

# **CHAPTER - 3**

# **FEATURE DETECTION**

**Prof. Mittal Darji**  
**IT, GCET**

# Segmentation

- Segmentation algorithms are often based on one of the following two basic properties of intensity values:
- Similarity
  - Partitioning an image into regions that are similar according to a set of predefined criteria.
- Discontinuity
  - Detecting boundaries of regions based on local discontinuity in intensity.

# Applications

- Object recognition & tracking
- Medical image processing (detecting tumor)
- Biometrics
- Processing Aerial or satellite images
- Automated traffic control
- Computer integrated surgery
- Treatment planning etc

# Four types of segmentation algorithms

## **Thresholding** - *Similarity*

Based on pixel intensities (shape of histogram is often used for automation).

## **Edge-based** - *Discontinuity*

Detecting edges that separate regions from each other.

## **Region-based** - *Similarity*

Grouping similar pixels (with e.g. region growing or merge & split).

## **Watershed segmentation** - *Discontinuity*

Find regions corresponding to local minima in intensity.

## **Match-based** - *Similarity*

Comparison to a given template.

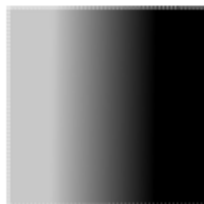
# Edge-based segmentation

Detection of sharp, local changes in intensity.

**Step**



**Ramp**



**Line**



**Point**



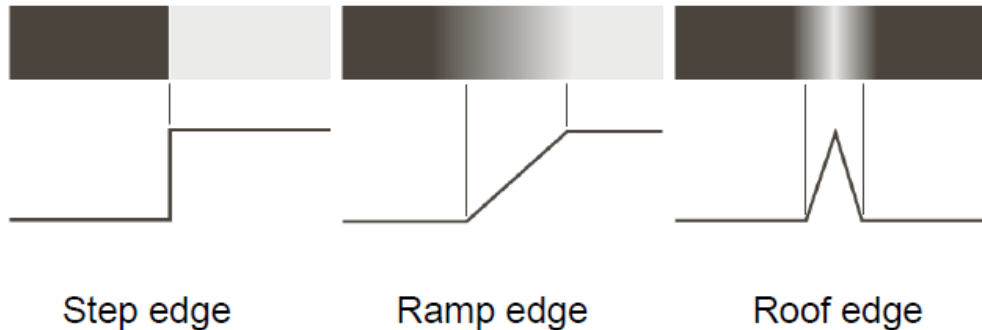
light

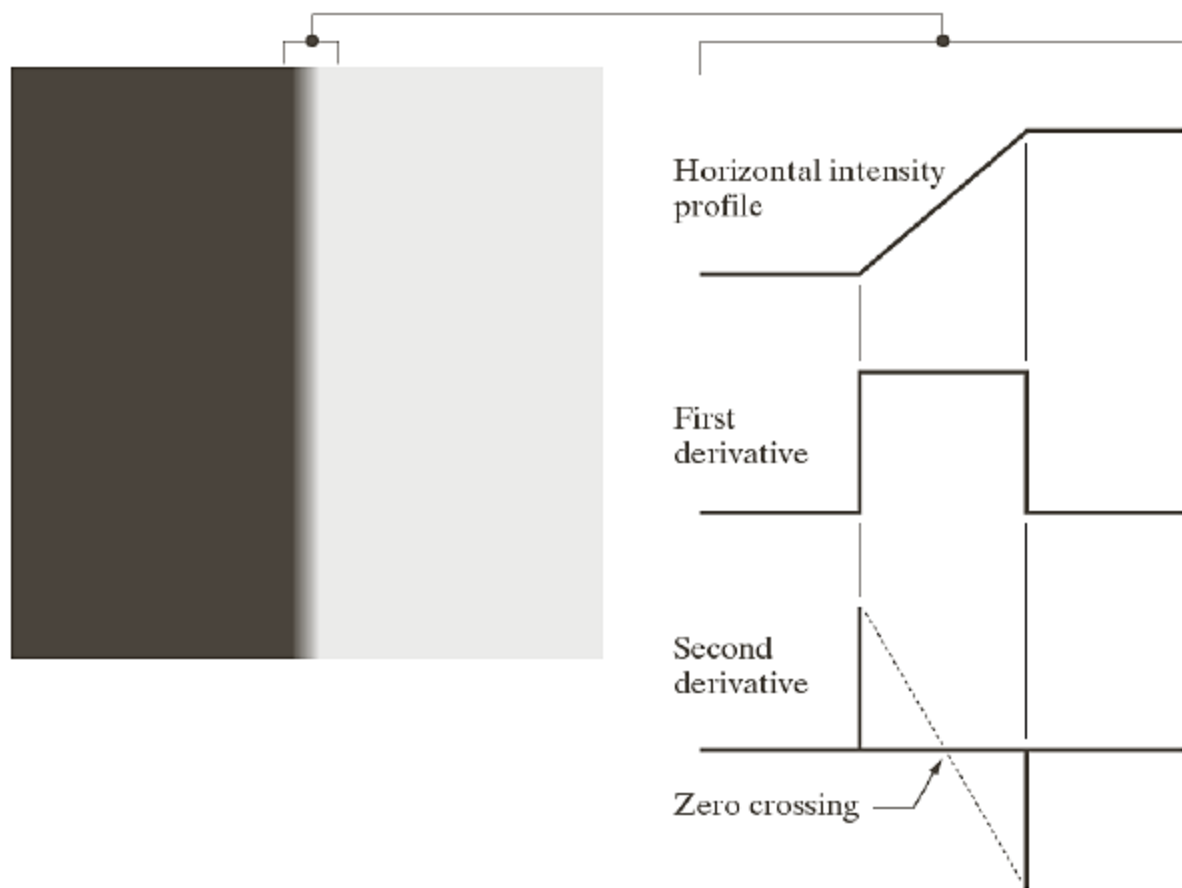
dark

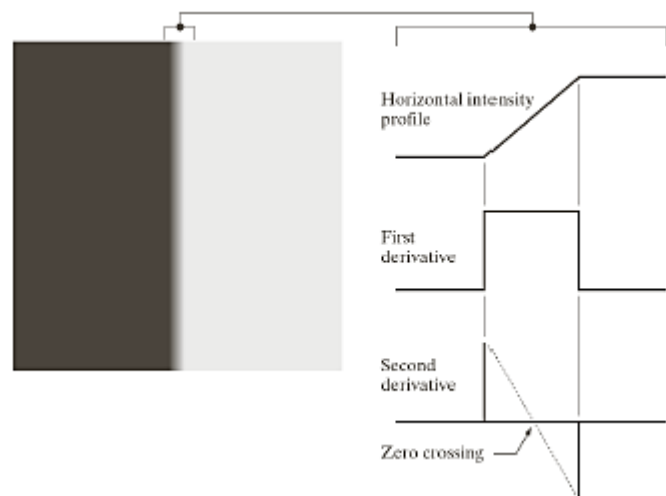


# Edge detection

- Edge models
  - Ideally, edges should be 1 pixel thin.
  - In practice, they are blurred and noisy.







## Edge point detection

- Magnitude of the first derivative.
- Sign change of the second derivative.

## Observations:

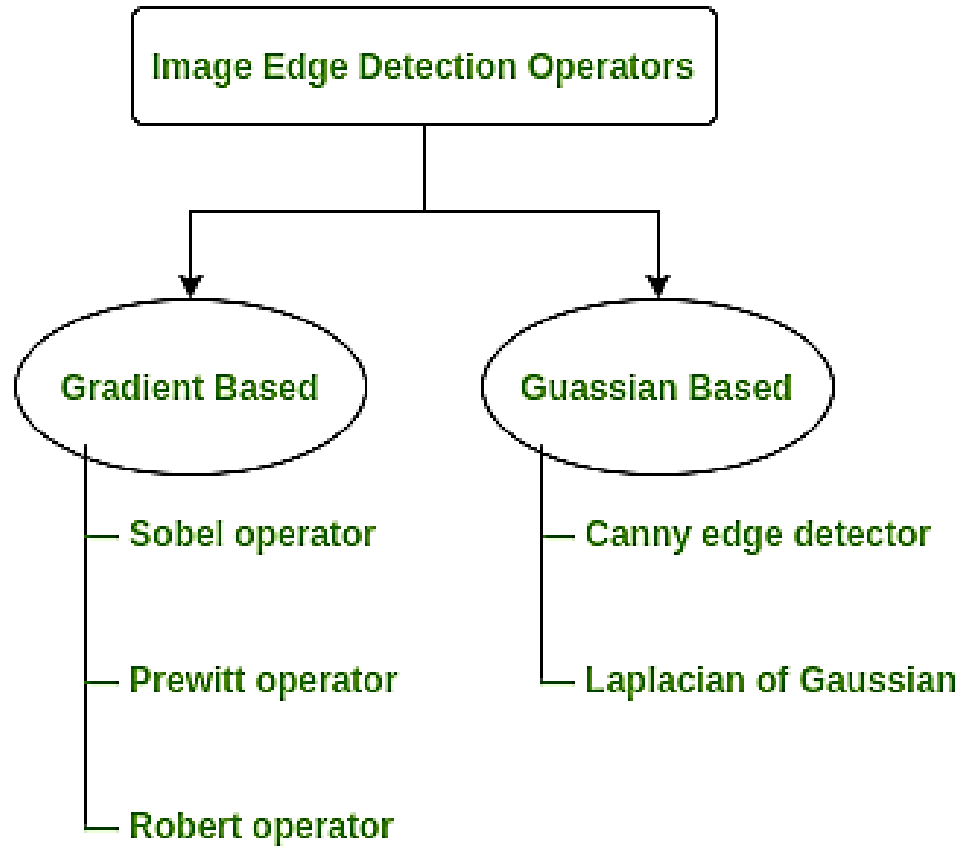
- Second derivative produces two values for an edge (undesirable).
- Its zero crossings may be used to locate the centres of thick edges.



# Edge detection operators

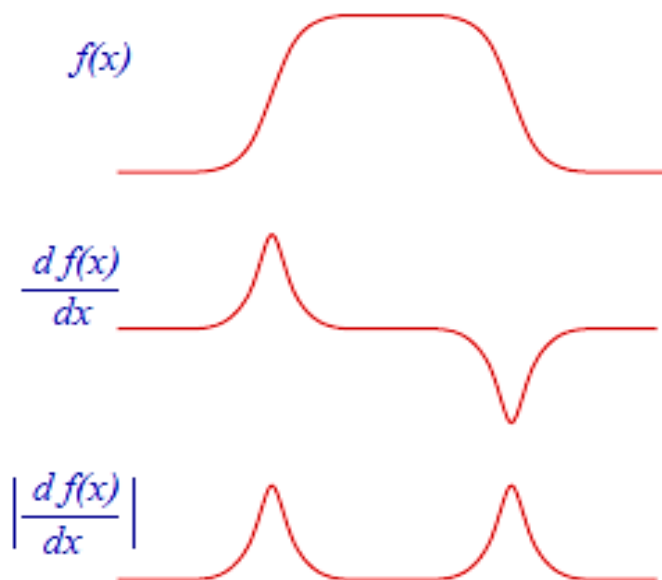
**Edge Detection Operators** are of two types:

- **Gradient** – based operator which computes first-order derivations in a digital image like, Sobel operator, Prewitt operator, Robert operator
- **Gaussian** – based operator which computes second-order derivations in a digital image like, Canny edge detector, Laplacian of Gaussian (LoG)



# First Order Derivatives

First Order Differentials: In One-Dimension we have



We can then detect the edge by a simple threshold of

$$\left| \frac{df(x)}{dx} \right| > T \Rightarrow \text{Edge}$$

but in two-dimensions things are more difficult:

$$\left| \frac{\partial f(x,y)}{\partial x} \right| \rightarrow \text{Vertical Edges} \quad \left| \frac{\partial f(x,y)}{\partial y} \right| \rightarrow \text{Horizontal Edges}$$

But we really want to detect edges in **all** directions.

In two-dimensions the first order differential  $\nabla f(x,y)$  is a vector, given by

$$\frac{\partial f(x,y)}{\partial x} \hat{i} + \frac{\partial f(x,y)}{\partial y} \hat{j}$$

so we need to calculate the modulus of the gradient, given by

$$|\nabla f(x,y)| = \sqrt{\left| \frac{\partial f(x,y)}{\partial x} \right|^2 + \left| \frac{\partial f(x,y)}{\partial y} \right|^2}$$

which is a *non-linear* operation.

We can now threshold to give the edges, with

$$\begin{array}{ll} |\nabla f(x,y)| > T & \text{Edge} \\ |\nabla f(x,y)| < T & \text{no Edge} \end{array}$$

**Real Space:** From previous we know we can form the first order differentials by Convolution, with

$$\frac{\partial f(i,j)}{\partial i} = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \odot f(i,j)$$

and

$$\frac{\partial f(i,j)}{\partial j} = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} \odot f(i,j)$$

so we can calculate the

$$|\nabla f(i,j)| = \sqrt{\left| \frac{\partial f(i,j)}{\partial i} \right|^2 + \left| \frac{\partial f(i,j)}{\partial j} \right|^2}$$

This implementation requires considerable numerical calculation. Note that square root must be calculated in floating point.

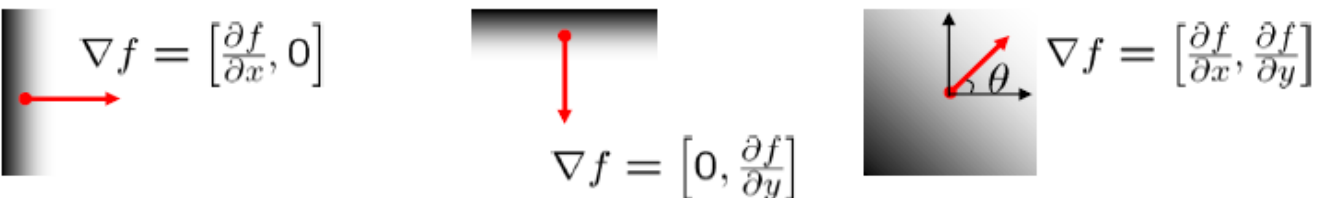
“Faster” approximation to the modulus of the gradient by

$$\left| \frac{\partial f(i,j)}{\partial i} \right| + \left| \frac{\partial f(i,j)}{\partial j} \right|$$

which is frequently a computational saving of 30%.

# Image Gradient

- The gradient of an image:  $\nabla f = \left[ \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y} \right]$

- 

The gradient points in the direction of most rapid increase in intensity.

