AI 330: Machine Learning

Fall 2023

Dr. Wessam EL-Behaidy

Associate Professor, Computer Science Department,

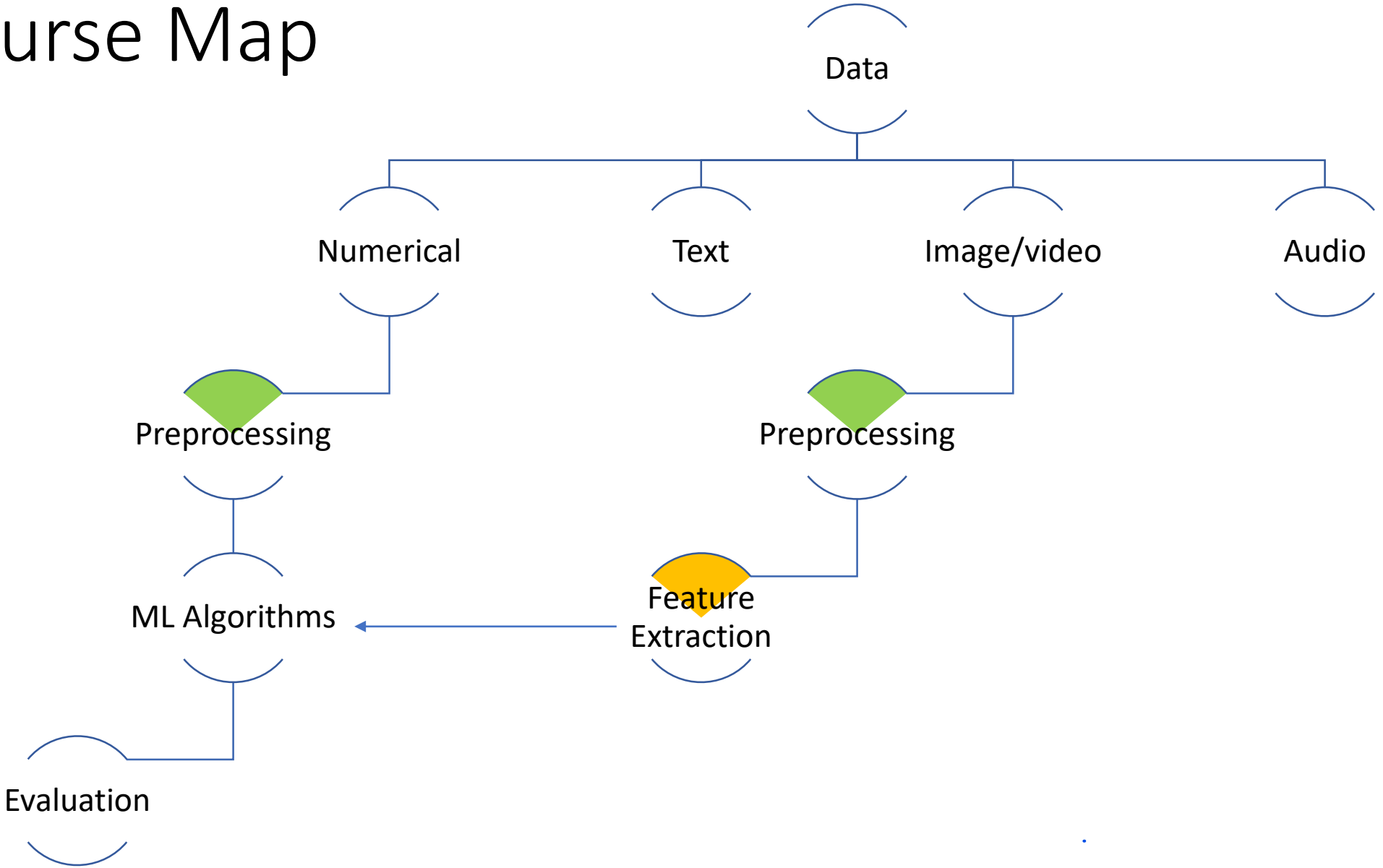
Faculty of Computers and Artificial Intelligence,

Helwan University.

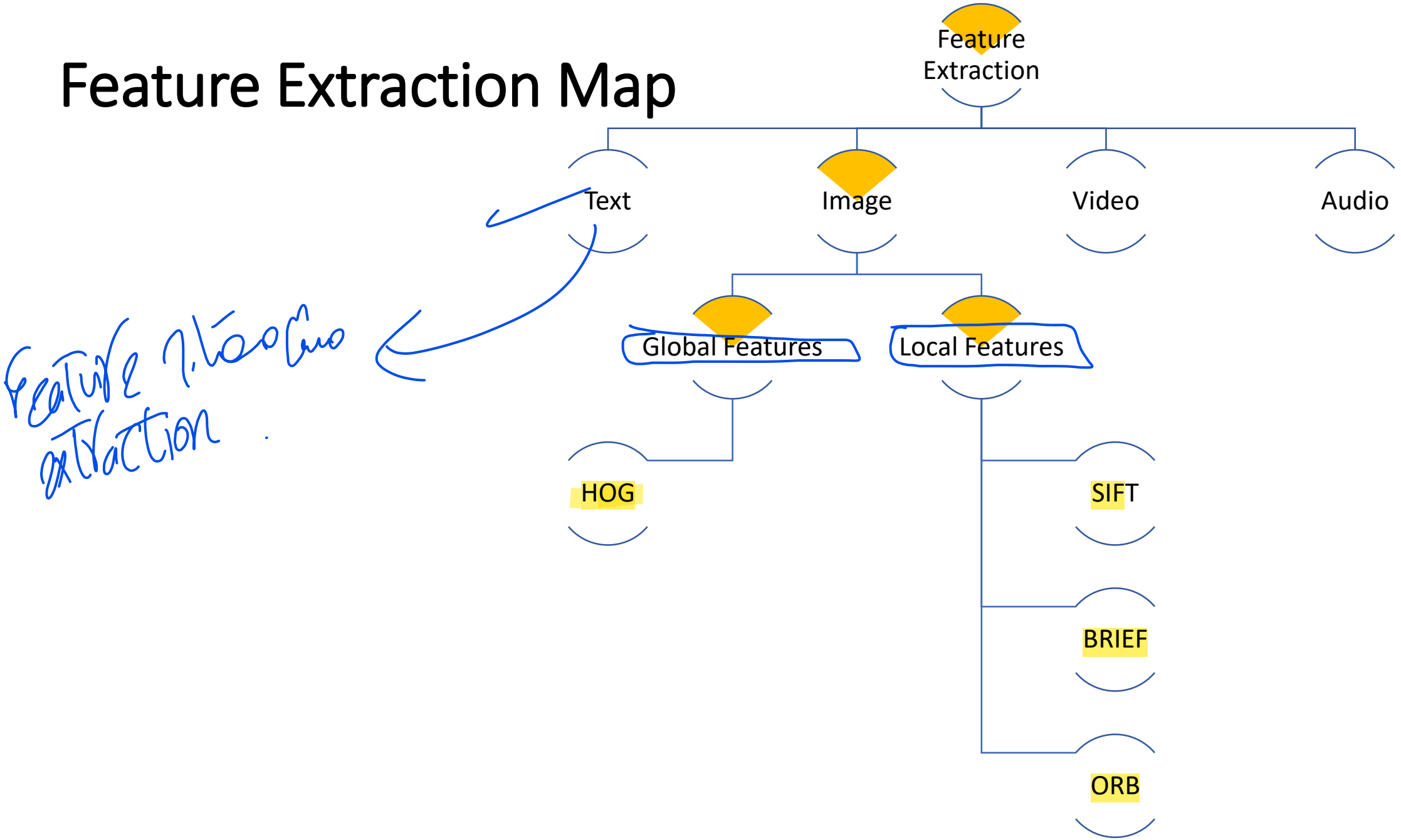
Dr. Ensaf Hussein

Associate Professor, Computer Science Department,
Faculty of Computers and Artificial Intelligence,
Helwan University.

Course Map


Data
Numerical
Text
Image/video
Audio
Preprocessing
Preprocessing
ML Algorithms
Feature Extraction
Evaluation

Feature Extraction Map



Lecture 3
Image Feature Extraction

Feature Extraction

- **Feature extraction** refers to the process of transforming raw data into numerical values (features).
- An image **feature descriptor** is a simplified representation of the image that contains only the most important information about the image, i.e., data unique to this specific object.
- Generally, features extracted from an image are of much lower dimension than the original image.
- A **good feature descriptor** is used to:
  - distinguish objects from one another. Ex: distinguish between cars and trucks.
  - match these features to a new image of the same object. Ex: recognize that this is the same building in two different images.

Feature Extraction

Feature extraction can be accomplished manually or automatically:

- **Manual feature extraction (Hand-crafted features)** requires identifying and describing the features that are **relevant for a given problem** and implementing a way to extract those features. Over decades of research, engineers and scientists have developed feature extraction methods for images, signals, and text.
- **Automated feature extraction** uses specialized algorithms or **deep networks** to extract features automatically from signals or images without the need for human intervention. This technique can be very useful when you want to move quickly from raw data to developing machine learning algorithms.

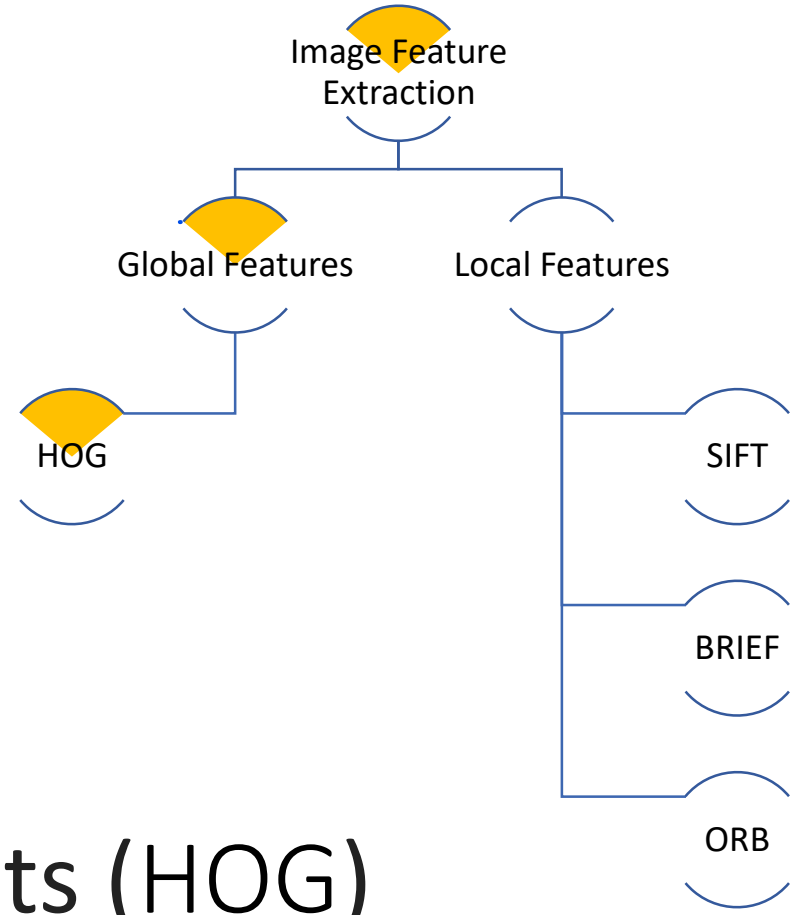
Feature extraction remains the first challenge that requires significant expertise before one can build effective predictive models.

Image Feature Descriptor

- **Global features** describe the image as a whole to generalize the entire object.
- Whereas the **local features** describe the image patches (key points in the image) of an object.
- Generally,
  - for **low-level applications** such as object detection, global features are used.
  - for **higher-level applications** such as object recognition, local features are used.

Global & local features

- **There are number of features for images, the most popular ones are:**
  - **Global features:**
    - **HOG** – Histogram of Oriented Gradients
  - **Local features:**
    - **SIFT** – Scale Invariant Feature Transform
    - **BRIEF** – Binary Robust Independent Elementary Features
    - **ORB** – Oriented FAST Rotated BRIEF



Global Feature:

Histogram of Oriented Gradients (HOG)

Slides References:

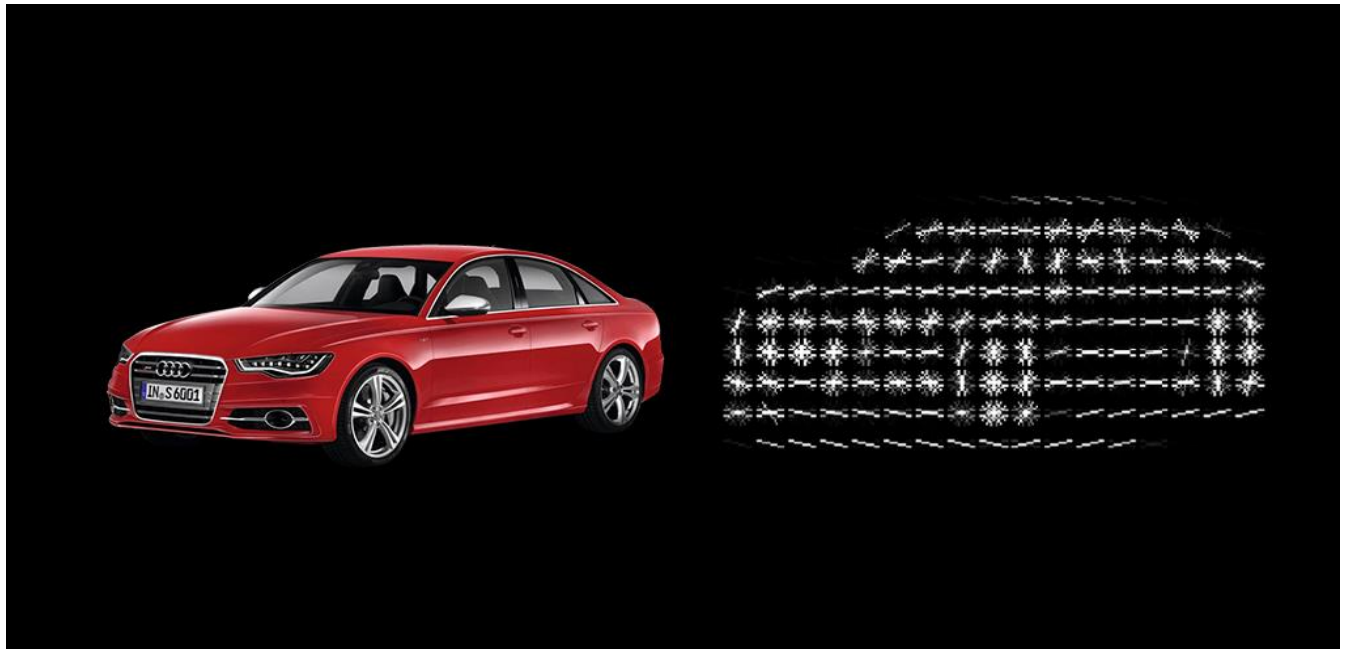
- https://www.analyticsvidhya.com/blog/2019/09/feature-engineering-images-introduction-hog-feature-descriptor/#:~:text=HOG%2C%20or%20Histogram%20of%20Oriented,vision%20tasks%20for%20object%20detection.
- https://learnopencv.com/histogram-of-oriented-gradients/

Histogram of Oriented Gradients

- HOG, or Histogram of Oriented Gradients, is a feature descriptor that is often used to extract **global features** from image data.
- It is widely used in computer vision tasks for object detection.

Histogram of Oriented Gradients

- The HOG descriptor:
  - focuses on the structure or the shape of an object. HOG is able to identify the edges and provide the edge direction as well. This is done by extracting the **gradient and orientation** (or you can say magnitude and direction) of the edges

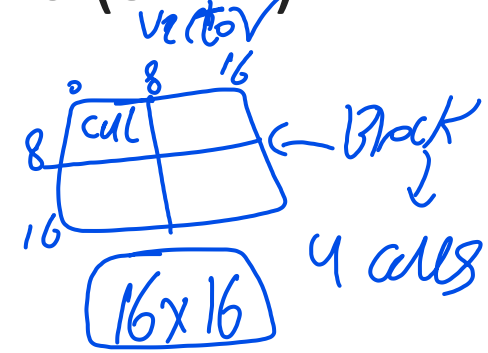


A formal definition:

The HOG feature descriptor counts the occurrences of gradient orientation in localized portions of an image.

