Basically, texture representation methods can be classified into two categories:
  - **Structural approach**: Texture is a set of primitive Texel's in some regular or repeated relationship.
    - Texel: A small geometric pattern that is repeated frequently on some surface resulting in a texture.
    - Work well for man-made and regular patterns
  - **Statistical approach**: Texture is a quantitative measure of the arrangement of intensities in a region.
    - Statistical methods analyze the spatial distribution of gray values, by computing local features at each point in the image, and deriving a set of statistics from the distributions of the local features.
    - More general and easier to compute and is used more often in practice.

Some Statistical Methods

- Some statistical approaches for texture:
  - **Corner Detection**
  - Co-occurrence matrices
  - Local binary patterns
  - Statistical moments
  - Autocorrelation
  - Markov random fields
  - Autoregressive models
  - Mathematical morphology
  - **Interest points – SIFT, SURF…**
  - Fourier power spectrum
  - Gabor filters

Texture Features - Corner

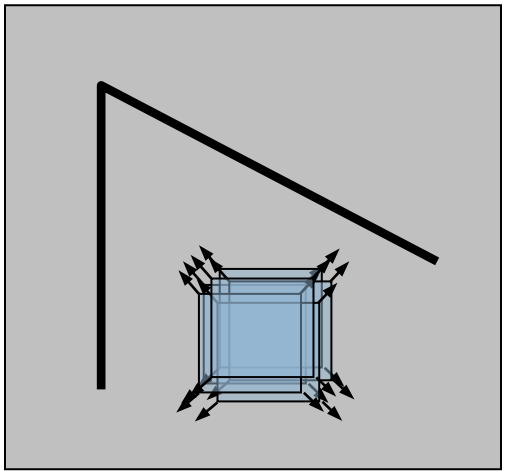
- A corner can be defined as the intersection of two edges.
- Can also be defined as a point for which there are two dominant and different edge directions in a local neighborhood of the point.
- Corner detection is frequently used in motion detection, image registration, video tracking, image matching, and object recognition.
- Edge detection that can be used with post-processing to detect corners.
  - Kirsch operator.
  - Frei-Chen masking set.

Corner Detection Approaches

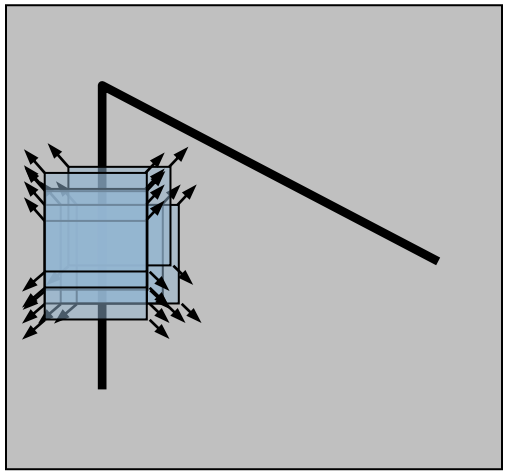
- Several proposed approaches for corner detection:
  - Moravec corner detection algorithm
  - **The Harris & Stephens corner detection algorithms**
  - The level curve curvature approach
  - Laplacian of Gaussian, differences of Gaussians and determinant of the Hessian scale-space interest points.
  - The Wang and Brady corner detection algorithm
  - The SUSAN corner detector
  - ……
- One determination of the quality of a corner detector is its ability to detect the same corner in multiple similar images, under conditions of different lighting, translation, rotation, Scaling, and other transforms.

Harris Detector - The Basic Idea

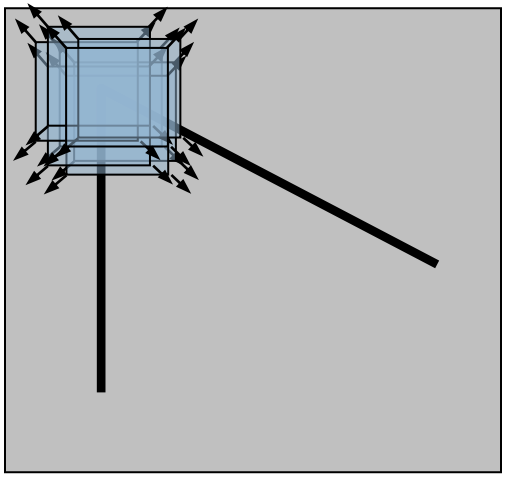
- We should easily recognize the point by looking through a small window
- Shifting a window in *any direction* should give *a large change* in intensity



"flat" region:
no change in
all directions



"edge":
no change along
the edge direction

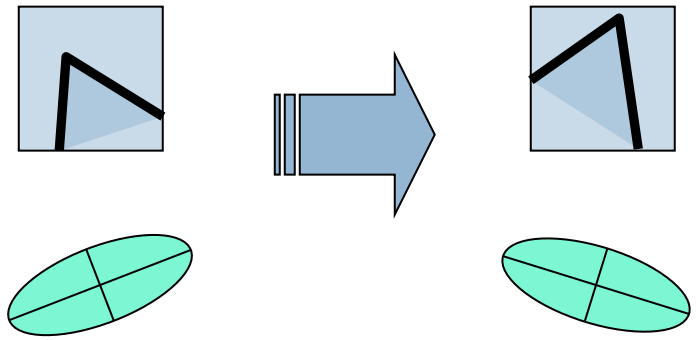


"corner":
significant change
in all directions

Find locations such that the minimum change caused by shifting the window in any direction is large

Harris Detector: Some Properties

- Rotation invariance



Ellipse rotates but its shape (i.e. eigenvalues) remains the same

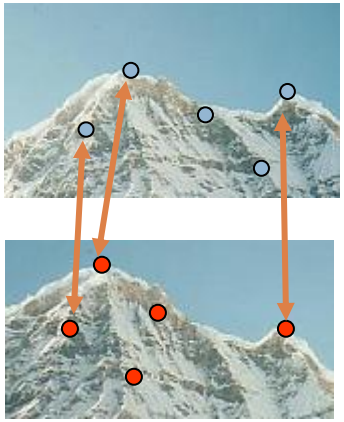
Corner response is invariant to image rotation

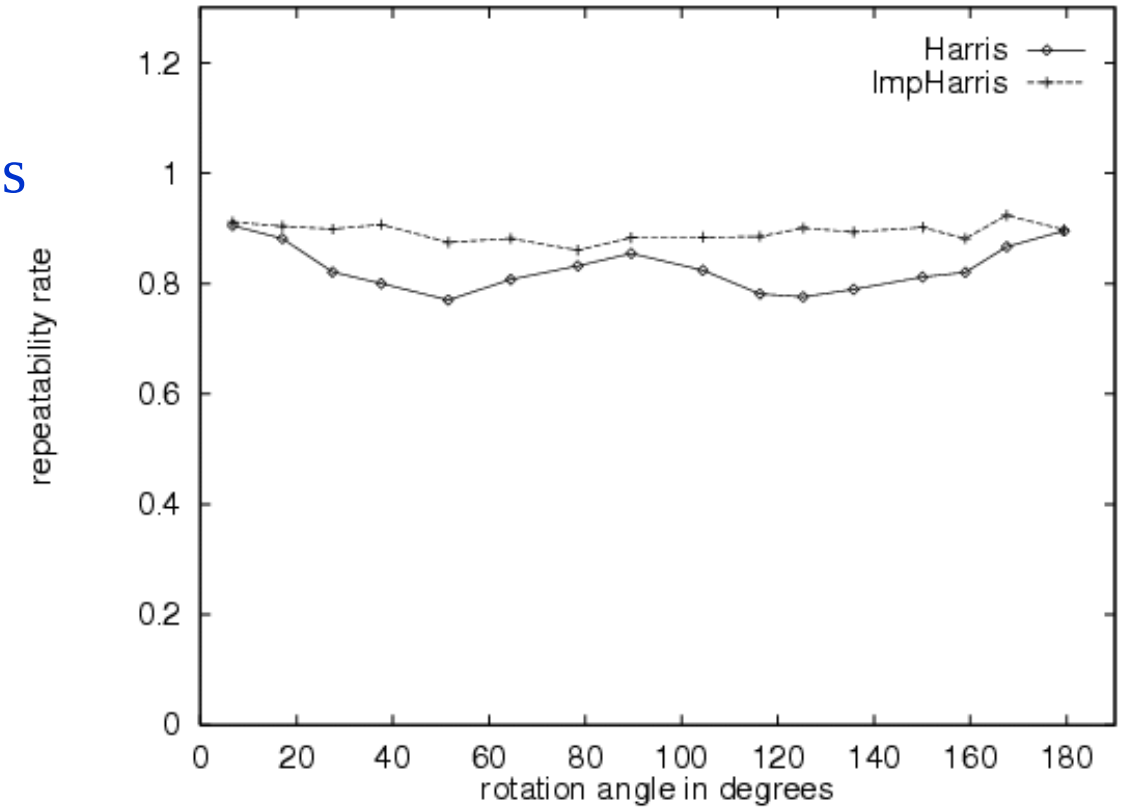
Harris Detector: Some Properties

- Rotation Invariant Detection

Repeatability rate:

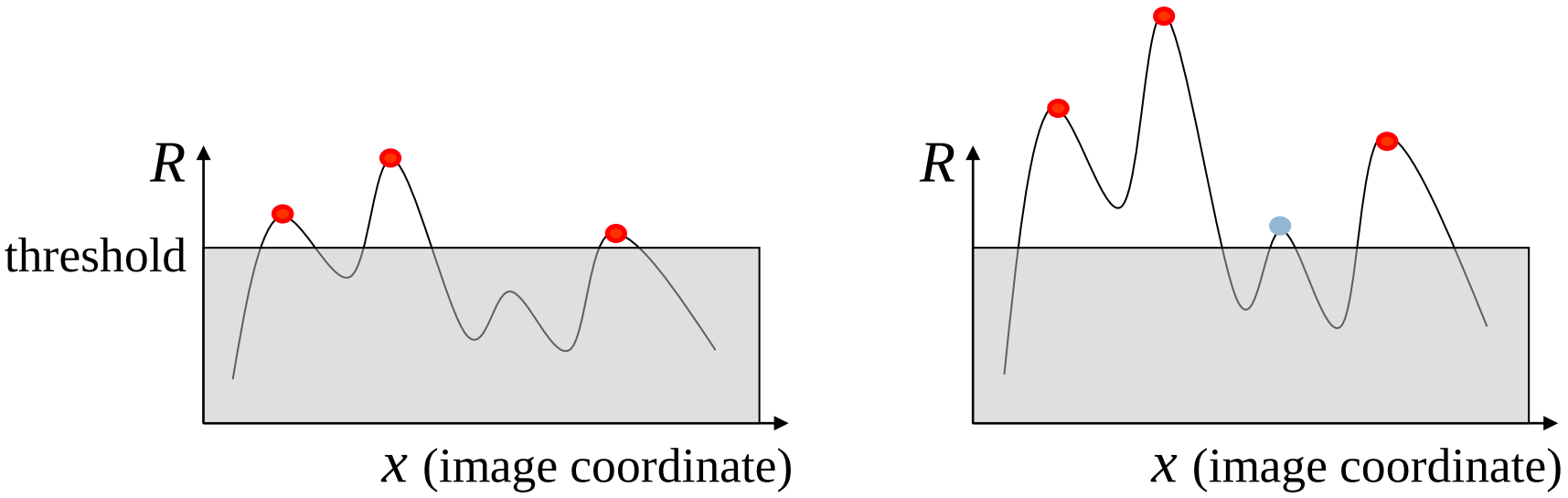
$$\frac{\text{\# correspondences}}{\text{\# possible correspondences}}$$





Harris Detector: Some Properties

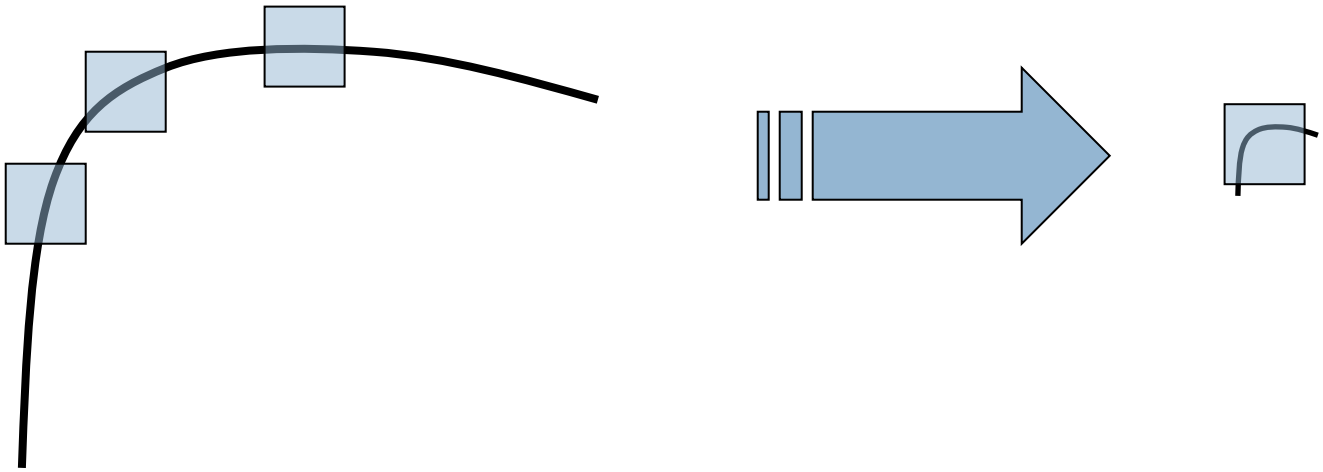
- Partial invariance to *affine intensity* change
  - ✓ Only derivatives are used => invariance to intensity shift $I \rightarrow I + b$
  - ✓ Intensity scale: $I \rightarrow a\ I$



***Partially invariant* to affine intensity change**

Harris Detector: Some Properties

- But: non-invariant to *image scale*!



All points will be classified as edges

Corner !

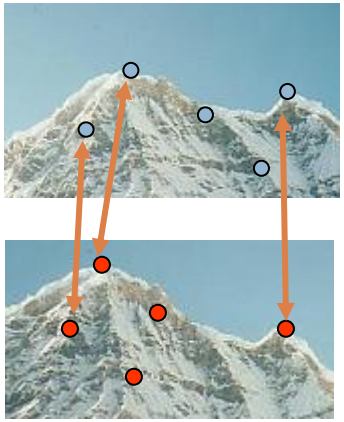
***Not invariant* to scaling**

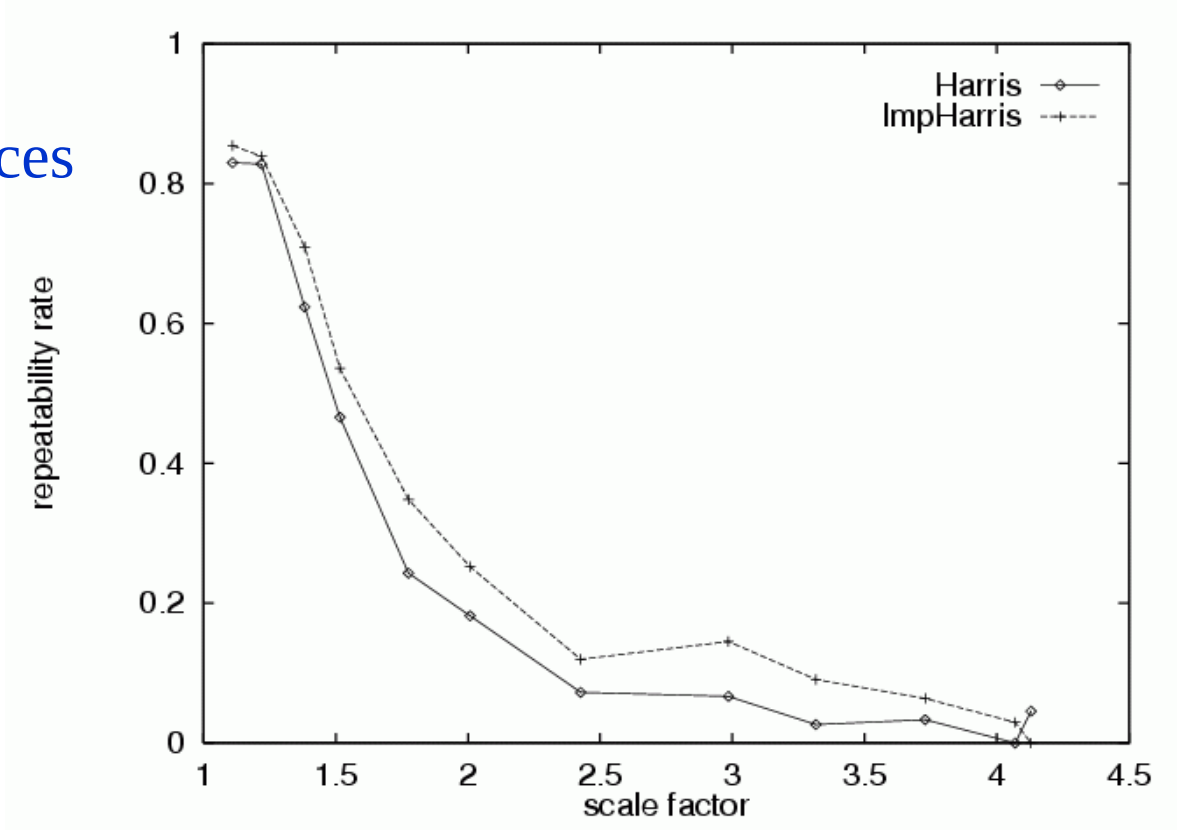
Harris Detector: Some Properties

- Quality of Harris detector for different scale changes

Repeatability rate:

$$\frac{\text{\# correspondences}}{\text{\# possible correspondences}}$$



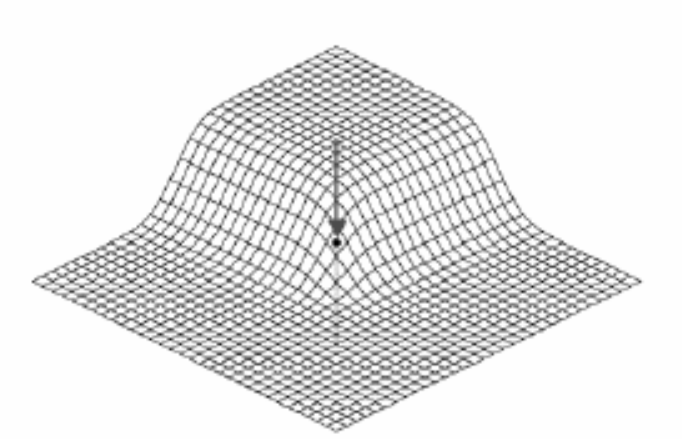


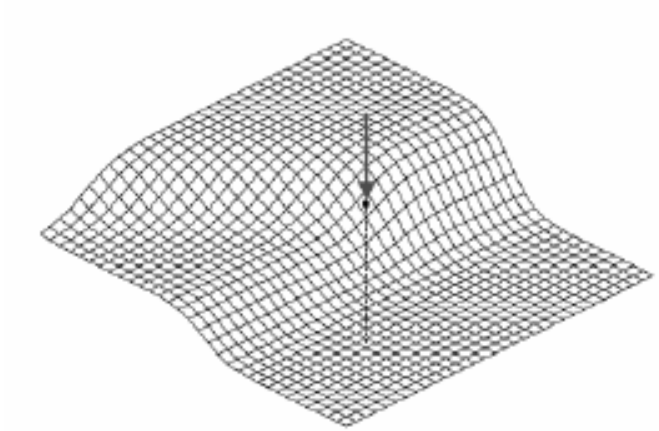
Outline

- Feature Extraction Overview
- Color Features
- Local Features
  - Edges
  - Corners
  - **Interests Point**
- Visual Bag of Words

Texture extraction by Interest Points

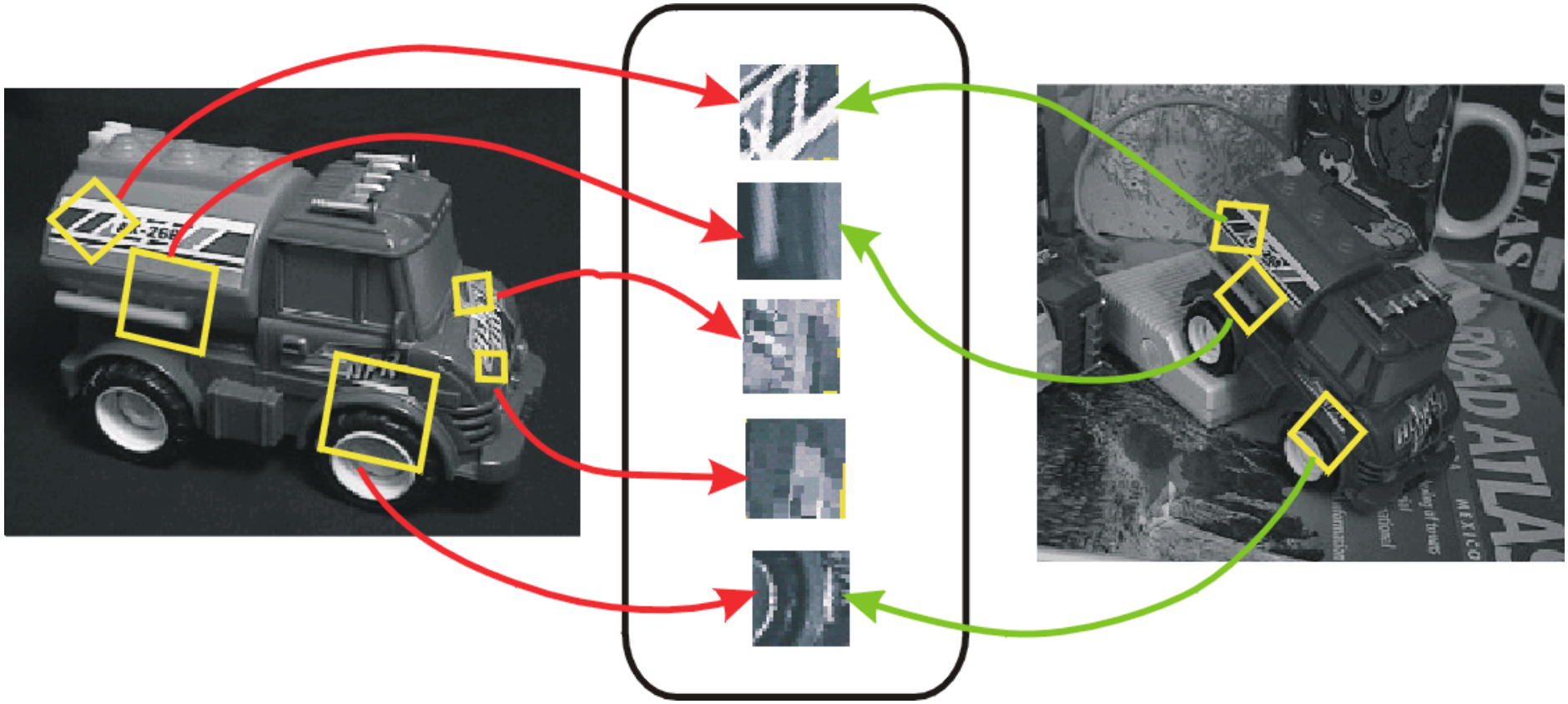
- What is an interest point
  - Expressive texture
    - The point at which the direction of the boundary of object changes abruptly
    - Intersection point between two or more edge segments
  - Goal: Detect points that are *repeatable* and *distinctive*