The gradient direction is given by  $\theta = \tan^{-1} \left( \frac{\partial f}{\partial y} / \frac{\partial f}{\partial x} \right)$

The *edge strength* is given by the gradient magnitude

$$\|\nabla f\| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}$$

# Edge gradient - Roberts operator

- In 1963, Roberts proposed this edge-finding operator.
- Roberts edge operator is a 2x2 template that uses the difference between two adjacent pixels in the diagonal direction.
- The template of Roberts operator is divided into horizontal direction and vertical direction.

# Edge gradient - Roberts operator

- one of the simplest operators, which uses local difference operators to find edges.
- It uses the difference between two adjacent pixels in the diagonal direction to approximate the gradient amplitude to detect edges.
- The **effect of detecting vertical edges is better than that of oblique edges**, the positioning accuracy is high, and it is sensitive to noise and cannot suppress the influence of noise.



# Gradient operators

$$\frac{\partial f(x, y)}{\partial x} = f(x+1, y) - f(x, y)$$

$$\frac{\partial f(x, y)}{\partial y} = f(x, y+1) - f(x, y)$$

Roberts operators

-1
1

-1	1
----	---

-1	0	0	-1
0	1	1	0

Roberts

# Edge gradient – Prewitt operator

- Prewitt operator is a kind of edge detection of first-order differential operator.
- It uses the gray difference between the upper and lower, left and right adjacent points of the pixel to reach the extreme value at the edge to detect the edge, remove some false edges, and smooth the noise.
- Since the Prewitt operator uses a  $3 * 3$  template to calculate the pixel value in the area, and the Robert operator's template is  $2 * 2$ , the edge detection results of the Prewitt operator are more obvious in the horizontal and vertical directions than the Robert operator.

# Edge gradient – Prewitt operator

- Prewitt operator is suitable for identifying images with more noise and gradual gray scale

-1	-1	-1	-1	0	1
0	0	0	-1	0	1
1	1	1	-1	0	1

Prewitt

# Prewitt operator - Simple and Diagonal

-1	-1	-1	-1	0	1
0	0	0	-1	0	1
1	1	1	-1	0	1

Prewitt

0	1	1	-1	-1	0
-1	0	1	-1	0	1
-1	-1	0	0	1	1

Prewitt

# Edge gradient – Sobel operator

- The Sobel operator adds the concept of weight on the basis of the Prewitt operator.
- It is believed that the distance between adjacent points has a different impact on the current pixel.
- The closer the pixel point corresponds to the current pixel, the greater the impact, so as to realize the image Sharpen and highlight edge contours.

# Edge gradient – Sobel operator

- The Sobel operator detects the edge based on the gray-scale weighted difference of the upper and lower, left and right adjacent points of the pixel, and reaches the extreme value at the edge.
- It has a smoothing effect on noise and provides more accurate edge direction information.
- Because the Sobel operator combines Gaussian smoothing and differential derivation, the result will have more noise resistance.
- When the accuracy requirements are not very high, Sobel operator is a more commonly used edge detection method.

# Edge gradient – Sobel operator

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

Sobel

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

Sobel

Image



Sobel  $|g_y|$



Sobel  $|g_x|$



Sobel  $|g_x| + |g_y|$



Image smoothed  
prior to edge  
detection.  
The wall bricks are  
smoothed out.

Image



Sobel  $|g_y|$



Sobel  $|g_x|$

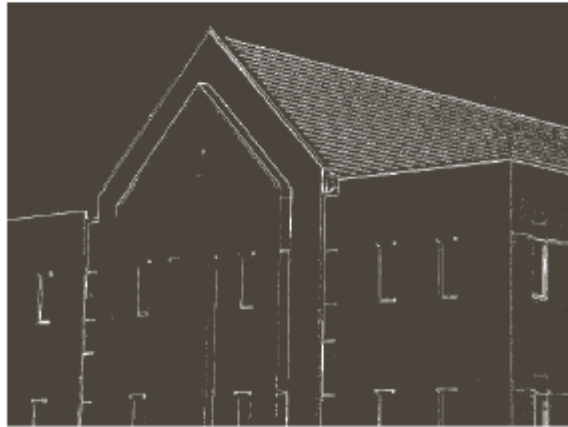


Sobel  $|g_x| + |g_y|$

## Diagonal Sobel filters



Thresholded Sobel gradient amplitudes at 33% of max value



Thresholded gradient

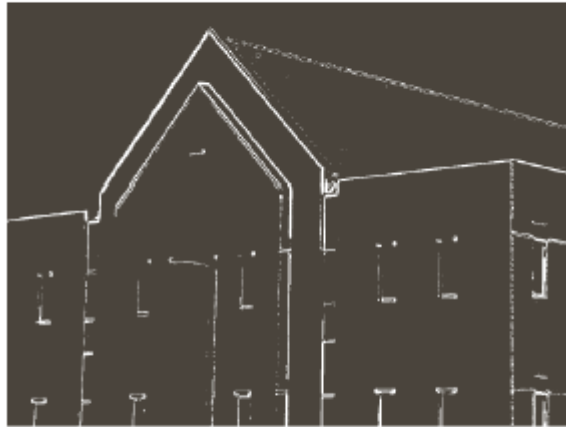
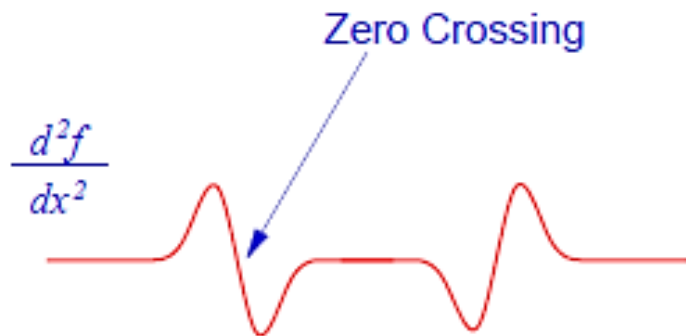


Image smoothing prior to  
gradient thresholding

# Second Order Differentials

Again in one dimension



the edge is then located by the zero crossing.

In two-dimensions we are required to calculate the Laplacian,

$$\nabla^2 f(x,y) = \frac{\partial^2 f(x,y)}{\partial x^2} + \frac{\partial^2 f(x,y)}{\partial y^2}$$

# Laplacian – Second order derivative - 1

Which, as seen previously, can be implemented by a single  $[3 \times 3]$  convolution of

$$\nabla^2 f(i,j) = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} \odot f(i,j)$$



Find the edges by location the zero crossings.

- **Thin Edges:** the edges occur *between* pixels. Always get thin edges, but difficult to display on a digital image.
- **Closed Loops:** edges always form closed loops, reduces break-up of edges, but can cause problems as corners.
- **Noise Problems:** Laplacian is a high pass filter, so enhances high frequencies, and thus noise.

Difficult to post process edge image, so to reduce the effect of noise we typically want to smooth the image *before* we form the Laplacian.

# Laplacian of Gaussian (LoG)

A good way to smooth an image is to convolve it with a Gaussian.

The Laplacian of the smoothed image is then

$$g(i, j) = \nabla^2 [h(i, j) \odot f(i, j)]$$

where  $h(x, y)$  is a Gaussian of the form

$$h(i, j) = \exp\left(-\frac{r^2}{2\sigma^2}\right)$$

where  $r^2 = i^2 + j^2$  and  $\sigma$  is the width of the Gaussian.

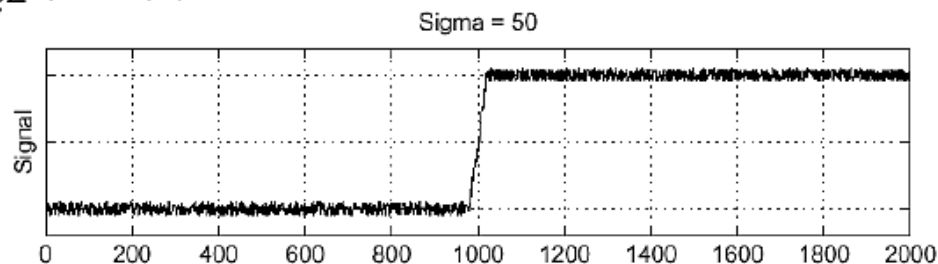
The convolution is linear, so we can write

$$g(i, j) = [\nabla^2 h(i, j)] \odot f(i, j)$$

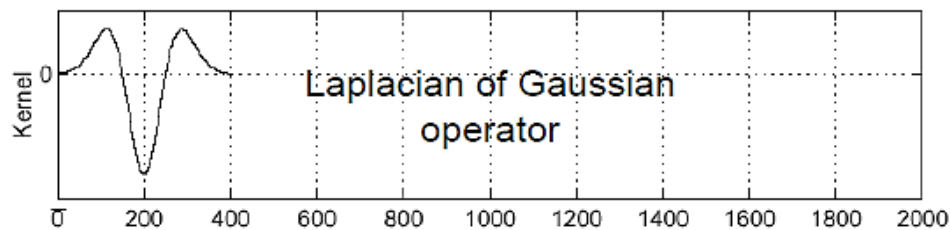
# Laplacian of Gaussian

Consider  $\frac{\partial^2}{\partial x^2}(h \star f)$

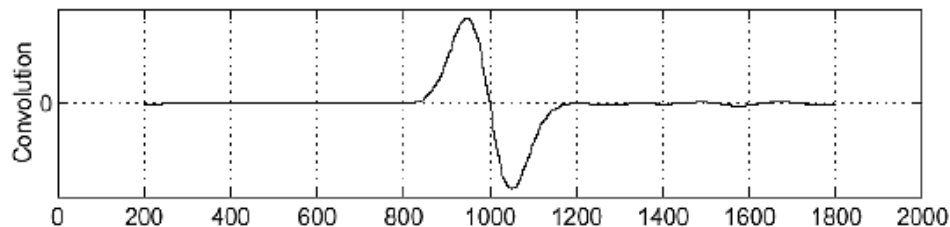
$f$



$\frac{\partial^2}{\partial x^2}h$



$(\frac{\partial^2}{\partial x^2}h) \star f$



Where is the edge? Zero-crossings of bottom graph





original image (Lena)



LoG followed by zero  
crossing detection

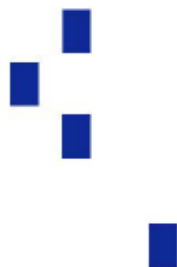
# Designing an optimal edge detector

Criteria for an “optimal” edge detector [Canny 1986]:

- **Good detection:** the optimal detector must minimize the probability of false positives (detecting spurious edges caused by noise), as well as that of false negatives (missing real edges).
- **Good localization:** the edges detected must be as close as possible to the true edges
- **Single response:** the detector must return one point only for each true edge point; that is, minimize the number of local maxima around the true edge.



True  
edge



Poor robustness  
to noise



Poor  
localization



Too many  
responses

# Canny edge detector

- The Canny edge detector was developed way back in 1986 by John F. Canny. And it's still widely used today was one of the default edge detectors in image processing.
- The Canny edge detection algorithm can be broken down into 5 steps:
  - Noise reduction using Gaussian filter
  - Gradient calculation along the horizontal and vertical axis
  - Non-Maximum suppression of false edges
  - Double thresholding for segregating strong and weak edges
  - Edge tracking by hysteresis

# Canny edge detector

- Algorithm is based on grayscale pictures.
- Therefore, the pre-requisite is to convert the image to grayscale before applying steps.

# 1 – Noise Reduction

- Since the mathematics involved behind the scene are mainly based on derivatives, edge detection results are highly sensitive to image noise.
- One way to get rid of the noise on the image, is by applying Gaussian blur to smooth it.
- To do so, image convolution technique is applied with a Gaussian Kernel (3x3, 5x5, 7x7 etc...).
- The kernel size depends on the expected blurring effect. Basically, the smallest the kernel, the less visible is the blur. In our example, we will use a 5 by 5 Gaussian kernel.

# 1 – Noise Reduction

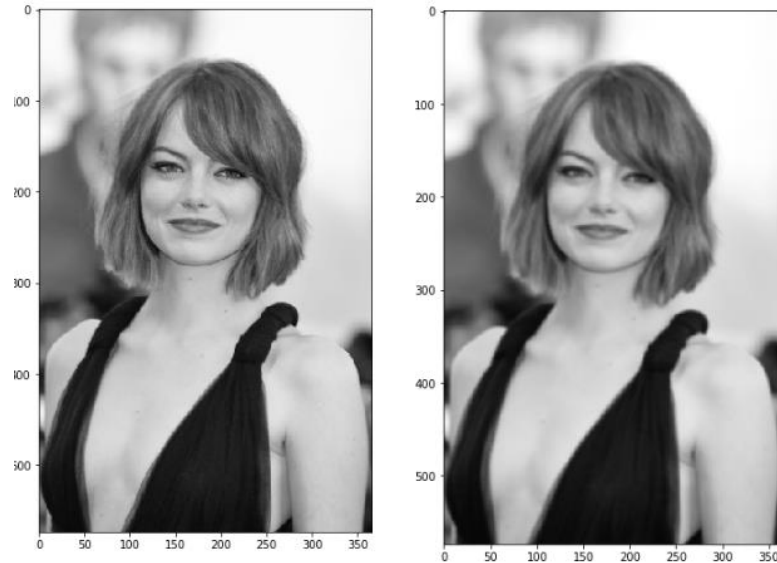
- The equation for a Gaussian filter kernel of size  $(2k+1) \times (2k+1)$  is given by:

$$H_{ij} = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{(i - (k+1))^2 + (j - (k+1))^2}{2\sigma^2}\right); 1 \leq i, j \leq (2k+1)$$

Gaussian filter kernel equation

# 1 – Noise Reduction

After applying the Gaussian blur, we get the following result:



Original image (left) — Blurred image with a Gaussian filter (sigma=1.4 and kernel size of 5x5)

## 2 – Gradient Calculation

- The Gradient calculation step detects the edge intensity and direction by calculating the gradient of the image using edge detection operators.
- Edges correspond to a change of pixels' intensity.
- To detect it, the easiest way is to apply filters that highlight this intensity change in both directions:  
horizontal (x) and vertical (y).