HOG Steps

- **Step 1:** Preprocessing the image (Crop and resize).
- **Step 2:** Calculate the Gradients of pixel values.
- **Step 3:** Calculate the Magnitude and Orientation of pixel values.
- **Step 4:** Calculate Histogram of Gradients in 8×8 cells (9×1)
- **Step 5:** Normalize gradients in 16×16 cell (36×1)
- **Step 6:** Features for the complete image

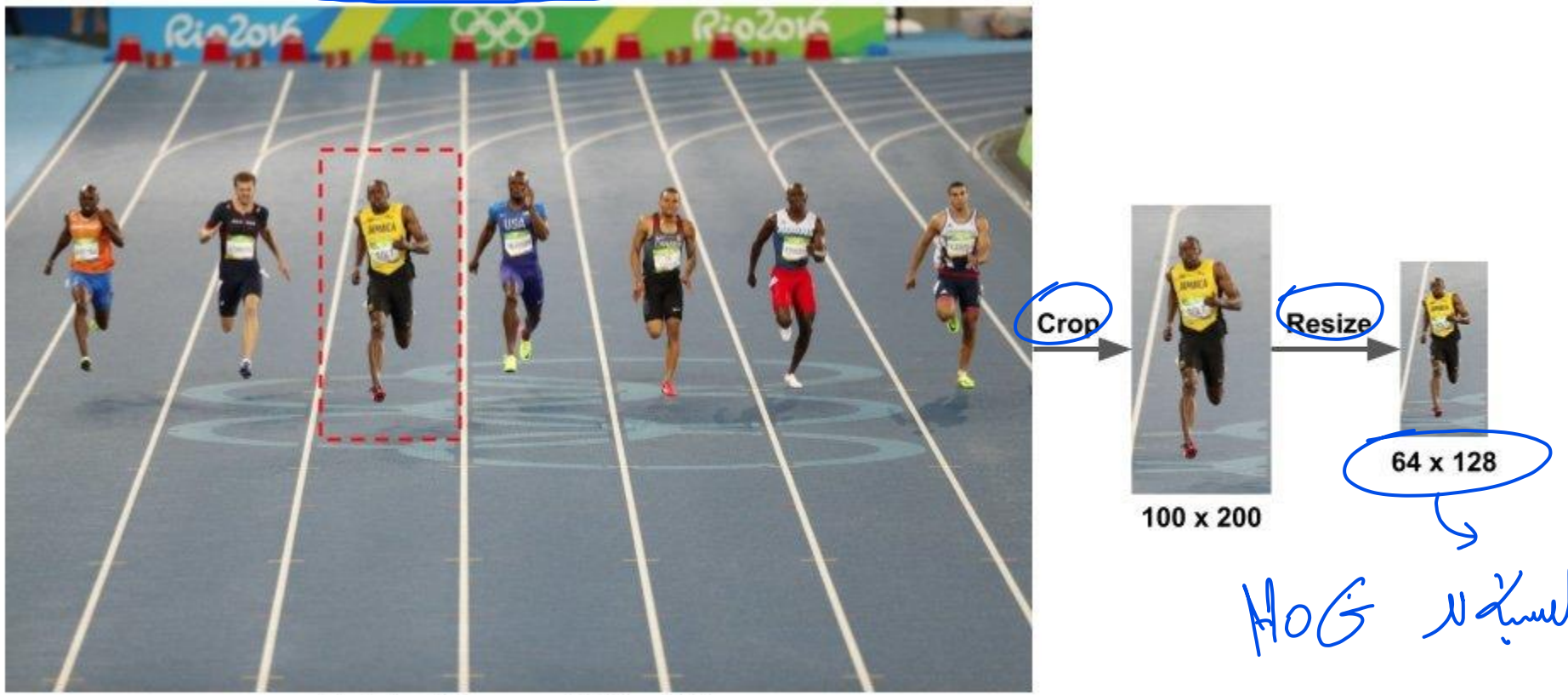


Step 1: Preprocessing the image

- HOG feature descriptor is calculated on a 64×128 patch of an image.
- Of course, an image may be of any size.
- Typically patches at multiple scales are analyzed at many image locations. The only constraint is that the patches need to have an aspect ratio of 1:2.
  - For example, they can be 100×200, 128×256, or 1000×2000 but not 101×205.

Step 1:

- Example: We used an image of size 720×475. We selected a patch of size 100×200 for calculating our HOG feature descriptor. This patch is cropped out of an image and resized to 64×128.



Step 2: Calculate the Gradients

- Gradients are the small change in the x and y directions.
- We calculate the gradients on each pixel in the image, by filtering it with a kernel, like Sobel with size 1.
- To determine the gradient (or change) in the x-direction

  Ex: Change in X direction(G_x) = | – 78 + 89 | = 11

- Similarly, to calculate the gradient in the y-direction

  Ex: Change in Y direction(G_y) = | – 76 + 68 | = 8

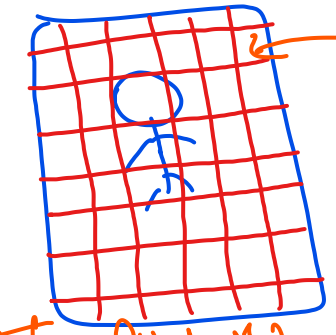
152	76	125
78	85	89
214	68	200

Sample of image pixel values

-1	0	1

-1
0
1

Sobel kernel of size 1



Step 2:

- We have calculated the gradients in both x and y directions separately. So, this process will give us **two new matrices** :
  - one storing gradients in the x-direction and
  - the other storing gradients in the y direction.

Step 3: Calculate the Magnitude and Orientation

- Using the gradients we calculated in the last step, we will now determine the **magnitude** and **direction (orientation)** for each pixel value using these formulas:

$$\text{Magnitude} = \sqrt{[(G_x)^2+(G_y)^2]}$$

$$\Phi = \arctan(Gy / Gx)$$

- So, for the previous example, we had Gx and Gy as 11 and 8.

$$\text{Magnitude} = \sqrt{[(11)^2+(8)^2]} = 13.6$$

$$\Phi = \arctan(8 / 11) = 36$$

For color images, the gradients of the three channels are evaluated. The **magnitude of gradient** at a pixel is the maximum of the magnitude of gradients of the three channels, and the **angle** is the angle corresponding to the maximum gradient

Step 3:

Absolute value of x-gradient

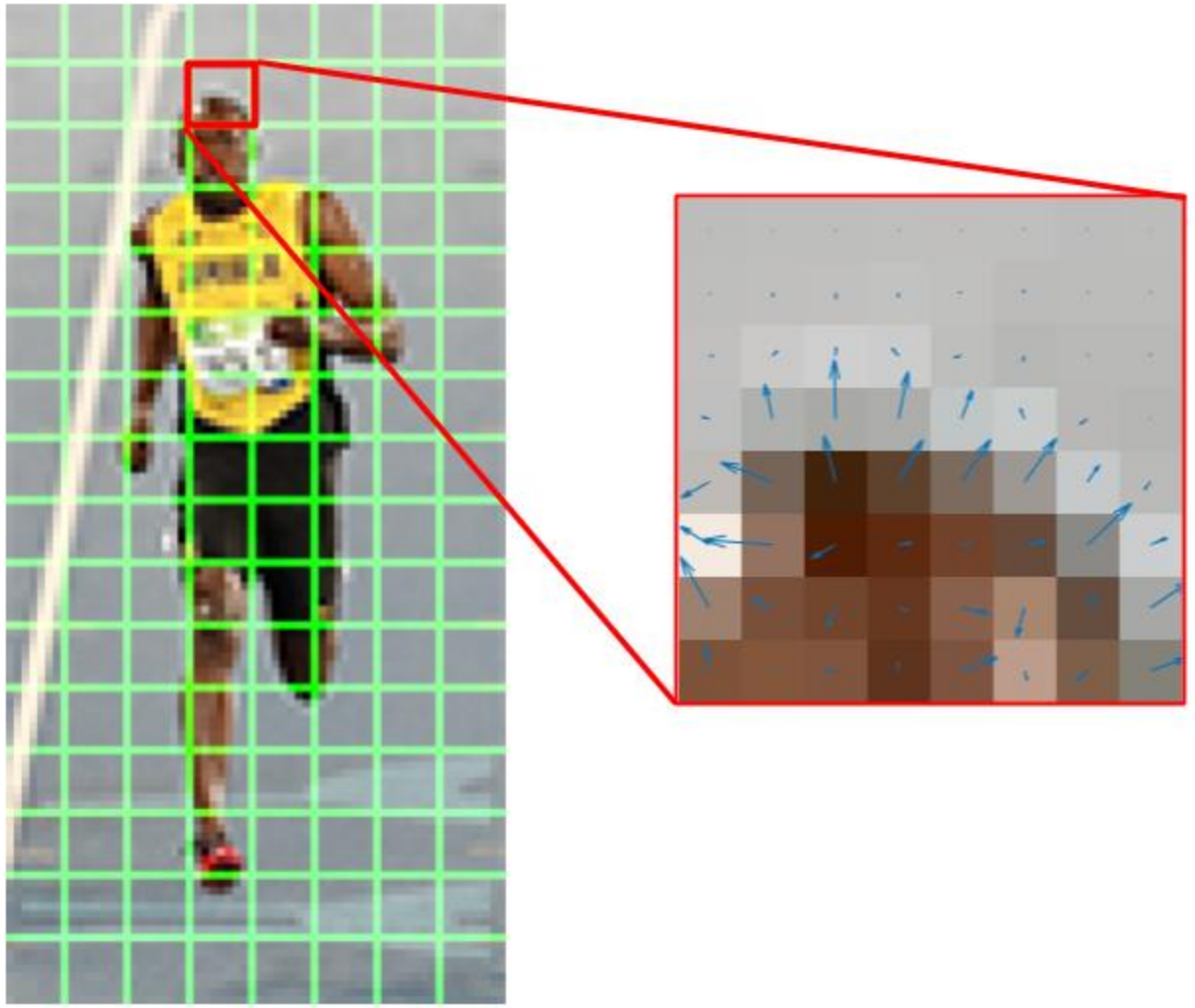
Absolute value of y-gradient

Magnitude of gradient

- Notice, that the x-gradient fires on vertical lines and the y-gradient fires on horizontal lines. The magnitude of gradient fires where ever there is a sharp change in intensity (as edges). None of them fire when the region is smooth.
- The gradient image removed a lot of non-essential information (e.g. constant colored background) but highlighted outlines

Step 4: Calculate Histogram of Gradients in 8×8 cells (9×1)

- In this step, the image is divided into 8×8 cells and a histogram of gradients is calculated for each 8×8 cell.
- We can certainly change this value here from 8 x 8 to 16 x 16 or 32 x 32.
- In the figure, the **direction of arrows** points to the direction of change in intensity, and the **magnitude** shows how big the difference is.



Step 4:

- We have 9 bins in the histogram.
- We select:
  - **bin**: based on the direction
  - **vote** (the value that goes into the bin): based on the magnitude.
- **Ex1:** the pixel is encircled in blue.

  angle (direction) = 80°, magnitude =2.

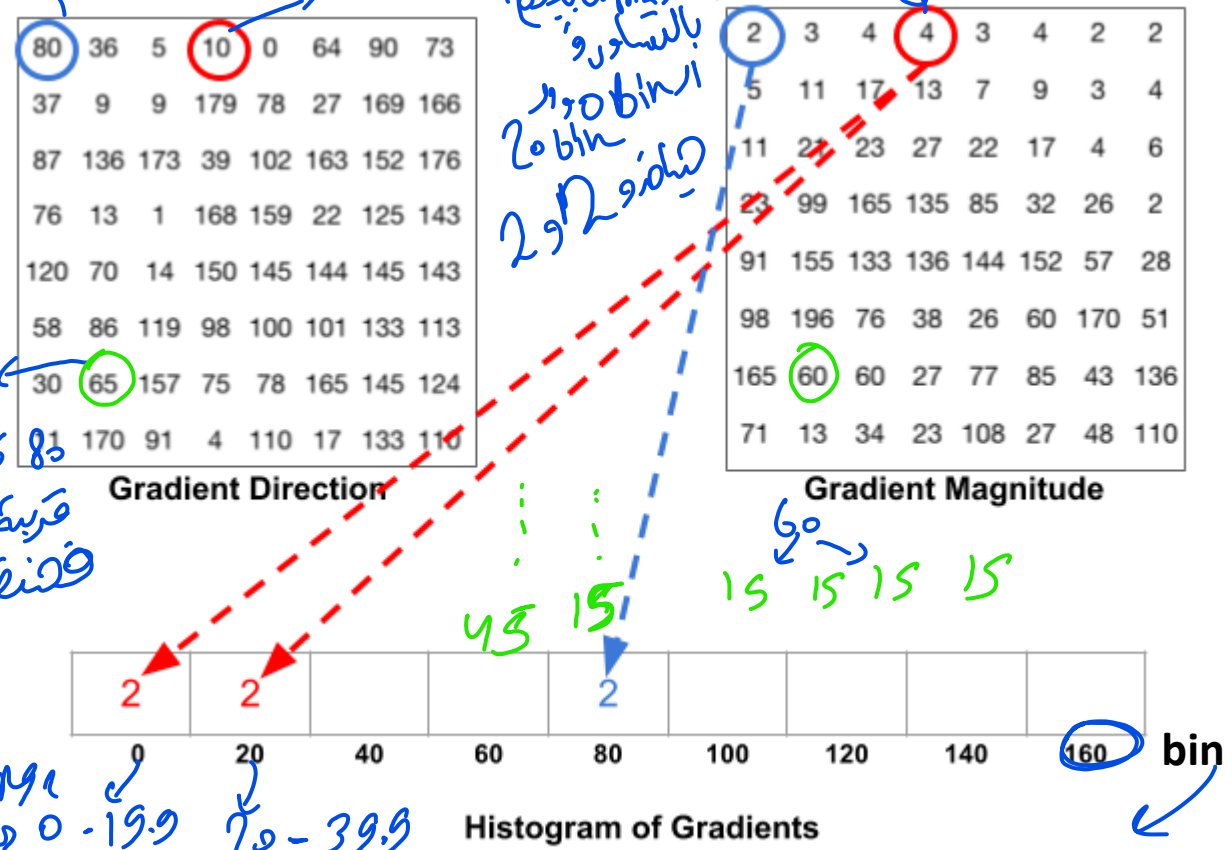
➔So it adds 2 to the 5th bin.

- **Ex2:** The pixel is encircled in red.

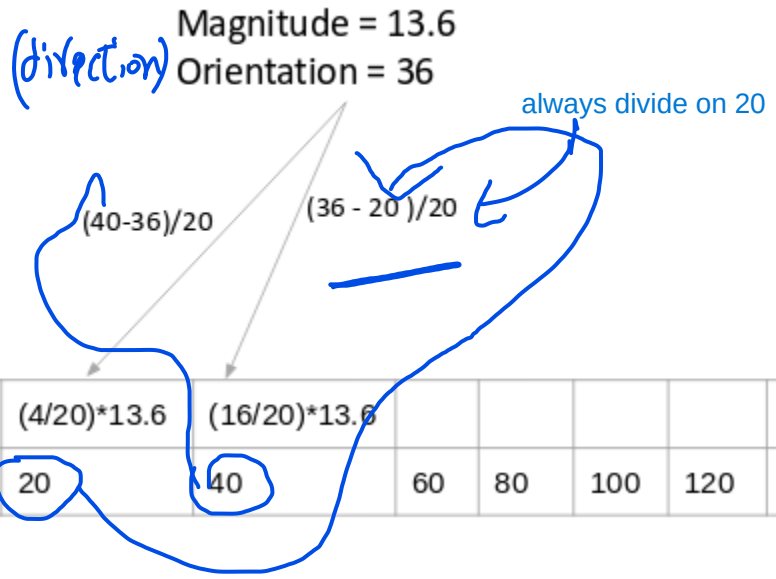
  angle = 10° , magnitude = 4.