Interest Point Detection: Main Idea

- Image content is transformed into local feature coordinates that are invariant to translation, rotation, scale, and other imaging parameters



Properties of Interest Point Detectors

- Detect all (or most) true interest points
- No false interest points
- Well localized.
- Robust with respect to noise.
- Efficient detection
- Invariant to transformation

Key trade-offs

Detection of interest points

More Repeatable

Robust detection
Precise localization

More Points

Robust to occlusion
Works with less texture

Description of patches

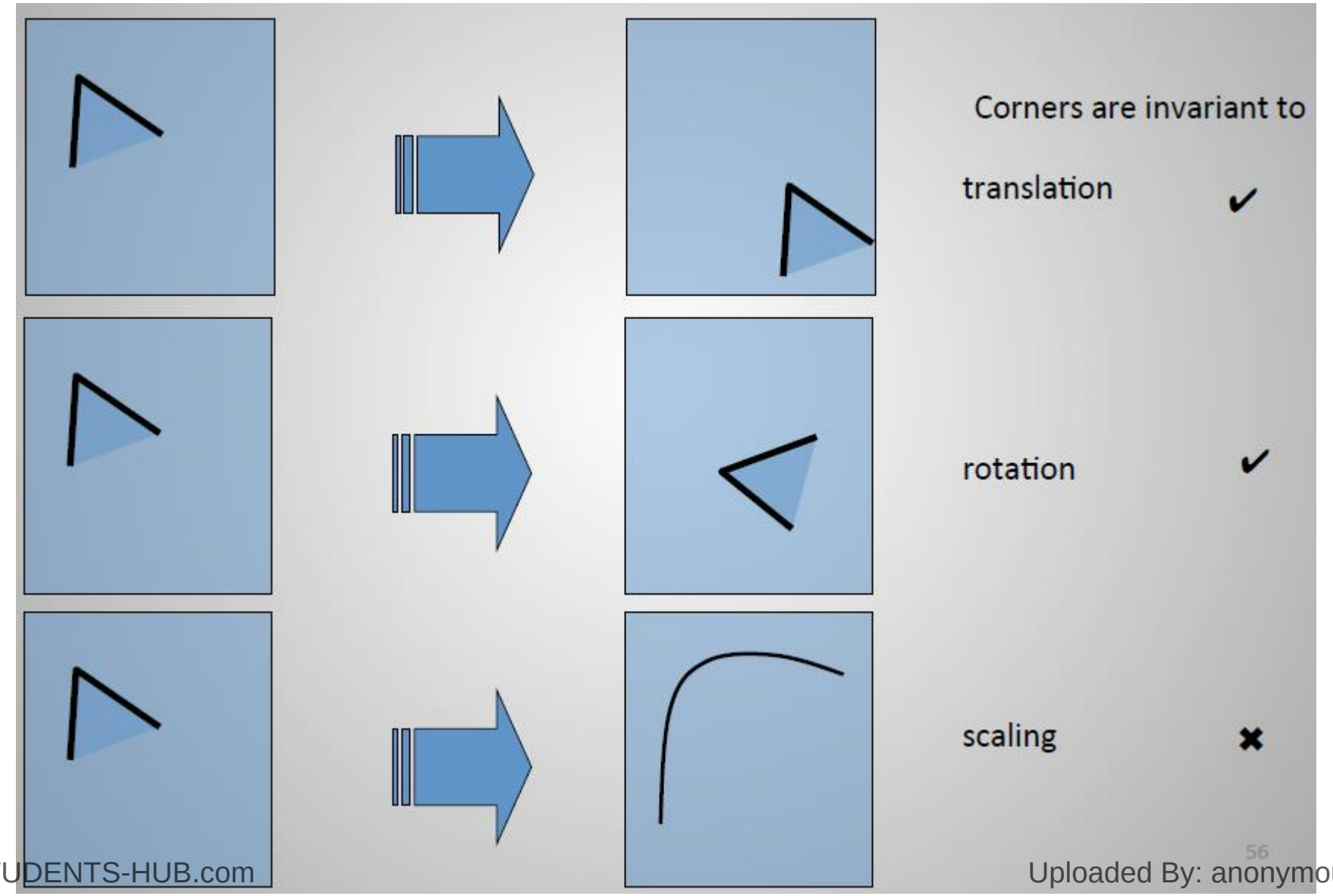
More Distinctive

Minimize wrong matches

More Flexible

Robust to expected variations
Maximize correct matches

Corner as an Interest Point



Uses for Interest Point

- Feature points are used also for:
  - Image alignment
  - 3D reconstruction
  - Motion tracking
  - Object recognition
  - Indexing and database retrieval
  - Robot navigation
  - … many others

SIFT - Scale Invariant Feature Transforms

- SIFT image features provide a set of features of an object that are not affected by many of the complications experienced in other methods, such as object scaling and rotation.
- While allowing for an object to be recognized in a larger image SIFT image features also allow for objects in multiple images of the same location, taken from different positions within the environment, to be recognized.
- SIFT features are also very resilient to the effects of "noise" in the image.
- The SIFT approach, for image feature generation, takes an image and transforms it into a "large collection of local feature vectors"

Overall Procedure at a High Level

- **Step 1:** Constructing a scale space
- **Step 2:** Laplacian of Gaussian approximation
- **Step 3:** Finding Keypoints
- **Step 4:** Eliminate edges and low contrast regions
- **Step 5:** Assign an orientation to the keypoints
- **Step 6:** Generate SIFT features

Step 1: **Constructing scale space**

- The first stage of the SIFT algorithm is to find image locations that are invariant to scale change.
- This is achieved by searching for stable features across all possible scales, using a function of scale known as *scale space*
  - Objects in unconstrained scenes will appear in different ways, depending on the scale at which images are captured
  - A reasonable approach is to work with all relevant scales simultaneously
- The scale space is defined by the function:

$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y)$$

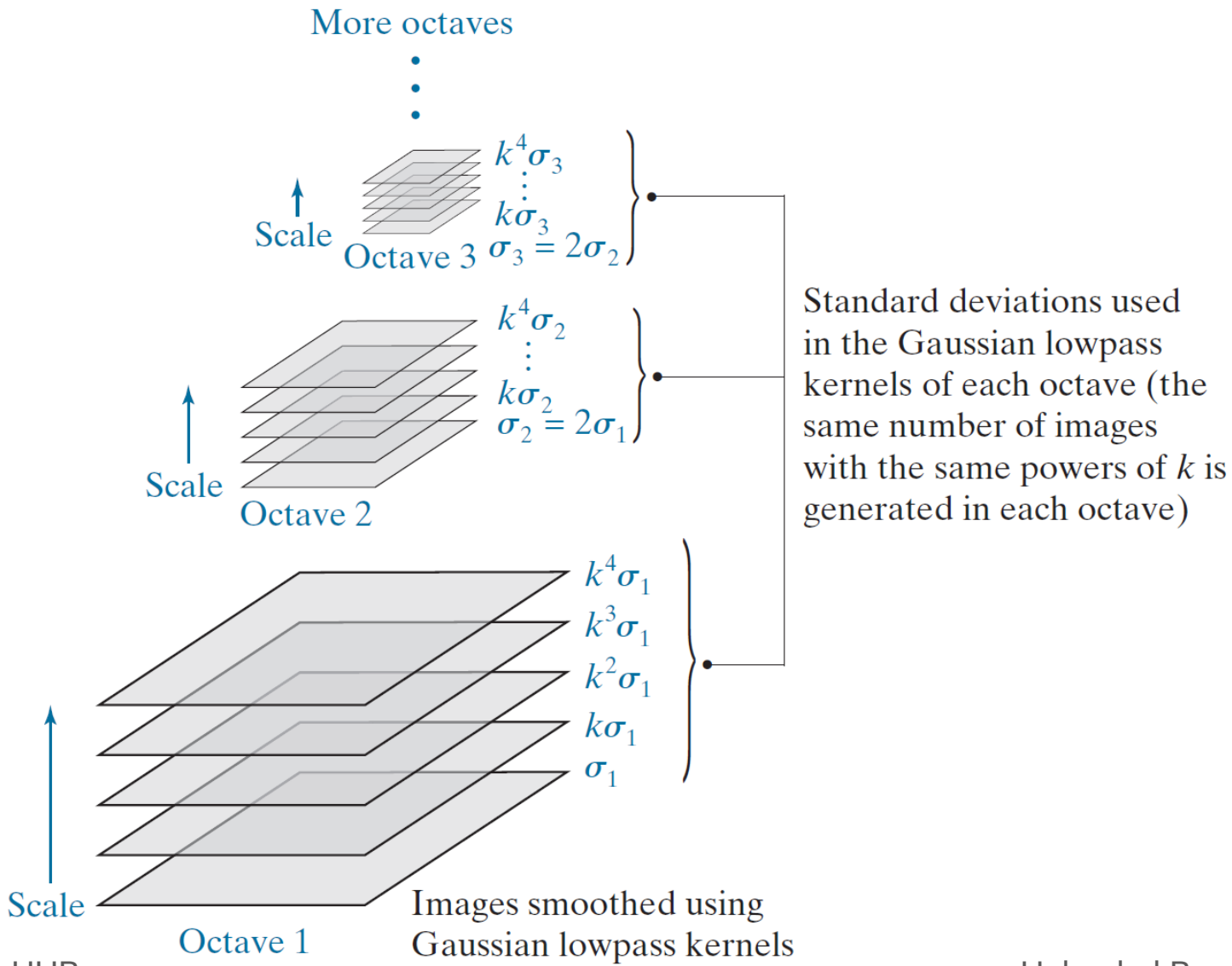
Where:

- * is the convolution operator,
- L is a blurred image
- G is the Gaussian Blur operator
- I is the input image.
- x,y are the location coordinates
- σ is the "scale" parameter. Think of it as the amount of blur. Greater the value, greater the blur.

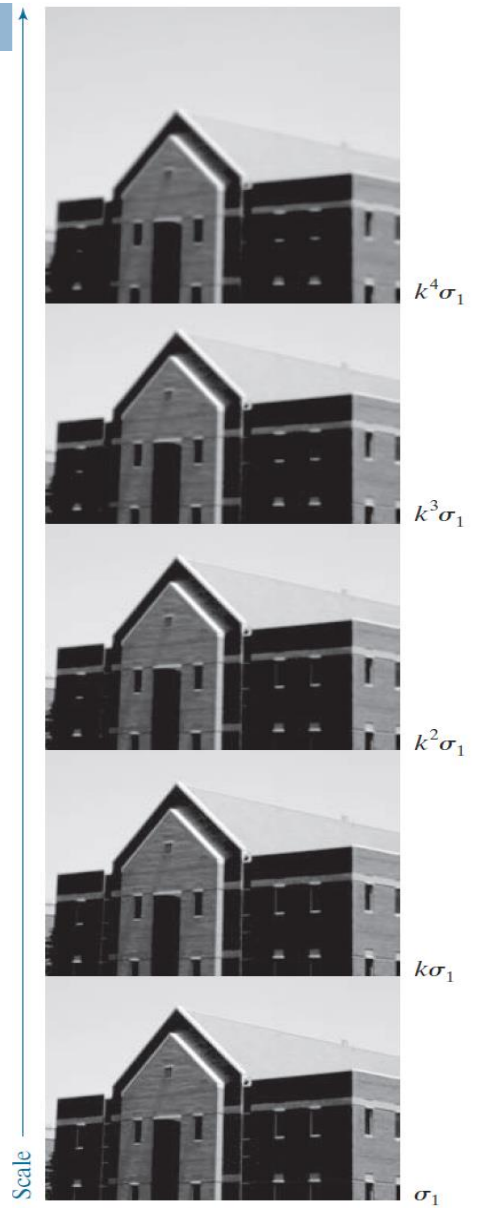
Constructing a scale space in SIFT

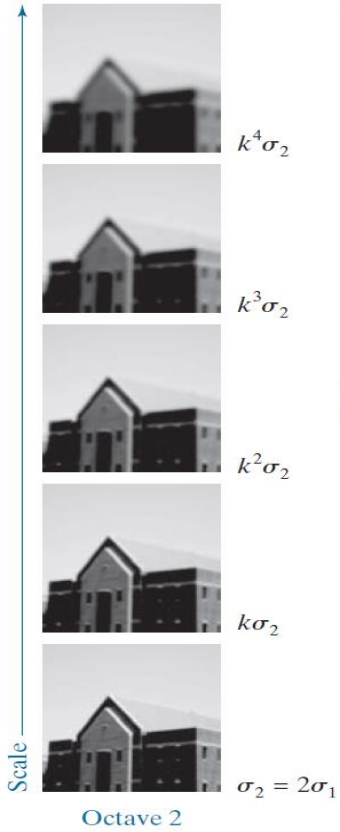
- The input image $f(x, y)$ is successively convolved with Gaussian kernels having standard deviations σ, $k\sigma$, $k^2\sigma$, $k^3\sigma$, . . . to generate a "stack" of Gaussian-filtered (smoothed) images that are separated by a constant factor k
- SIFT subdivides scale space into *octaves*, with each octave corresponding to a doubling of σ.
- SIFT further subdivides each octave into an integer number, s, of intervals, so that an interval of 1 consists of two images, an interval of 2 consists of three images, and so forth
- The *first* image in the *second* octave is formed by down sampling the original image (by skipping every other row and column), and then smoothing it using a kernel with twice the standard deviation used in the first octave

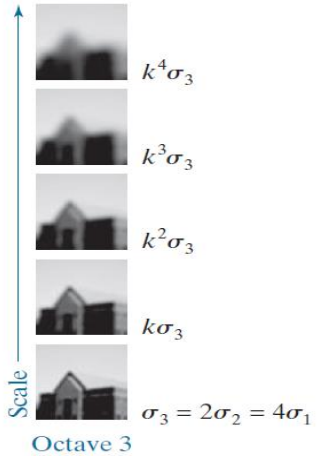
Constructing a scale space in SIFT



Constructing a scale space in SIFT







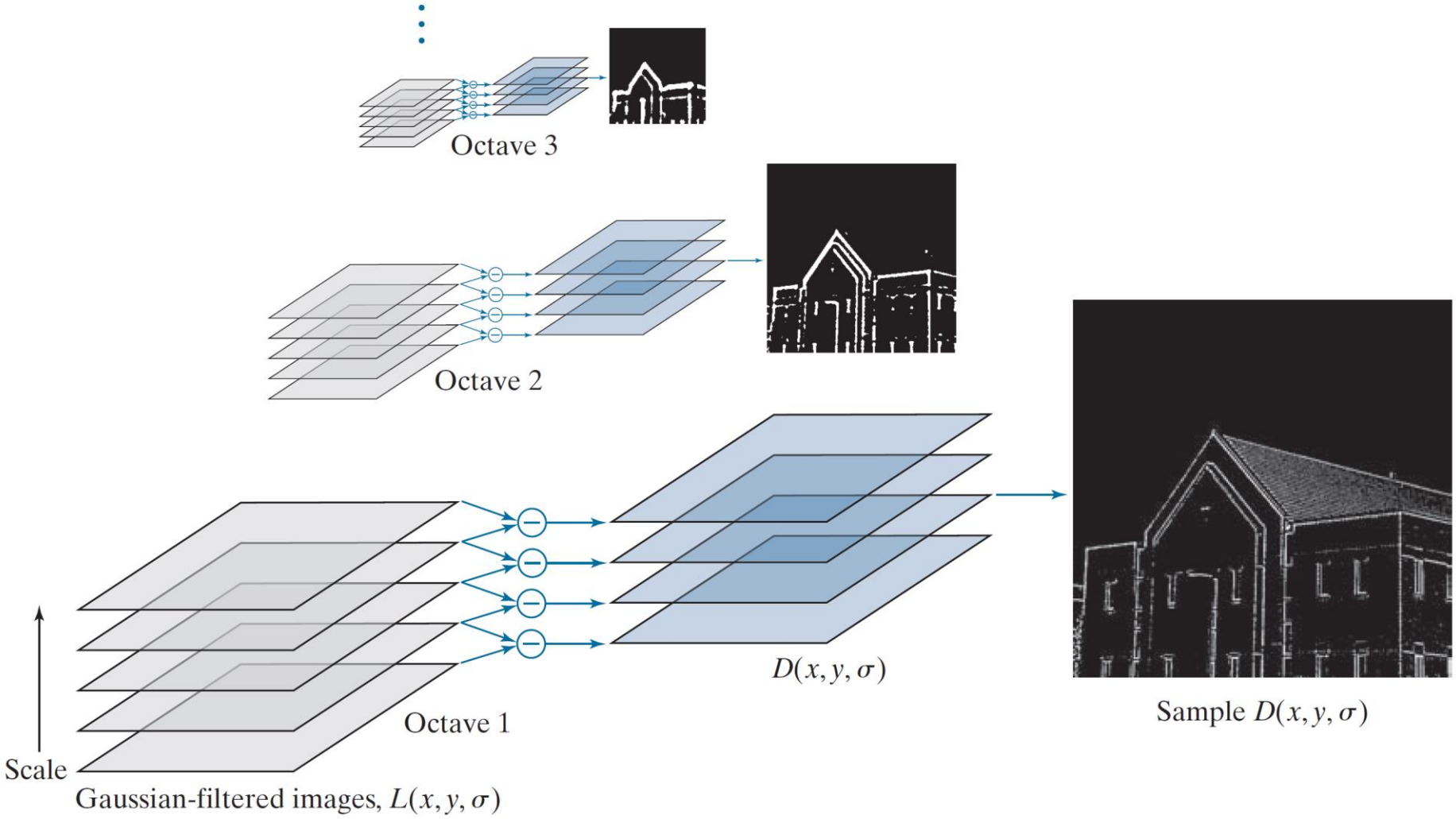
$\sigma_1 = \sqrt{2}/2 = 0.707 \qquad k = \sqrt{2} = 1.414$

Octave	Scale 1	2	3	4	5
1	0.707	1.000	1.414	2.000	2.828
2	1.414	2.000	2.828	4.000	5.657
3	2.828	4.000	5.657	8.000	11.314

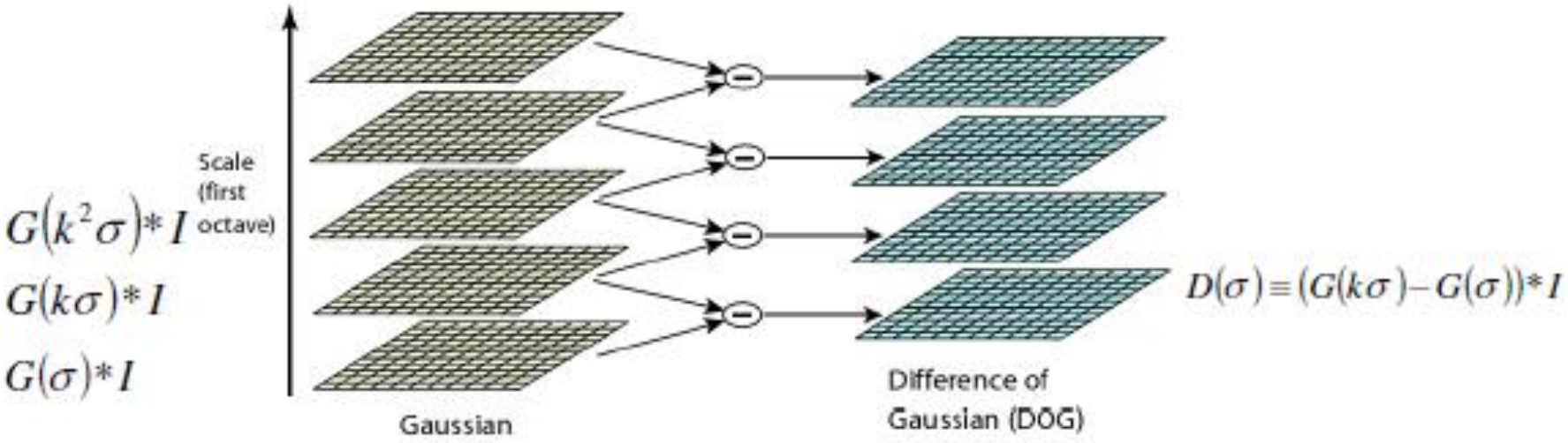
Step 2: Laplacian of Gaussian approximation