## 2 – Gradient Calculation

- When the image is smoothed, the derivatives  $I_x$  and  $I_y$  w.r.t.  $x$  and  $y$  are calculated. It can be implemented by convolving  $I$  with Sobel kernels  $K_x$  and  $K_y$ , respectively:

$$K_x = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}, K_y = \begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix}.$$

Sobel filters for both direction (horizontal and vertical)

## 2 – Gradient Calculation

- Then, the magnitude  $G$  and the slope  $\theta$  of the gradient are calculated as follow:

$$|G| = \sqrt{I_x^2 + I_y^2},$$
$$\theta(x, y) = \arctan \left( \frac{I_y}{I_x} \right)$$

Gradient intensity and Edge direction

## 2 – Gradient Calculation



Blurred image (left) — Gradient intensity (right)

## 2 – Gradient Calculation



Blurred image (left) — Gradient intensity (right)

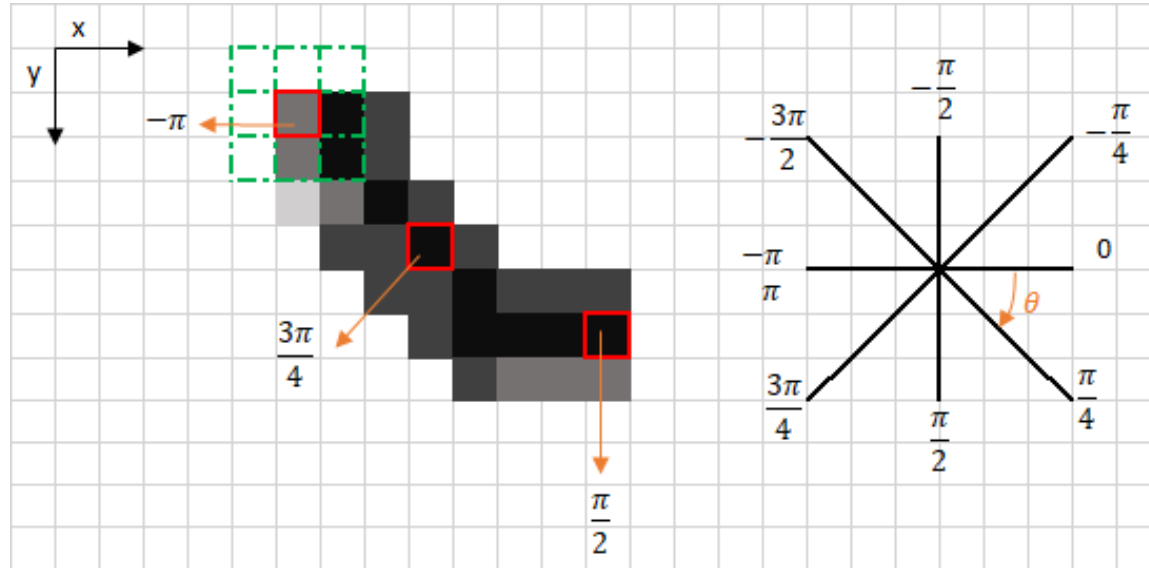
we can see that some of the edges are thick and others are thin. Non-Max Suppression step will help us mitigate the thick ones.

Moreover, the gradient intensity level is between 0 and 255 which is not uniform. The edges on the final result should have the same intensity (i-e. white pixel = 255).

# 3 – Non-Maximum Suppression

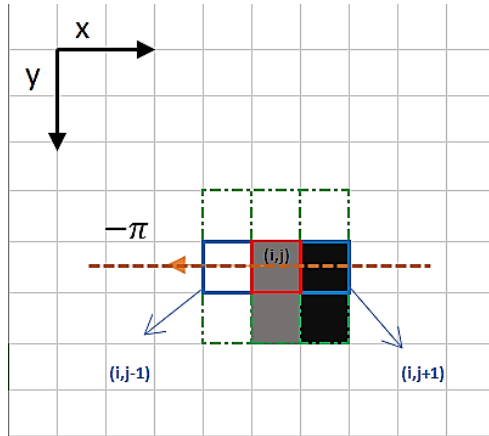
- Ideally, the final image should have thin edges. Thus, we must perform non-maximum suppression **to thin out the edges**.
- The principle is simple:
- the algorithm goes through all the points on the gradient intensity matrix and finds the pixels with the maximum value in the edge directions.

# 3 – Non-Maximum Suppression



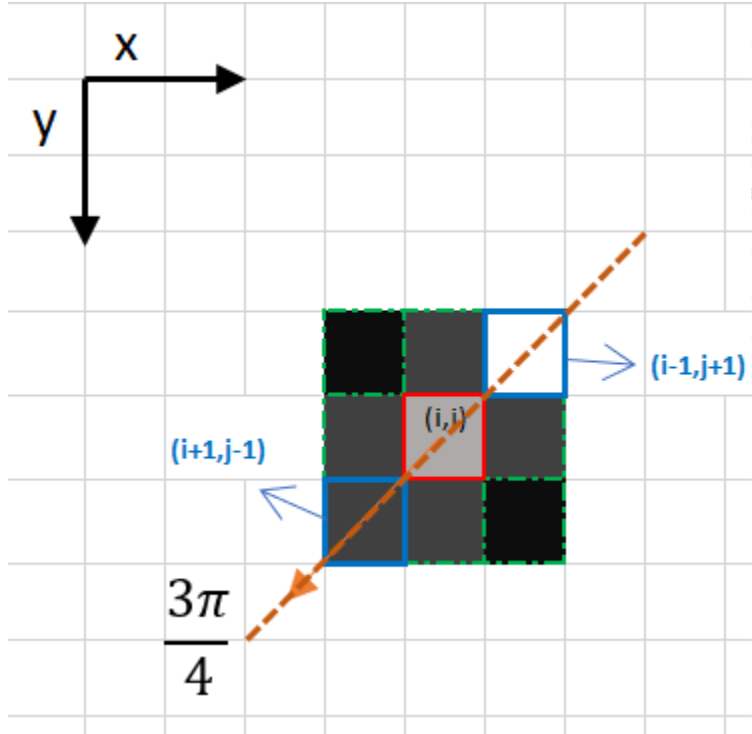
- The upper left corner red box present on the above image, represents an intensity pixel of the Gradient Intensity matrix being processed. The corresponding edge direction is represented by the orange arrow with an angle of  $-\pi$  radians ( $\pm 180$  degrees).

# 3 – Non-Maximum Suppression



- The edge direction is the orange dotted line (horizontal from left to right). The purpose of the algorithm is to check if the pixels on the same direction are more or less intense than the ones being processed.
- In the example above, the pixel  $(i, j)$  is being processed, and the pixels on the same direction are highlighted in blue  $(i, j-1)$  and  $(i, j+1)$ .
- If one of those two pixels is more intense than the one being processed, then only the more intense one is kept. Pixel  $(i, j-1)$  seems to be more intense, because it is white (value of 255). Hence, the intensity value of the current pixel  $(i, j)$  is set to 0.
- If there are no pixels in the edge direction having more intense values, then the value of the current pixel is kept.

# 3 – Non-Maximum Suppression



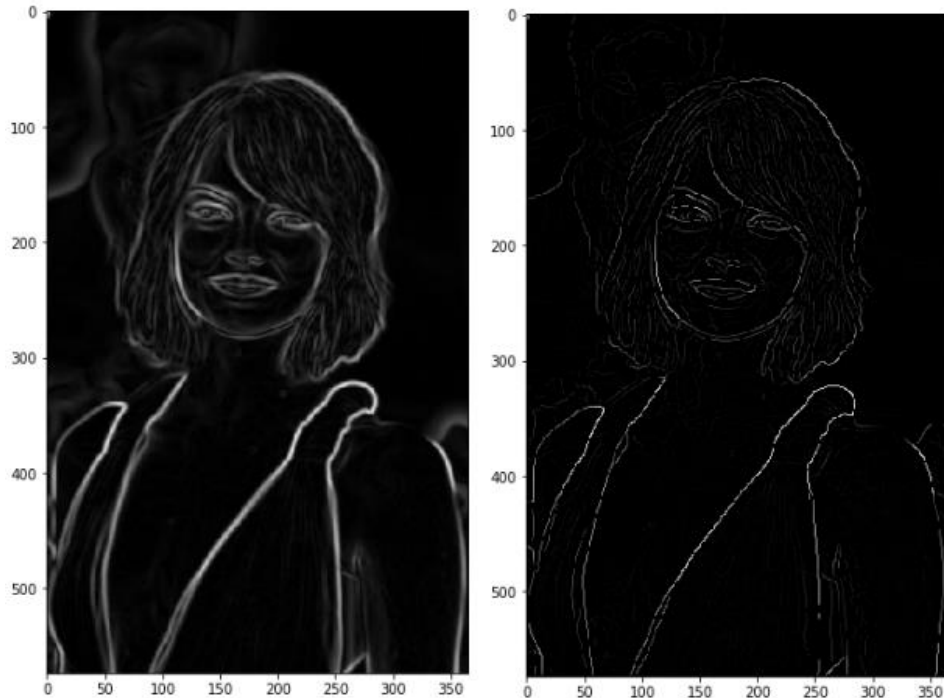
In this case the direction is the orange dotted diagonal line. Therefore, the most intense pixel in this direction is the pixel  $(i-1, j+1)$ .



# 3 – Non-Maximum Suppression

- Each pixel has 2 main criteria (edge direction in radians, and pixel intensity (between 0–255)).
- Based on these inputs the non-max-suppression steps are:
  - Create a matrix initialized to 0 of the same size of the original gradient intensity matrix;
  - Identify the edge direction based on the angle value from the angle matrix;
  - Check if the pixel in the same direction has a higher intensity than the pixel that is currently processed;
  - Return the image processed with the non-max suppression algorithm.

# 3 – Non-Maximum Suppression



The result is the same image with thinner edges. We can however still notice some variation regarding the edges' intensity: some pixels seem to be brighter than others, and we will try to cover this shortcoming with the two final steps.

# 4 – Double Threshold

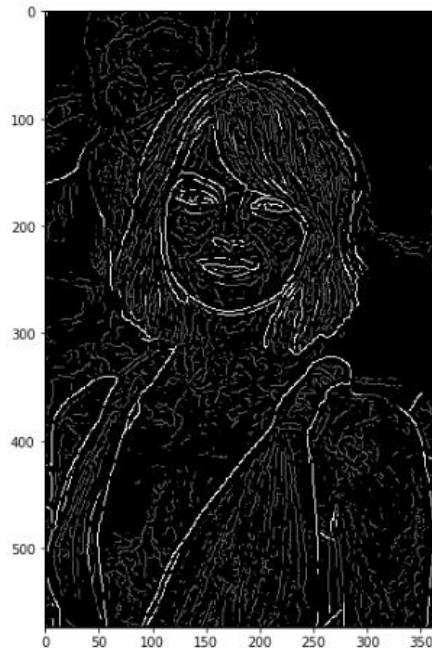
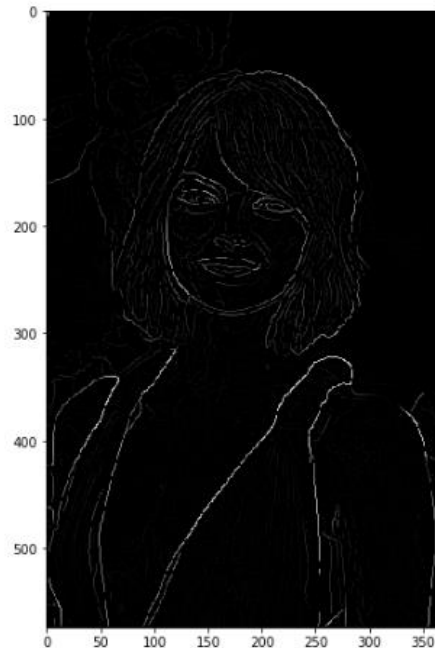
- The double threshold step aims at identifying 3 kinds of pixels: **strong, weak, and non-relevant**:
- Strong pixels are pixels that have an intensity so high that we are sure they contribute to the final edge.
- Weak pixels are pixels that have an intensity value that is not enough to be considered as strong ones, but yet not small enough to be considered as non-relevant for the edge detection.
- Other pixels are considered as non-relevant for the edge.

# 4 – Double Threshold

- The double thresholds holds for:
  - High threshold is used to identify the strong pixels (intensity higher than the high threshold)
  - Low threshold is used to identify the non-relevant pixels (intensity lower than the low threshold)
  - All pixels having intensity between both thresholds are flagged as weak and the Hysteresis mechanism (next step) will help us identify the ones that could be considered as strong and the ones that are considered as non-relevant.

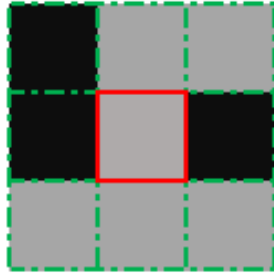
# 4 – Double Threshold

- The result of this step is an image with only 2 pixel intensity values (strong and weak):

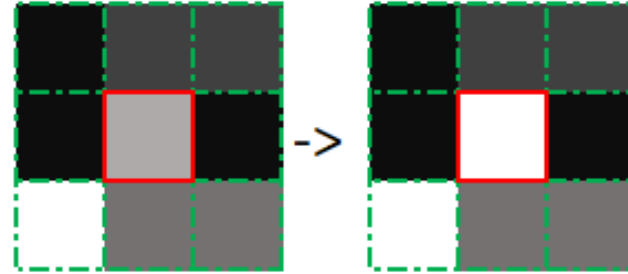


# 5 – Edge Tracking by Hysteresis

Based on the threshold results, the hysteresis consists of transforming weak pixels into strong ones, if and only if at least one of the pixels around the one being processed is a strong one, as described below:

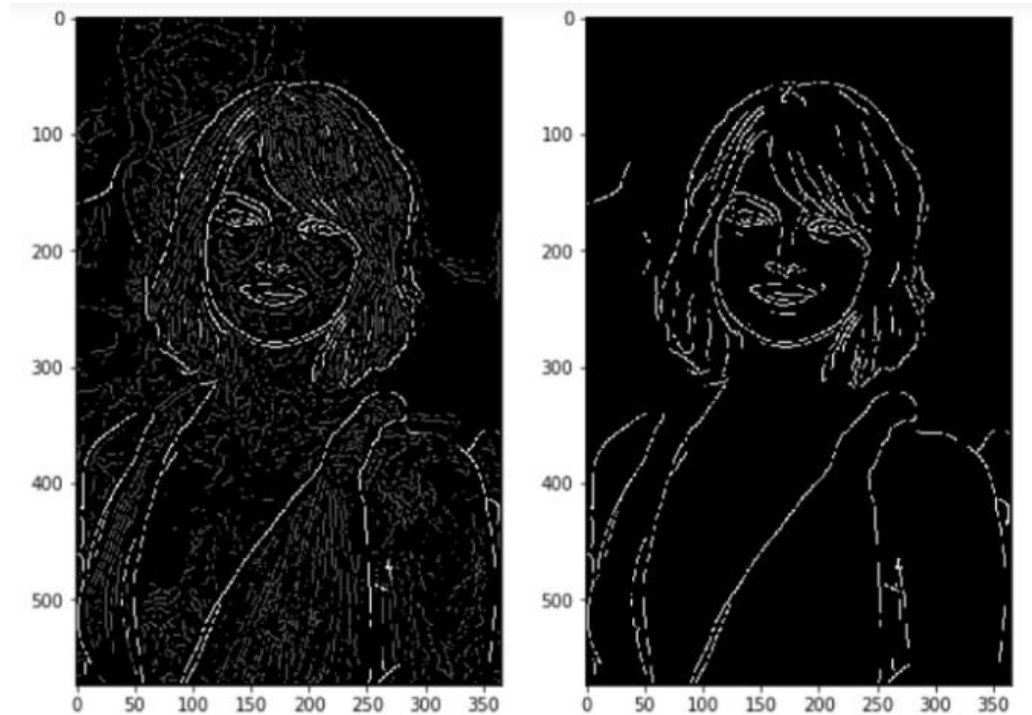


No strong pixels around



One strong pixel around

# 5 – Edge Tracking by Hysteresis



Results of hysteresis process

# Corner Detection

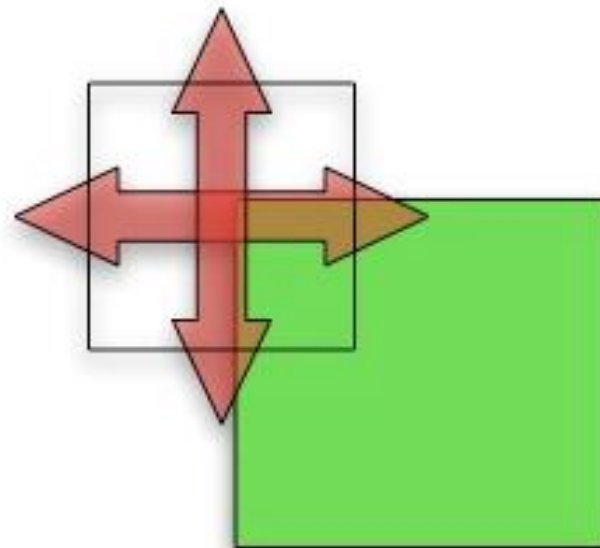
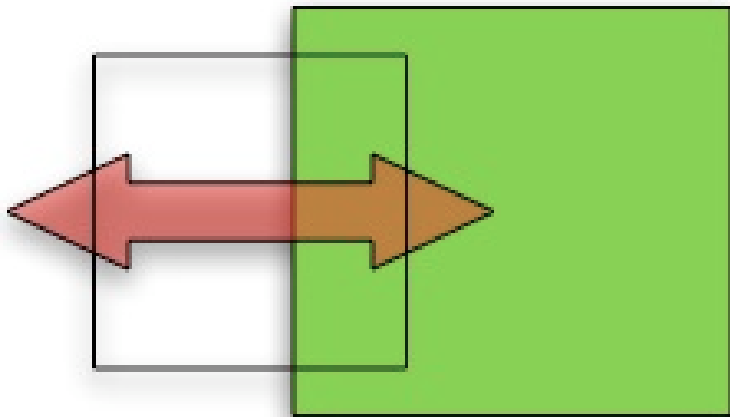


# Corner Detection

- Corners are very characteristic points of images.
- Intuitively, they are characterized by strong two or multidirectional signal variations.
- Because of this feature, corner points are highly discriminative and are often used for image matching or object detection.
- However, many types of corners can be defined, and there are many methods for their detection in digital images.

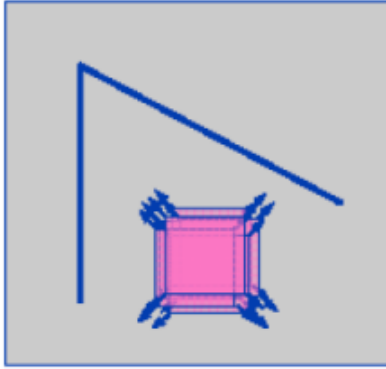
# How it works

- Corner detection works on the principle that if you place a small window over an image, if that window is placed on a corner then if it is moved in any direction there will be a large change in intensity. This is illustrated below with some diagrams.
- If the window is over a flat area of the image then there will be obviously be no intensity change when the window moves. If the window is over an edge there will only be an intensity change if the window moves in one direction.

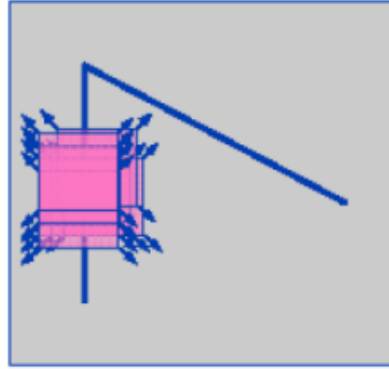


- If the window is over a corner then there will be a change in all directions, and therefore we know there must be a corner.
- The Harris corner detector, demonstrated above, measures the strength of detected corners, and only marks those above a given strength as actual corners.

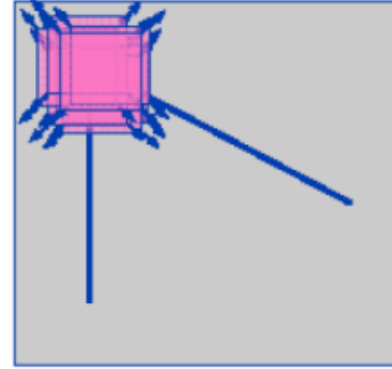
# Harris Corner Detector: Basic Idea



**Flat region:** no intensity change in all directions



**Edge:** no change along edge directions

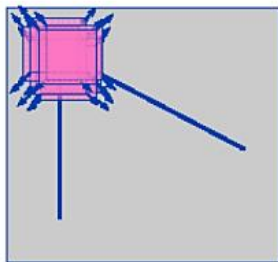


**Corner:** significant intensity change in many directions

- Harris corner detector gives mathematical approach for determining which case holds

# Harris Detector: The Mathematics

- Shift patch by  $[u,v]$  and compute change in intensity
- Change of intensity for shift  $[u,v]$



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

Window  
function

Shifted  
intensity

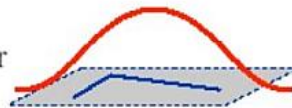
Intensity

Window function  $w(x, y) =$



1 in window, 0 outside

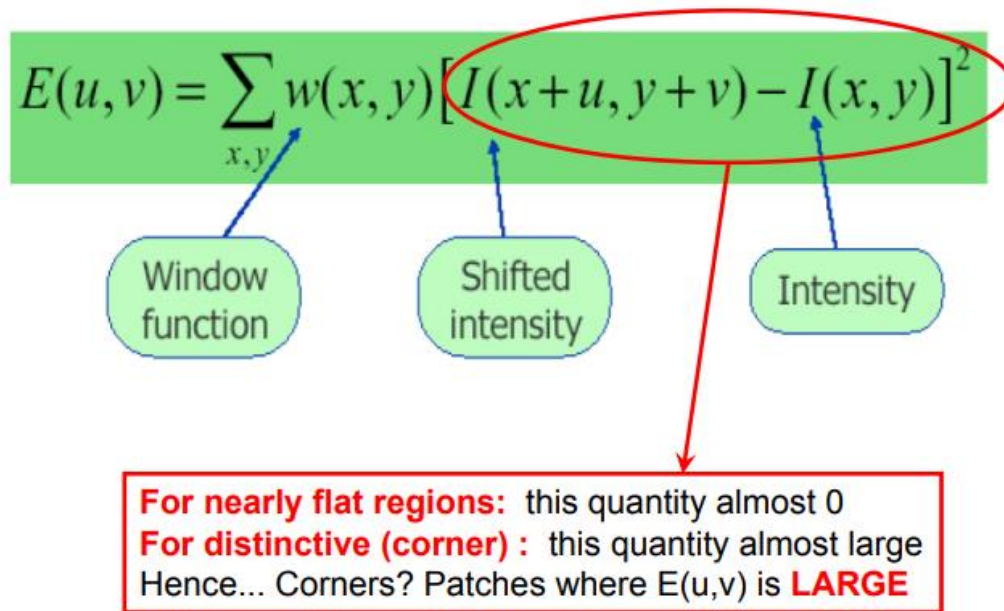
or



Gaussian

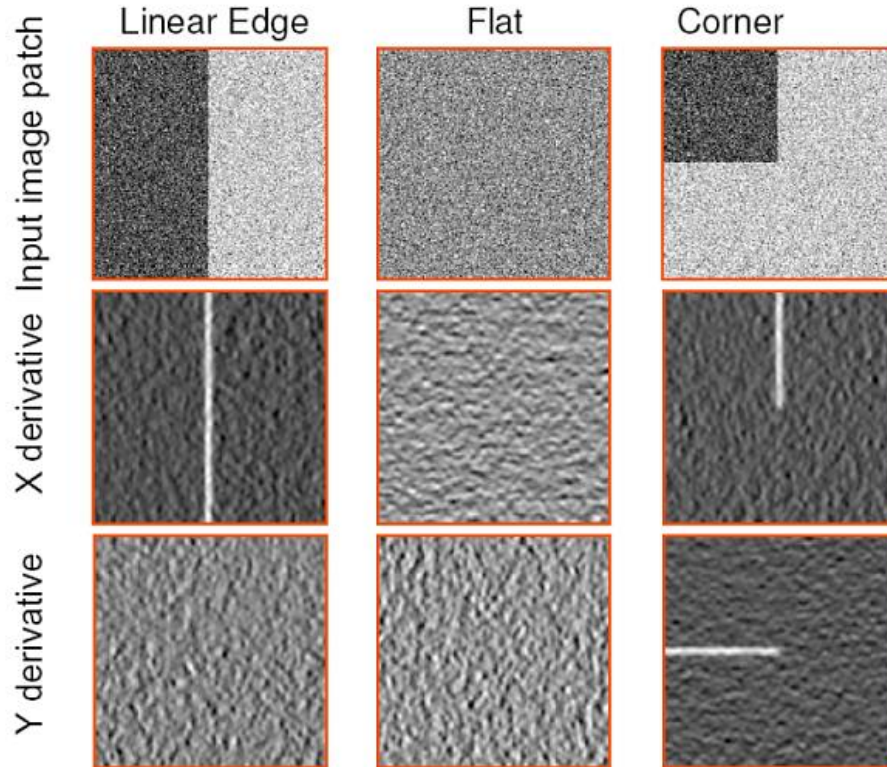
# Harris Detector: The Intuition

- Change of intensity for shift  $[u,v]$



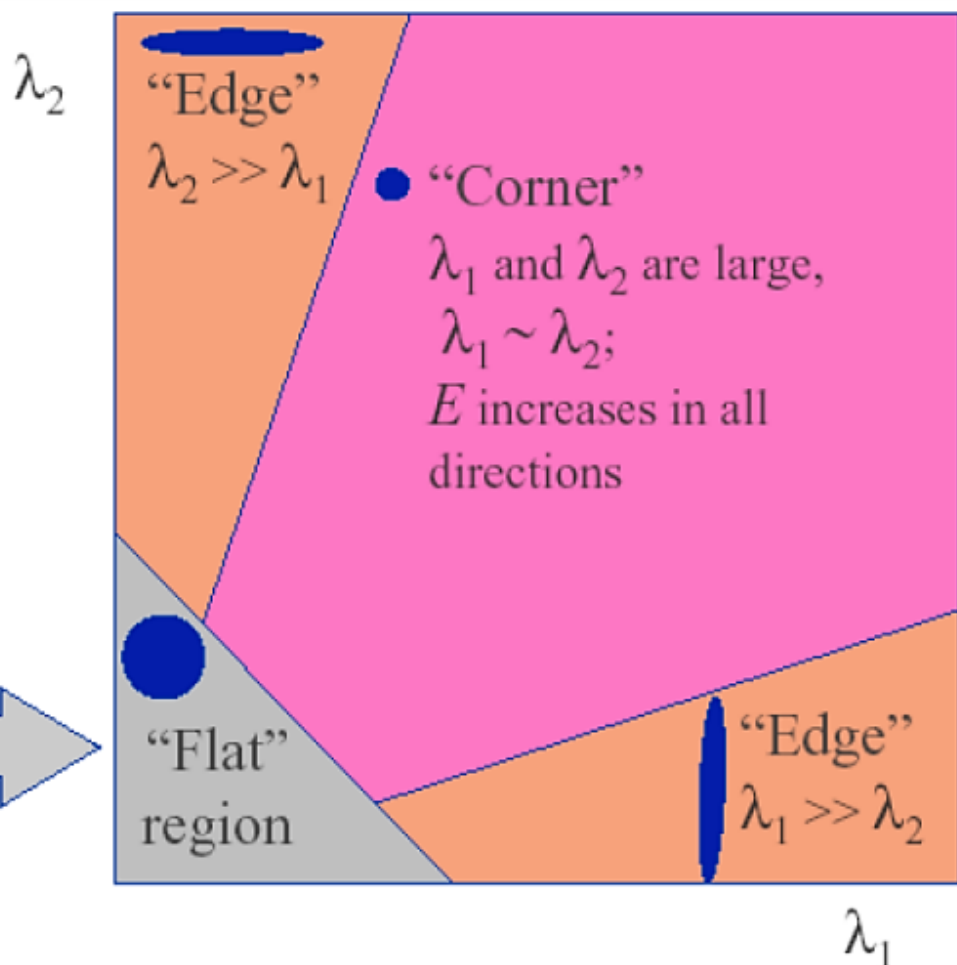
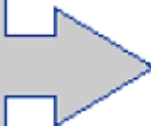
# Harris Corner Detection Intuition

## Example: Cases and 2D Derivatives



Classification of  
image points using  
eigenvalues of  $M$ :

$\lambda_1$  and  $\lambda_2$  are small;  
 $E$  is almost constant  
in all directions





# Harris Corner Detector

- $$\bar{M}' = \begin{pmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{pmatrix} \quad \lambda_{1,2} = \frac{\text{trace}(\bar{M})}{2} \pm \sqrt{\left(\frac{\text{trace}(\bar{M})}{2}\right)^2 - \det(\bar{M})}$$
$$= \frac{1}{2} \left( \bar{A} + \bar{B} \pm \sqrt{\bar{A}^2 - 2\bar{A}\bar{B} + \bar{B}^2 + 4\bar{C}^2} \right)$$