Since 10° is halfway between 0 and 20,

➔ the vote by the pixel splits evenly into the two bins

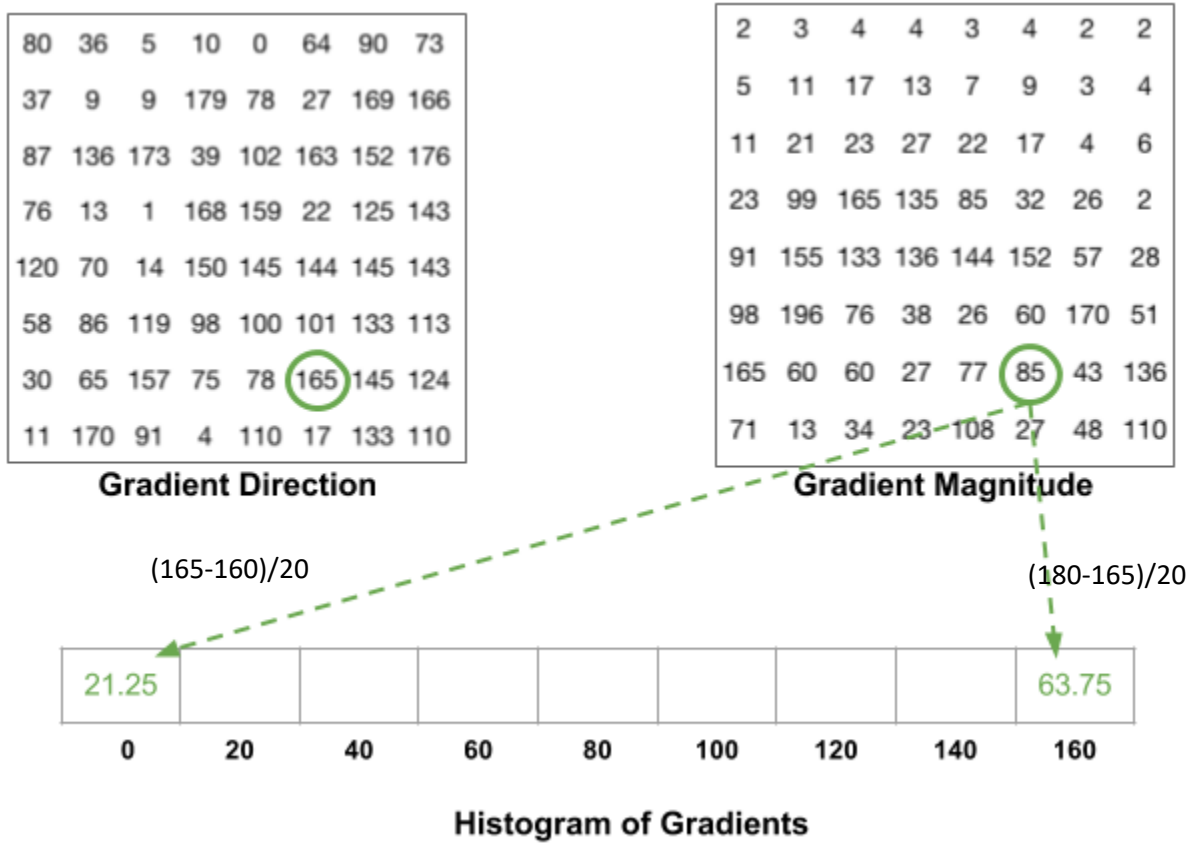


Step 4:



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

Ex3: Uneven split in direction (orientation)

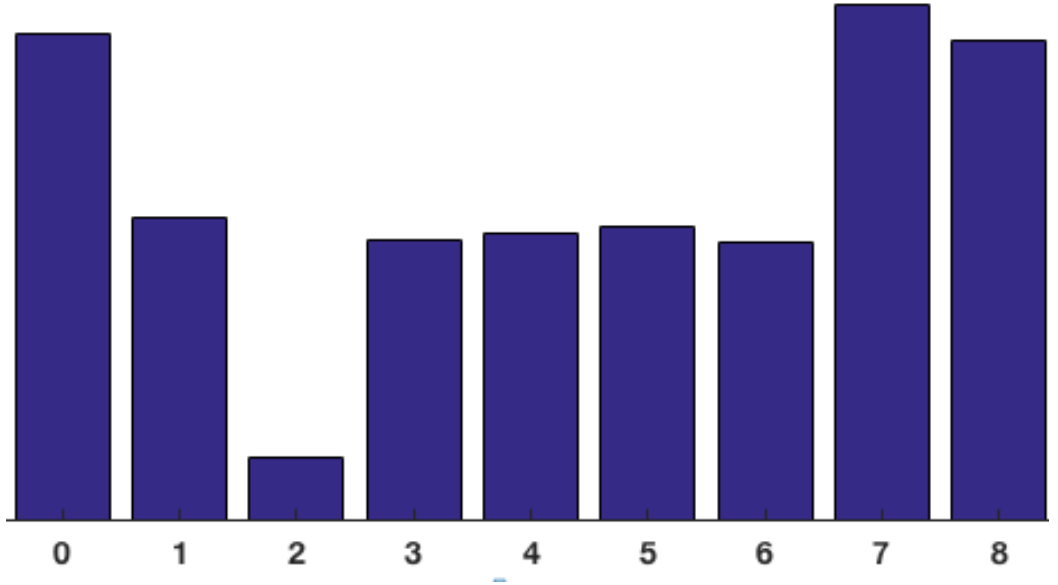


Ex4: **If the angle is greater than 160 degrees**

Remember, the higher contribution should be to the bin value which is closer to the orientation

Step 4:

- The contributions of all the pixels in the 8×8 cells are added up to create the 9-bin histogram.
- we will get a 9 x 1 matrix for each cell, and its 9-bin histogram looks like this:



Step 5: Normalize gradients in 16×16 cell (36×1)

- In the previous step, we created a histogram based on the gradient of the image.
  - Gradients of an image are sensitive to overall lighting.
- Ideally, we want our descriptor to be independent of lighting variations.
  - We can reduce this lighting variation by "normalizing" the gradients over a bigger sized block of 16×16.

Step 5:

- **To perform a 16 x 16 block,** we will be combining four 8×8 cells to create a 16×16 block.
  - ➔We have four 9×1 matrices (9x1 matrix from 8 x 8 cell)
  - ➔A 16×16 block can be concatenated to form a 36 x 1 matrix
- **To normalize this matrix**, we will divide each of these values by the square root of the sum of squares of the values.

Example,

for a given vector V: V = [a1, a2, a3, ….a36]

We calculate the root of the sum of squares:

$$k = \sqrt{(a1)^2 + (a2)^2 + (a3)^2 + \ldots (a36)^2}$$

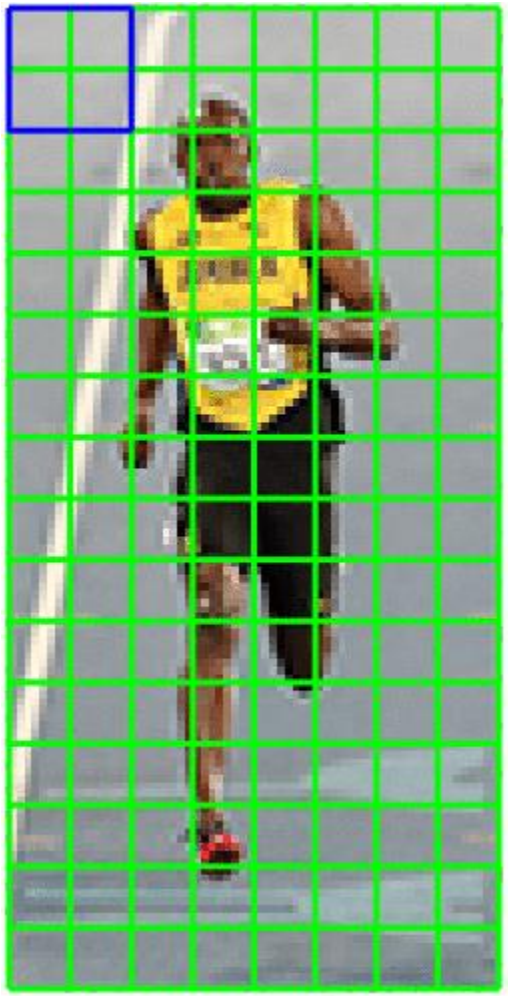
And divide all the values in the vector V with this value k:

$$\text{Normalised Vector} = \left(\frac{a_1}{k}, \frac{a_2}{k}, \frac{a_3}{k}, \ldots \frac{a_{36}}{k} \right)$$

The resultant would be a normalized vector of size 36×1

Step 5:

- The window is then moved by 8 pixels (see animation) and a normalized 36×1 vector is calculated over this window and the process is repeated.



Step 6: Features for the complete image

- To calculate the final feature vector for the entire image patch, the 36×1 vectors are concatenated into one giant vector.
- **Let us calculate**

1. How many positions of the 16×16 blocks do we get for a single 64×128 image?

There are 7 horizontal and 15 vertical positions ➔ 7 x 15 = 105 positions.

2. Each 16×16 block is represented by a 36×1 vector.

➔The feature vector = 36×105 = **3780** dimensional-vector.

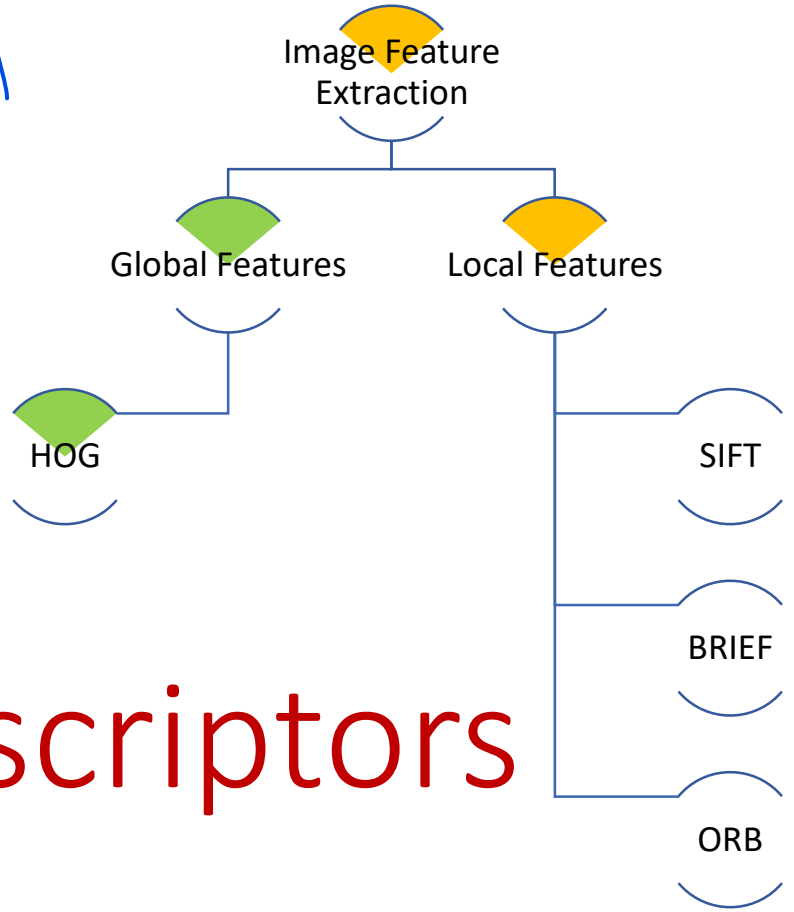
- **Hence, the total features for the image would be 3780 features.**

Visualizing Histogram of Oriented Gradients

- The HOG descriptor of an image patch is usually visualized by plotting the 9×1 normalized histograms in the 8×8 cells.
- See the image on the side. You will notice that the dominant direction of the histogram captures the shape of the person, especially around the torso and legs.

Local features:

Keypoints & Descriptors



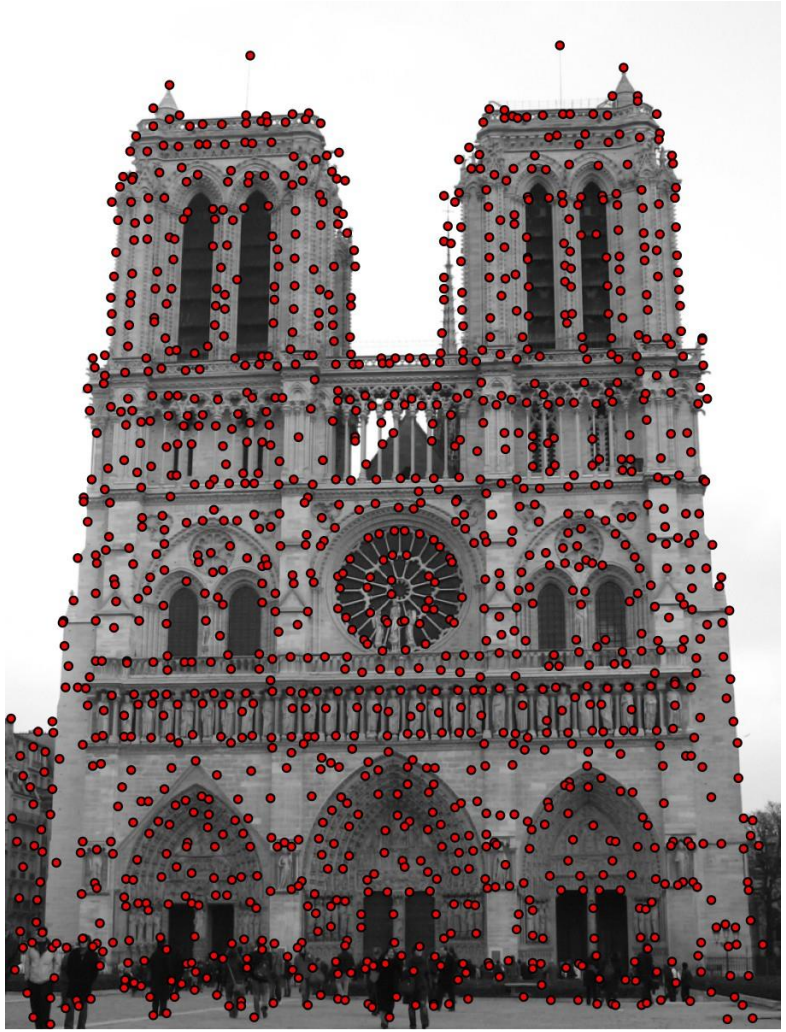
Slides References:
https://www.analyticsvidhya.com/blog/2019/10/detailed-guide-powerful-sift-technique-image-matching-python/

University of Bonn , Prof. Cyrill Stachniss:
https://www.youtube.com/watch?v=nGya59Je4Bs&t=607s
https://www.youtube.com/watch?v=CMolhcwtGAU