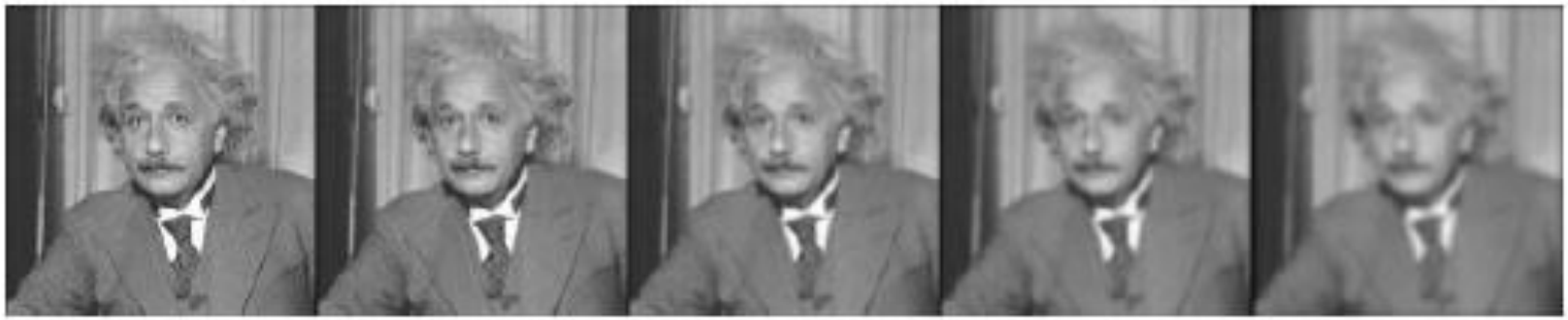
- To find key points use Laplacian of Gaussian (LoG)
  - Take an image, and blur it a little.
  - Then calculate second order derivatives on it (or, the “laplacian”).
- The problem is, calculating all those second order derivatives is computationally intensive.
- Solution, use the Difference of Gaussians (DoG) approximation.
  - We use the scale space (from previous step).
  - We calculate the difference between two consecutive scales.
    - These Difference of Gaussian images are approximately equivalent to the Laplacian of Gaussian, and we’ve replaced a computationally intensive process with a simple subtraction (fast and efficient).
  - These DoG images are a great for finding out interesting key points in the image

Step 2: Laplacian of Gaussian approximation



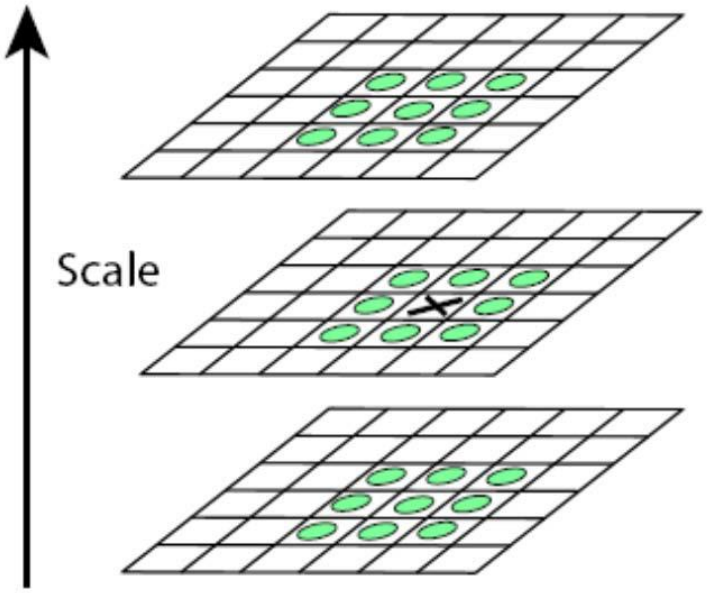
Step 2: Laplacian of Gaussian approximation





Step 3: Finding Keypoints

- To detect the local maxima and minima of D(x, y, σ)
  - Each point is compared with its 8 neighbors at the same scale, and its 9 neighbors up and down one scale.
    - X is marked as a “key point” if it is the greatest or least of all 26 Neighbours
- Large number of extrema,
  - Computationally expensive
  - Detect the most stable subset with a coarse sampling of scales

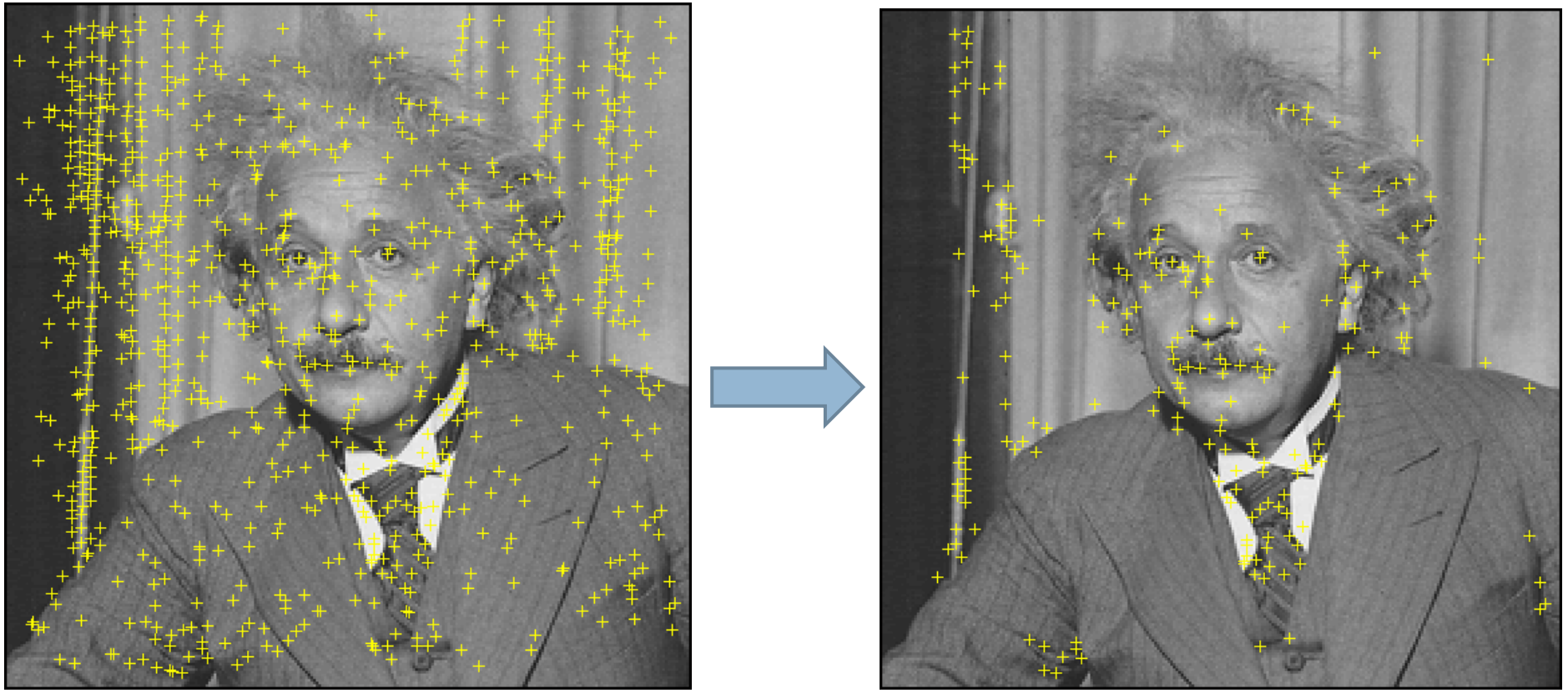


Step 4: Eliminate edges and low contrast regions

- Key points generated in the previous step produce a lot of key points. Some of them lie along an edge, or they don't have enough contrast. In both cases, they are not useful as features, so we need to get rid of them.

- Two Approaches:
  - Reject points with bad contrast:
    - DoG smaller than 0.03 (image values in [0,1])
  - Reject edges
    - Use Harris detector and keep only corners

Step 4: Eliminate edges and low contrast regions



Step 5: Assign an orientation to the keypoints

- After step 4 (legitimate key points), we already know the scale at which the keypoint was detected (it's the same as the scale of the blurred image). So we have scale invariance.
- The next thing is to assign an orientation to each keypoint. This orientation provides rotation invariance
- This step aims to assign a consistent orientation to the keypoints based on local image properties.
- The keypoint descriptor, can then be represented relative to this orientation, achieving invariance to rotation.
- The idea is to collect gradient magnitude and orientation around each keypoint (widow 16*16). Then we figure out the most prominent orientation(s) in that region. And we assign this orientation(s) to the keypoint.

Step 5: Assign an orientation to the keypoints

- The scale of the keypoint is used to select the Gaussian smoothed image, *L*, that is closest to that scale. In this way, all orientation computations are performed in a scale-invariant manner.
- For each pixel in the widow around Keypoint compute gradient magnitude and orientation using Sobel filter:

$$G_x = \frac{\partial f}{\partial x} = (Z_7 + 2Z_8 + Z_9) - (Z_1 + 2Z_2 + Z_3)$$

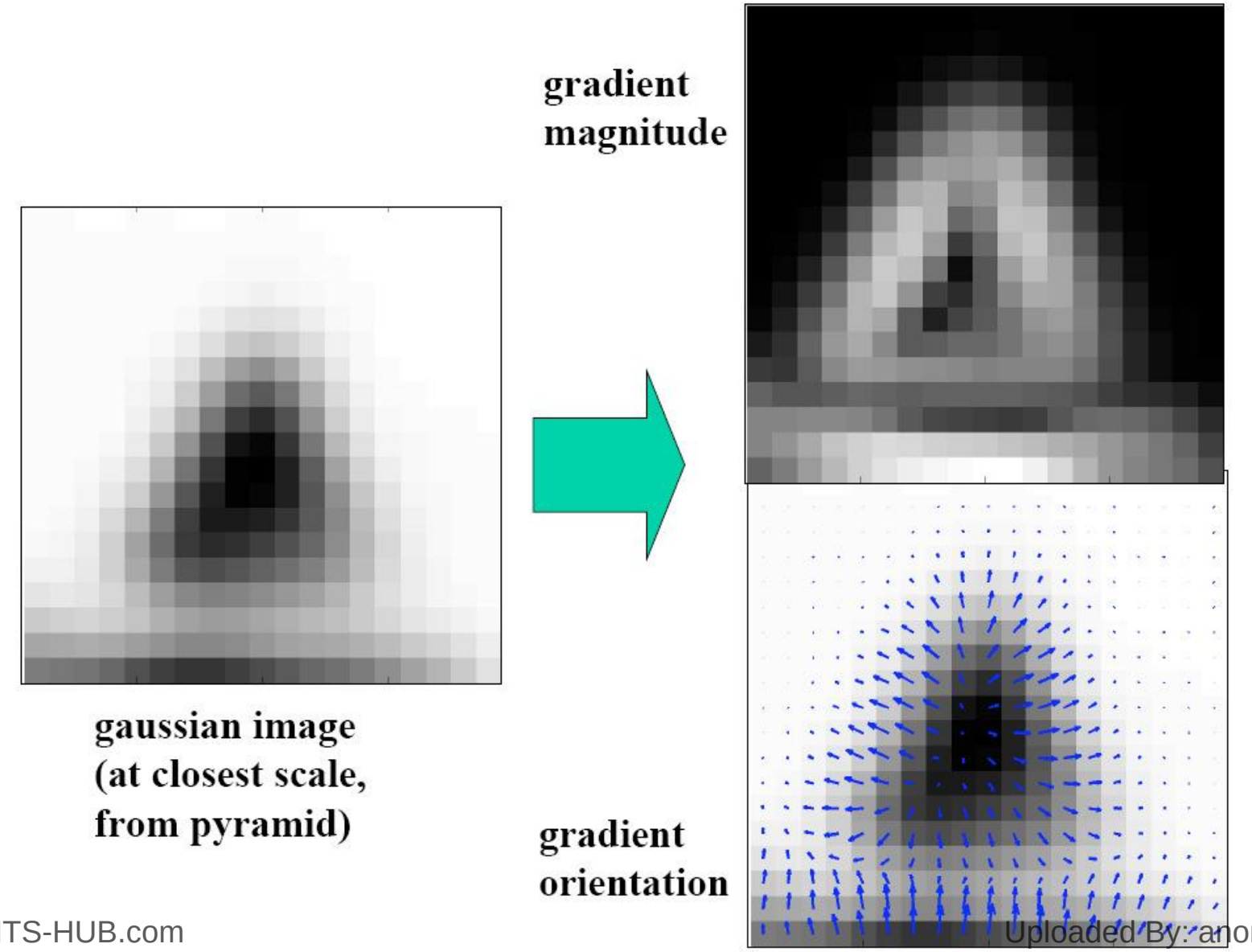
$$G_y = \frac{\partial f}{\partial y} = (Z_3 + 2Z_6 + Z_9) - (Z_1 + 2Z_4 + Z_7)$$

$$M(x,y) = \left[\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2\right]^{1/2}$$

$$\alpha = \tan^{-1}\left(\frac{G_y}{G_x}\right)$$

- Some SIFT implementations precompute the gradient magnitude and direction for all pixels in the image pyramid during scale space construction. This avoids recalculating them for each keypoint.

Orientation assignment

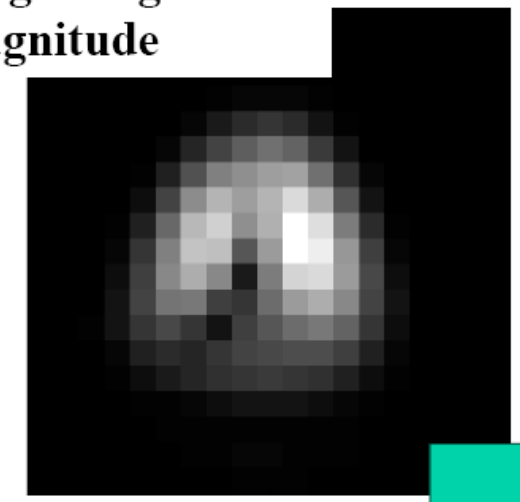


Step 5: Assign an orientation to the keypoints

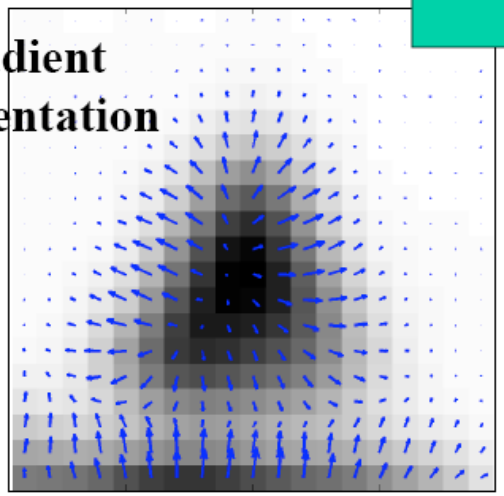
- The magnitude and orientation is calculated for all pixels around the keypoint as following:
  - Create a weighted direction histogram in a neighborhood of a key point.
  - In this histogram, the 360 degrees of orientation are broken into 36 bins (each 10 degrees).
    - Lets say the gradient direction at a certain point (in the "orientation collection region") is 18.759 degrees, then it will go into the 10-19 degree bin. And the "amount/weight" that is added to the bin is proportional to the magnitude of gradient at that point.
  - The size of the "orientation collection region" around the keypoint depends on it's scale. The bigger the scale, the bigger the collection region
    - The window size, or the "orientation collection region", is equal to the size of the kernel for Gaussian Blur of amount 1.5*sigma.

Orientation assignment

weighted gradient magnitude

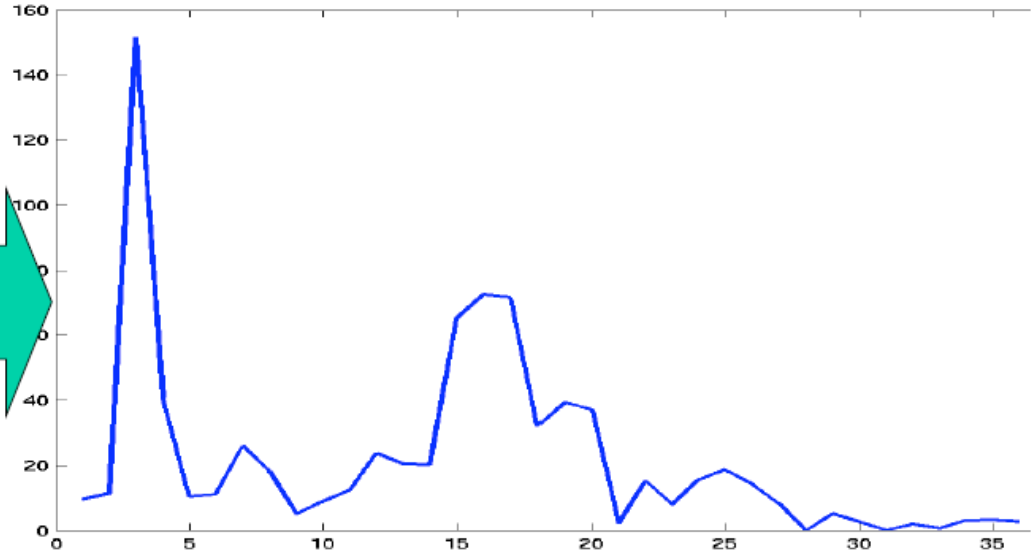


gradient orientation



weighted orientation histogram.

Each bucket contains sum of weighted gradient magnitudes corresponding to angles that fall within that bucket.



36 buckets

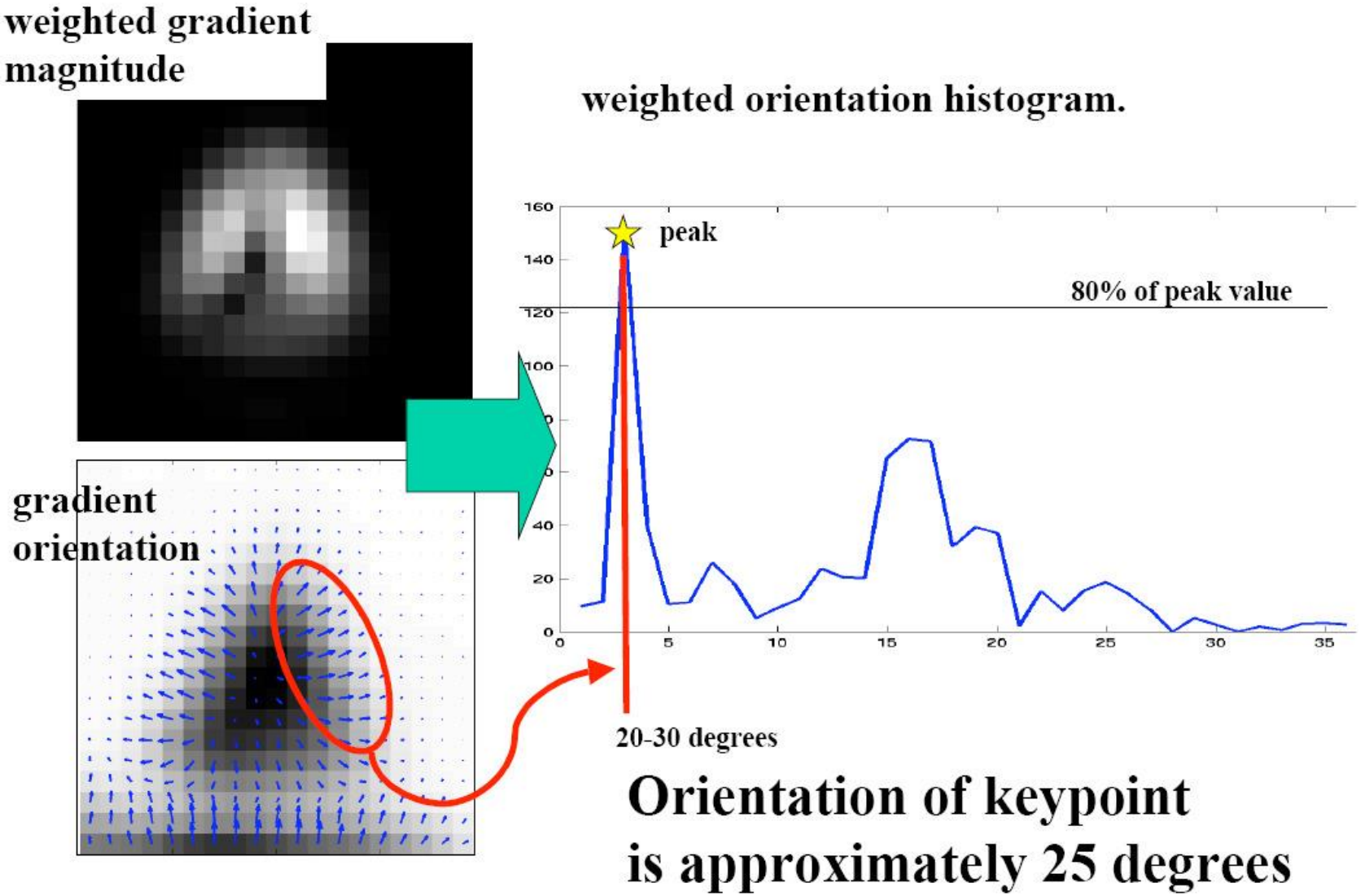
10 degree range of angles in each bucket, i.e.