- Eigenvalues  $\lambda_1, \lambda_2$  encode **edge strength**
  - Flat (uniform regions of image)  $\bar{M} = 0$ . Therefore  $\lambda_1 = \lambda_2 = 0$
  - For ideal ramp,  $\lambda_1 > 0, \lambda_2 > 0$
- Associated eigenvectors represent **edge orientation**
- A corner should have:
  - Strong edge in main direction (corresponding to larger of  $\lambda_1, \lambda_2$ )

# Harris Corner Detector

- $$\lambda_{1,2} = \frac{\text{trace}(\bar{M})}{2} \pm \sqrt{\left(\frac{\text{trace}(\bar{M})}{2}\right)^2 - \det(\bar{M})}$$
$$= \frac{1}{2} \left( \bar{A} + \bar{B} \pm \sqrt{\bar{A}^2 - 2\bar{A}\bar{B} + \bar{B}^2 + 4\bar{C}^2} \right)$$
- For corner, second edge must be significant (corresponds to  $\text{trace}(\bar{M})/2 - \sqrt{\dots}$ )
- Function for corner detection:

$$\bar{M}' = \begin{pmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{pmatrix}$$

$$Q(u, v) = \det(\bar{M}) - \alpha \cdot (\text{trace}(\bar{M}))^2$$
$$= (\bar{A}\bar{B} - \bar{C}^2) - \alpha \cdot (\bar{A} + \bar{B})^2$$

$$\det M = \lambda_1 \lambda_2$$
$$\text{trace } M = \lambda_1 + \lambda_2$$

- In practice  $\alpha$  assigned fixed value in range 0.04 – 0.06 (max 0.25)
- Larger values of  $\alpha$  makes detector function less sensitive (fewer

# Harris Corner Detector

- An image location  $(u,v)$  is candidate for corner point when

$$Q(u, v) > t_H$$

- $t_H$  is threshold selected based on image content, typically lies in range 10,000 – 1,000,000
- Detected corners inserted into set and sorted in descending order (i.e.  $q_i \geq q_{i+1}$ ) based on their corner strength

$$Corners = [c_1, c_2, \dots c_N]$$

- Many false corners occur in neighborhood of real corner
- Traverse sorted list, delete false corners towards end of list

# Corner Detection Summary

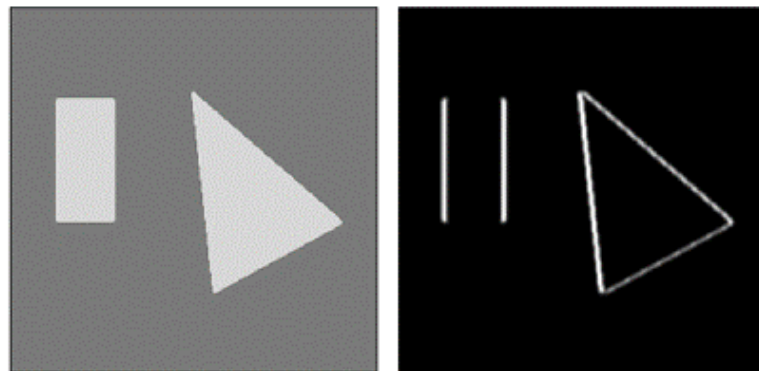
- if the area is a region of constant intensity, both eigenvalues will be very small.
- if it contains an edge, there will be one large and one small eigenvalue (the eigenvector associated with the large eigenvalue will be parallel to the image gradient).
- if it contains edges at two or more orientations (i.e., a corner), there will be two large eigenvalues (the eigenvectors will be parallel to the image gradients).

# Corner Detection Algorithm

Input: image  $f$ , threshold  $t$  for  $\lambda_2$ , size of  $Q$

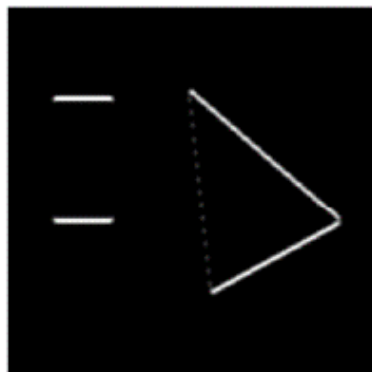
- (1) Compute the gradient over the entire image  $f$
- (2) For each image point  $p$ :
  - (2.1) form the matrix  $C$  over the neighborhood  $Q$  of  $p$
  - (2.2) compute  $\lambda_2$ , the smaller eigenvalue of  $C$
  - (2.3) if  $\lambda_2 > t$ , save the coordinates of  $p$  in a list  $L$
- (3) Sort the list in decreasing order of  $\lambda_2$
- (4) Scanning the sorted list top to bottom: delete all the points that appear in the list that are in the same neighborhood  $Q$  with  $p$

# Harris Corner Detection Example

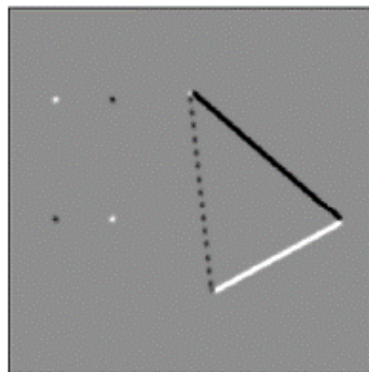


$I(u, v)$

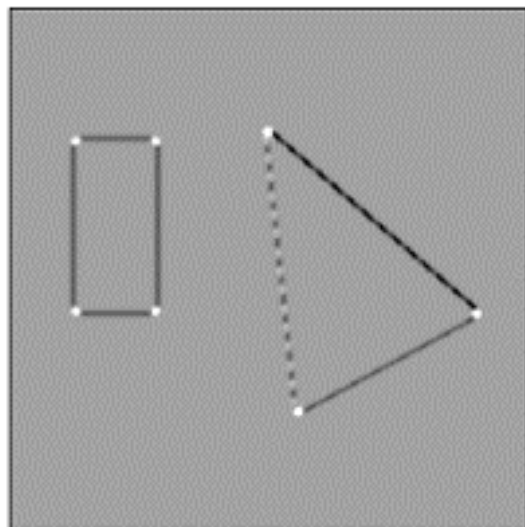
$A = I_x^2(u, v)$



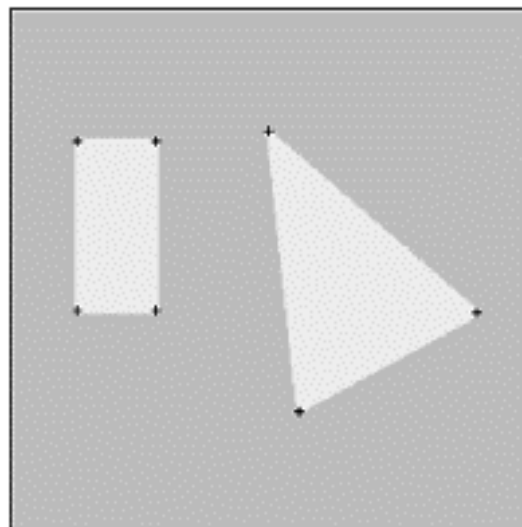
$B = I_y^2(u, v)$



$C = I_x I_y(u, v)$



$Q(u, v)$

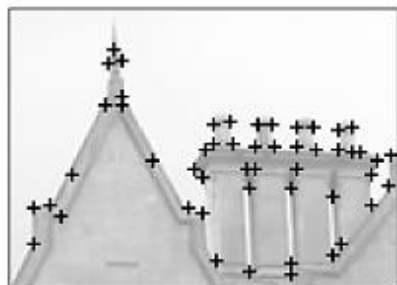


detected corners

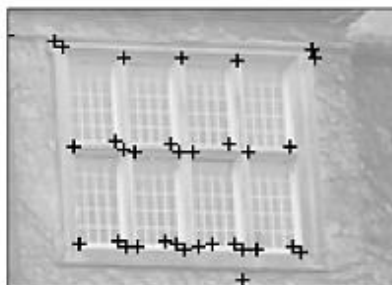


**Image with final  
corner points marked**



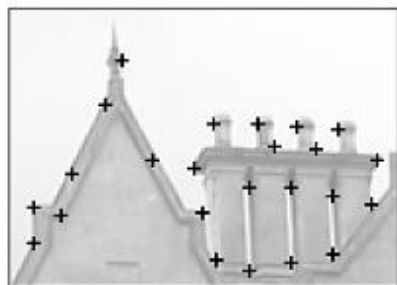


(b)

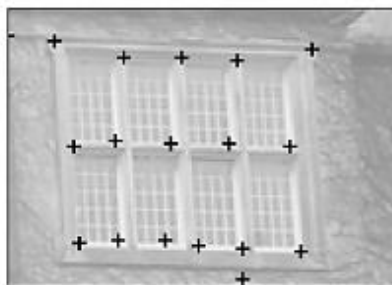


(c)

**Before thresholding**



(d)



(e)

**After thresholding**

# Criteria

1. Detection – a good corner detector should detect all corner points, even the ones that are not characterized by a strong signal response. At the same time it should be insensitive to noise.
2. Localization – corners should be detected and marked in the positions of their true occurrence.
3. Stability – detected points should persist at their locations even on multiple acquisitions under varying conditions or some geometric transformations of the same scene. Stability is often measured by a repeatability measure [373].
4. Speed – it is obvious that the faster the method the better. However, sometimes the speed factor is in opposition to other parameters of a detector, such as good localization for instance.

# Morphological operations

# Morphological Filtering

- Binary images (consists of pixels that can have one of exactly two colors, usually black and white) may contain numerous imperfections.
- In particular, noises and textures can be distorted when the binary regions produced by simple threshold.
- Morphological Operations in Image Processing pursues the goals of removing these imperfections by accounting for the form and structure of the image.

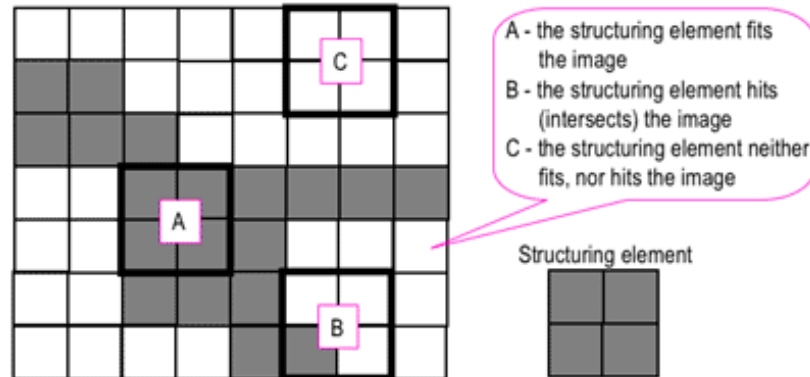
# What Are Morphological Operators?

- Local pixel transformations for processing region shape
- Most often used on binary images
- Logical transformations based on comparison of pixel neighborhoods with a pattern.

# Morphological Filtering

A small shape or template called a structuring element, a matrix that identifies the pixel in the image being processed and defines the neighborhood used in the processing of each pixel is used to probe an image in these Morphological techniques.

It is positioned at all possible locations in the input image and compared with the corresponding neighborhood of pixels.



# Morphological Filtering

- Commonly these structuring elements have odd dimensions and the origin defined as the center of the matrix.
- These are playing in morphological image processing the same role as convolution kernels in linear image filtering.

## **Fundamental operations –**

- Dilation
- Erosion

## **Compound Operations –**

- Opening
- Closing
- Gradient
- Black hat or Bottom hat
- Top hat

# Dilation

**Dilation** (represented by the symbol  $\oplus$ ) -

The assigned structuring element is used for probing and expanding the shapes contained in the input image. In Specific, it acts like **local maximum filter**.

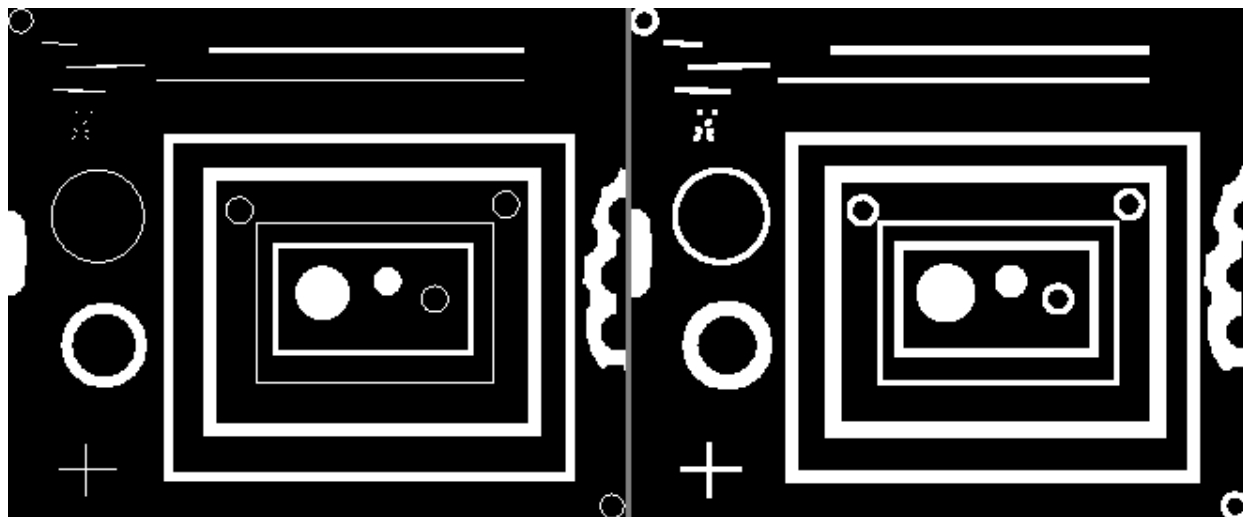
It adds a layer of pixels to both the inner and outer boundaries of regions.

That is, the value of the output pixel is the **maximum value of all pixels in the neighborhood**.