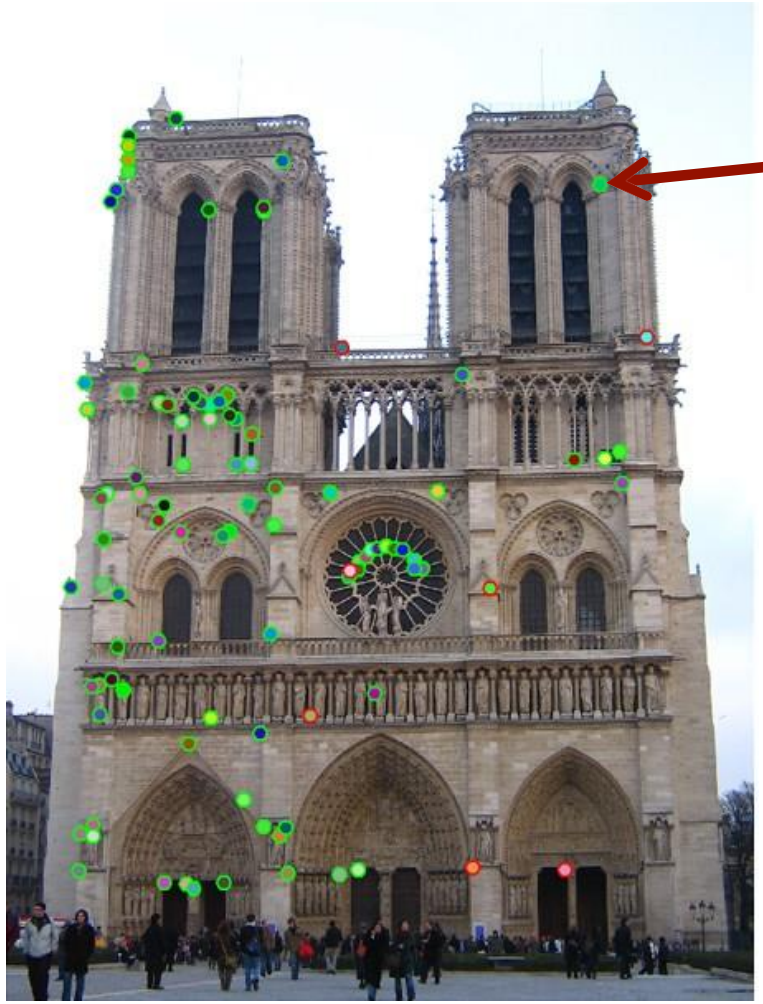
Motivation

- Finding locally distinct points

  ➔ keypoints



Keypoints and Descriptors



keypoint

descriptor at the keypoint

$$f = \begin{bmatrix} 0.02 \\ 0.04 \\ 0.1 \\ 0.03 \\ 0 \\ \dots \end{bmatrix}$$

Keypoints and Descriptors

§ **Keypoint** is a (locally) distinct location in an image

§ The feature **descriptor** summarizes the local structure around the keypoint

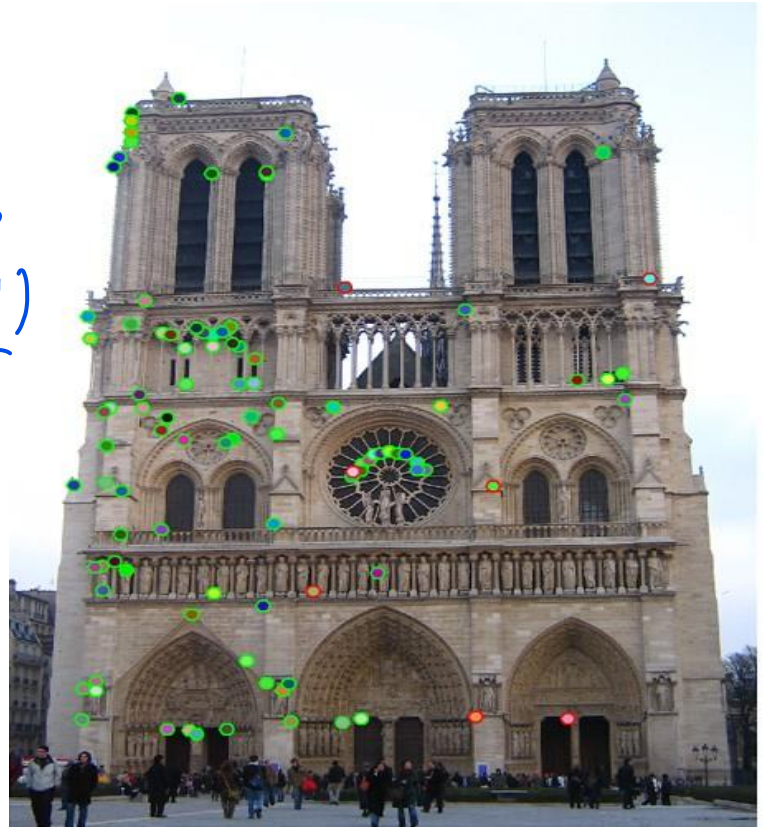
Keypoint

+

Descriptor

=

visual features

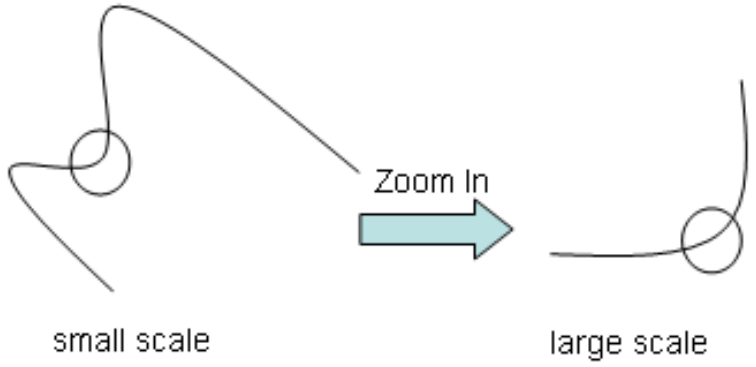


Keypoints:

- § Corners are often highly distinct points
- § Corners are invariant to translation, rotation, and illumination.
- § **Corner** = **two edges** in roughly orthogonal directions
- § **Edge** = a sudden **brightness change**

Keypoints:

- § Two groups of approaches for finding locally distinct points:
  1. **Corners via structure matrix**
     - § Harris, Shi-Tomasi, Förstner
  2. **Difference of Gaussians (DoG)**
     - § Finds corners and blobs over scales



Scaling Problem in Harris

- § These approaches are key ingredients of most hand-designed features

Local features:
Keypoints & Descriptors

Difference of Gaussians (DoG) Keypoint Detection

Difference of Gaussians Keypoints

- A variant of corner detection
- Provides responses at corners, edges, and blobs
- **Blob** = mainly constant region but different to its surroundings

Keypoints: DoG Over Scale-Space Pyramid

Procedure Over different image pyramid levels

Step 1: Gaussian smoothing (blurring)

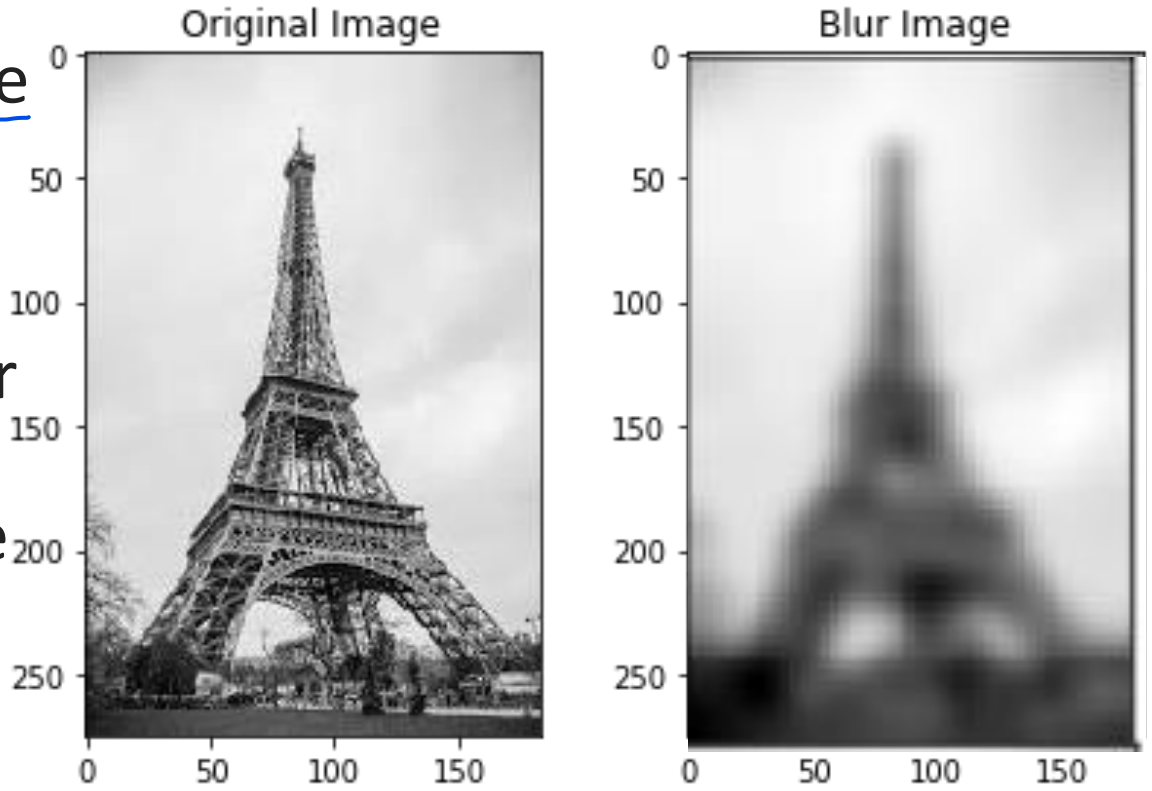
Step 2: Difference-of-Gaussians: find extrema (over smoothing scales)

Step 3: maxima suppression at edges (i.e., Keypoint Localization)

Step 1: Gaussian smoothing (Blurring)

- For every pixel in an image, the Gaussian smoothing calculates a value based on its neighboring pixels.

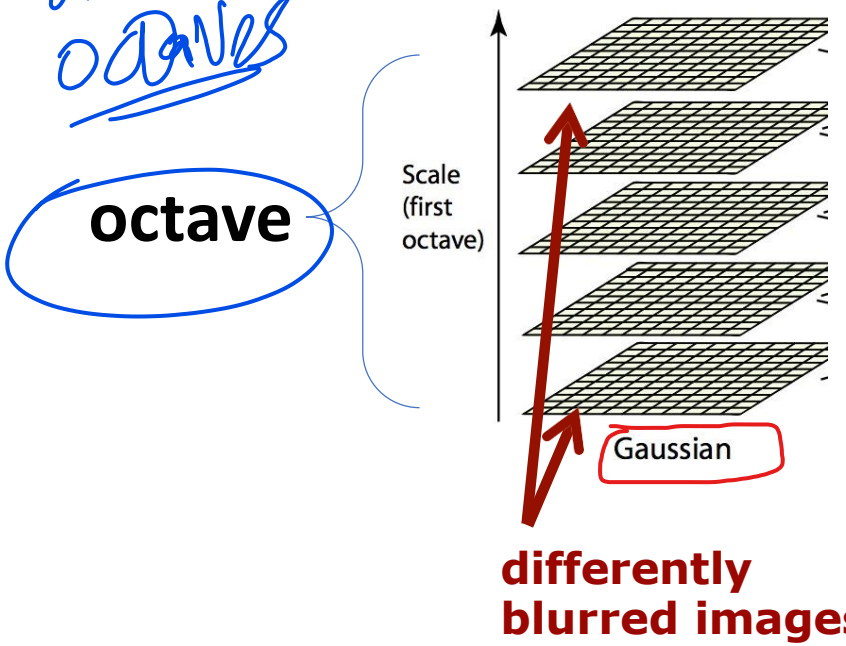
- As you can see, the texture and minor details are removed from the image, and only the relevant information like the shape and edges remain.



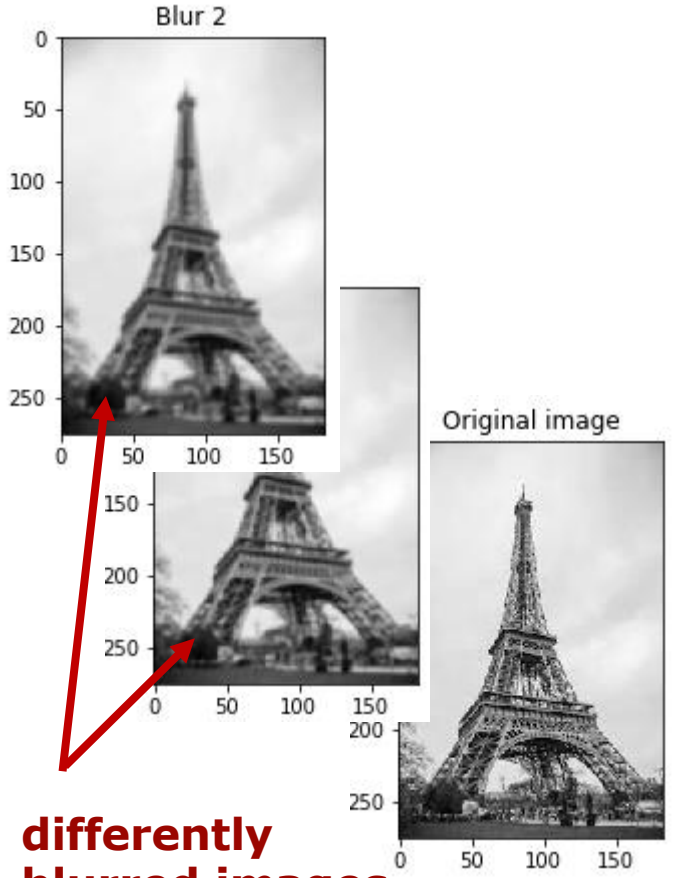
https://www.analyticsvidhya.com/blog/2019/10/detailed-guide-powerful-sift-technique-image-matching-python/

Illustration

- Gaussian smoothing reduces noise and highlights the important features of the image.
- Blur images with different σ



differently blurred images



differently blurred images

Image courtesy: Lowe38

Scale Space

Now, *we need to ensure that these features* ***must not be scale-dependent***

➔ searching for these features on multiple scales.

"**Scale space**" is a collection of images having different scales, generated from a single image.

Illustration

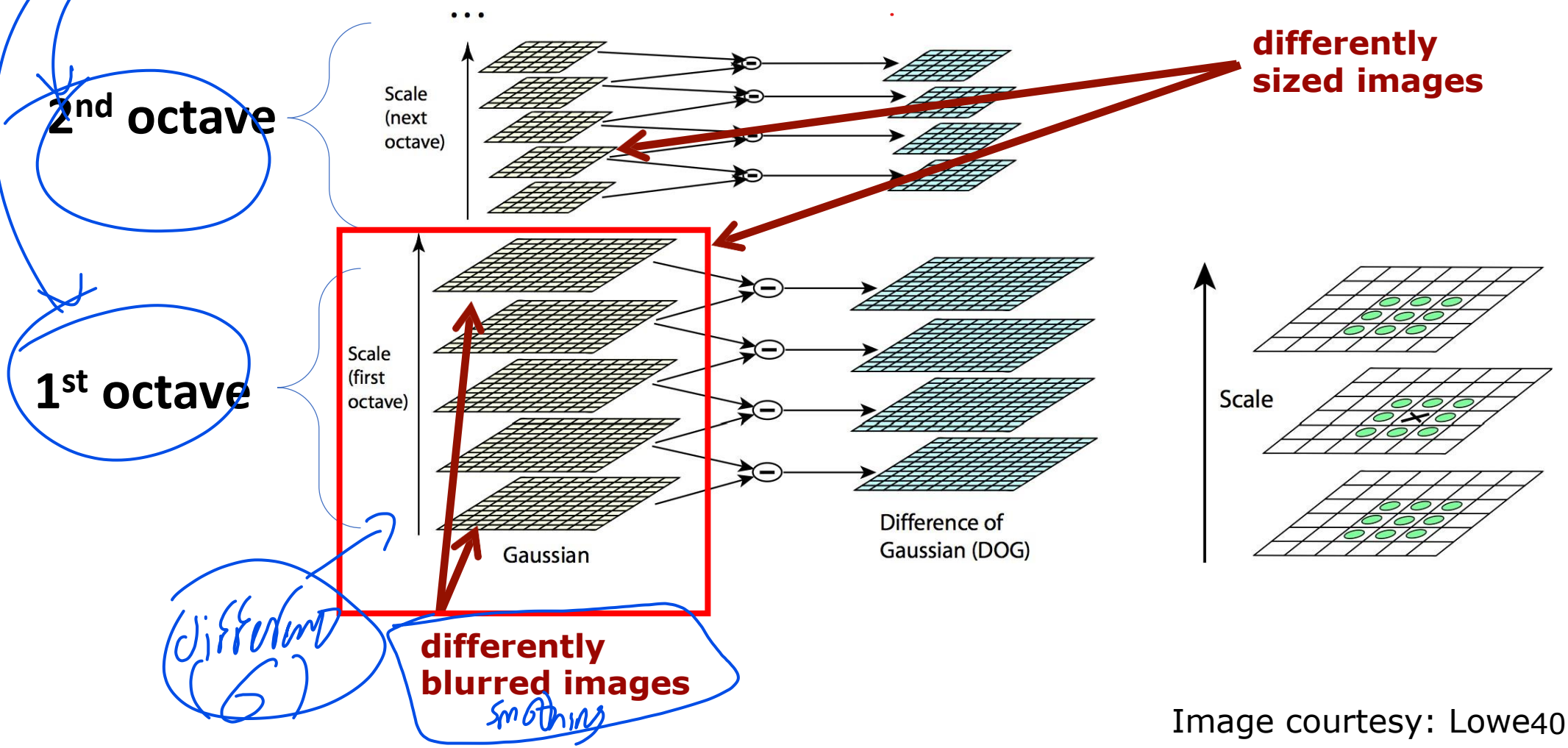
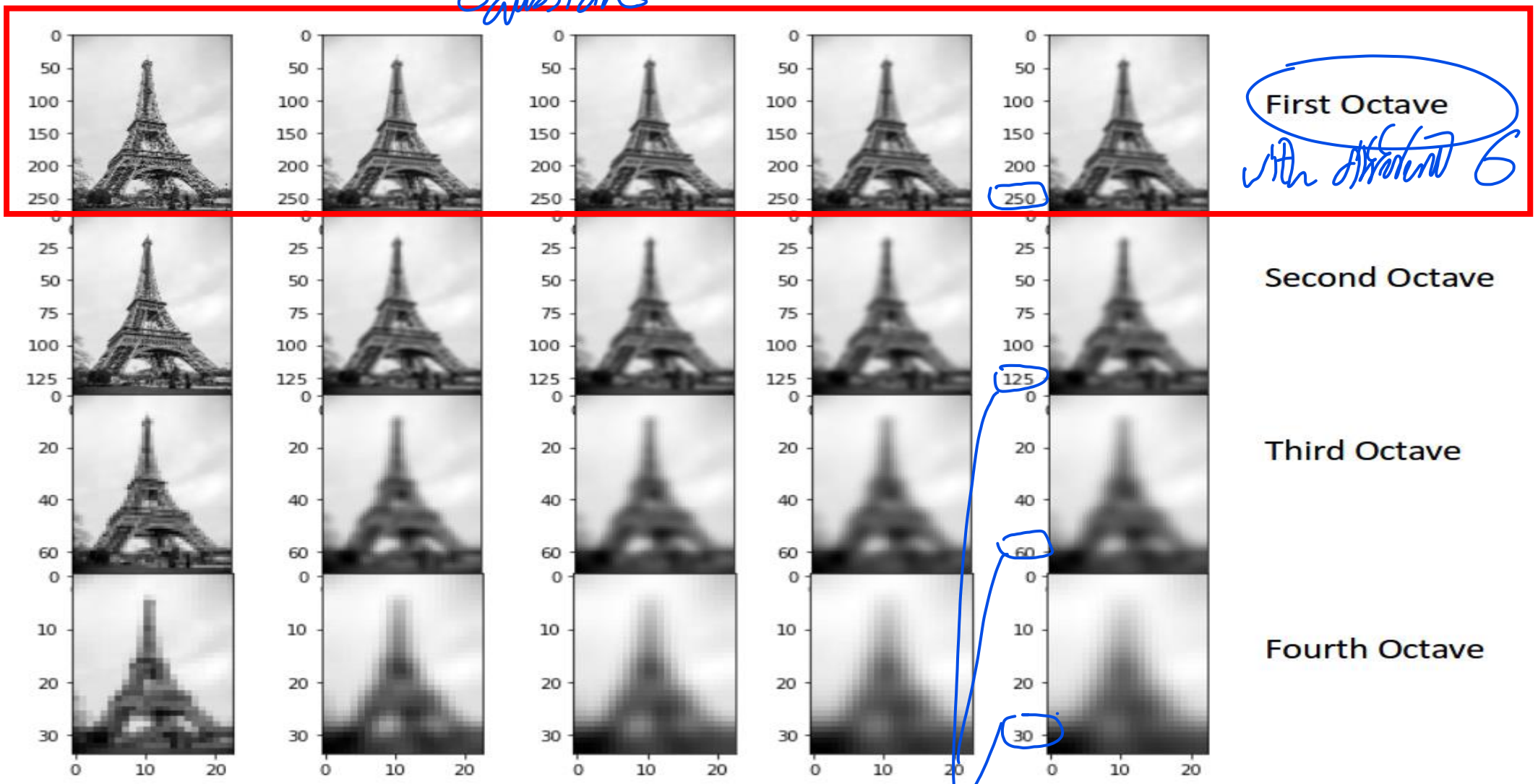