0 <=ang<10 : bucket 1

10<=ang<20 : bucket 2

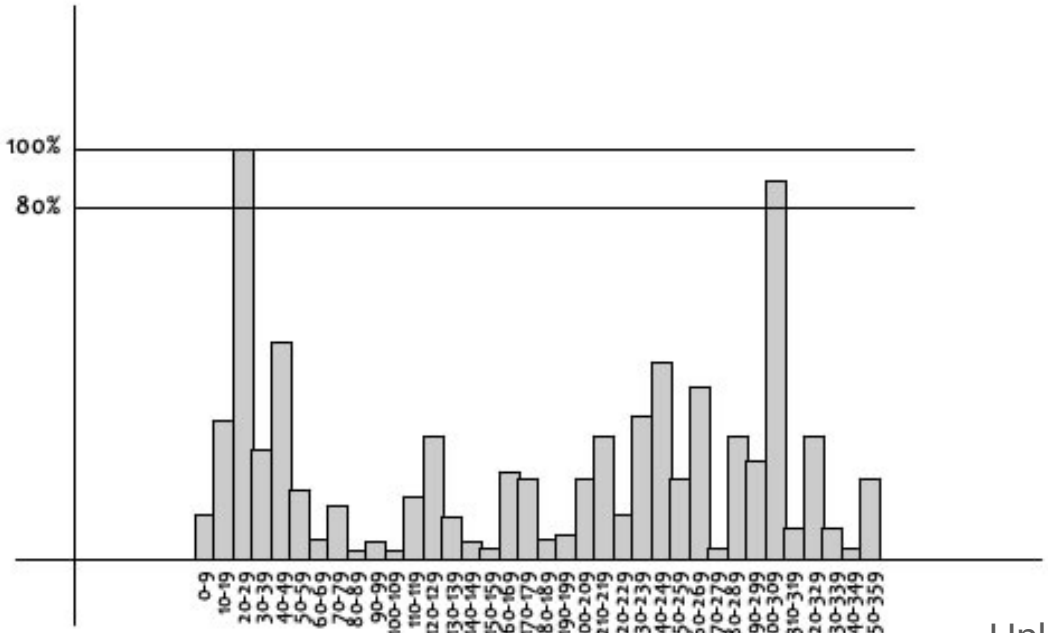
20<=ang<30 : bucket 3 …

Orientation assignment



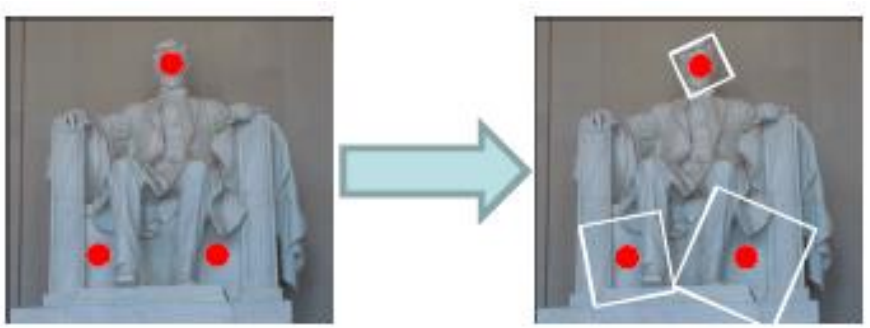
Step 5: Assign an orientation to the keypoints

- In the histogram below, the peaks at 20-29 degrees. So, the keypoint is assigned orientation 3 (the third bin). And the “amount” that is added to the bin is proportional to the magnitude of gradient at that point.
- Also, any peaks above 80% of the highest peak are converted into a new keypoint. This new keypoint has the same location and scale as the original. But it’s orientation is equal to the other peak. So, orientation can split up one keypoint into multiple keypoints.

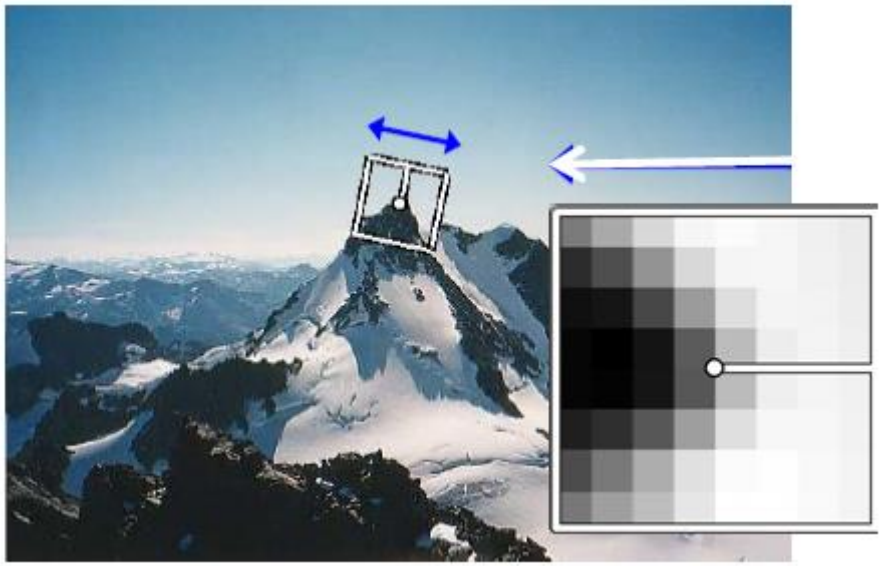


Making descriptor rotation invariant

- Rotate patch (window around keypoint) according to its dominant gradient orientation to the horizontal orientation
  - The dominant orientation will be horizontal orientation
  - This puts the patches into a canonical orientation.

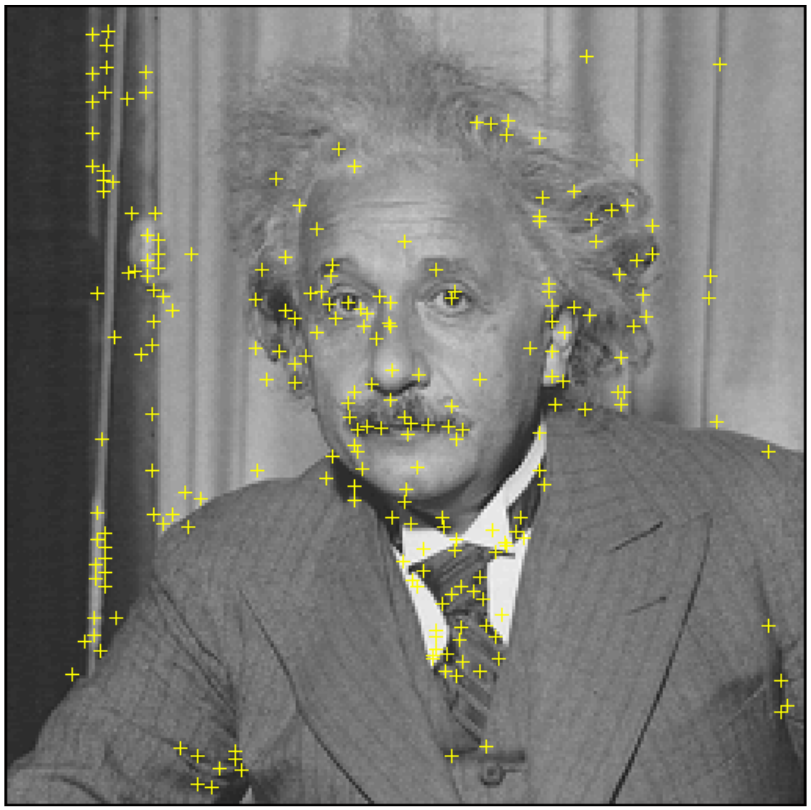


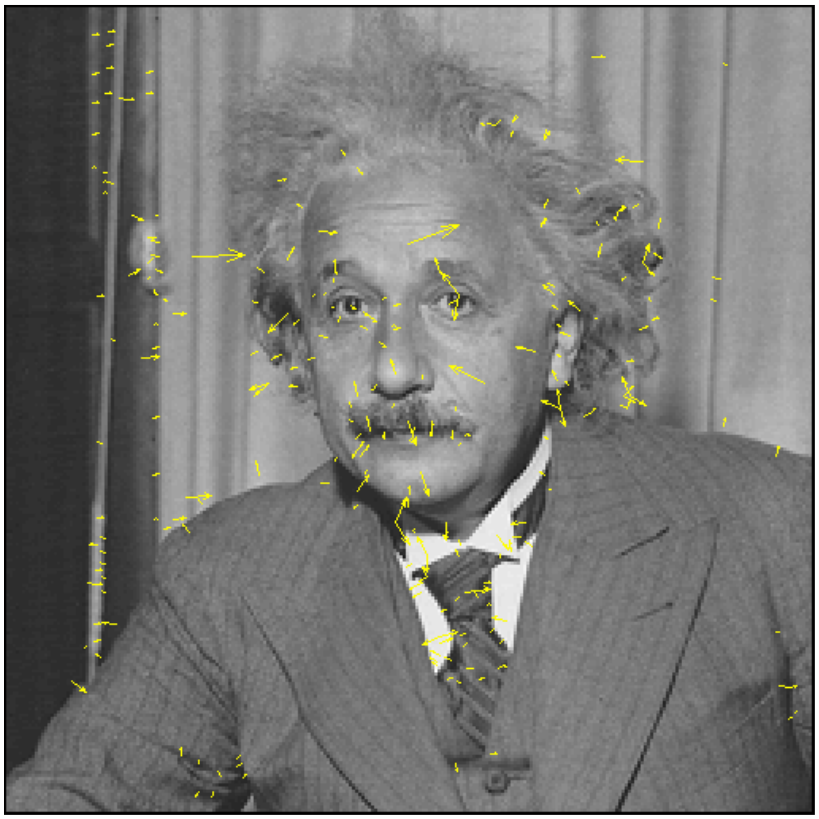
- Make scaling according to the arrow length
  - Eliminate scaling problem



Orientation assignment

Orientation Visualization



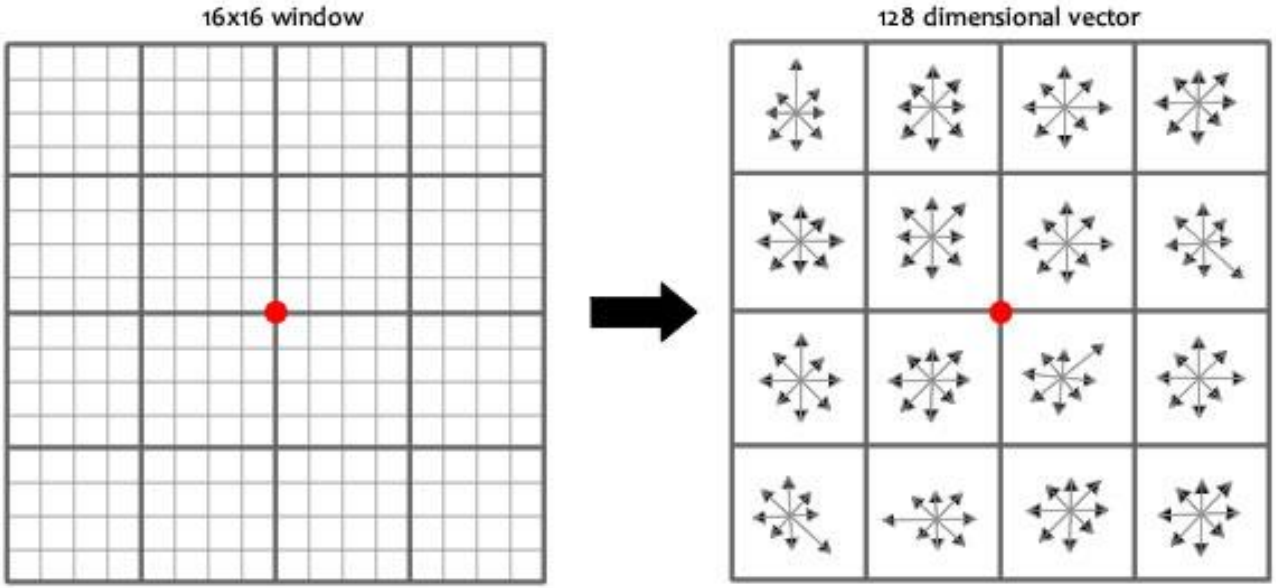


Step 6: Generate SIFT features

- Now we create a fingerprint for each keypoint. This is to identify a keypoint.
- Each point so far has x, y, σ, m, θ
  - Location x,y
  - Scale: σ
  - Gradient magnitude and orientation: m, θ
- Now we need a descriptor for the region
  - Could sample intensities around point, but…
    - Sensitive to lighting changes
    - Sensitive to slight errors in x, y, θ

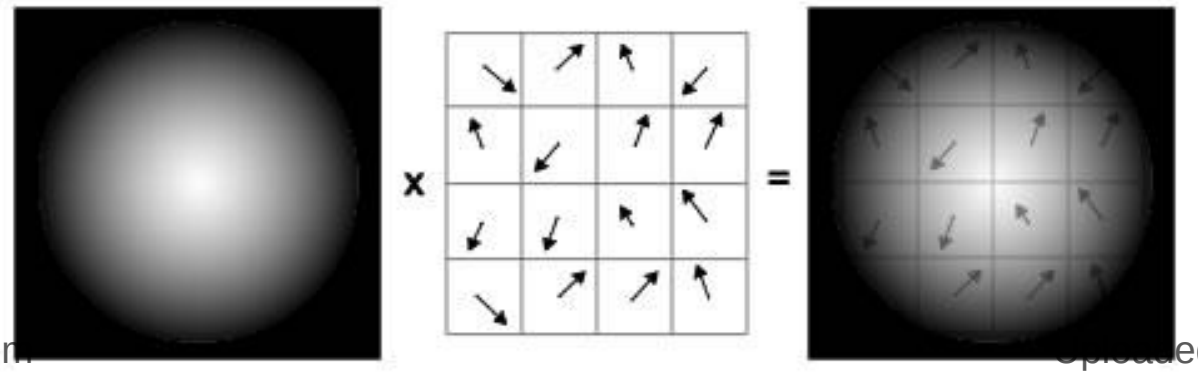
Step 6: Generate SIFT features

- The idea:
  - We want to generate a very unique fingerprint for the keypoint.
  - It should be easy to calculate.
  - We also want it to be relatively lenient when it is being compared against other keypoints.
- To do this, a 16x16 window around the keypoint were identified.
- This 16x16 window is broken into sixteen 4x4 windows.



Step 6: Generate SIFT features

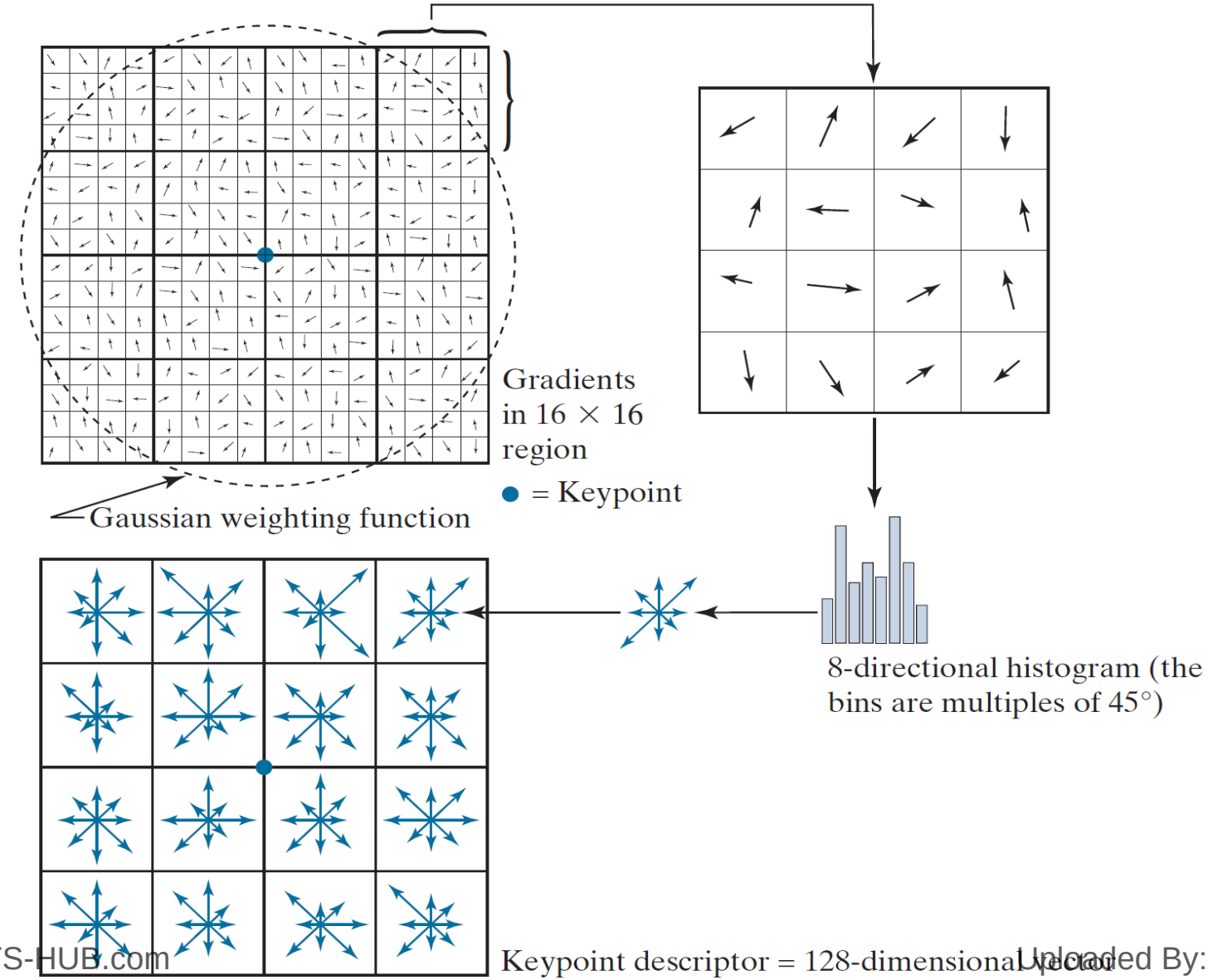
- Within each 4x4 window, gradient magnitudes and orientations are calculated.
- These orientations are put into an 8 bin histogram.
  - Any gradient orientation in the range 0-44 degrees add to the first bin. 45-89 add to the next bin. And so on.
- The amount added to the bin depends on the magnitude of the gradient.
- Unlike the past, the amount added also depends on the distance from the keypoint.
  - So gradients that are far away from the keypoint will add smaller values to the histogram.
  - This is done using a "gaussian weighting function". This function simply generates a gradient (it's like a 2D bell curve). You multiple it with the magnitude of orientations, and you get a weighted thingy. The farther away, the lesser the magnitude.



Step 6: Generate SIFT features

- Doing this for all 16 pixels, you would've "compiled" 16 totally random orientations into 8 predetermined bins. You do this for all sixteen 4x4 regions.
- So you end up with 4x4x8 = 128 numbers.
- Once you have all 128 numbers, you normalize them (just like you would normalize a vector in school, divide by root of sum of squares).
- These 128 numbers form the "feature vector".
- This keypoint is uniquely identified by this feature vector.

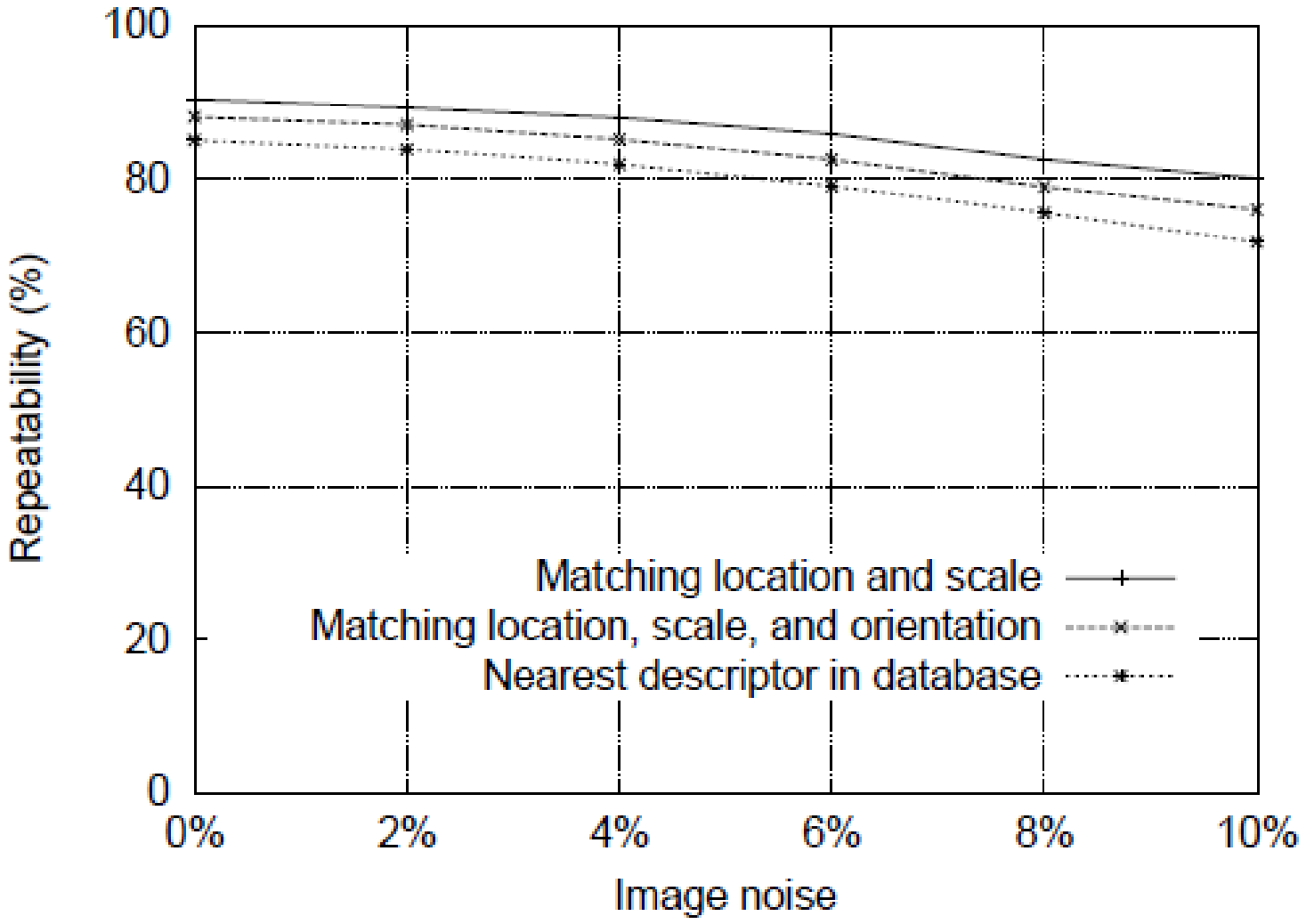
Step 6: Generate SIFT features - Summary