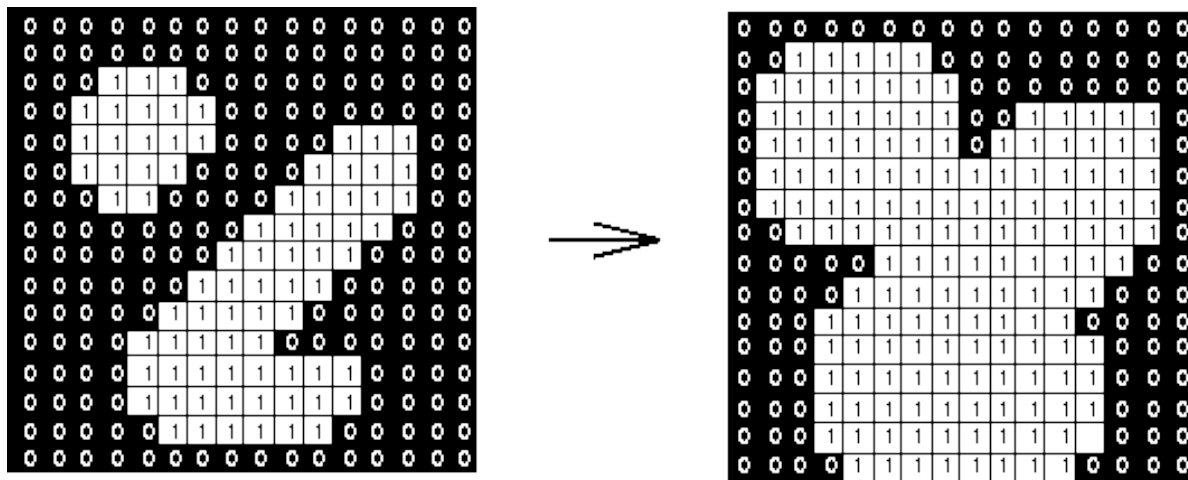
In a binary image, a pixel is set to 1 if any of the neighboring pixels have the value 1. Morphological dilation makes objects more visible and fills in small holes in objects.



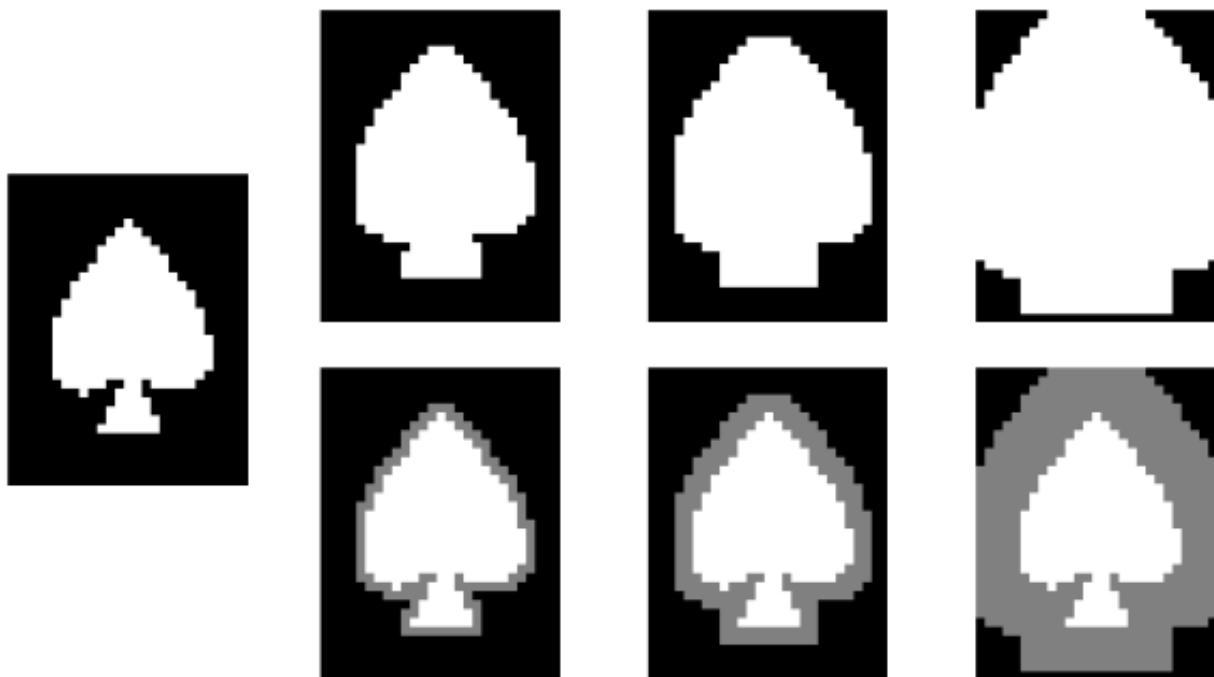
# Dilation



# Dilation



# 8-neighbour dilate



Dilate  $\times 1$

Dilate  $\times 2$

Dilate  $\times 5$

# Erosion

**Erosion** (represented by the symbol  $\ominus$ ) -

Straight opposite to the dilation. The assigned structuring element is used for probing and reducing the shapes contained in the input image. In Specific, it acts like **local minimum filter**.

Also, the structuring elements shrinks an image by stripping away a layer of pixels from both the inner and outer boundaries of regions.

By using erosion, we can **eliminate the holes and gaps** between different regions become larger, and small details.

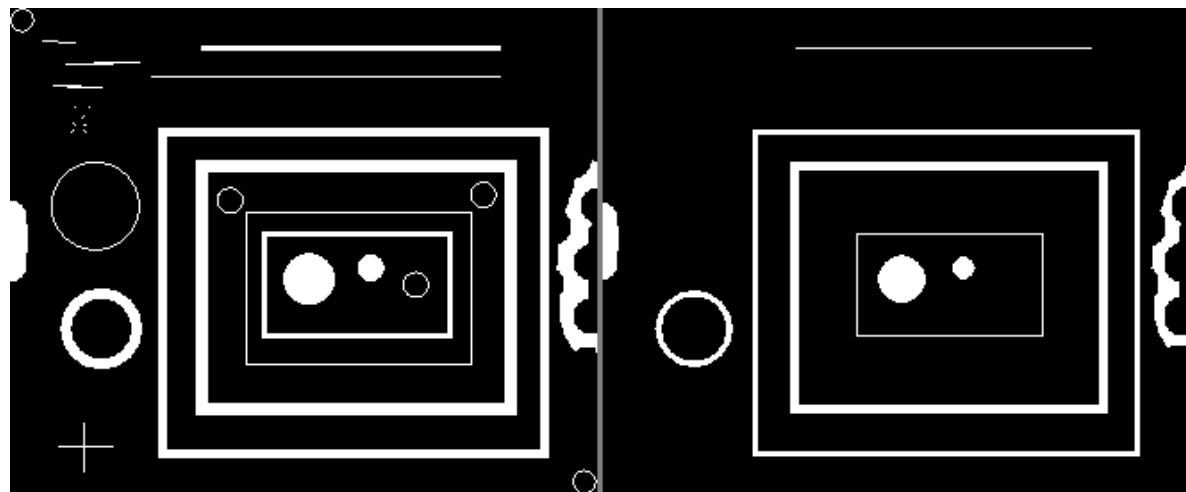
# Erosion

The value of the output pixel is the minimum value of all pixels in the neighborhood.

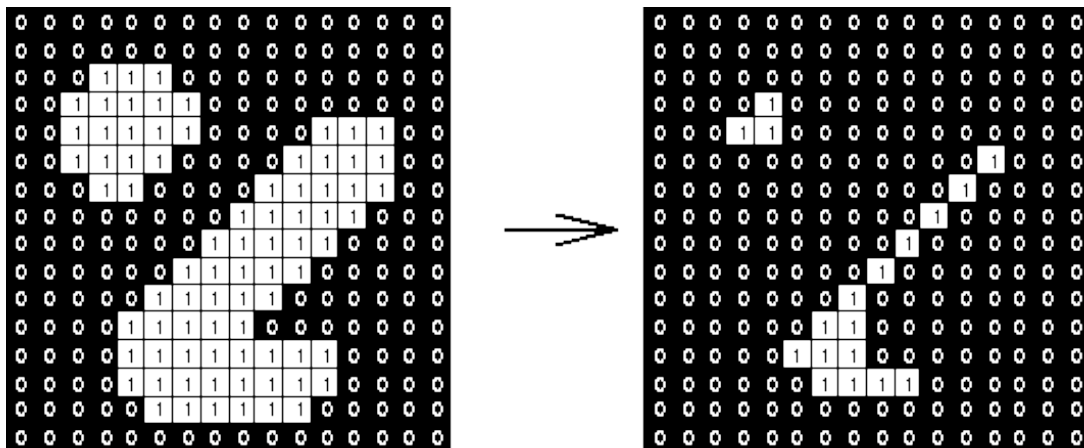
In a binary image, a pixel is set to 0 if any of the neighboring pixels have the value 0.

Morphological erosion removes islands and small objects so that only substantive objects remain.

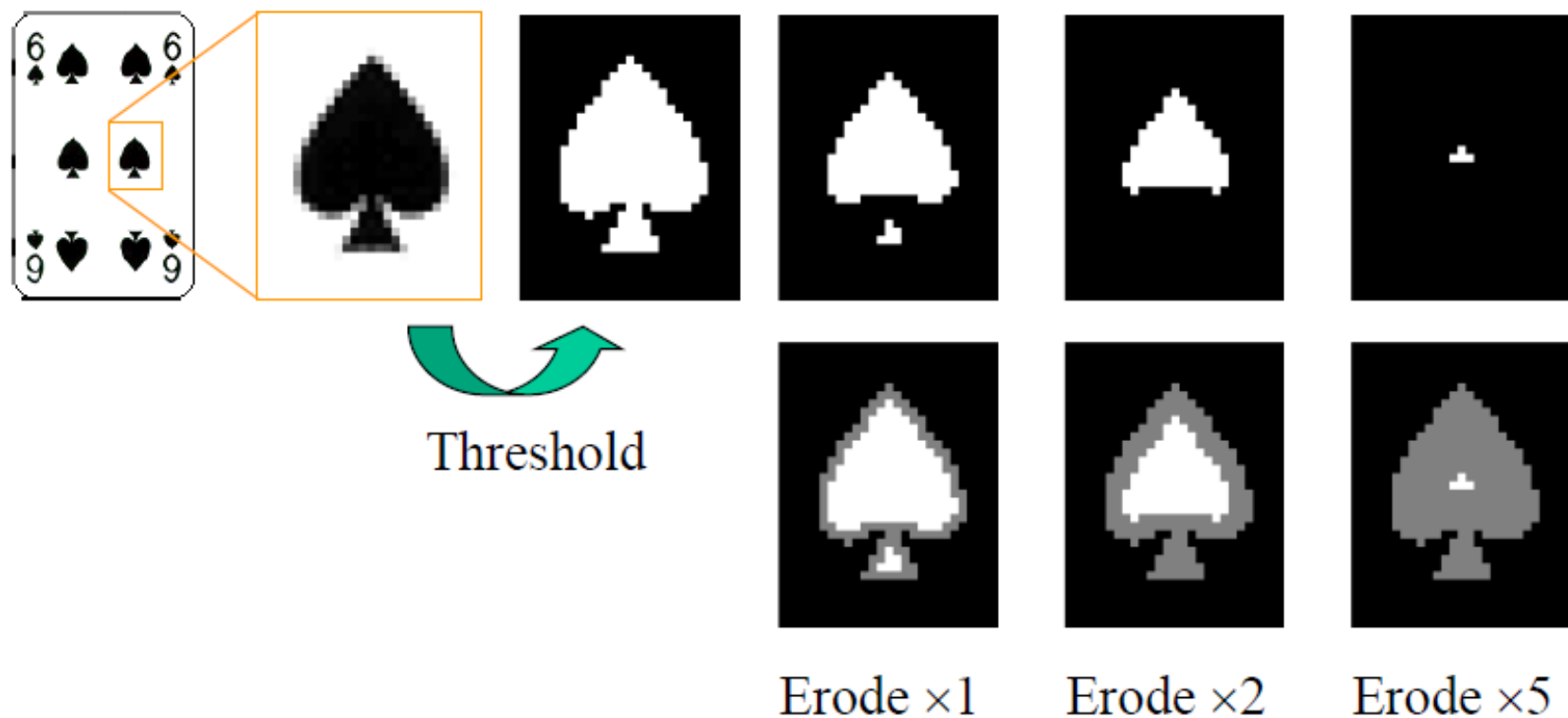
# Erosion



# Erosion



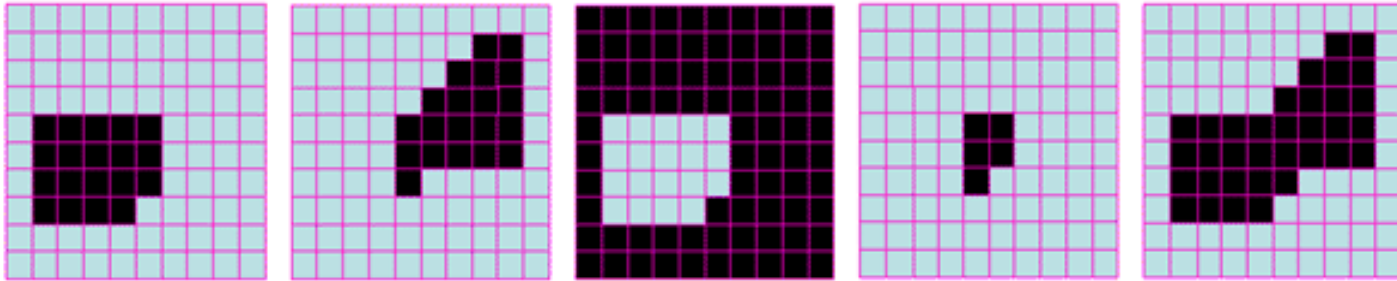
# 8-neighbour erode





# Compound operations

Morphological operations are represented as combinations of erosion, dilation, and simple set-theoretic operations such as the complement of a binary image, intersection and union of two or more binary images.