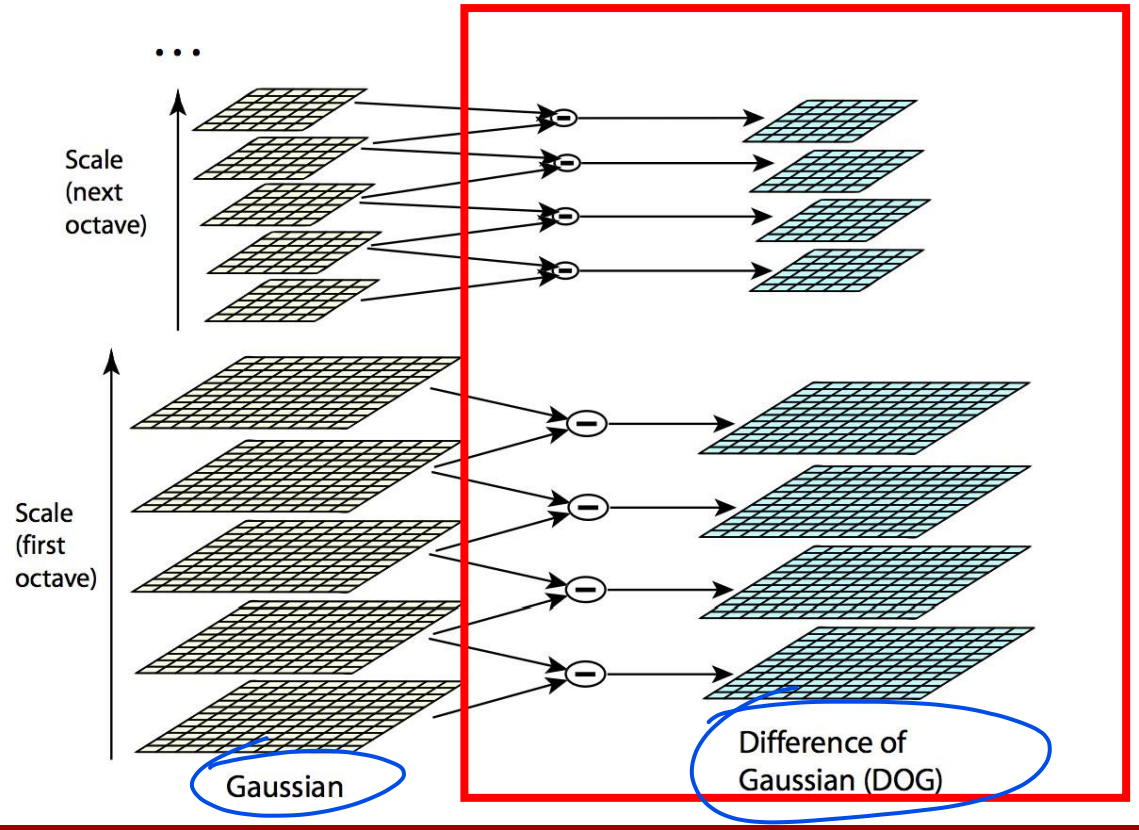
Image courtesy: Lowe40

The ideal number of octaves should be 4, and for each octave, the number of blur images should be 5.



Step 2: Difference of Gaussians

- DoG is a feature enhancement algorithm that creates another set of images, for each octave, by subtracting every image from the previous image on the same scale.



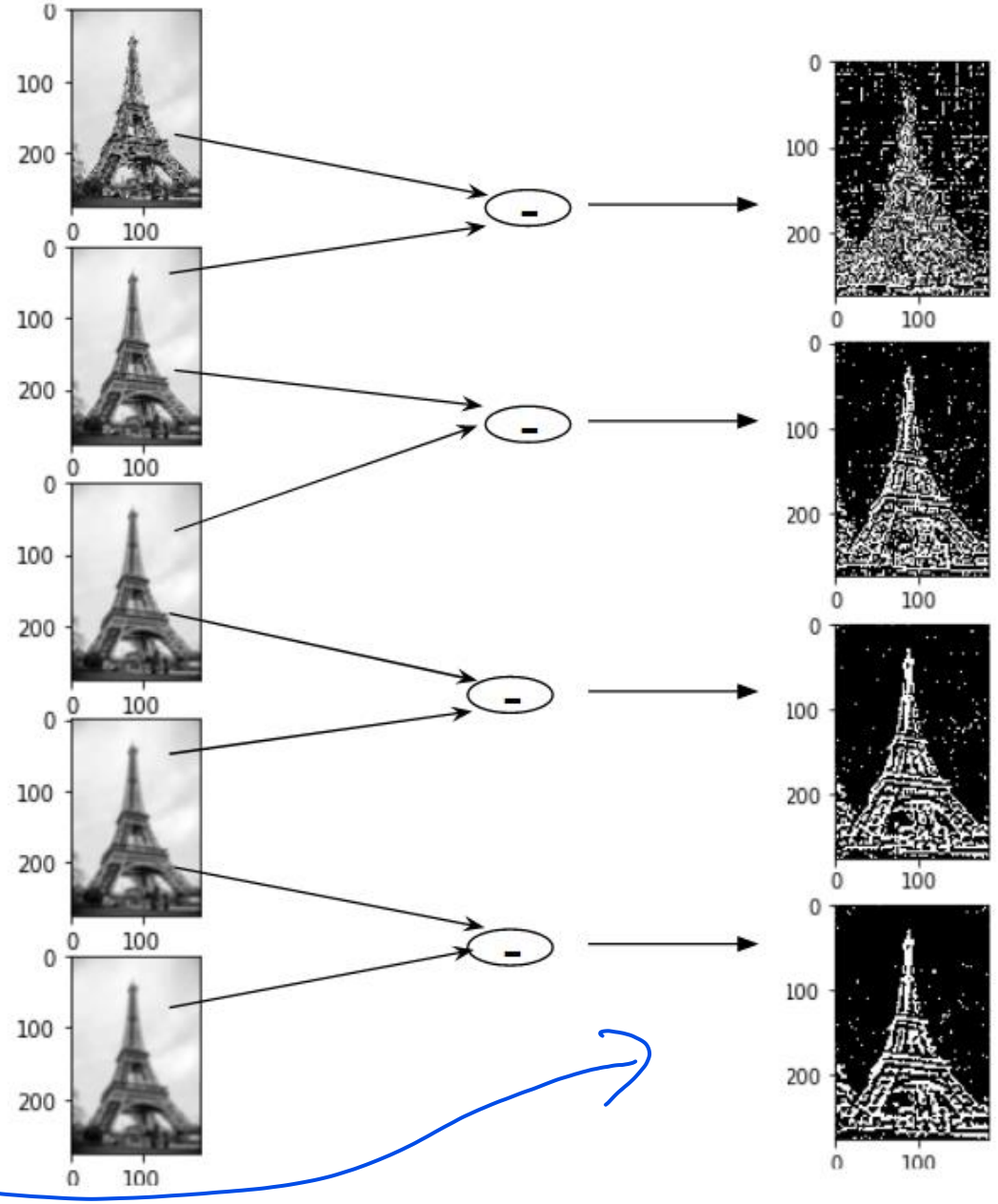
Step 2:

- § Blurring filters out high-frequencies (noise)
- § Subtracting differently blurred images from each other only keeps the frequencies that lie between the blur level of both images
- § DoG acts as a **band-pass** filter

Step 2: Illustration

- On the left, we have 5 images, all from the first octave (thus having the same scale). Each subsequent image is created by applying the Gaussian blur over the previous image.
- On the right, we have four images generated by subtracting the consecutive Gaussians.

➔ **Increases visibility of corners, edges, and other detail present in the image**



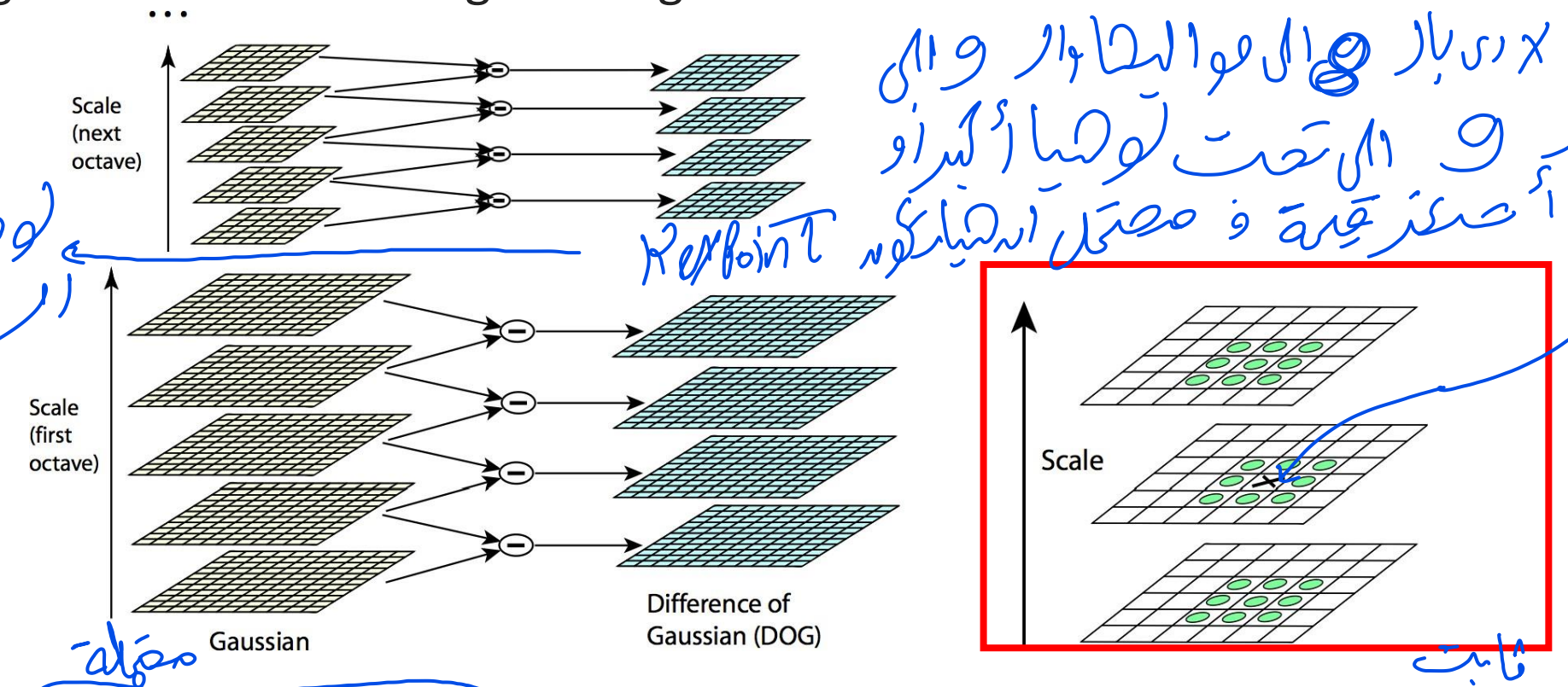
Step 2: Find Extrema

- This step is to find the important keypoints from the image that can be used for feature matching.
- **The idea is to find extrema, i.e., the local maxima and minima for the images.**
- To locate the local maxima and minima (extrema):
  - we go through every pixel in the image and compare it with its neighboring pixels.
  - Neighboring not only includes the surrounding pixels of that image (in which the pixel lies), but also the nine pixels for the previous and next image in the octave.

This means that every pixel value is **compared with 26 other pixel** values to find whether it is the local maxima/minima.

Step 2: Find Extrema

- For example, in the below diagram, we have three images from the first octave. The pixel marked ***x*** is compared with the neighboring pixels (in green) and is selected as a keypoint if it is the highest or lowest among the neighbors.



- We now have **potential keypoints** that represent the images and are **scale-invariant**.

Step 3: Extrema suppression (Keypoint Localization)

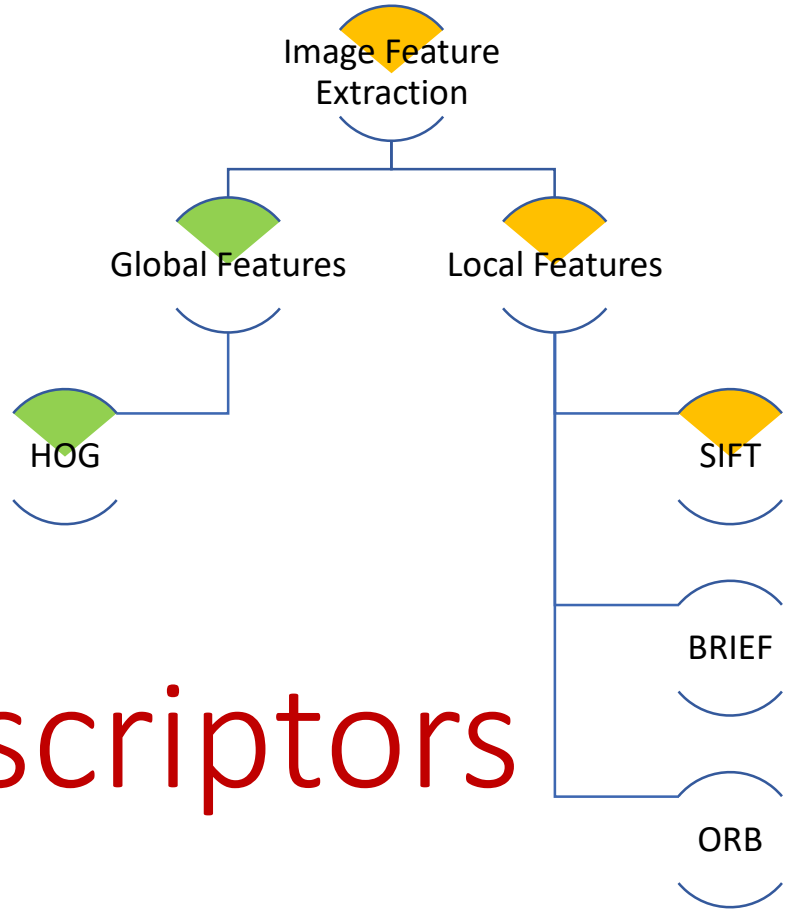
- § The DoG finds blob-like and corner-like image structures but also leads to strong responses along edges
- **§ Corners and blobs are good keypoints, but edges are bad for matching.**

➔ Hence, we will eliminate the keypoints that may not be robust to noise such as **low contrast, and edges.**

Step 3:

- **To deal with the low contrast keypoints,** a second-order Taylor expansion is computed for each keypoint. If the resulting value is less than 0.03 (in magnitude), we reject the keypoint.
- **To deal with edge keypoints**, a second-order Hessian matrix is used to identify such keypoints.