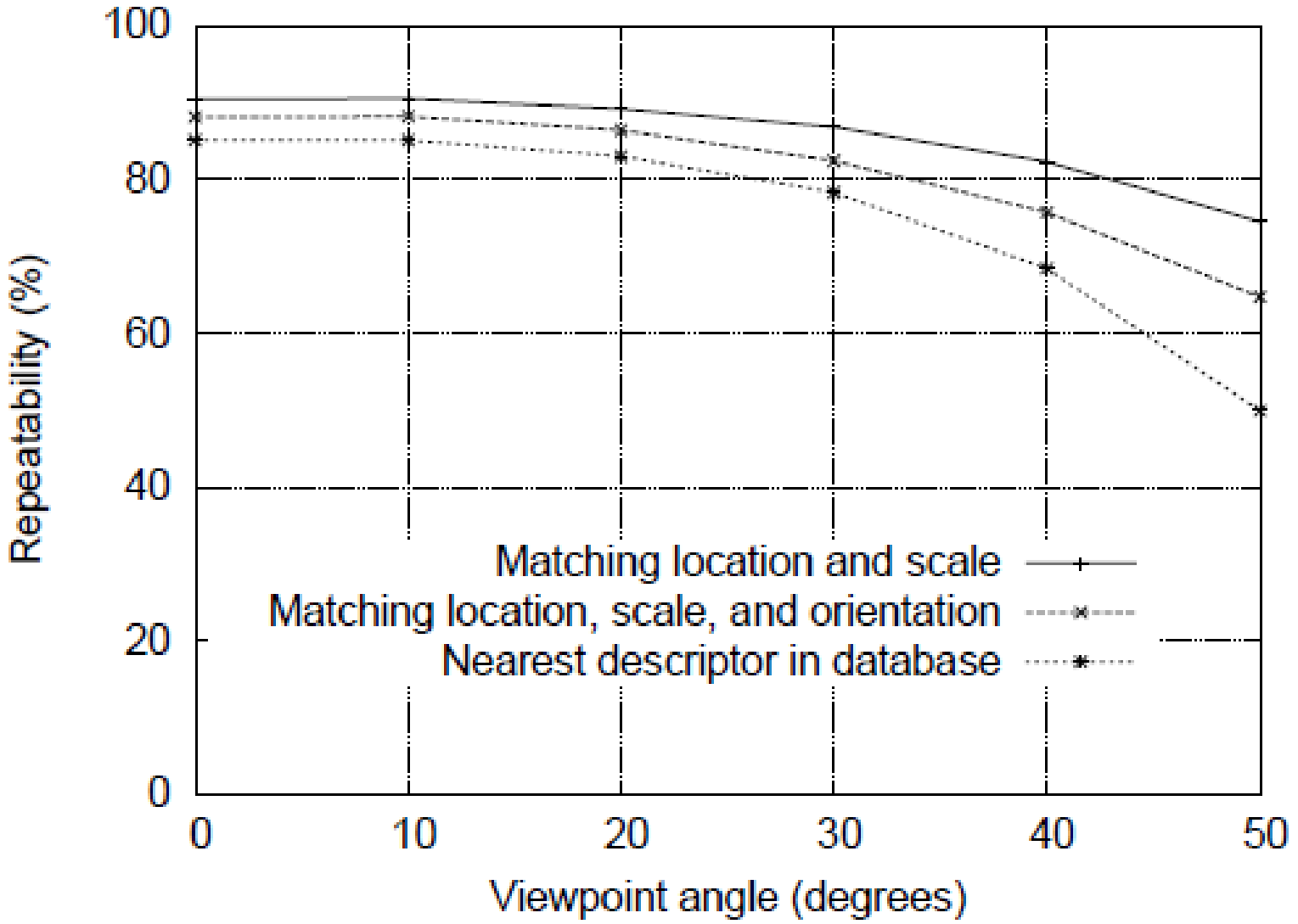
SIFT Keypoint Descriptor Summery

- Descriptor: 128-D
  - 4 by 4 patches, each with 8-D gradient angle histogram: 4×4×8 = 128
  - Normalized to reduce the effects of illumination change.
- Position: (x, y)
  - Where the feature is located at.
- Scale
  - Control the region size for descriptor extraction.
- Orientation
  - To achieve rotation-invariant descriptor.

Effect of Noise on SIFT



Effect of Orientation on SIFT

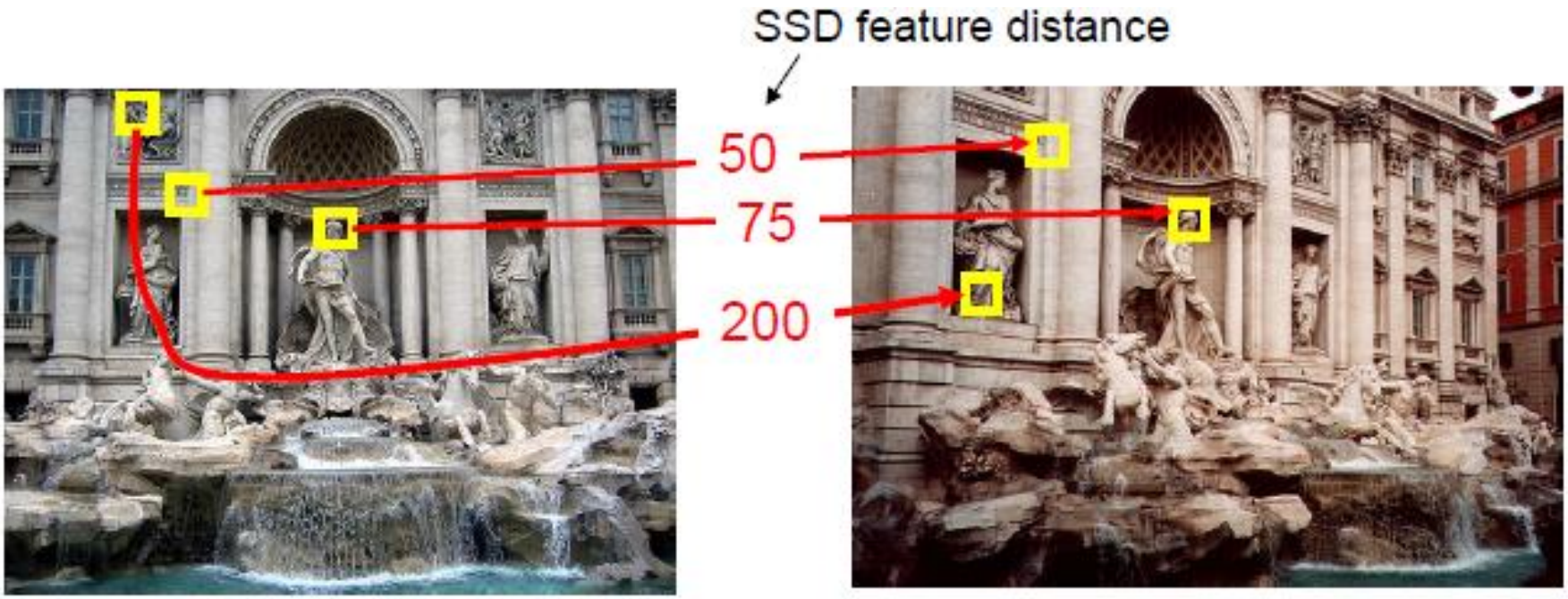


SIFT application: Image Matching

- Image matching using involves detecting keypoints and computing descriptors for these keypoints in two images.
- The keypoints and descriptors can then be matched to find correspondences between the images.
- Given a feature in I1, how to find the best match in I2?
  1. Define distance function that compares two descriptors
  2. Test all the features in I2, find the one with min distance
- How to define the difference between two features f_1, f_2?
  - Simple approach is SSD(f_1, f_2)
    - Sum of square differences between entries of the two descriptors
    - Can give good scores to very ambiguous (bad) matches

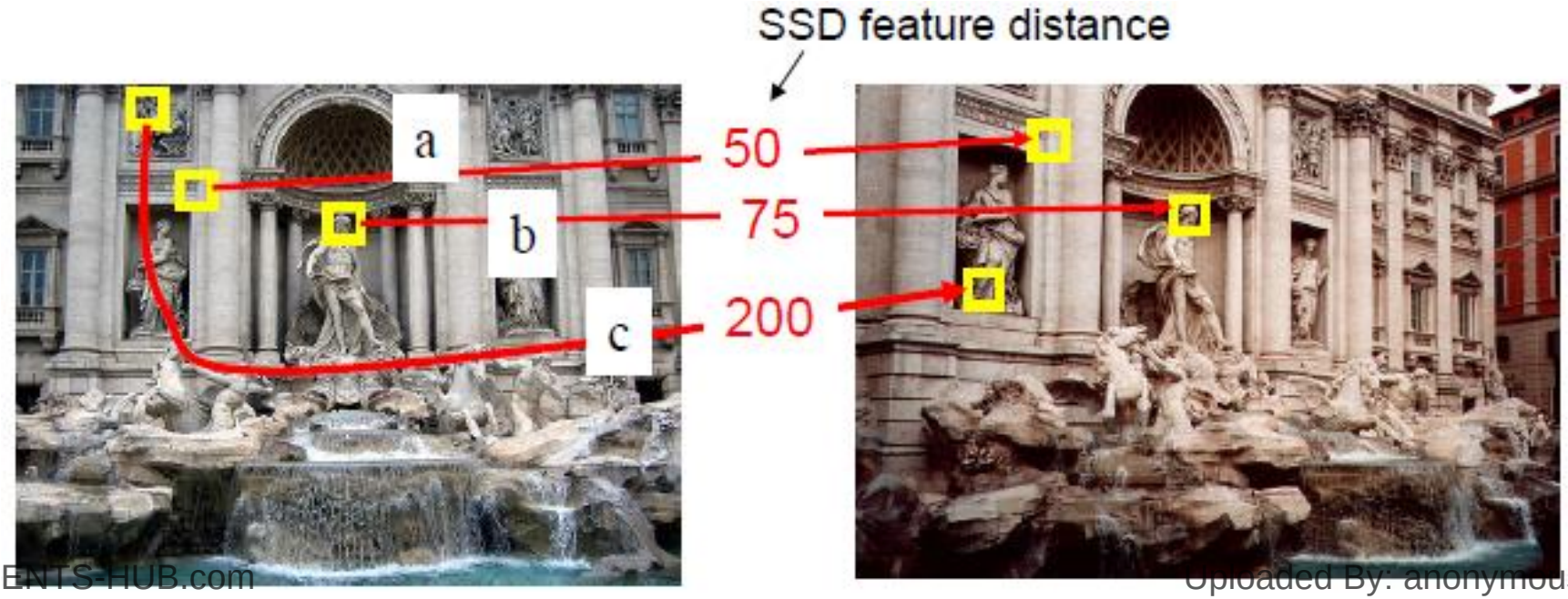
Keypoints Matching

- Suppose we use SSD
  - Small values are possible matches but how small?
  - Decision rule: Accept match if SSD < T, where T is a threshold
  - What is the effect of choosing a particular T?



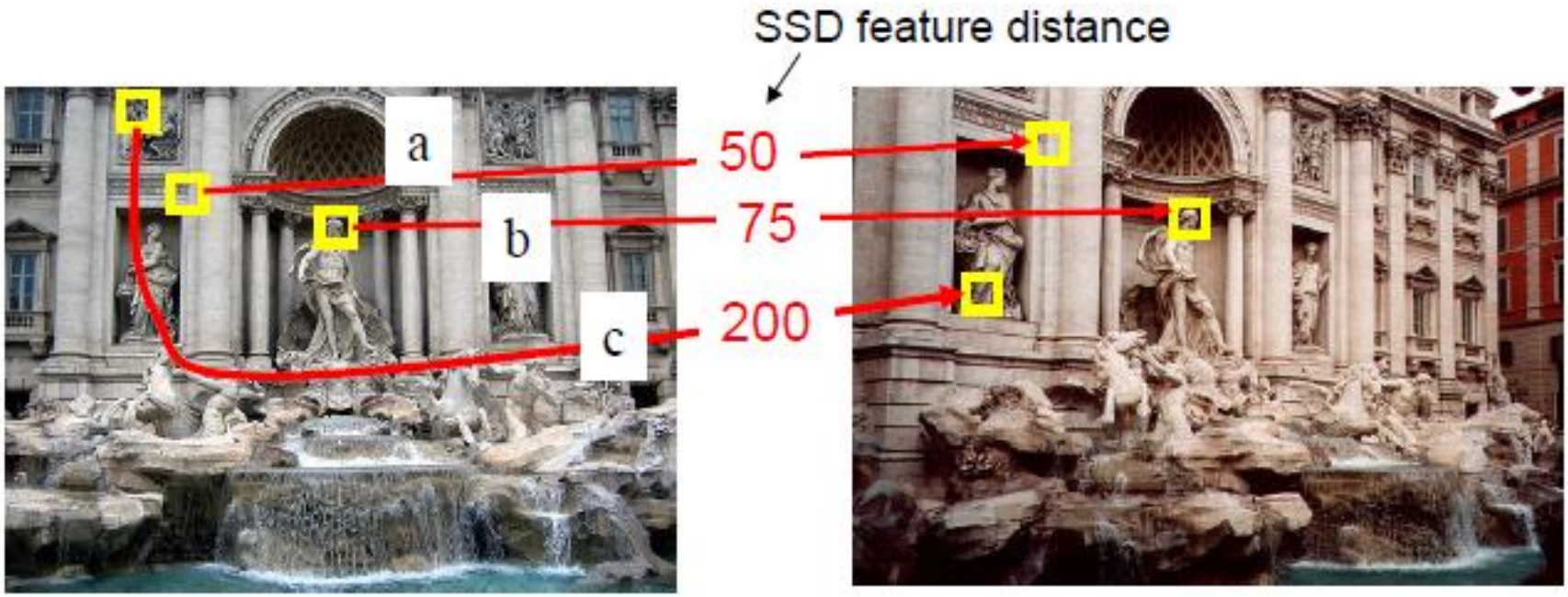
Keypoints Matching

- Distance **Decision rule:**
  - **Accept match if SSD < T**
  - Example: **Large T**, T = 250 ⇒
    - a, b, c are all accepted as matches
    - a and b are true matches (“**true positives**”) –they are actually matches
    - c is a false match (“**false positive**”) –actually not a match



Keypoints Matching

- **Decision rule:**
  - **Accept match if SSD < T**
  - Example: **Smaller T,** T = 100 ⇒
    - only a and b are accepted as matches
    - a and b are true matches (“true positives”)
    - c is no longer a “false positive”(it is a “true negative”)



Summary: SIFT Pros

- **Scale and Rotation Invariance:**
  - SIFT is designed to be invariant to scale and rotation changes, making it effective for matching objects under different viewing conditions.
- **Distinctiveness:**
  - SIFT descriptors are designed to be distinctive, allowing for robust matching of keypoints even in the presence of occlusions or changes in illumination.
- **Locality:**
  - SIFT descriptors are computed based on local image regions around keypoints, making them well-suited for capturing local structure and details.
- **Robustness to Noise:**
  - SIFT is relatively robust to noise and can handle images with moderate levels of noise.
- **Quantity:**
  - Many features can be generated for even small objects

Summary: SIFT Cons

- **Computationally Intensive:**
  - SIFT involves the computation of gradient orientations, image convolutions, and the generation of scale-space representations, making it computationally intensive. This can be a limitation in real-time applications or on resource-constrained devices.
- **Memory Usage:**
  - The generation of scale-space images and the storage of keypoint descriptors can consume significant memory, especially when dealing with large images or a large number of keypoints.
- **Sensitivity to Parameters:**
  - SIFT performance can be sensitive to parameter choices, such as the number of histogram orientations, and may require tuning for optimal results on different types of images.
- **Not Fully Rotation Invariant:**
  - While SIFT is designed to be robust to rotations, extreme rotations might still pose challenges.

HoG vs SIFT

Feature	HoG (Histogram of Oriented Gradients)	SIFT (Scale-Invariant Feature Transform)
Descriptor Focus	Distribution of gradient orientations	Keypoints and surroundings using histograms of gradient orientations
Scale Invariance	Not inherently scale-invariant (can be combined with scale spaces)	Inherently scale-invariant due to scale-space pyramid
Rotation Invariance	Not inherently rotation-invariant (can be combined with additional techniques)	Inherently rotation-invariant
Localization	Uses sliding windows for local region computation	Uses Difference of Gaussians (DoG) for scale-space representation and keypoint localization
Computational Complexity	Generally less computationally intensive, suitable for real-time applications	More computationally intensive, less suitable for real-time applications
Common Applications	Object detection, pedestrian detection	Image stitching, object recognition, image retrieval

Other Keypoints Detectors and Descriptors

- **PCA-SIFT**
- **SURF:** Speeded Up Robust Features
- **FREAK:** Fast Retina Keypoint
- **BRIEF:** Binary Robust Independent Elementary Features
- **ORB:** Oriented FAST and Rotated BRIEF - an efficient alternative to SIFT or SURF
- **BRISK:** Binary Robust Invariant Scalable Keypoints

Other Keypoints Detectors and Descriptors

Feature	SIFT	SURF	ORB	AKAZE	BRIEF
Accuracy	Excellent	Good	Good	Good	Medium
Robustness to scale and rotation changes	Excellent	Good	Good	Good	Medium
Robustness to illumination changes	Good	Good	Good	Good	Medium
Computational cost	Medium	Low	Very low	Very low	Very low
Implementation complexity	Medium	Low	Low	Low	Low

Outline

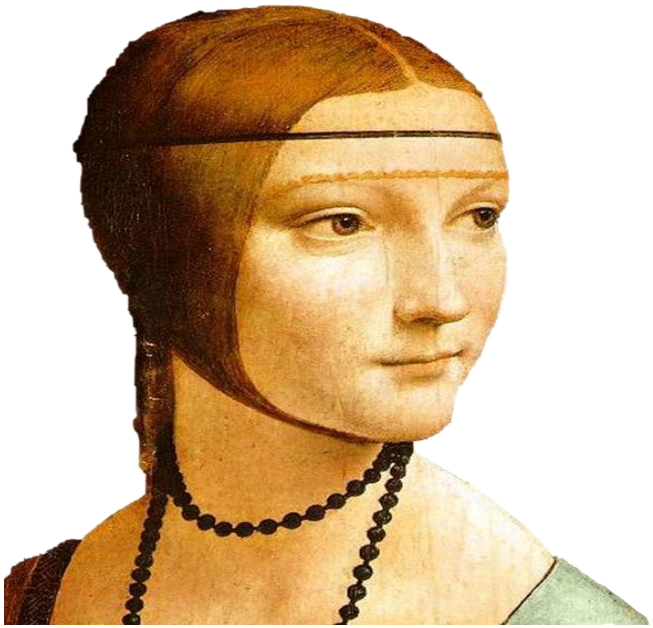
- Feature Extraction Overview
- Color Features
- Local Features
  - Edges
  - Corners
  - Interests Point
- **Visual Bag of Words**

Visual Bag of Words

- "Bag of Words" is a way to simplify object representation as a collection of their subparts
- The model originated in natural language processing, where we consider texts such as documents, paragraphs, and sentences as collections of words-effectively "bags" of words.
- In Computer Vision, we can consider an image to be a collection of image features. By incorporating frequency counts of these features, we can apply the "Bag of Words" model towards images and use this for prediction tasks such as image classification and face detection.