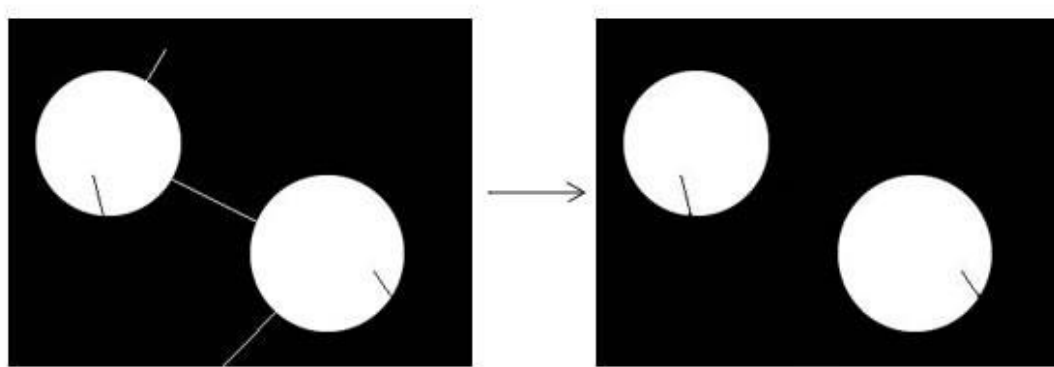
First two at your left are two images, 3rd one is the complement of 1st Image, 4th one is the intersection of 1st two images and the last one at you right most position is the union of the 1st two images

# 1 - Opening

**Opening**  $(A \circ B = (A \ominus B) \oplus B)$  –

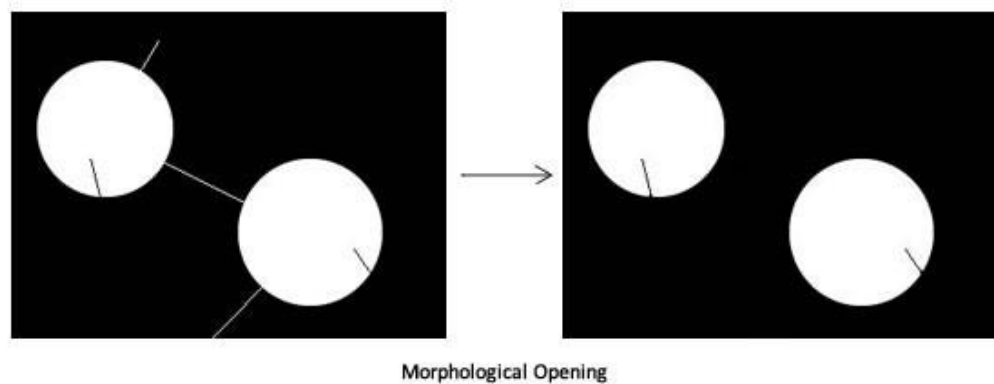
It is achieved by **first eroding** an image and **then dilating** it.

Opening **removes any narrow connections** and lines between two regions.



Morphological Opening

# 1 - Opening



The opening operation erodes an image and then dilates the eroded image, using the same structuring element for both operations.

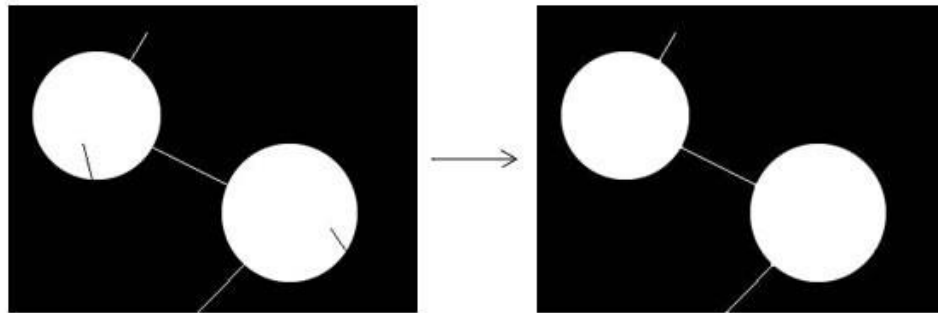
Morphological opening is **useful for removing small objects** from an image while preserving the shape and size of larger objects in the image.

## 2 - Closing

**Closing**  $(A \bullet B = (A \oplus B) \ominus B)$  -

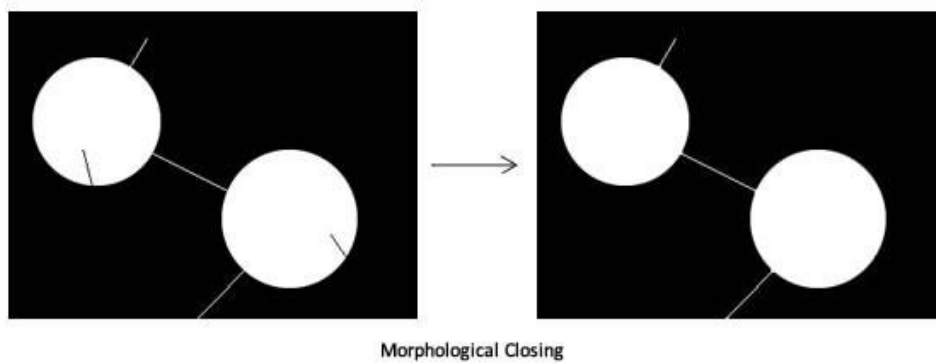
It is done by **first dilating** the image and **then eroding** it.

The order is the reverse of opening. Closing **fills up any narrow black regions or holes** in the image.



Morphological Closing

## 2 - Closing



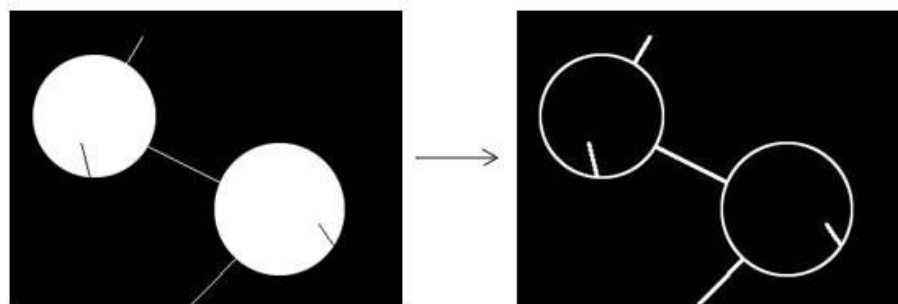
The closing operation dilates an image and then erodes the dilated image, using the same structuring element for both operations.

Morphological closing is **useful for filling small holes from an image** while preserving the shape and size of the objects in the image.

# 3 - Gradient

The difference between the dilation and the erosion of the image. This is used **to find boundaries or edges in an image**.

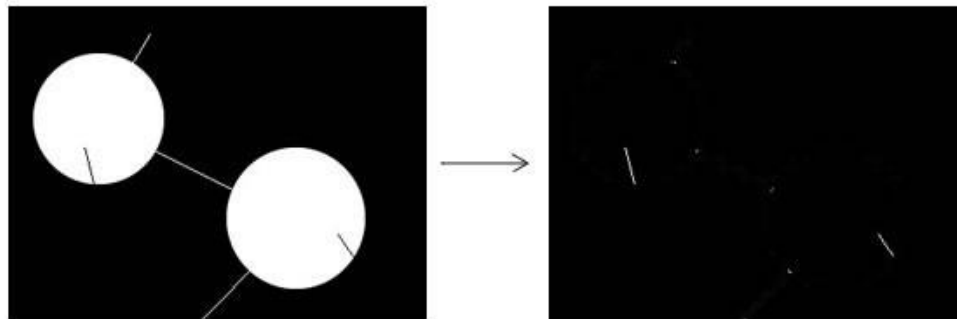
It recommended to apply some filtering before calculating the gradient because it is **very sensitive to noise**.



Morphological Gradient

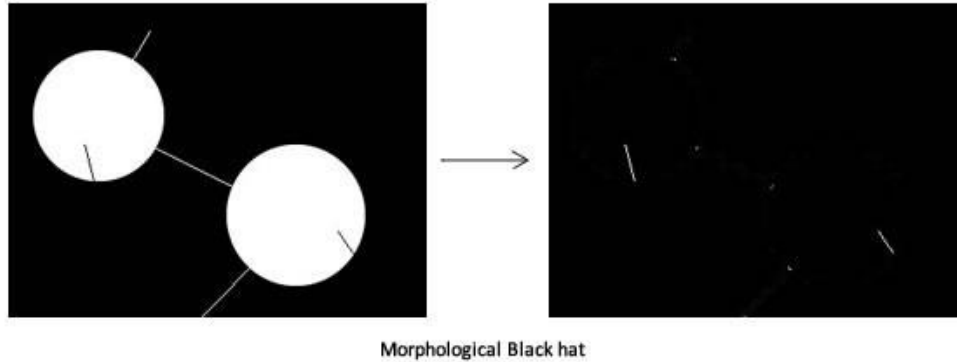
## 4 - Black hat or Bottom hat

The **difference** between the **closing** of an image and the input **image** itself. This **highlights the narrow black regions** in the image.



Morphological Black hat

## 4 - Black hat or Bottom hat



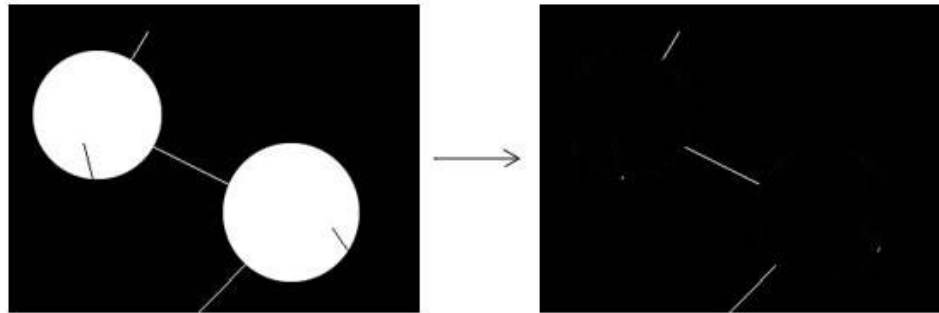
Morphological Black hat

The bottom hat transform closes an image, then subtracts the original image from the closed image. The bottom hat transform can be **used to find intensity troughs** in a grayscale image.



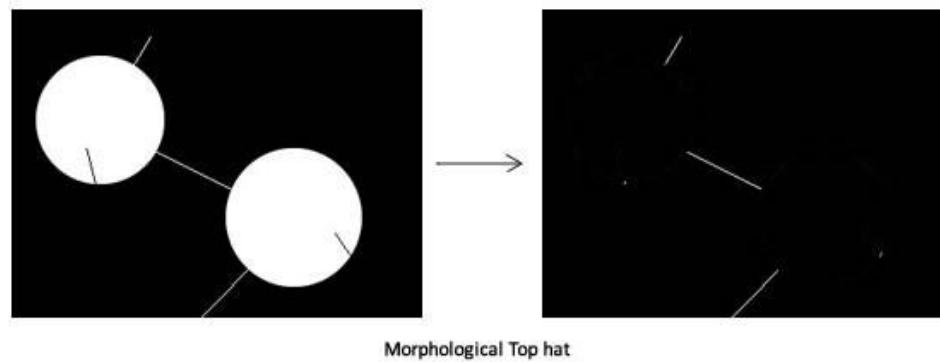
## 5 – Top hat

The difference of the source or input image and the opening of the source or input image. It highlights the narrow pathways between different regions.



Morphological Top hat

## 5 – Top hat



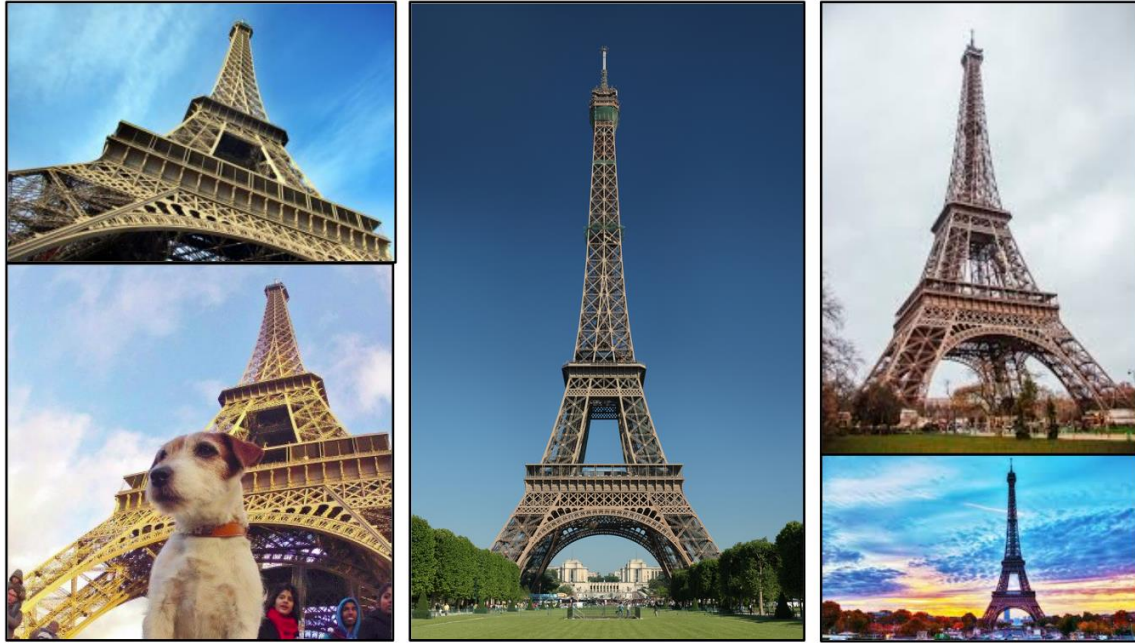
The top hat transform opens an image, then subtracts the opened image from the original image.

The top hat transform can be **used to enhance contrast in a grayscale image with non uniform illumination**. The transform can also isolate small bright objects in an image.

# SIFT descriptors

# Introduction

- Take a look at the below collection of images and think of the common element between them:



- The resplendent Eiffel Tower, of course! The keen-eyed among you will also have noticed that each image has a different background, is captured from different angles, and also has different objects in the foreground (in some cases).
- I'm sure all of this took you a fraction of a second to figure out. It doesn't matter if the image is rotated at a weird angle or zoomed in to show only half of the Tower. This is primarily because you have seen the images of the Eiffel Tower multiple times and your memory easily recalls its features. We naturally understand that the scale or angle of the image may change but the object remains the same.

- But machines have an almighty struggle with the same idea. It's a challenge for them to identify the object in an image if we change certain things (like the angle or the scale). Here's the good news – machines are super flexible and we can teach them to identify images at an almost human-level.
- This is one of the most exciting aspects of working in [computer vision](#)!
- So, we will talk about an image matching algorithm that identifies the key features from the images and is able to match these features to a new image of the same object

# Introduction to SIFT

- SIFT, or Scale Invariant Feature Transform, is a feature detection algorithm in Computer Vision.
- SIFT helps locate the local features in an image, commonly known as the 'keypoints' of the image. These keypoints are scale & rotation invariant that can be used for various computer vision applications, like image matching, object detection, scene detection, etc.
- We can also use the keypoints generated using SIFT as features for the image during model training. The major advantage of SIFT features, over edge features or hog features, is that they are not affected by the size or orientation of the image.