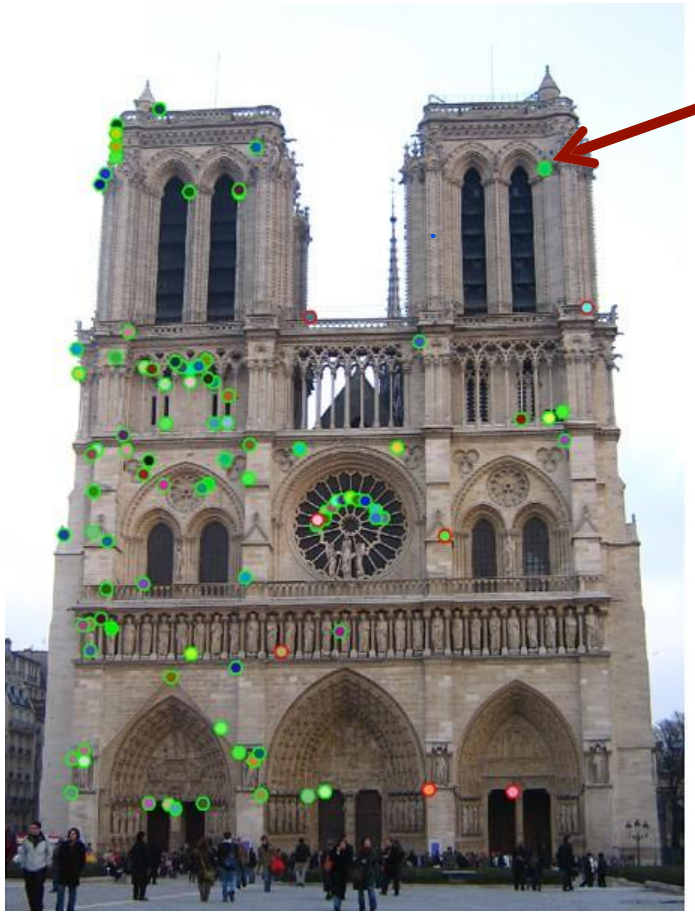
➔ we have accurate keypoints to represent the image features with **scale-invariant.**



Local features: Keypoints & Descriptors

SIFT Descriptor

Keypoint Done.
What about a Descriptor?



keypoint

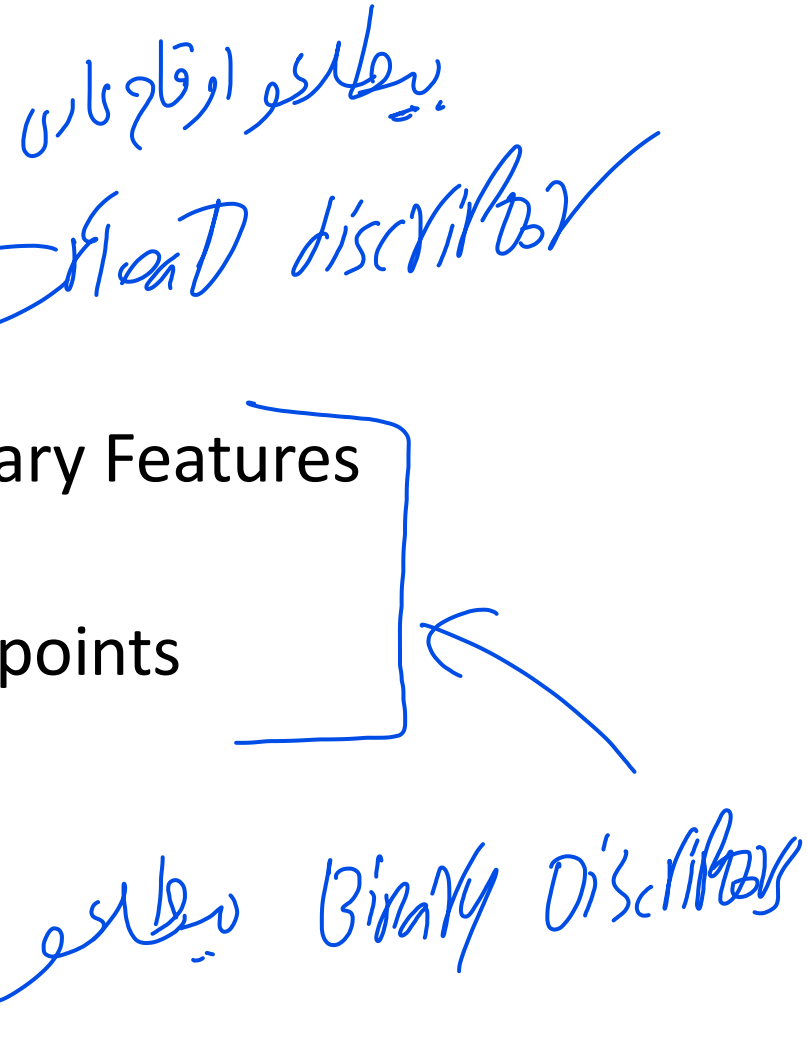
descriptor at the keypoint

$$f = \begin{bmatrix} 0.02 \\ 0.04 \\ 0.1 \\ 0.03 \\ 0 \\ \dots \end{bmatrix}$$

Local Descriptors

§ Features: Describing a keypoint

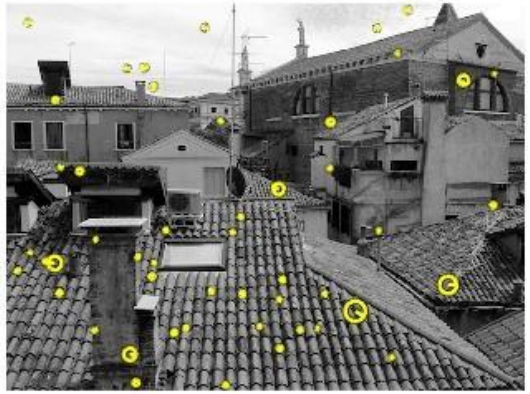
- § **SIFT** – Scale Invariant Feature Transform
- § **SURF** – Speeded-Up Robust Features
- § **BRIEF** – Binary Robust Independent Elementary Features
- § **ORB** – Oriented FAST Rotated BRIEF
- § **BRISK** – Binary Robust Invariant Scalable Keypoints
- § **FREAK** – Fast REtinA Keypoint



51

SIFT Considers the Distribution of Gradients Around Keypoints

input



key points

descriptor (regions)

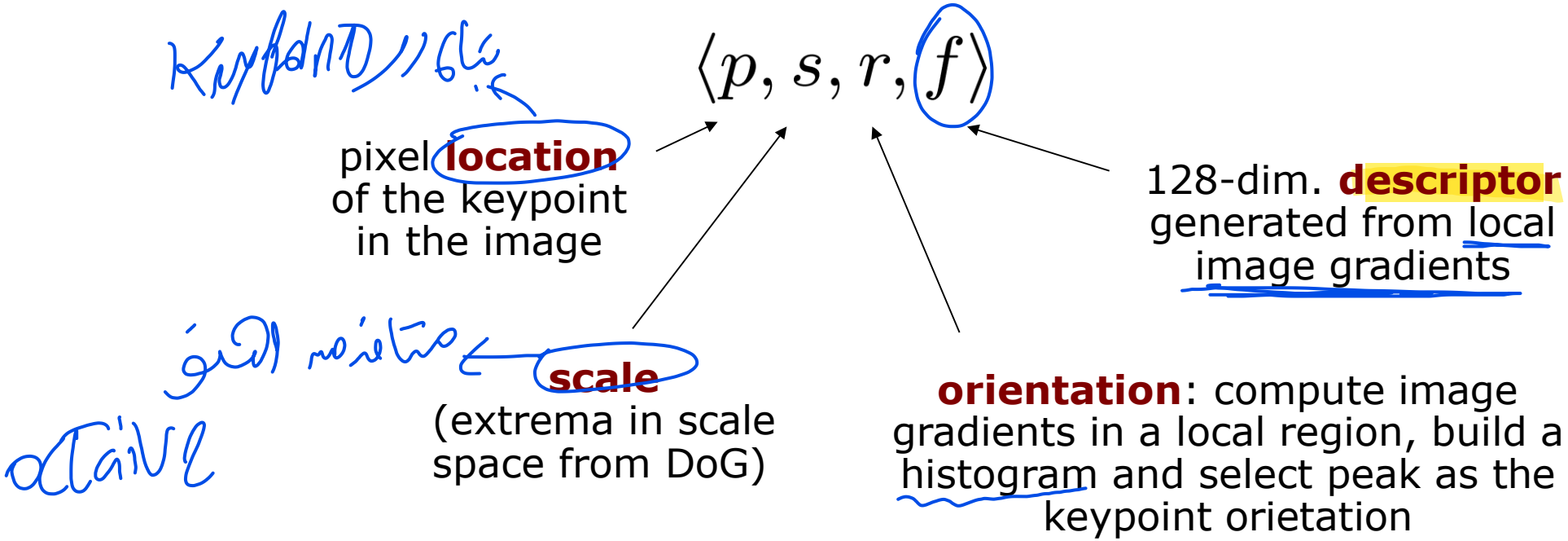
Image courtesy: Vedaldi and Fulkerson 19

SIFT Descriptor

- Image content is transformed into features that are **invariant to**
  - image translation,
  - rotation, and
  - scale
- They are **partially invariant to**
  - illumination changes and
  - affine transformations and 3D projections
- Suitable for detecting visual landmarks
  - **from different angles and distances**
  - **with a different illumination**

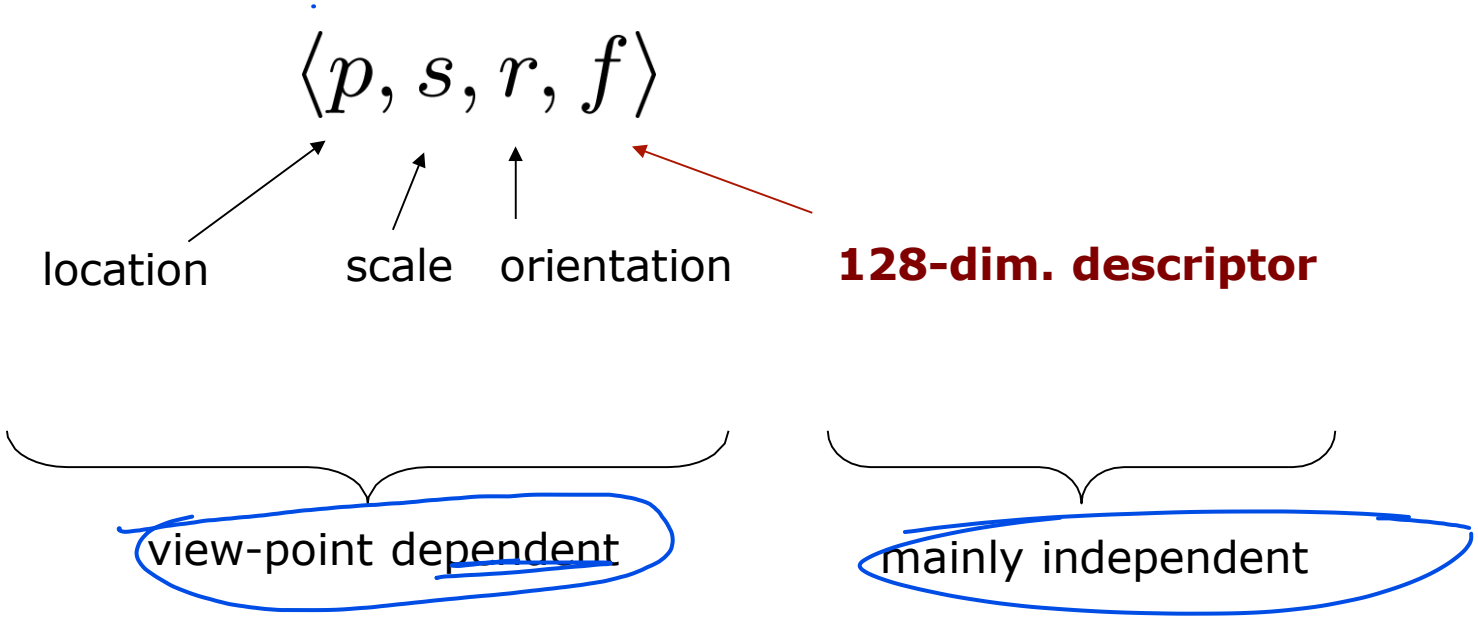
SIFT Features

- A SIFT feature is given by a vector computed at a local extreme point in the scale space



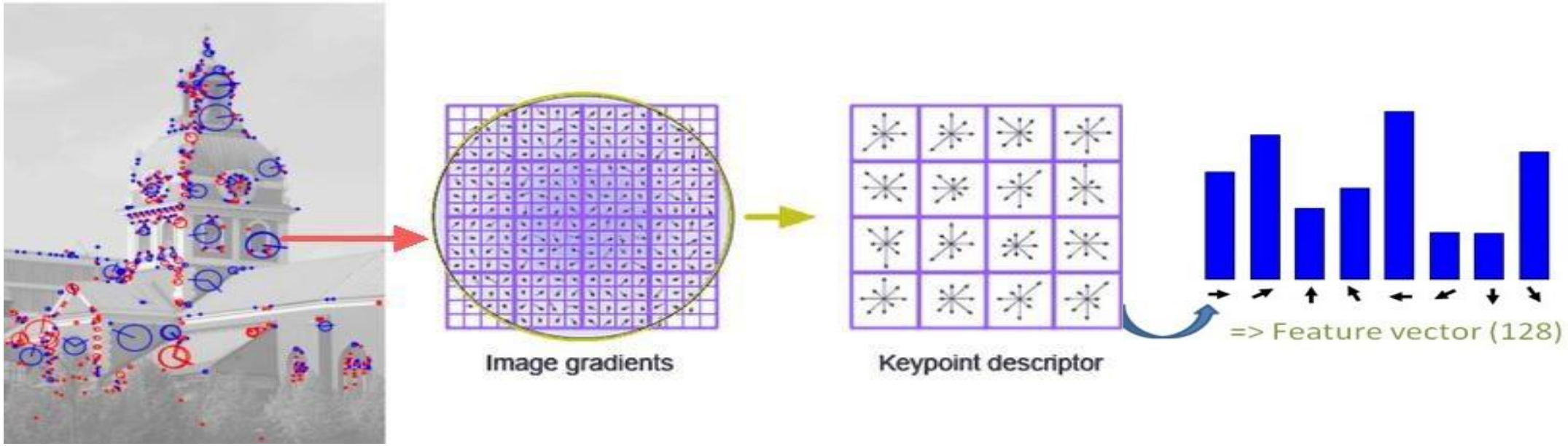
SIFT Features

- A SIFT feature is given by a vector computed at a local extreme point in the scale space