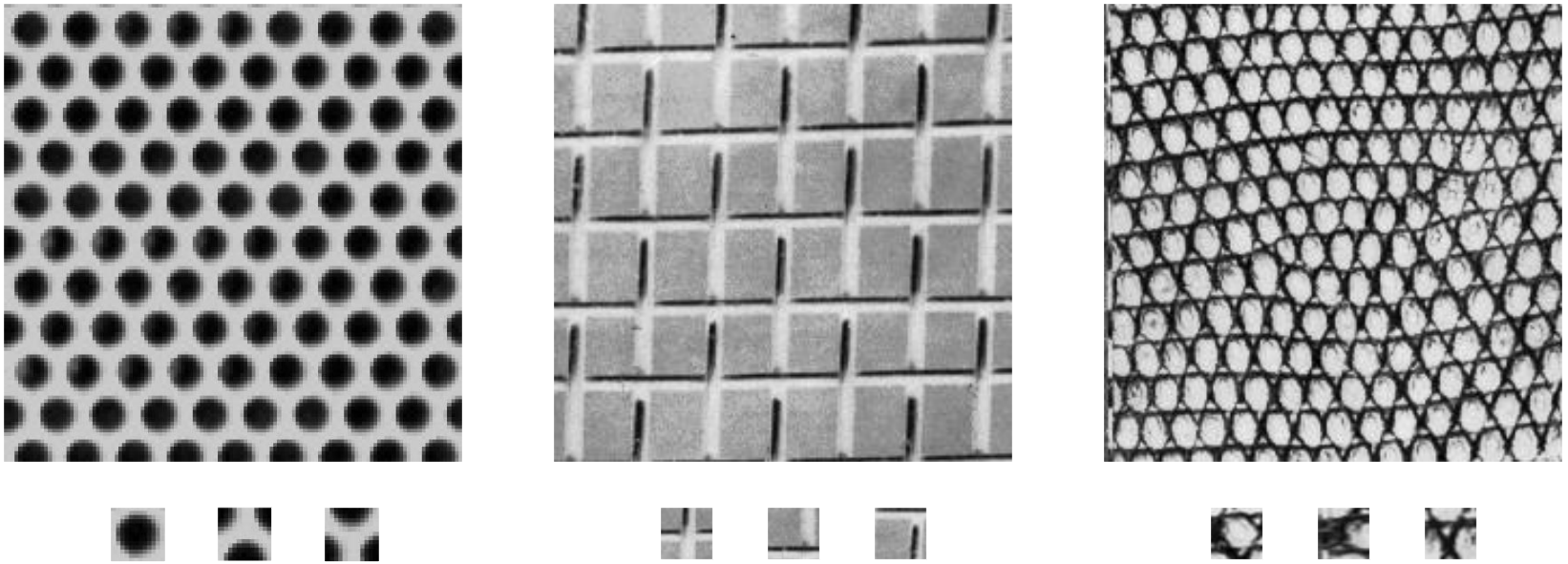
Visual Bag of Words

Object → Bag of 'words'

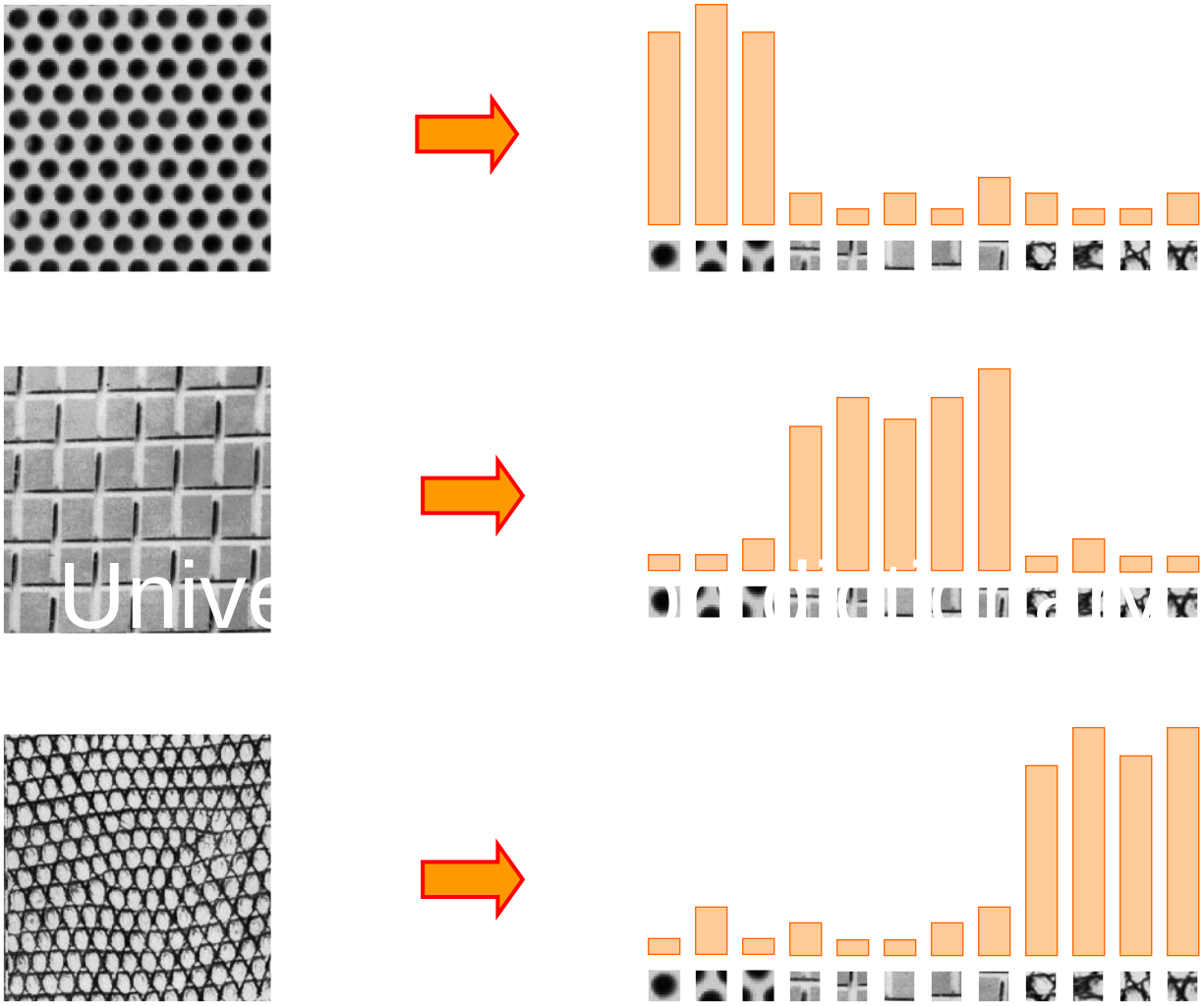


Example: Texture recognition

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



Texture recognition

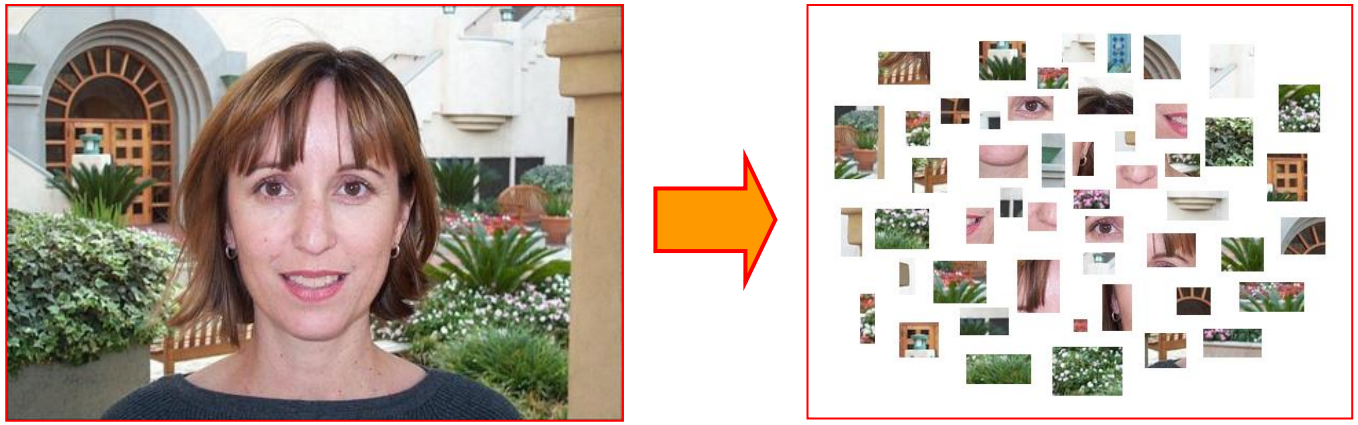


If we were to consider each texton a feature, then each image could be represented as a histogram across these features

Origin: Bag-of-words models

- Orderless document representation: frequencies of words from a dictionary Salton & McGill (1983)
- Documents consist of words which can be considered their features. Thus, every document is represented by a histogram across the words in the dictionary
- Thus, a **"bag of words"** can be viewed as a histogram representing frequencies across a vocabulary developed over a set of images or documents - new data then can be represented with this model and used for prediction tasks.

Bags of features for object recognition

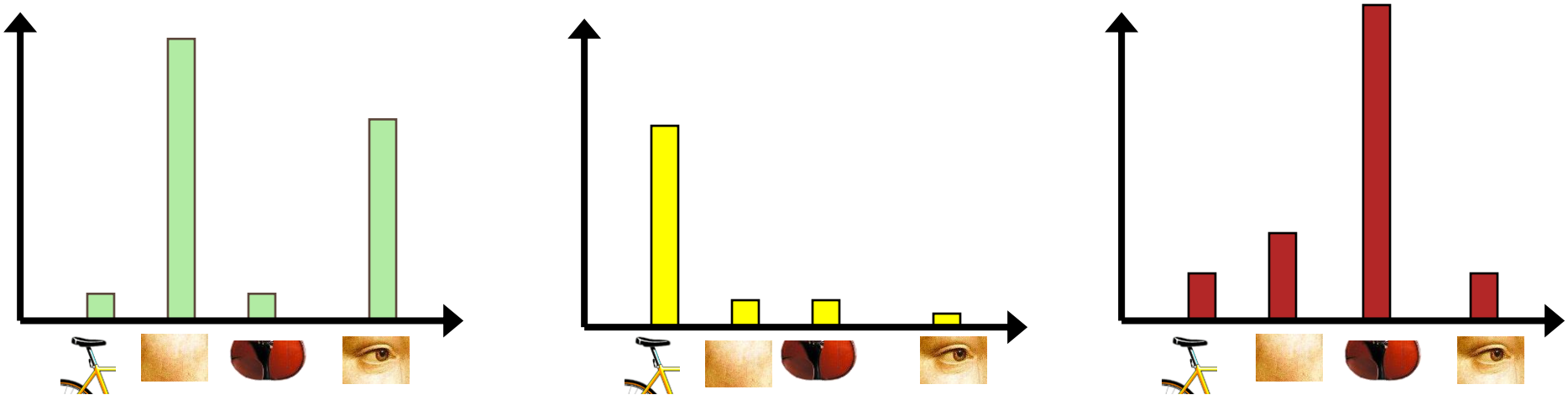


face, flowers, building

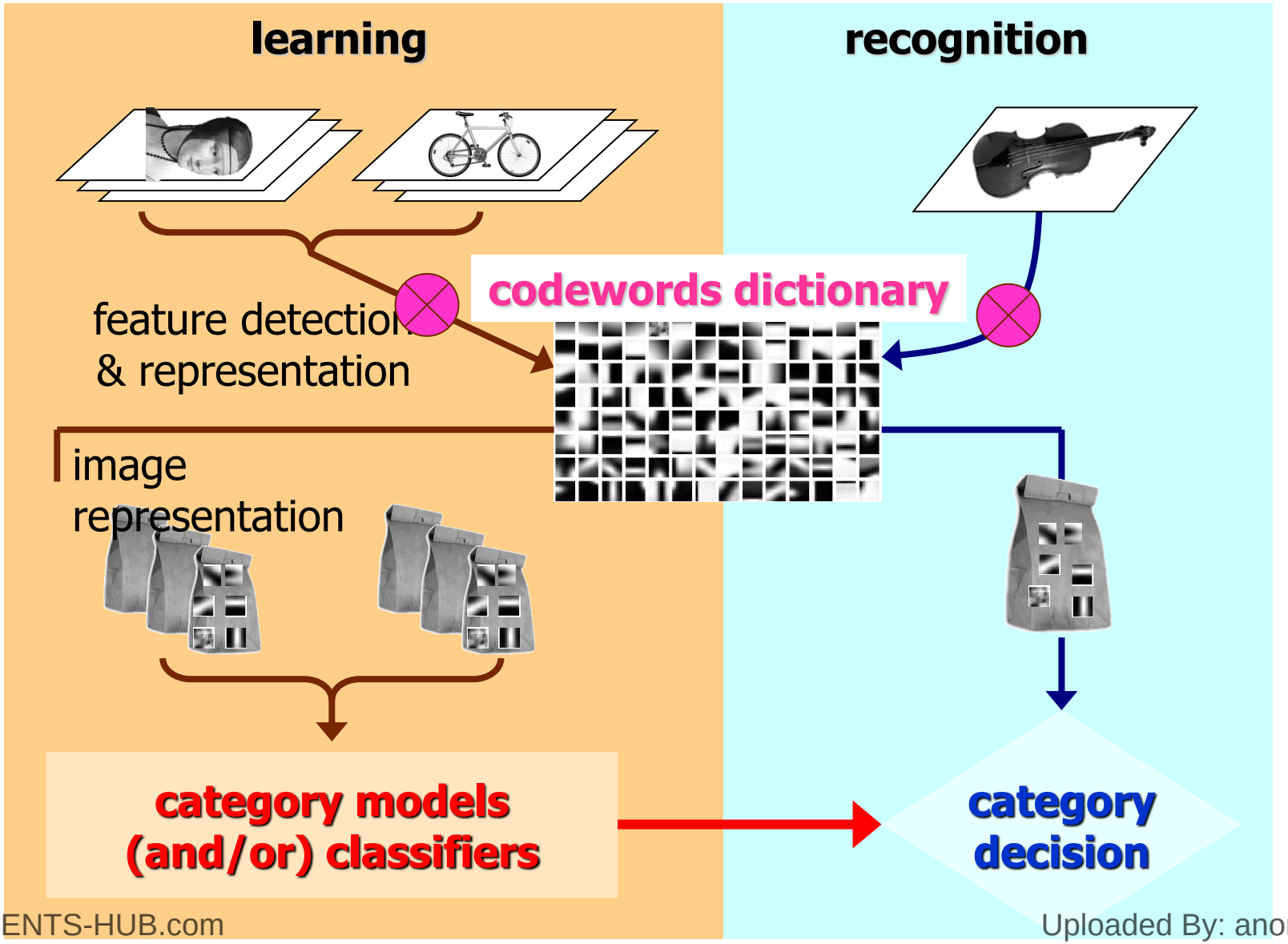
- Works pretty well for image-level classification and for recognizing object *instances*

Image classification

- Given the bag-of-features representations of images from different classes, how do we learn a model for distinguishing them?



Bag of features



Bag of features – Main Algorithm

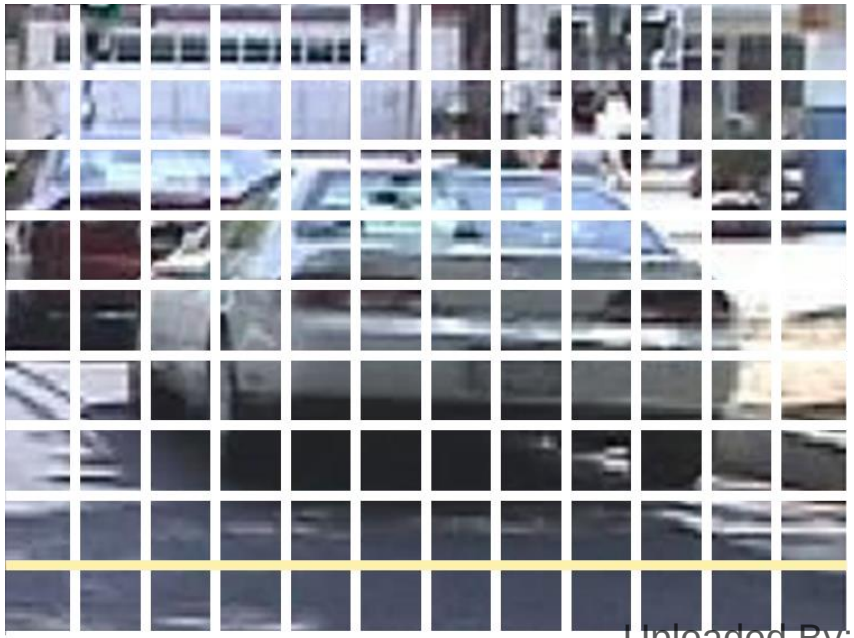
1. First, take a bunch of images, extract features, and build up a "dictionary" or "visual vocabulary" – a list of common features
   - Extract features
   - Learn "visual vocabulary"
   - Quantize features using visual vocabulary
   - Represent images by frequencies of "visual words"

2. Given a new image, extract features and build a histogram – for each feature, find the closest visual word in the dictionary

Extracting Interesting Features

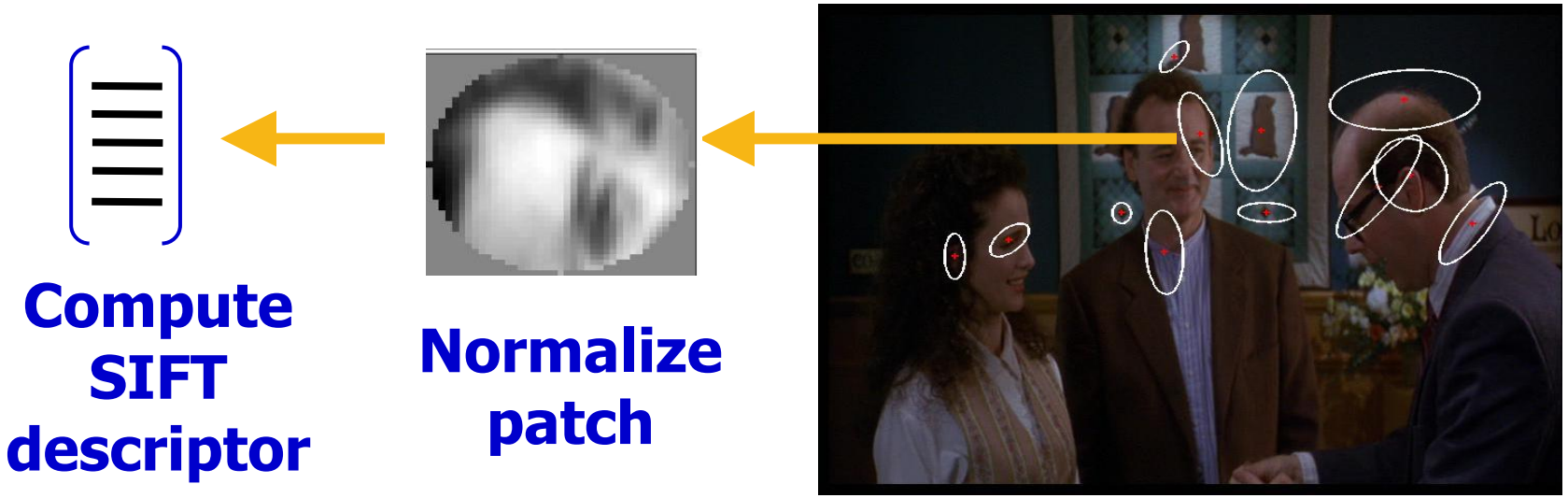
- We eventually use these features to find the most common features across our dataset of images.
- We can choose any type of feature we want to find our features. For example, we can simply split our images into a grid and grab the sub-images as features (shown below). Or, we can use corner detection of SIFT features as our features.

Regular grid for sub-images as features



Extracting Interesting Features

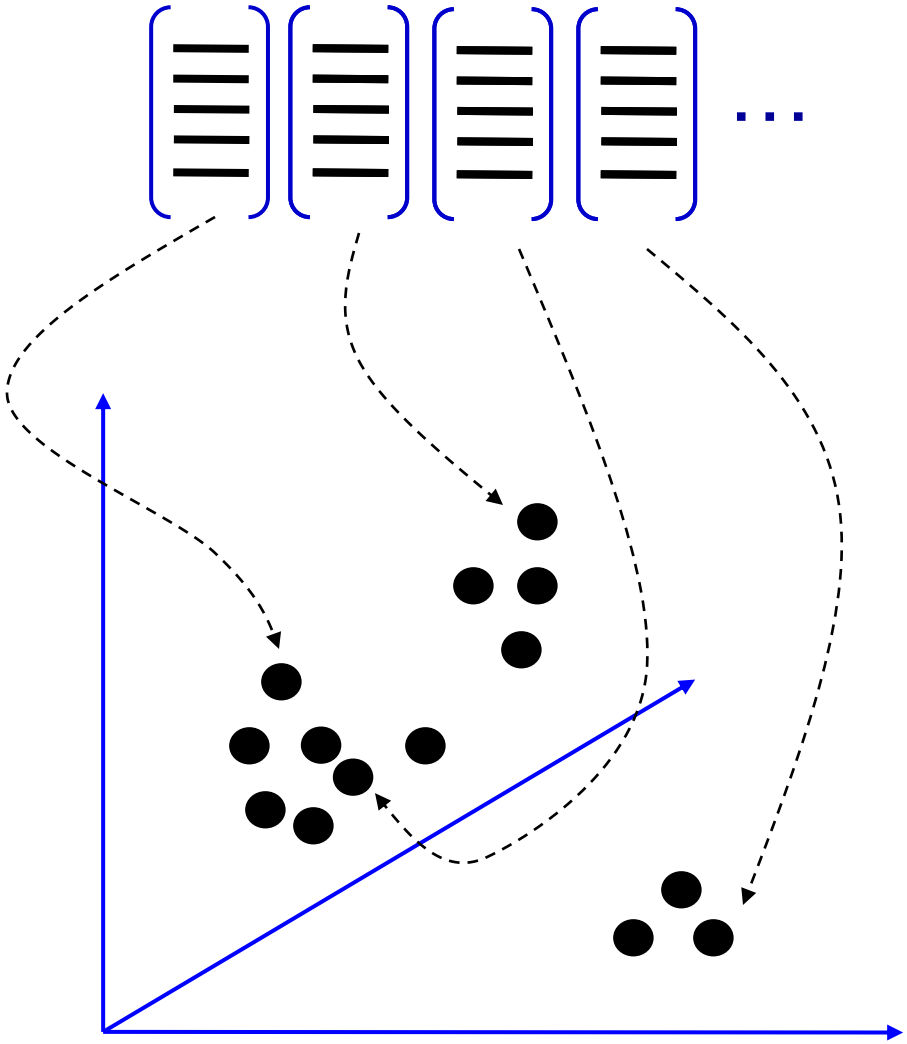
Interest Point Features



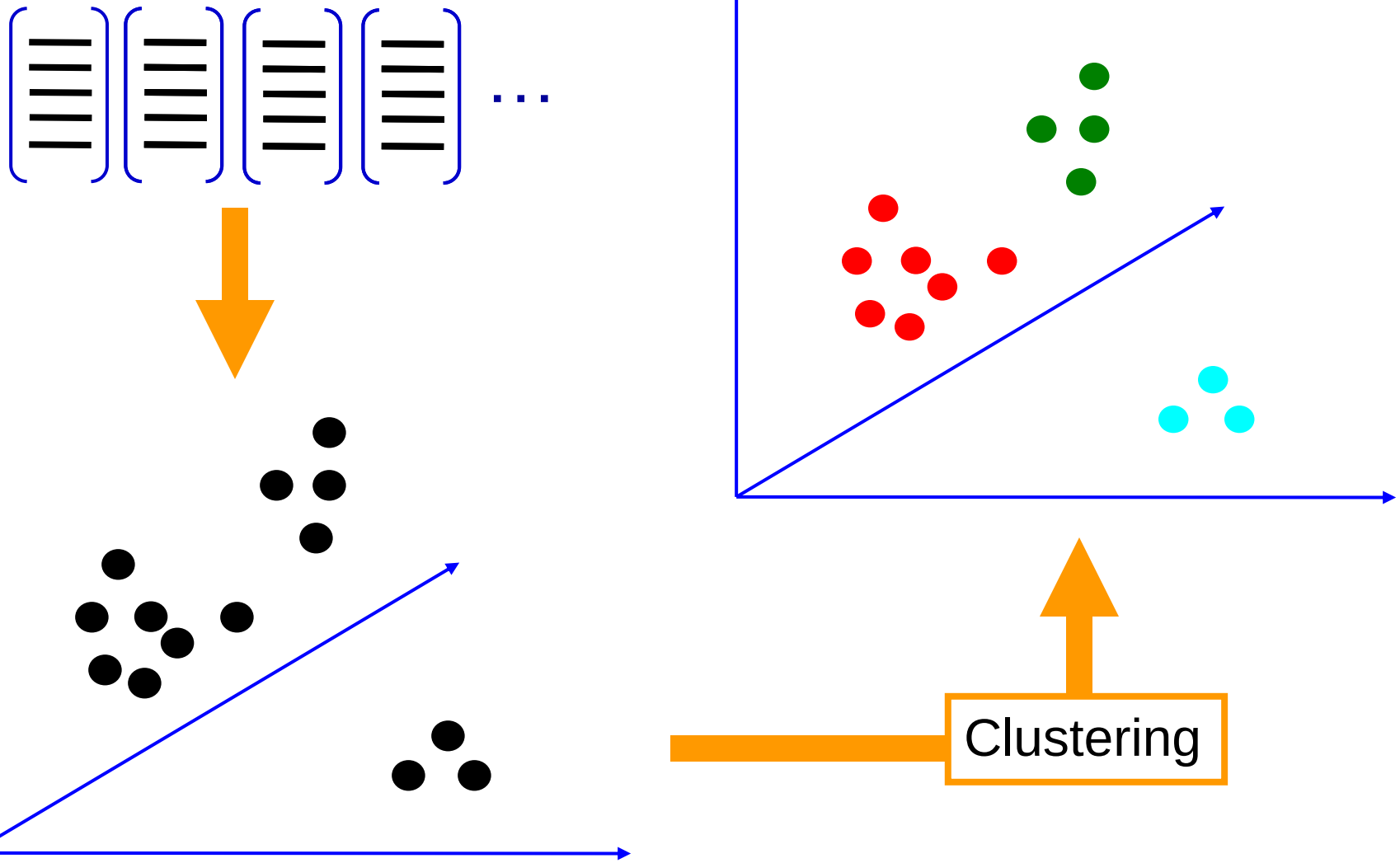
Learning Visual Vocabulary

- Once we have our features, we must turn this large feature set into a small set of "themes".
- These "themes" are analogous to the "words" in the Natural Language Processing version of the algorithm.
- In the Computer Vision application, the "words" are called textons.
- To find textons, we simply cluster our features. We can use any clustering technique (K-Means is most common) to cluster the features.
- We then use the centers of each cluster as the textons. Our set of textons is known as a visual vocabulary.
- Each cluster center produced by k-means becomes a codevector.

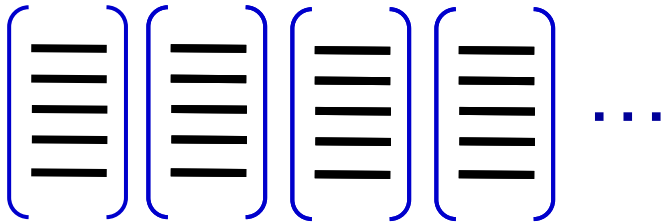
Learning the visual vocabulary

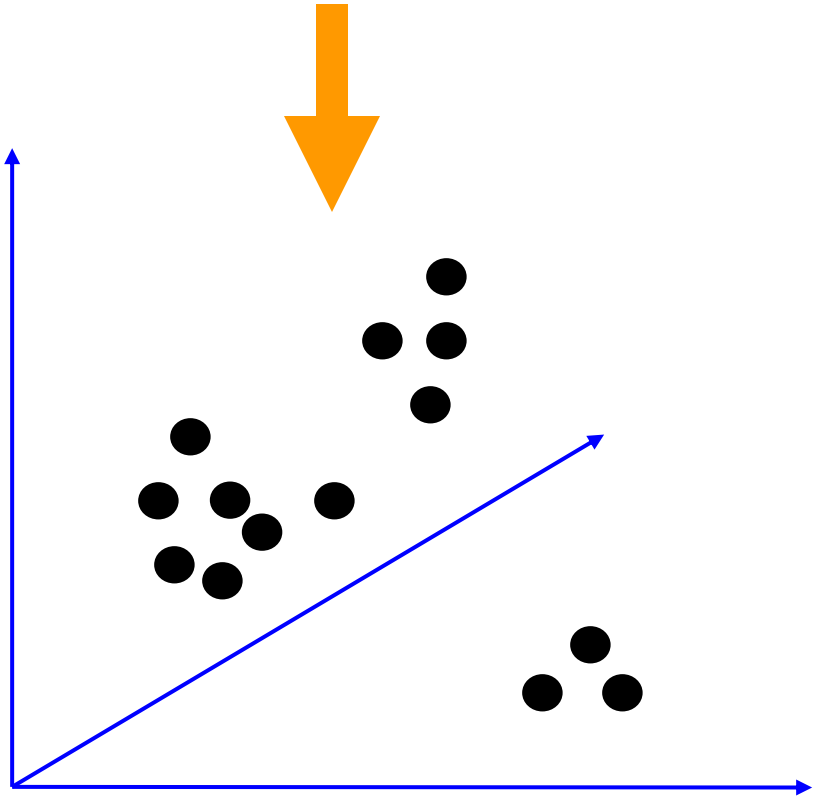


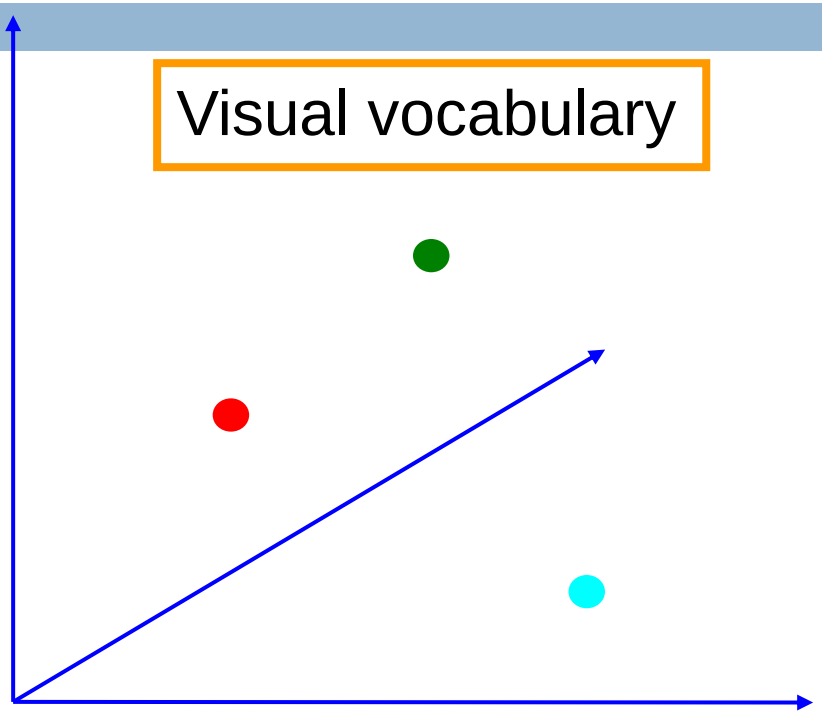
Learning the visual vocabulary



Learning the visual vocabulary





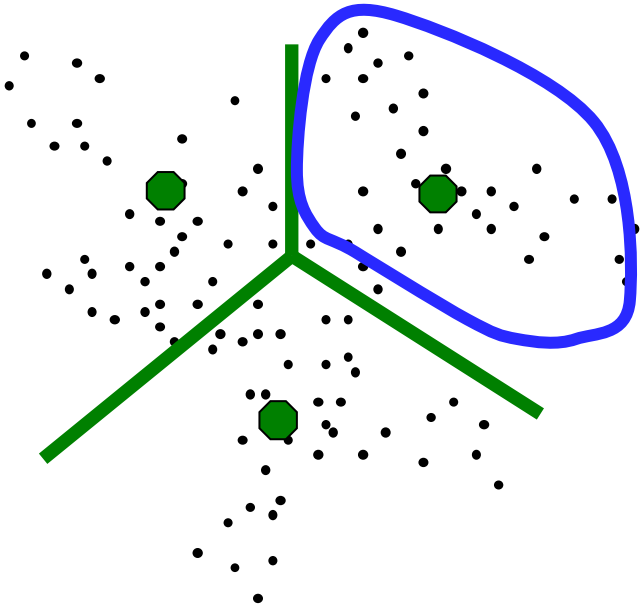


Clustering

Slide credit: Josef Sivic

Visual words

- Example: each group of patches belongs to the same visual word



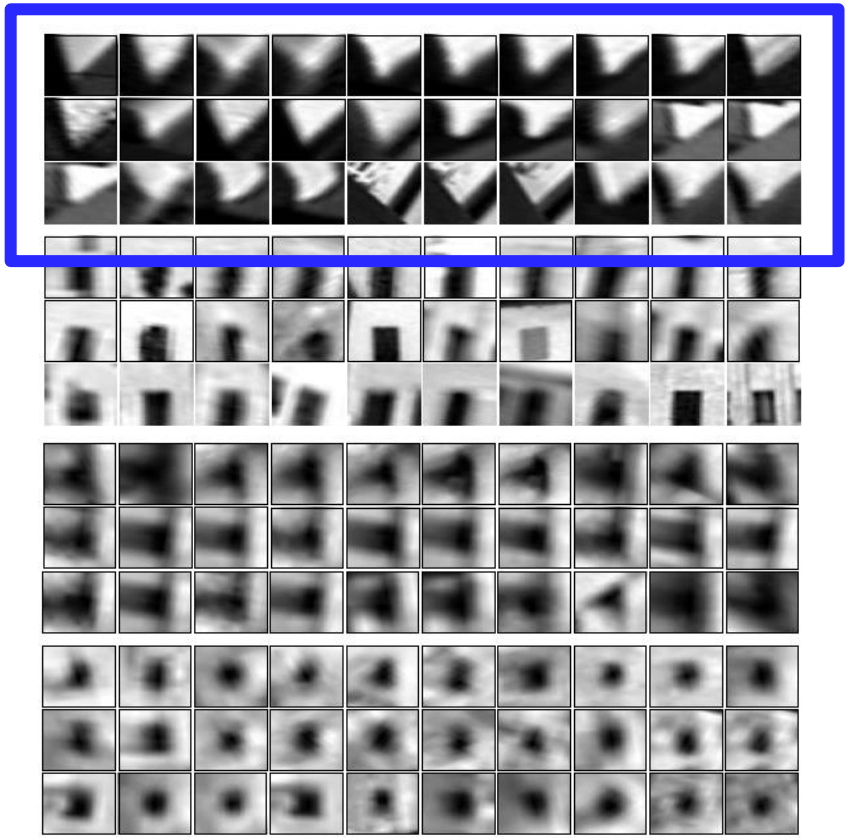
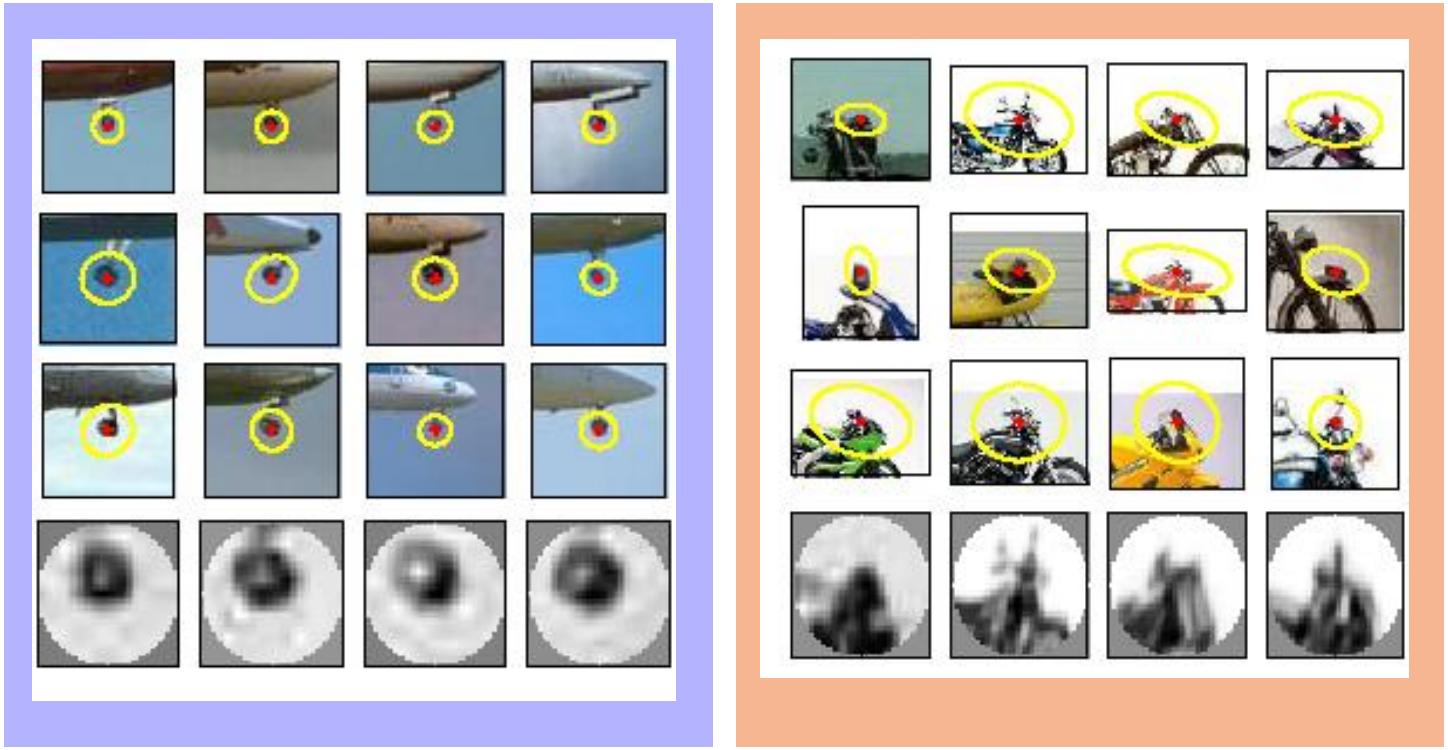
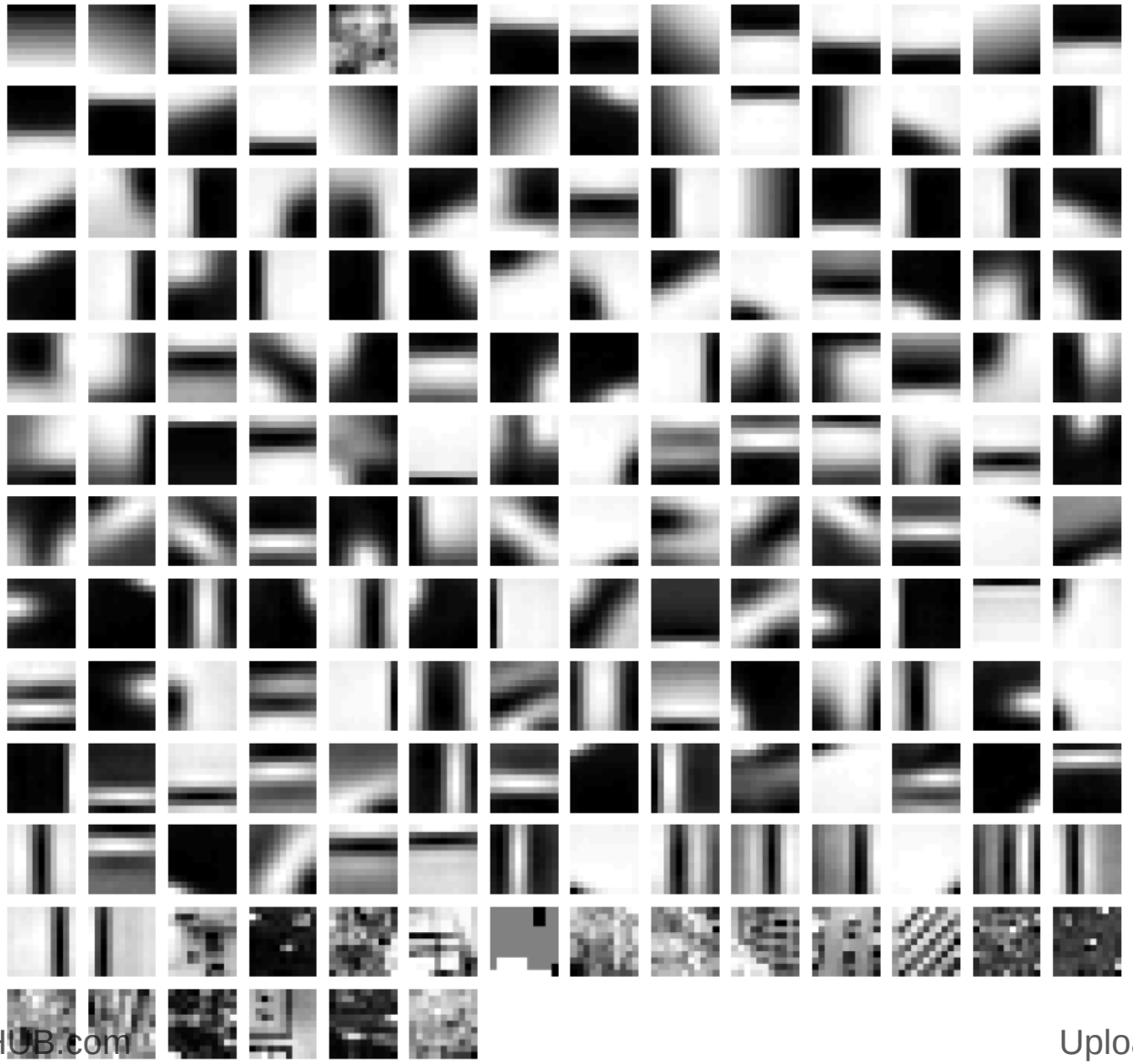


Image patch examples of visual words



Example visual vocabulary



Quantize features using visual vocabulary

- 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 = dictionary
  - Codevector = visual word = center of each feature 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

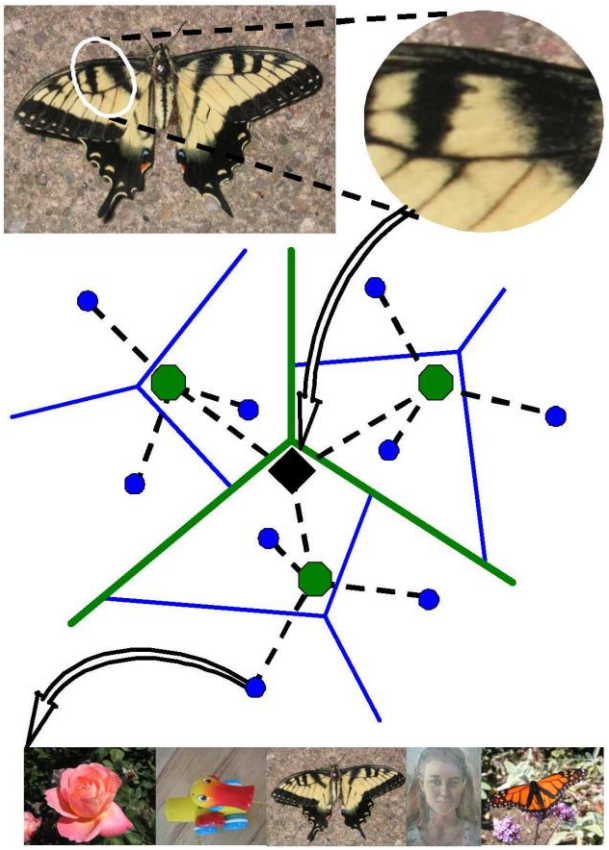
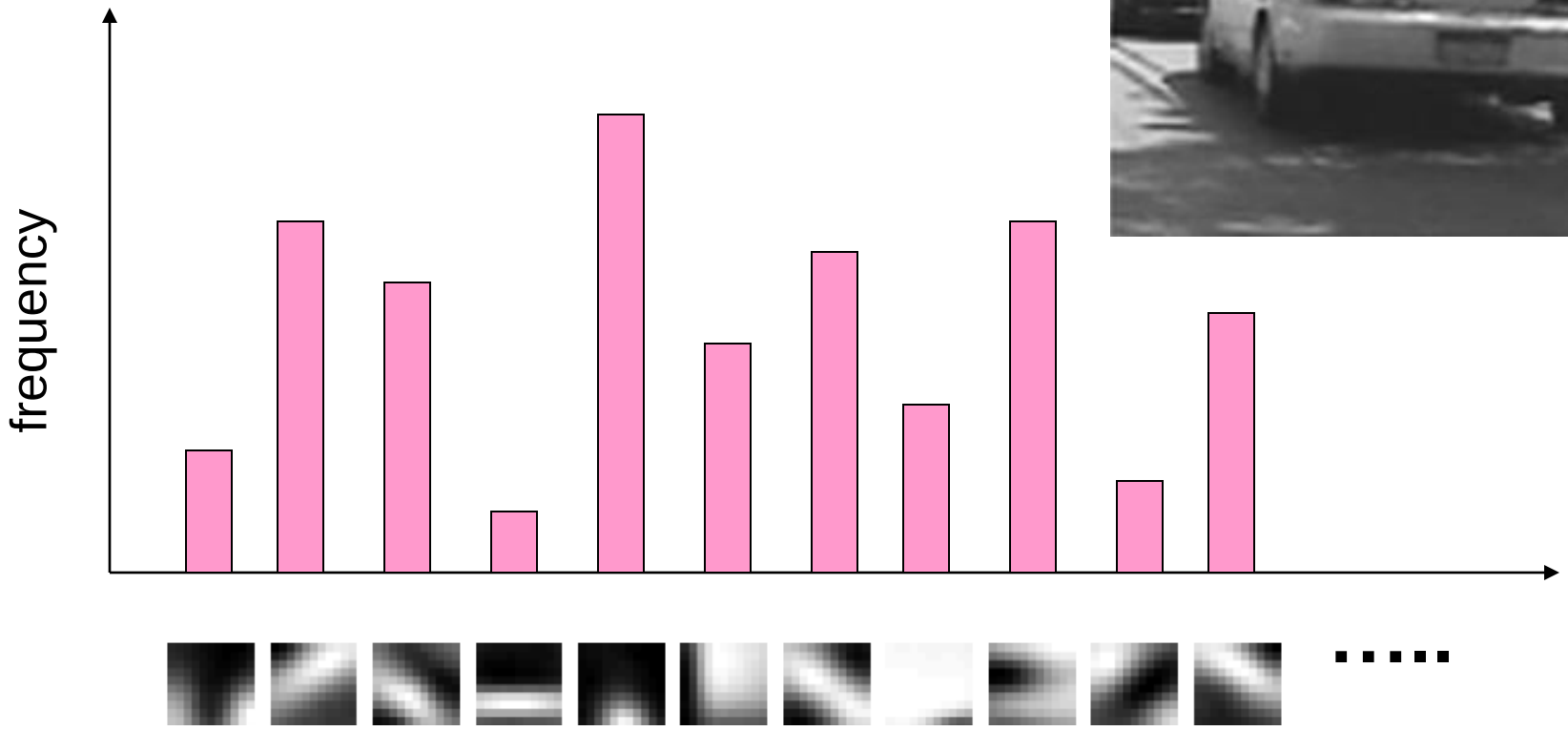


Image representation

- Once we have built our codebook, we can use it to do interesting things.
- First, we can represent every image in our dataset as a histogram of codevector frequencies. We use feature quantization to accomplish this.
- Then, we have two options, depending on our type of problem.
  - If we have a supervised learning problem (i.e. our data has labels), we can train a classifier on the histograms. This classifier will then be trained on the appearance of the textons and hence will be a robust way to distinguish between classes.
  - If we have an unsupervised learning problem (i.e. our data does not have labels), we can further cluster the histograms to find visual themes/groups within our dataset.