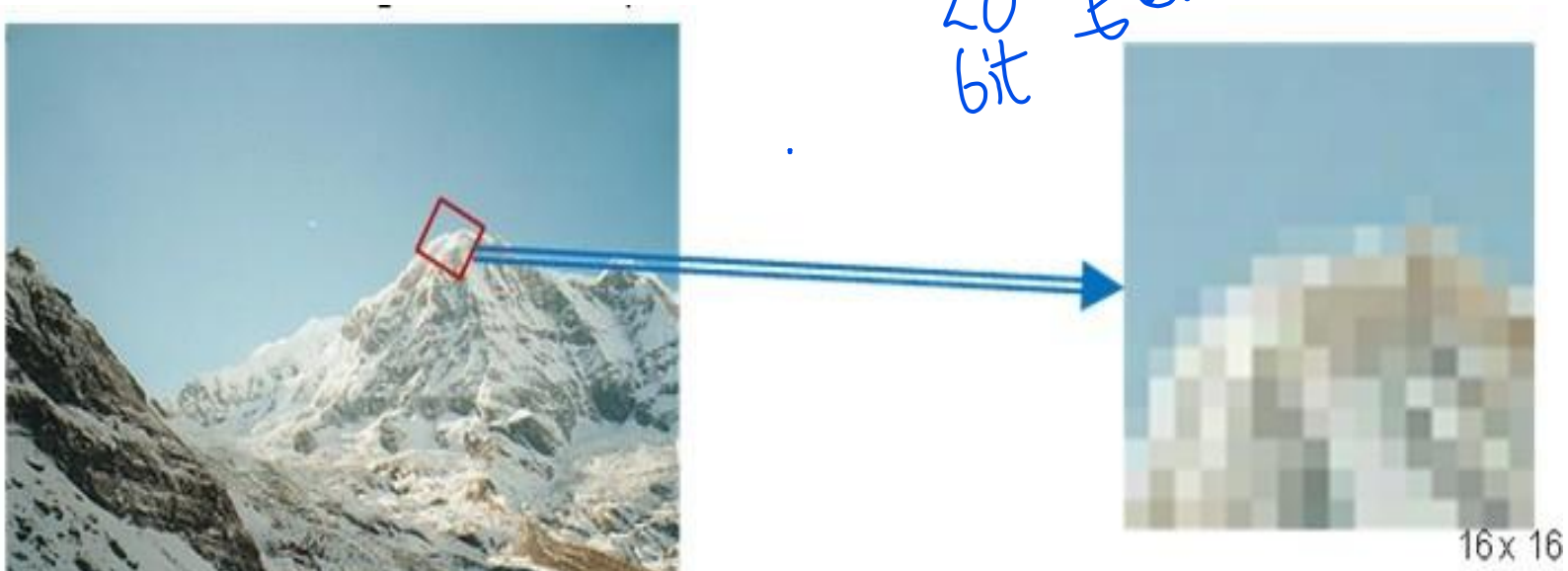
SIFT Descriptor Steps

- **Step1:** Compute image gradients in local 16x16 area at the selected scale
- **Step2:** Create an array of orientation histograms
- **Step3:** 8 orientations x 4x4 histogram array = 128 dimensions (yields best results)



Step 1: Compute image gradients in local 16x16 area at the selected scale

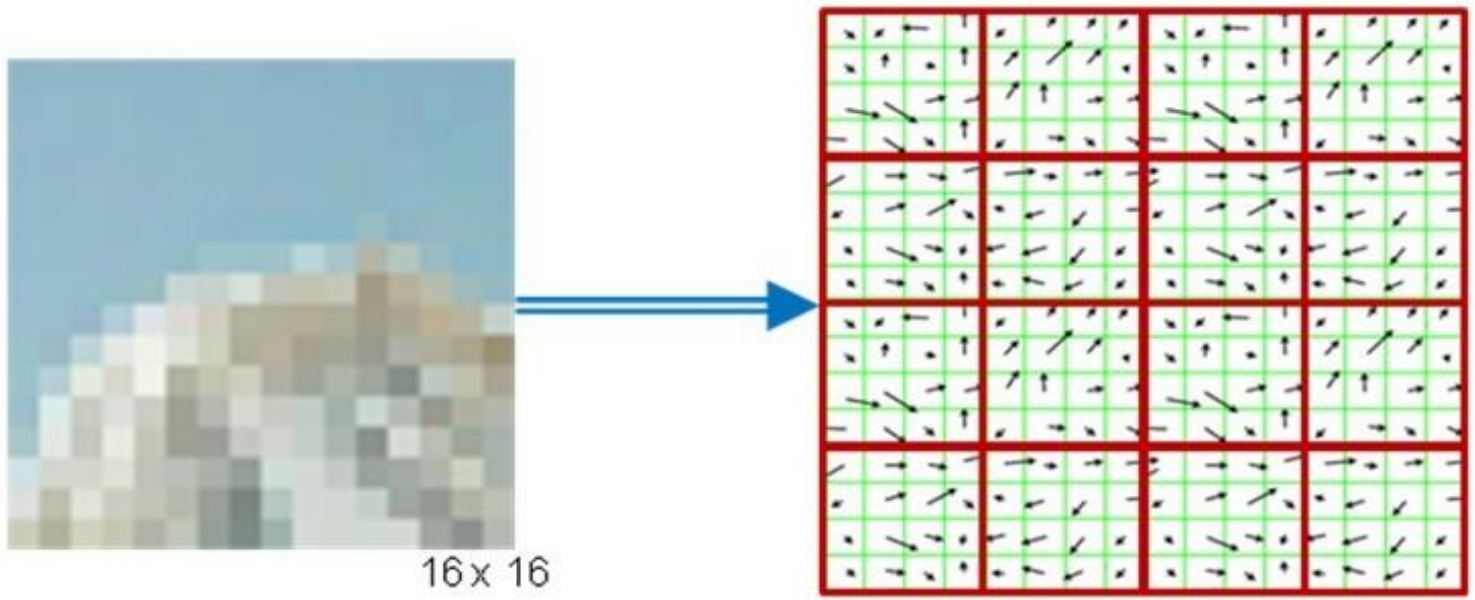
- For a given keypoint, warp the region around it to orientation and scale and resize the region to 16X16 pixels.



- Since we use the surrounding pixels, the descriptors will be partially invariant to illumination or brightness of the images.

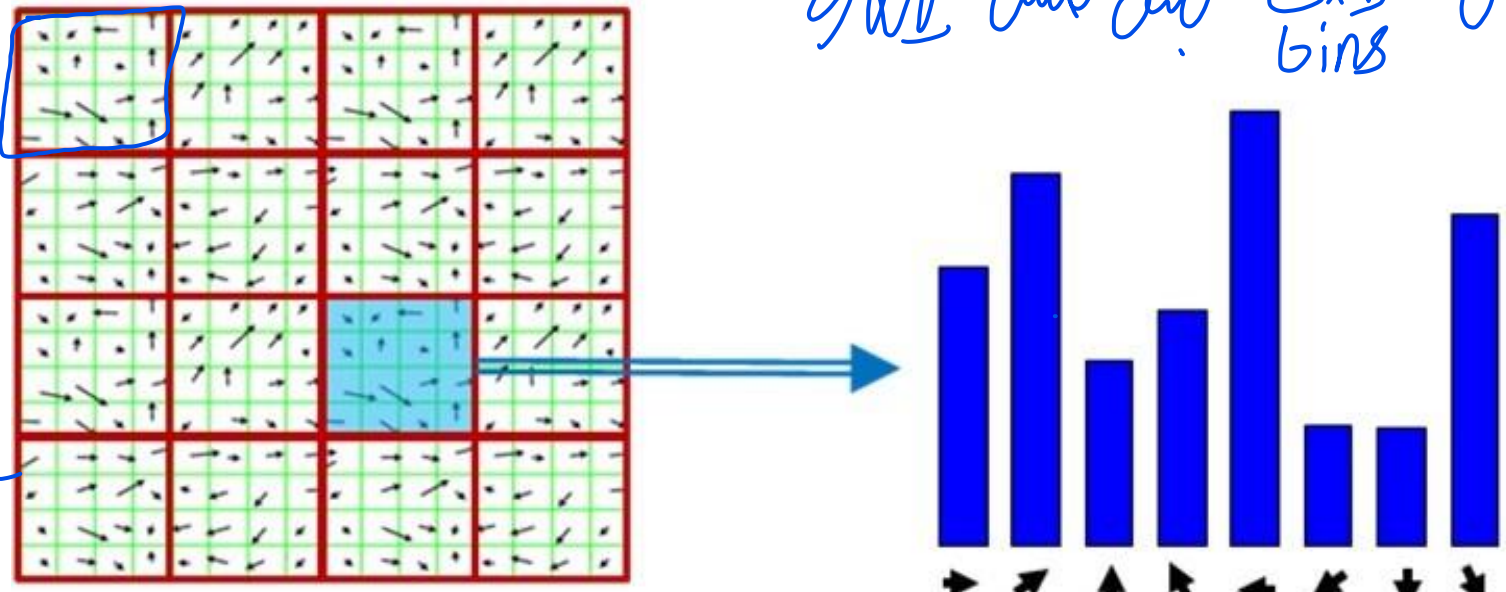
Step 1:

- Compute the gradients for each pixel (orientation and magnitude) to generate a unique fingerprint for this keypoint.
- Then, divide the pixels into 16, 4X4 pixels squares.



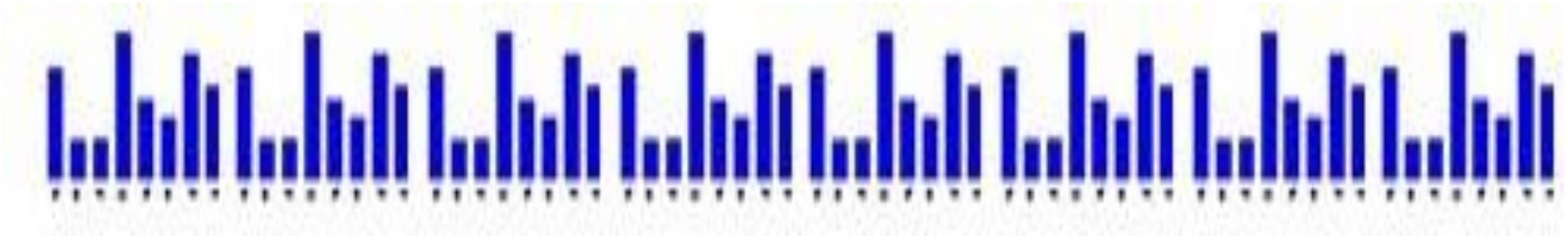
Step 2: Create an array of orientation histograms

- For each square, compute gradient direction histogram over 8 directions. There is an 8x1 vector for each 4X4 pixels square.

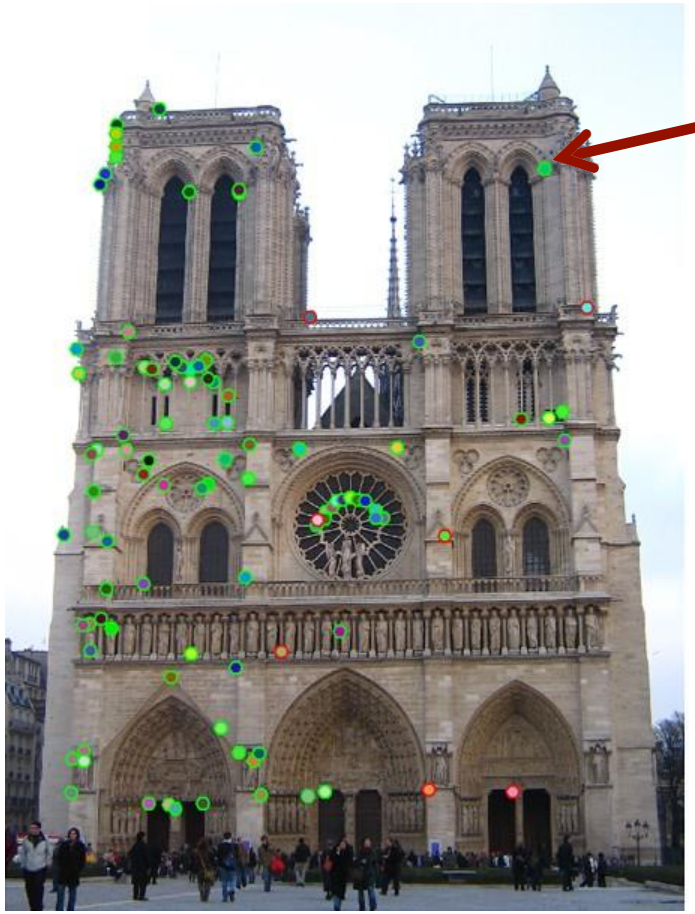


Step 3: 8 orientations x 4x4 histogram array

- Concatenate the histograms to obtain a 128 (16*8) dimensional feature vector:



SIFT Approach Done!

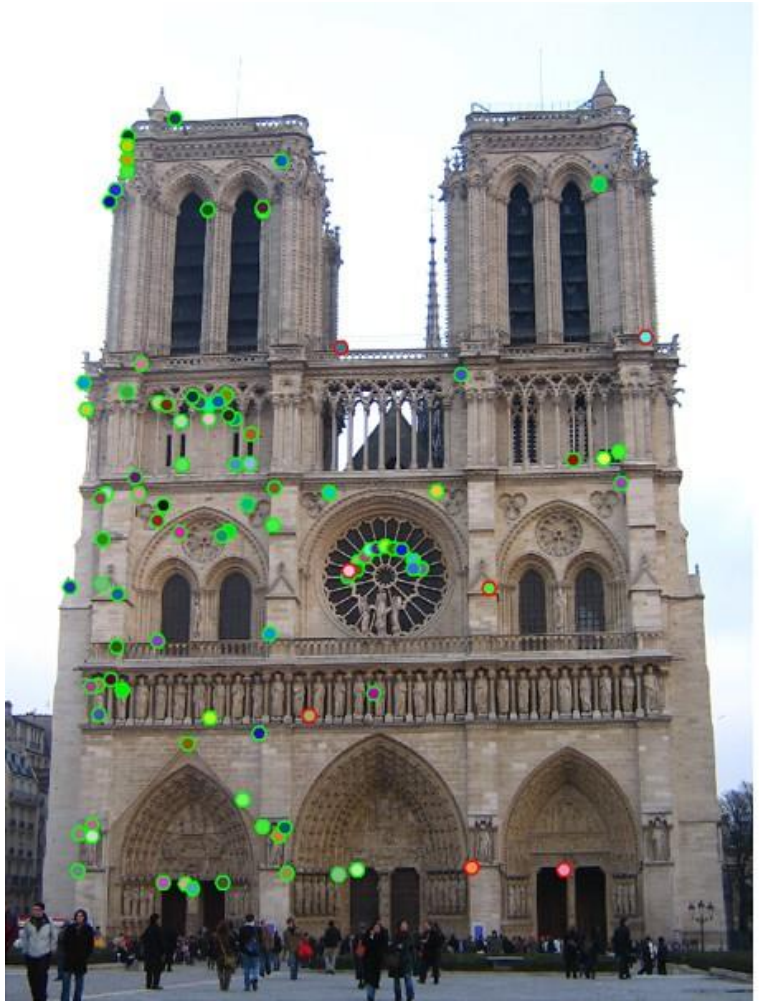


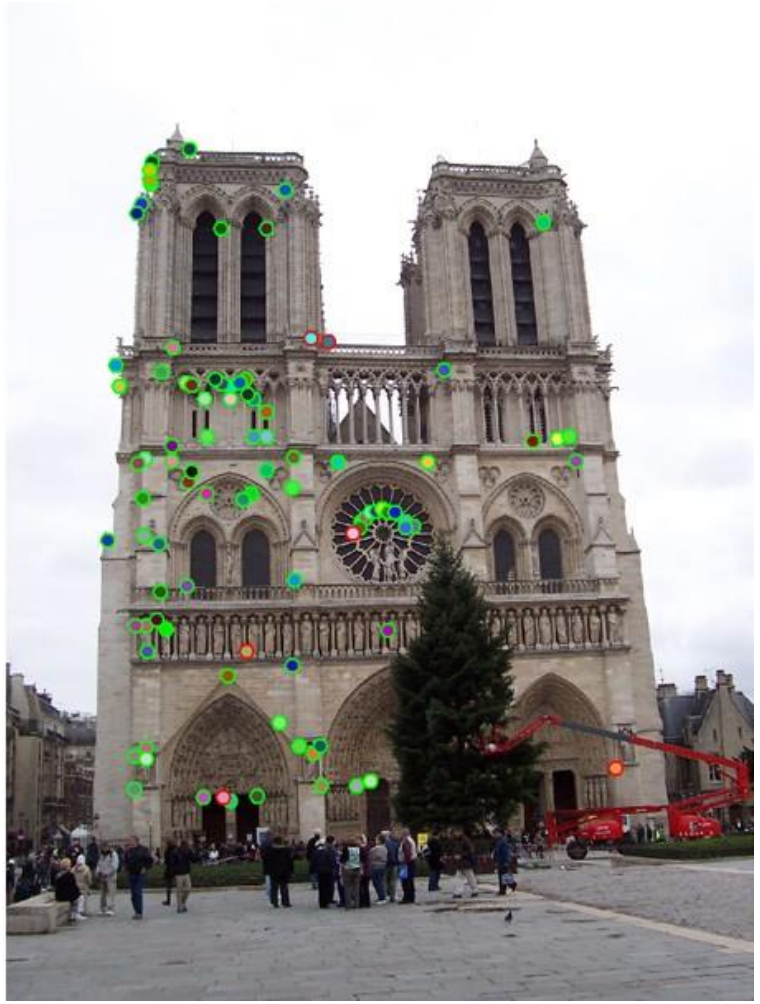
keypoint (via DoG)

descriptor (via gradient histogram)