Image representation



Weighting the words

- Just as with text, some visual words are more discriminative than others

the, and, or vs. ***cow, AT&T, Cher***

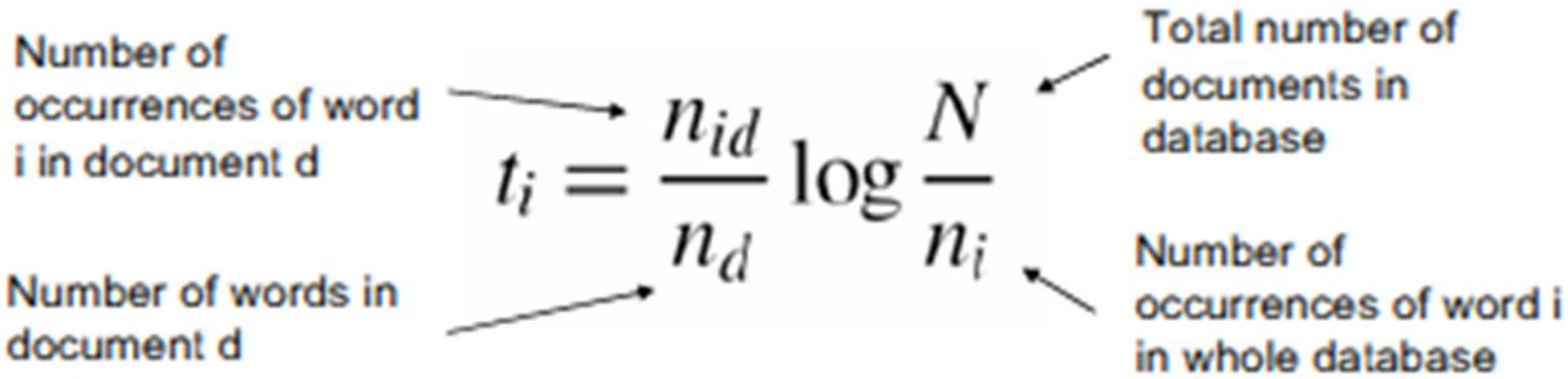
- The bigger fraction of the documents a word appears in, the less useful it is for matching
  - e.g., a word that appears in *all* documents is not helping us

TF-IDF weighting

- Instead of computing a regular histogram distance, we'll weight each word by it's *inverse document frequency*
- inverse document frequency (IDF) of word *j* =

$$\log \frac{\text{number of documents}}{\text{number of documents in which } j \text{ appears}}$$

- To compute the value of bin *j* in image *I*:



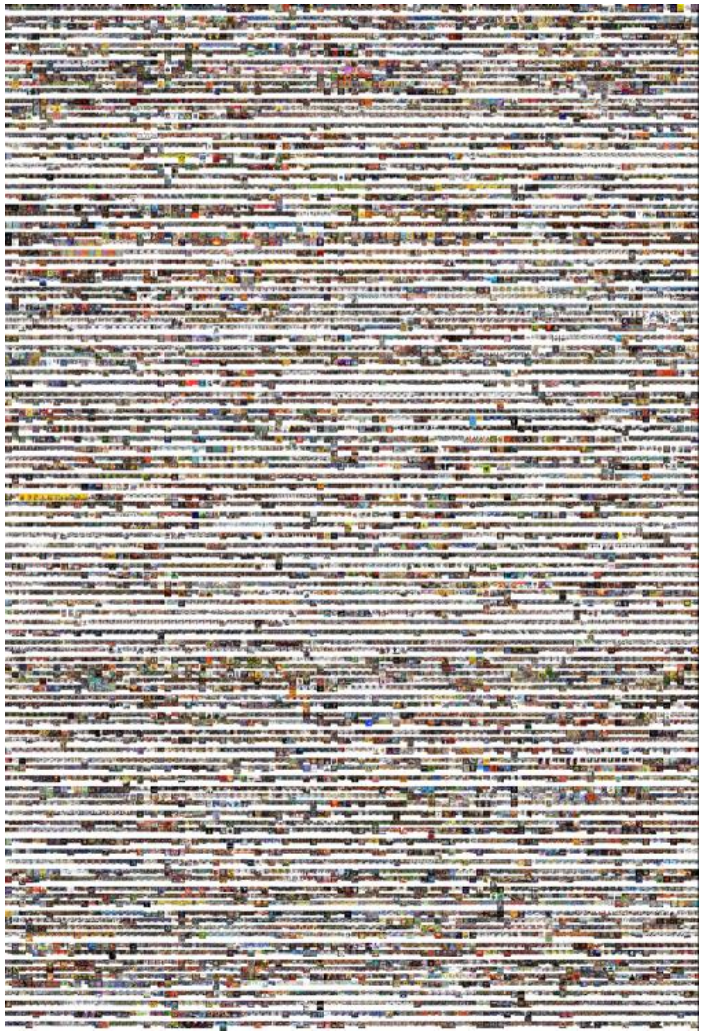
Large-scale image matching

11,400 images of game covers
(Caltech games dataset)

- Bag-of-words models have been useful in matching an image to a large database of object *instances*

how do I find this image in the database?

Large-scale image search



Build the database:

- Extract features from the database images
- Learn a vocabulary using k-means (typical k: 100,000)
- Compute *weights* for each word
- Create an inverted file mapping words → images

Inverted file

- Each image has ~1,000 features
- We have ~100,000 visual words
  - → each histogram is extremely sparse (mostly zeros)

- Inverted file
  - mapping from words to documents

```
"a":      {2}
"banana": {2}
"is":     {0, 1, 2}
"it":     {0, 1, 2}
"what":   {0, 1}
```

- Can quickly use the inverted file to compute similarity between a new image and all the images in the database
  - Only consider database images whose bins overlap the query image

Inverted file index

Image #1

Image #2

Image #3

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

…

- Can quickly use the inverted file to compute similarity between a new image and all the images in the database
  - Only consider database images whose bins overlap the query image

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

Inverted file index

New query image

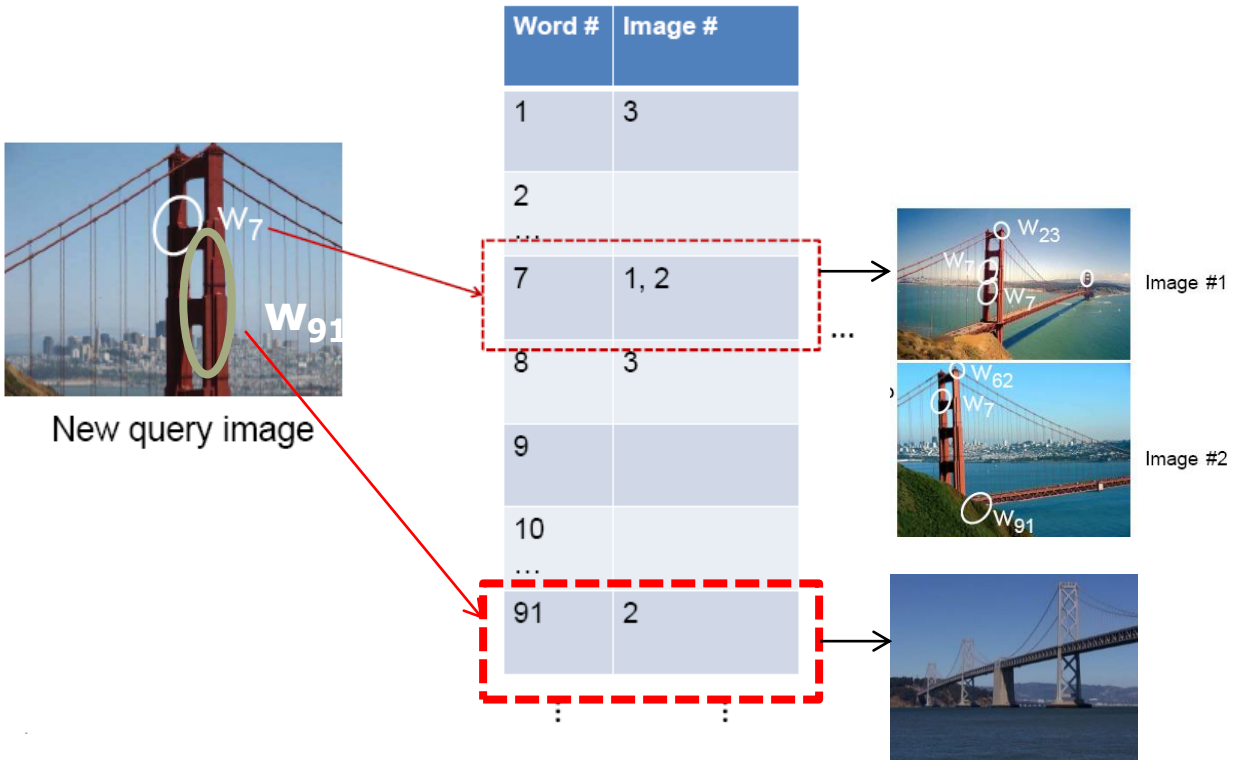
Word #	Image #
1	3
2	
…	
7	1, 2
8	3
9	
10	
…	
91	2
⋮	⋮

Image #1

Image #2

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

Inverted file index and bags of words similarity



1. Extract words in query
2. Inverted file index to find relevant frames
3. Compare word counts

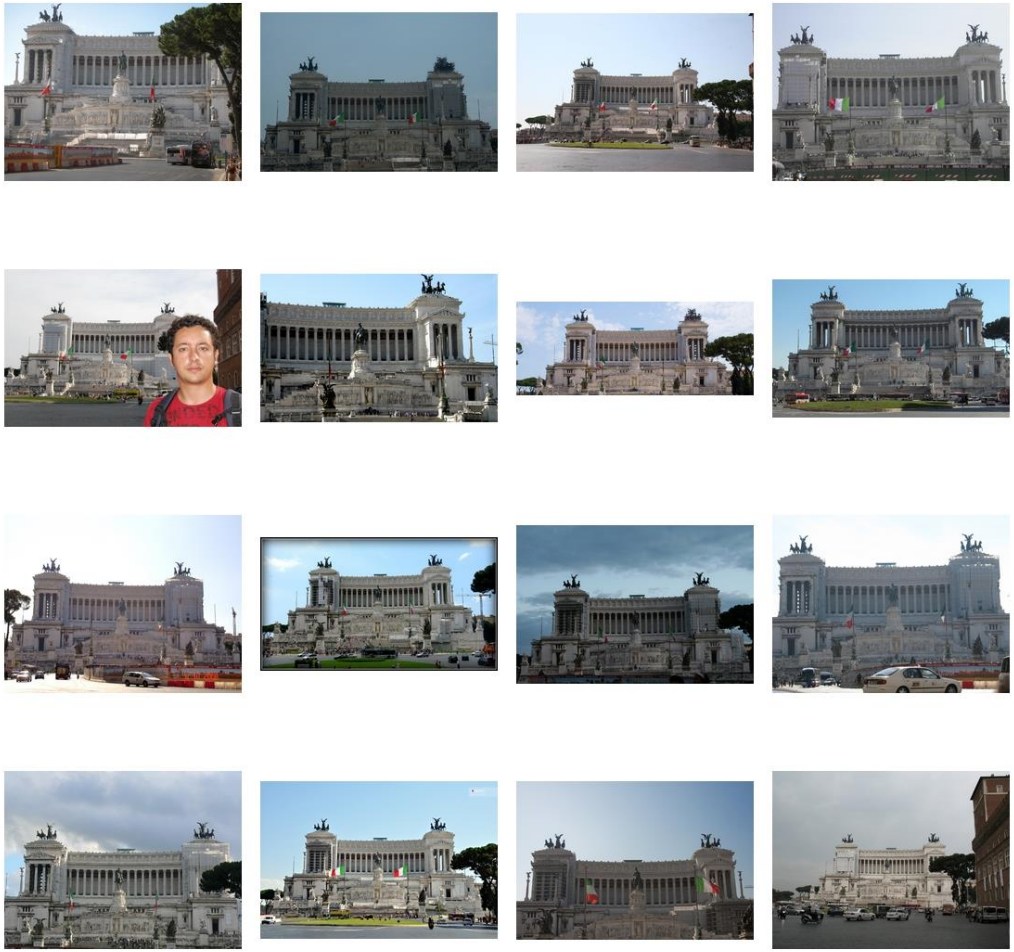
Large-scale image search

query image

top 6 results

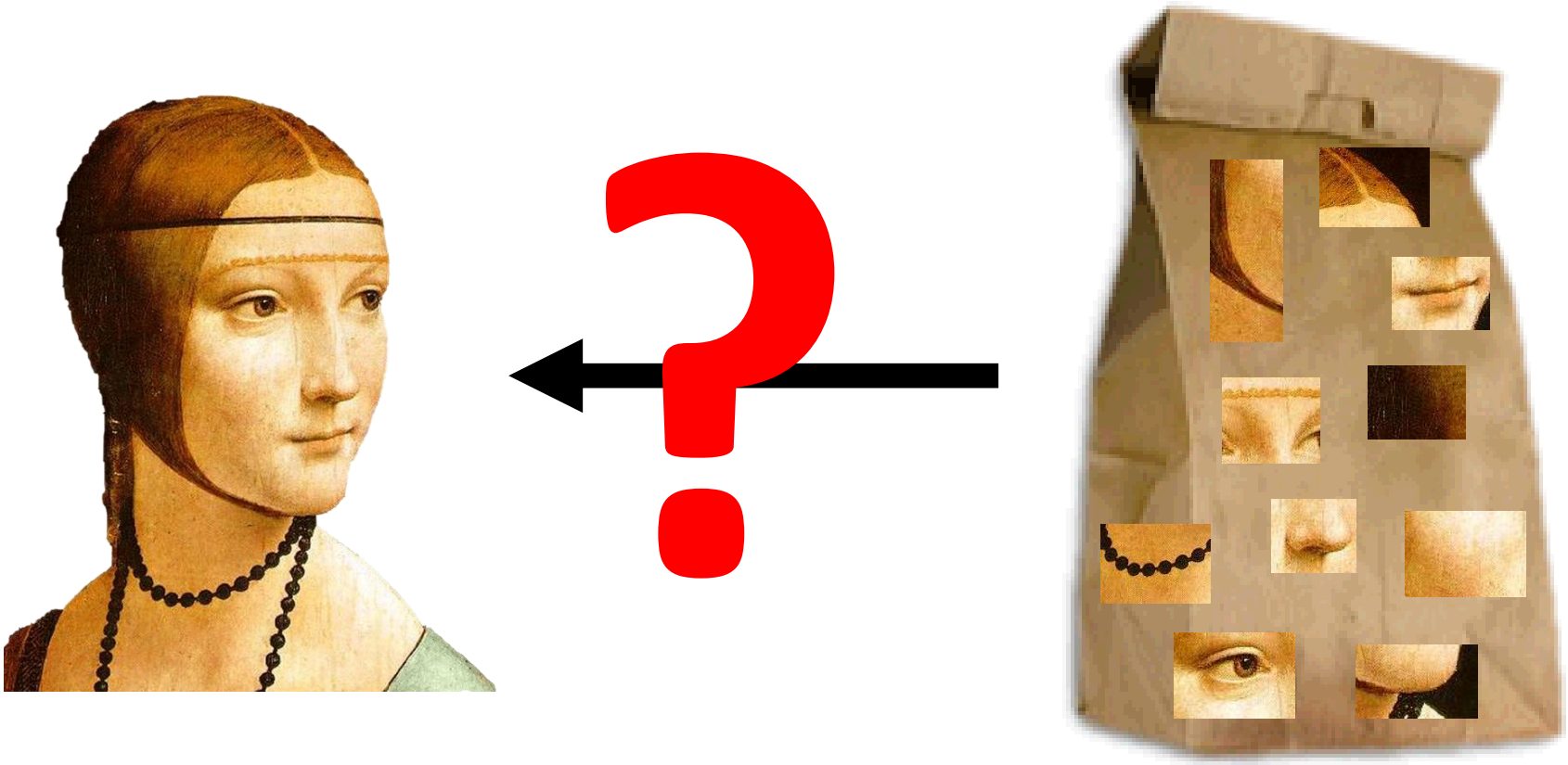
- Cons: performance degrades as the database grows

Example bag-of-words matches



Example bag-of-words matches

What about spatial info?

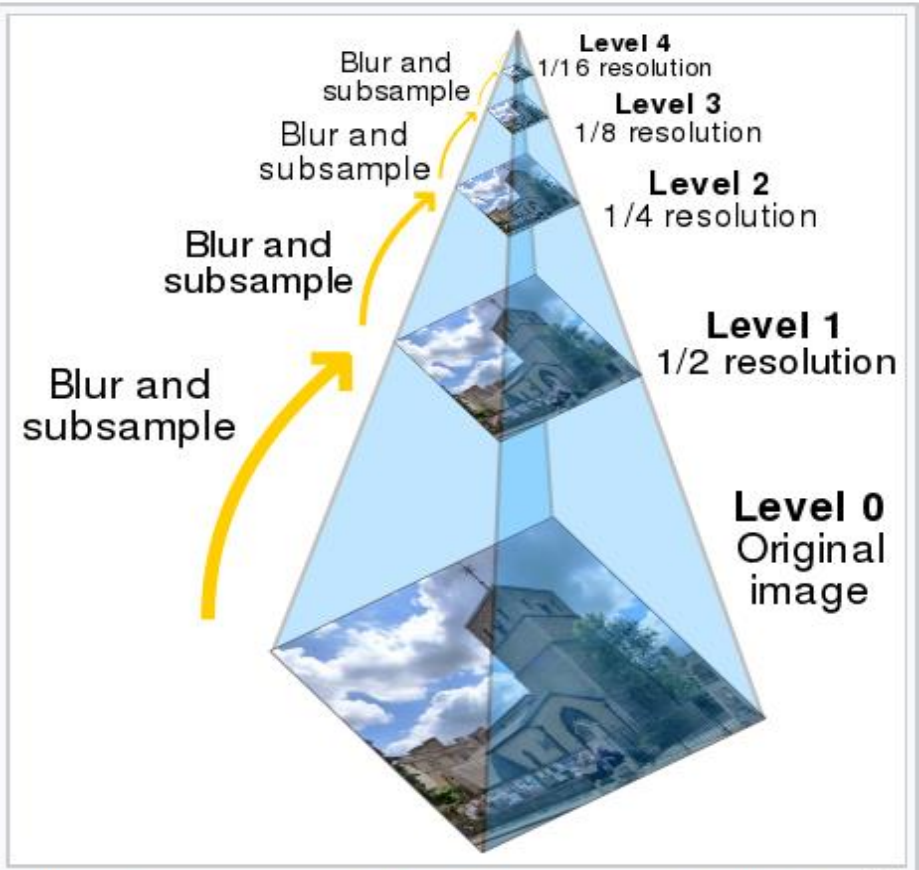


Bag of Words + Pyramids

- Bag of Words alone doesn't discriminate if a patch was obtained from the top, middle or bottom of the image because it doesn't save any spatial information.
- We need smart method to incorporate the spatial information in the BoW model
- **Spatial Pyramid Matching**
  - Very useful for representing images.
  - Spatial pyramid matching partitions the image into increasingly fine sub-regions and allows us to computes histograms (BoW) of local features inside each sub-region.
  - Strong features (ie.larger vocabulary size) is better than weaker features (ie. smaller vocabulary size).

Spatial Pyramids

- Very useful 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.

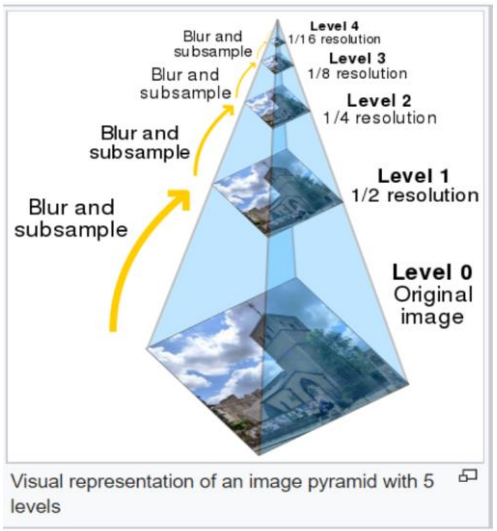


Visual representation of an image pyramid with 5 levels

Steps for Bag of Words with Spatial Pyramids:

1. **Image Division:**
   - Divide the image into a grid or a set of spatial bins. Commonly used divisions include 1x1 (no spatial pyramid), 2x2, 3x3, or more, forming different levels.
2. **Bag of Words for Each Spatial Bin:**
   - Apply the Bag of Words model independently to each spatial bin. This involves feature extraction, codebook generation, feature quantization, and histogram representation.
3. **Concatenation:**
   - Concatenate the histogram representations from all levels of the spatial pyramid. This creates a multi-level histogram representation that encodes both local and spatial information.
4. **Normalization (Optional):**
   - Optionally, normalize the concatenated histogram to ensure that the representation is invariant to changes in image scale or overall intensity.

Bag of words + pyramids



Visual representation of an image pyramid with 5 levels