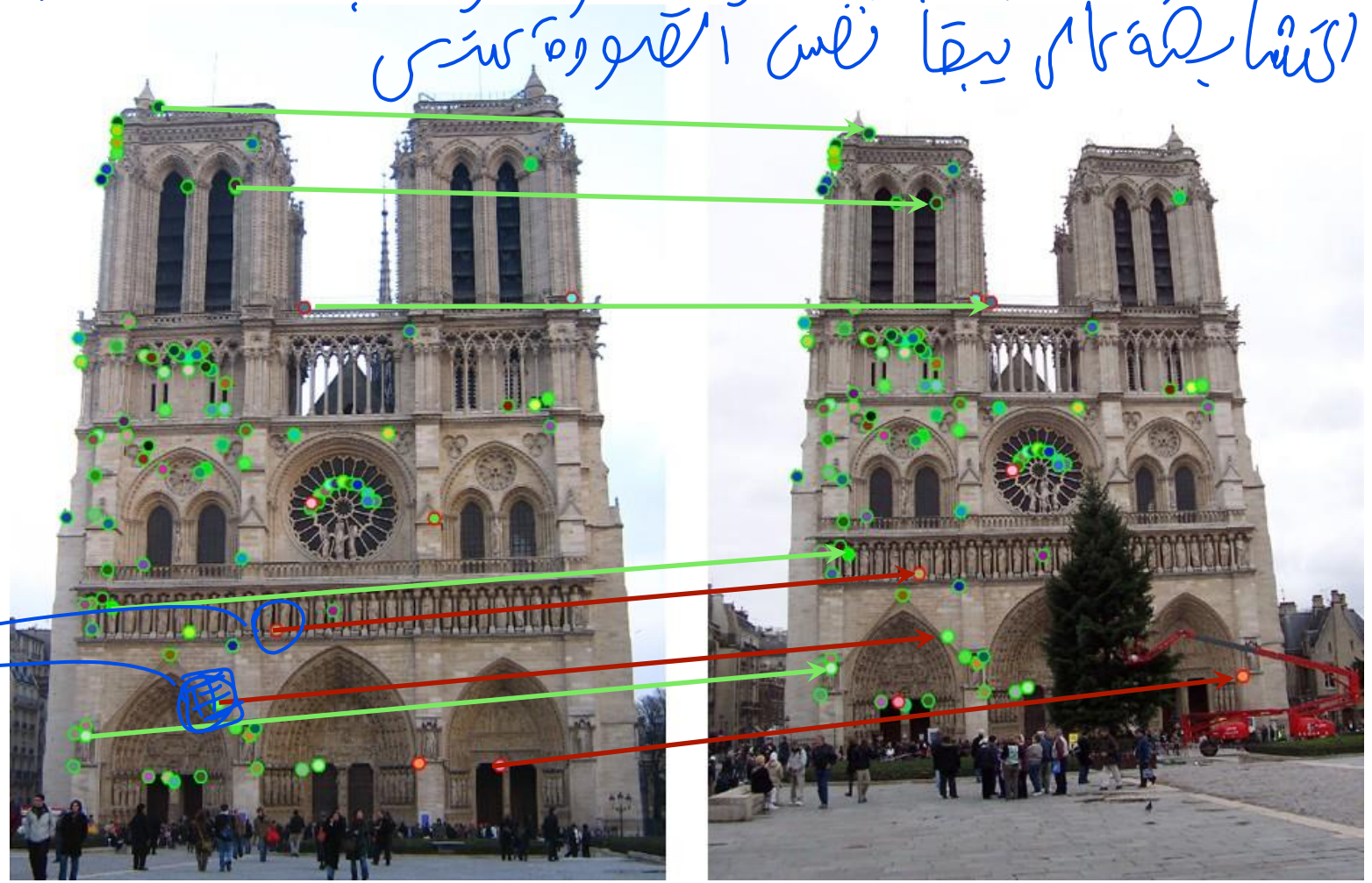
$$f = \begin{bmatrix} 0.02 \\ 0.04 \\ 0.1 \\ 0.03 \\ 0 \\ \dots \end{bmatrix}$$

How To Match Them?





Based on Descriptor Difference



Lowe's Ratio Test

§ 3 Steps test to eliminate ambiguous matches for a query feature *q*:

1. Step: Find the closest two descriptors to ***q***, called ***p*₁** and ***p*₂** based on the Euclidean distance ***d***

$$d(\mathbf{p}, \mathbf{q}) = \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2}$$

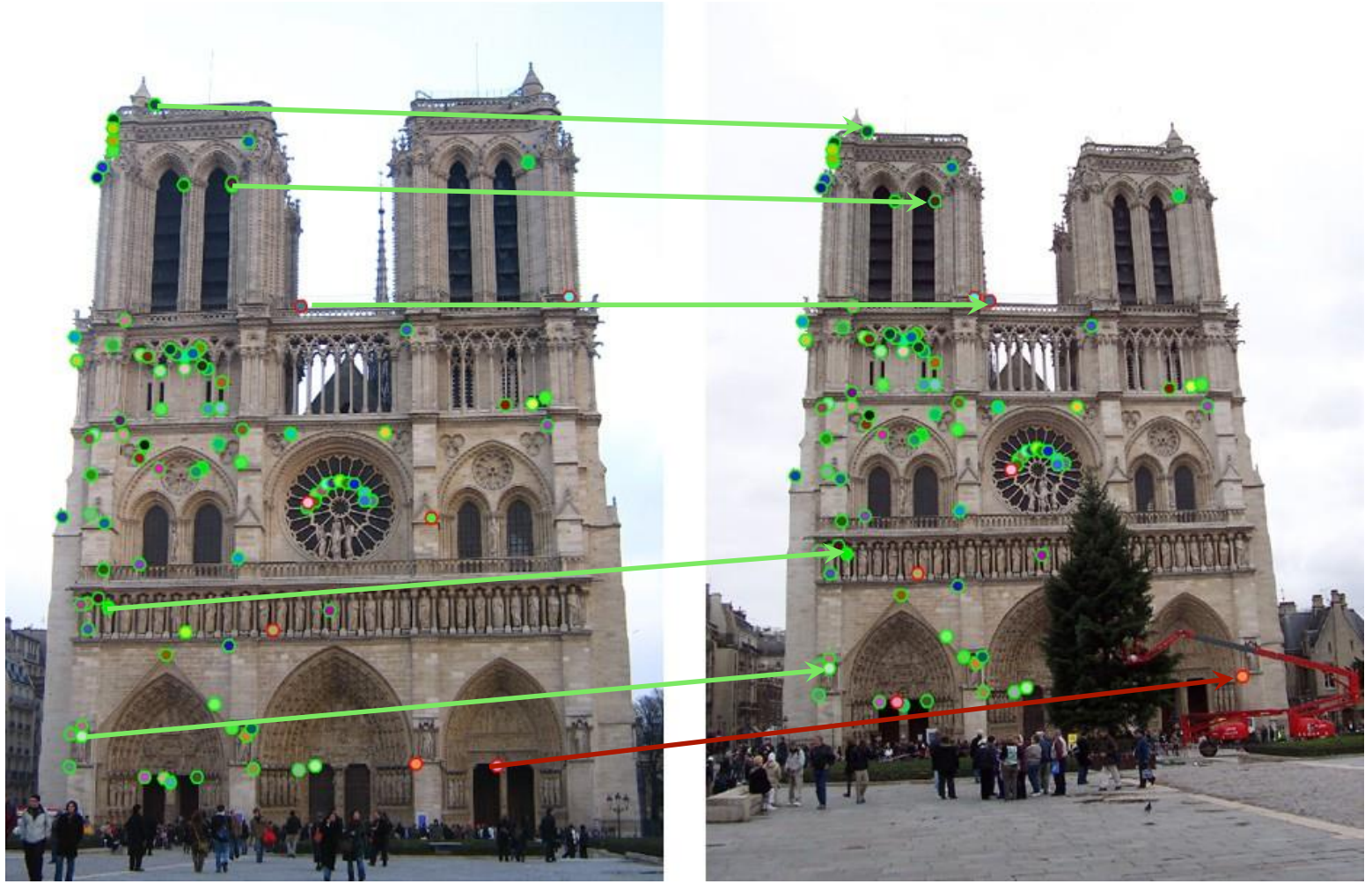
2. Step: Test if the distance to the best match is smaller than a threshold:

$$d(q, p_1) < T$$

3. Step: Accept match only if the best match is substantially better than second:

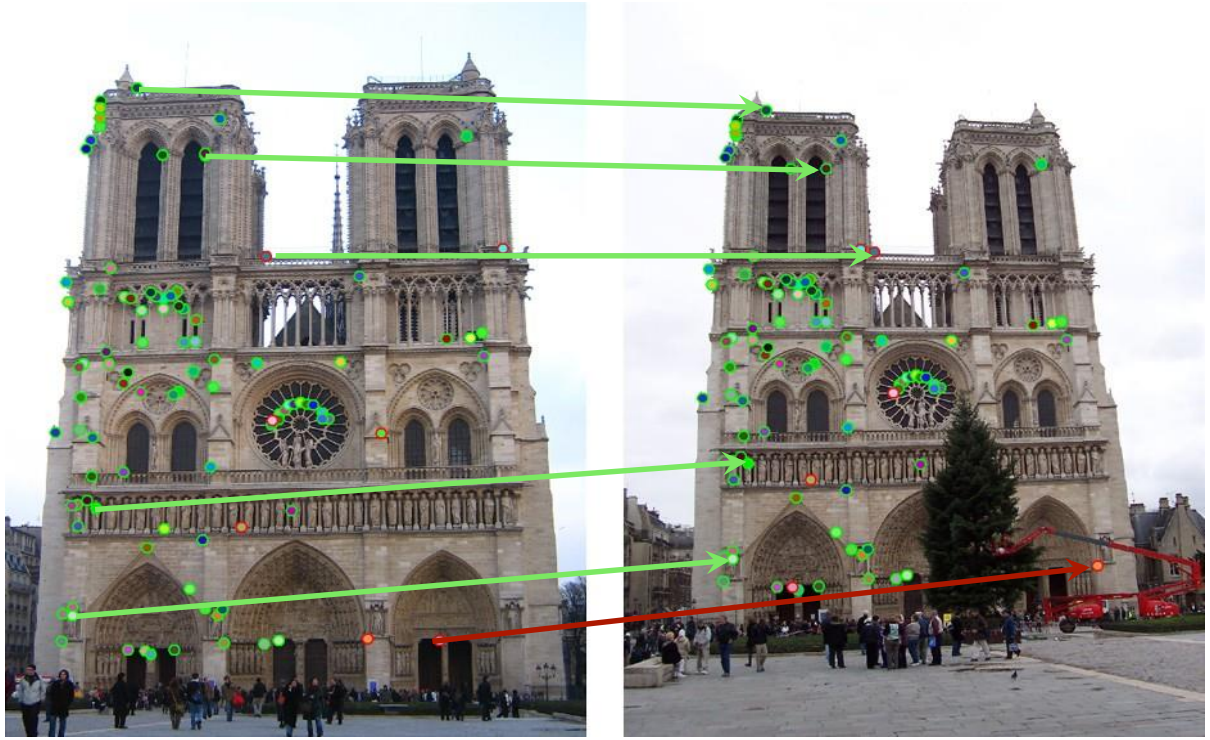
$$\frac{d(q, p_1)}{d(q, p_2)} < \frac{1}{2}$$

Based on RatioTest

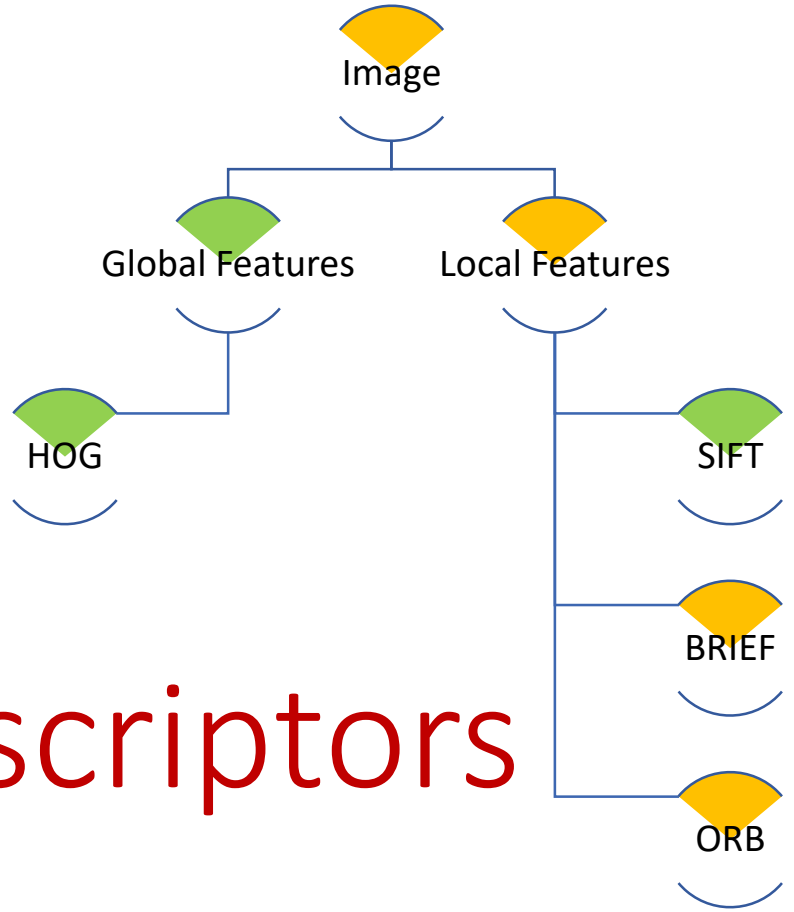


Outliers

- § Lowe’s Ratio test works well
- § There will still remain few outliers
- § Outliers require an extra treatment



Local features:
Keypoints & Descriptors



Binary Descriptors: BRIEF, ORB

Why Binary Descriptors?

- Complex features such as SIFT work well and are a gold standard
- SIFT is expansive to compute
- Binary descriptors aim at generating **small binary strings** that are **easy to compute and compare**

Key Advantages of Binary Descriptors

- § **Compact descriptor**
  - The number of pairs gives the length in bits

- § **Fast to compute**
  - Simply intensity value comparisons

- § **Trivial and fast to compare**
  - Hamming distance

$$d_{\text{Hamming}}(B_1, B_2) = \text{sum}(\text{xor}(B_1, B_2))$$

Examples of Binary descriptors

- **BRIEF** – Binary Robust Independent Elementary Features
- **ORB** – Oriented FAST Rotated BRIEF
- **BRISK** – Binary Robust Invariant Scalable Keypoints
- **FREAK** – Fast REtinA Keypoint

Summary

- Keypoints and descriptor together define common visual features
- Keypoint defines the location
- Descriptor describes the appearance
- Several descriptors operating on gradient histograms (SIFT, SURF, …)
- Binary descriptors for efficiency (BRIEF, ORB, ...)

DataCamp Courses

CAREER TRACK: Machine Learning Scientist with Python

Phase 1: (5 points) due Fri. 24 Nov.

1- Supervised Learning with scikit-learn (1)

2- Preprocessing for Machine Learning in Python (8)

3- Feature Engineering for Machine Learning in Python (10)

4- Linear Classifiers in Python (3)

Phase 2: (5 points) due Fri. 22 Dec.

1- Machine Learning with Tree-Based Models in Python (4)

2- Cluster Analysis in Python (6)

3- Dimensionality Reduction in Python (7)

4- Model Validation in Python (11)

Thanks