Smaller Image of Eiffel Tower



Rotated Image of Eiffel Tower



- For example, here is another image of the Eiffel Tower along with its smaller version. The keypoints of the object in the first image are matched with the keypoints found in the second image. The same goes for two images when the object in the other image is slightly rotated. Amazing, right?
- Let's understand how these keypoints are identified and what are the techniques used to ensure the scale and rotation invariance.



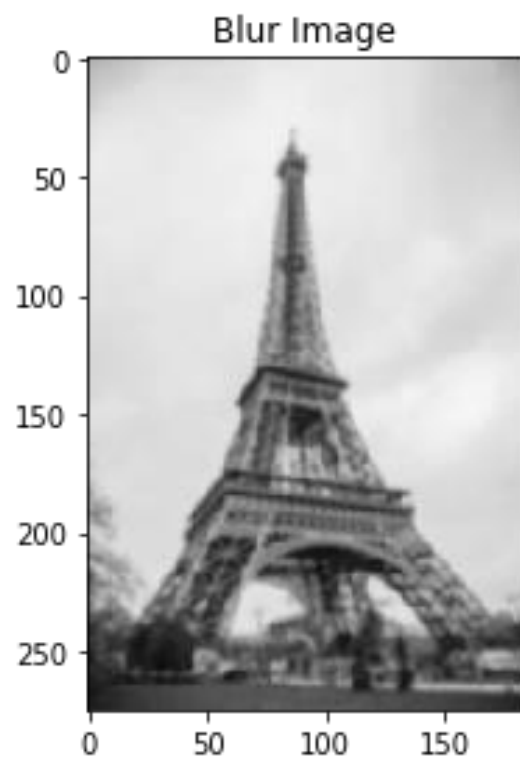
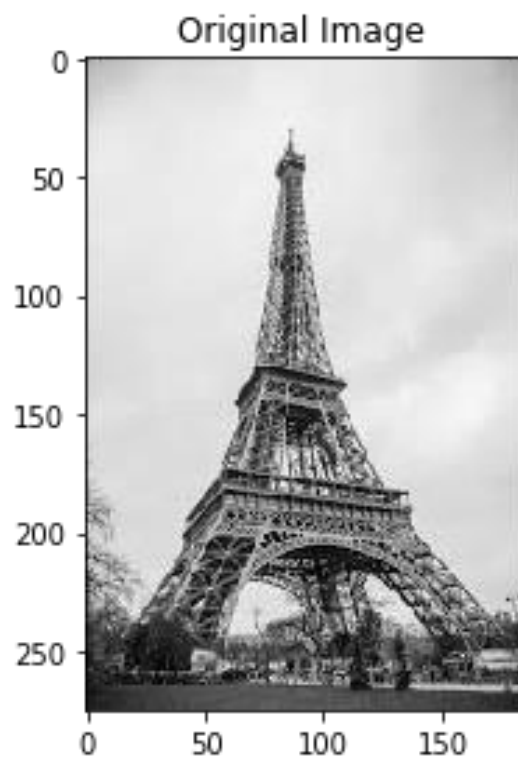
Broadly speaking, the entire process can be divided into 4 parts:

- **Constructing a Scale Space:** To make sure that features are scale-independent
- **Keypoint Localisation:** Identifying the suitable features or keypoints
- **Orientation Assignment:** Ensure the keypoints are rotation invariant
- **Keypoint Descriptor:** Assign a unique fingerprint to each keypoint

Finally, we can use these keypoints for feature matching!

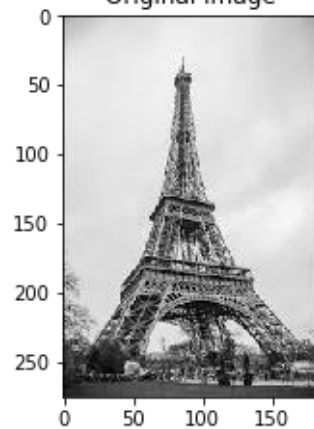
# Constructing the Scale Space

- We need to identify the most distinct features in a given image while ignoring any noise. Additionally, we need to ensure that the features are not scale-dependent. These are critical concepts so let's talk about them one-by-one.
- We use the Gaussian Blurring technique to reduce the noise in an image.
- So, for every pixel in an image, the Gaussian Blur calculates a value based on its neighboring pixels. Below is an example of image before and after applying the Gaussian Blur. 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:

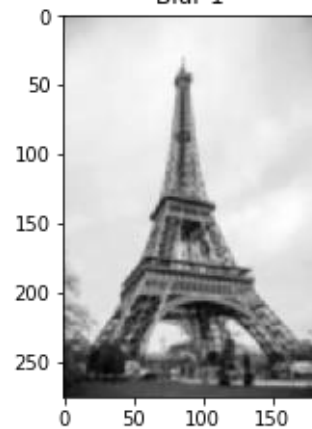


- Gaussian Blur successfully removed the noise from the images and we have highlighted the important features of the image. Now, we need to ensure that these features must not be scale-dependent. This means we will be searching for these features on multiple scales, by creating a 'scale space'.
- Scale space is a collection of images having different scales, generated from a single image.
- Hence, these blur images are created for multiple scales. To create a new set of images of different scales, we will take the original image and reduce the scale by half. For each new image, we will create blur versions as we saw above.
- Here is an example to understand it in a better manner. We have the original image of size (275, 183) and a scaled image of dimension (138, 92). For both the images, two blur images are created:

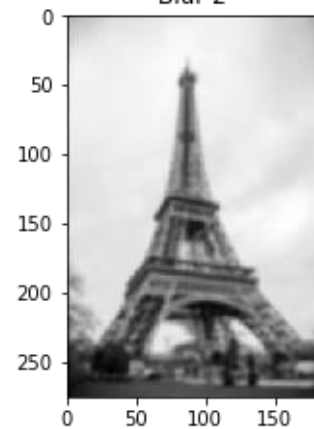
Original image



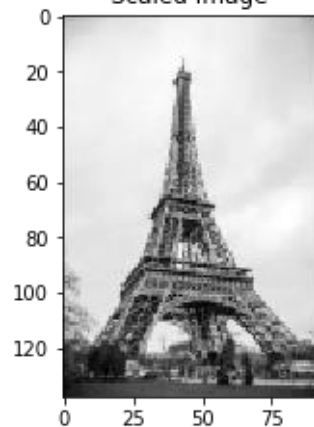
Blur 1



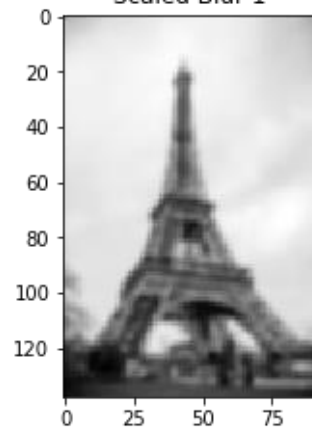
Blur 2



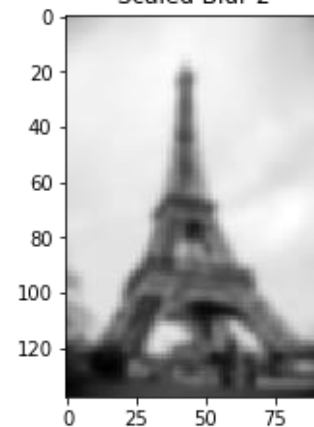
Scaled Image



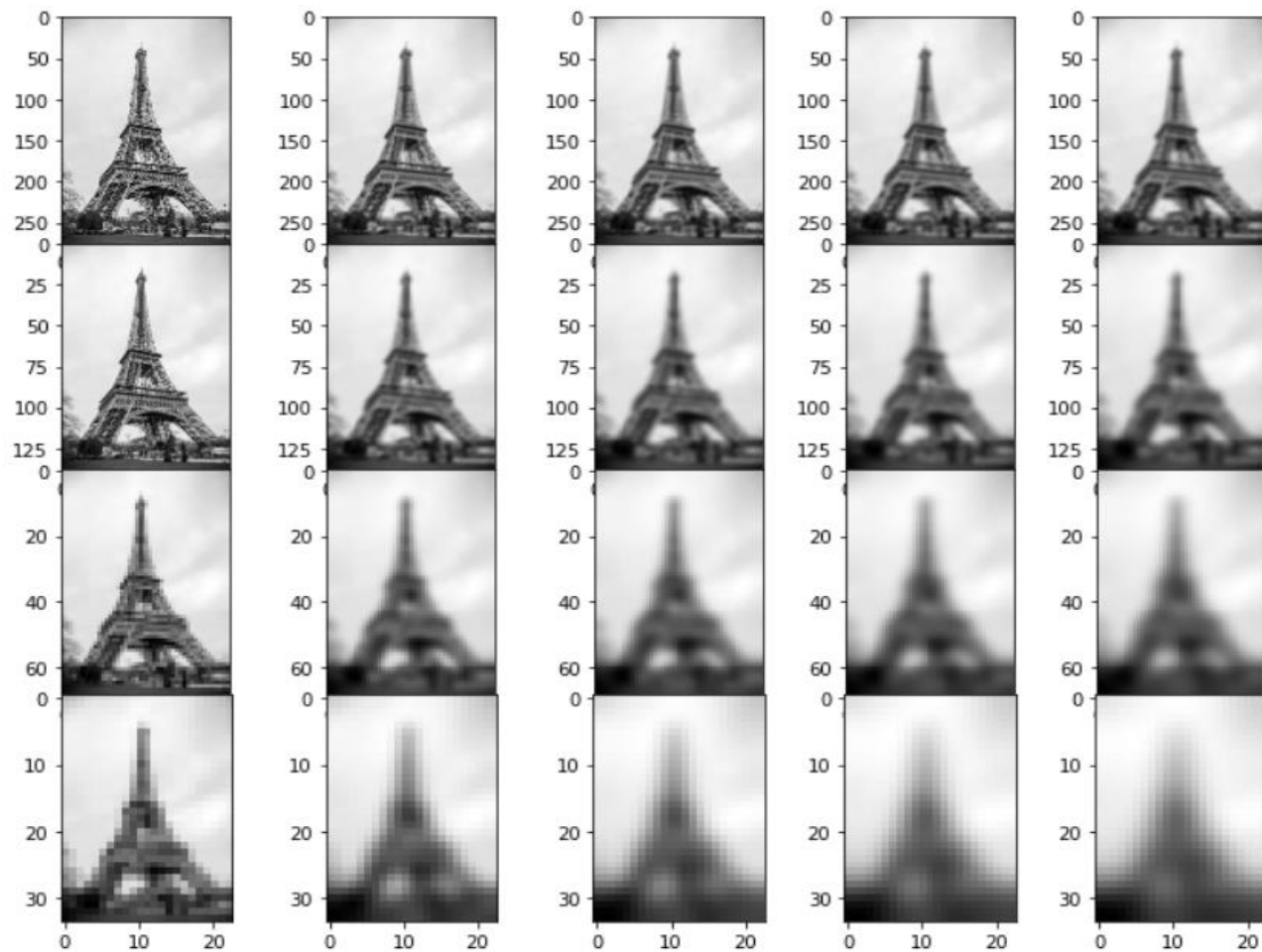
Scaled Blur 1



Scaled Blur 2



- You might be thinking – how many times do we need to scale the image and how many subsequent blur images need to be created for each scaled image? **The ideal number of octaves should be four**, and for each octave, the number of blur images should be five.



First Octave

Second Octave

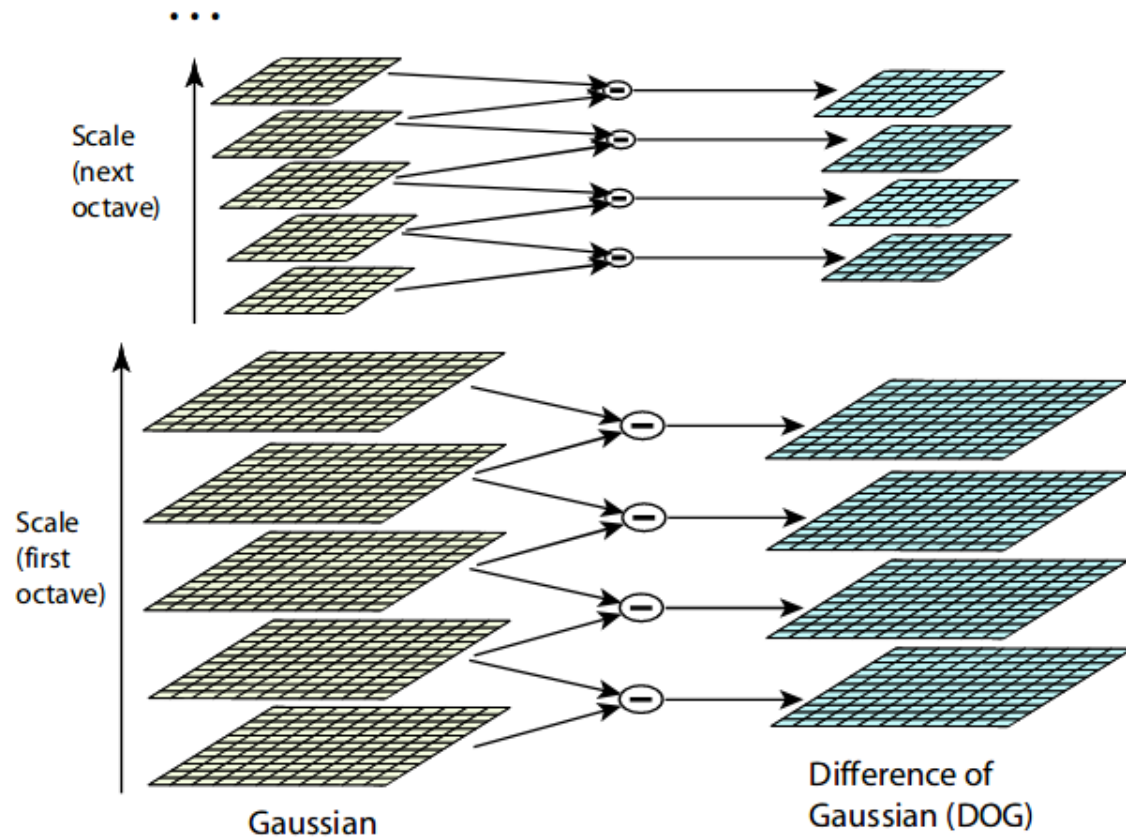
Third Octave

Fourth Octave

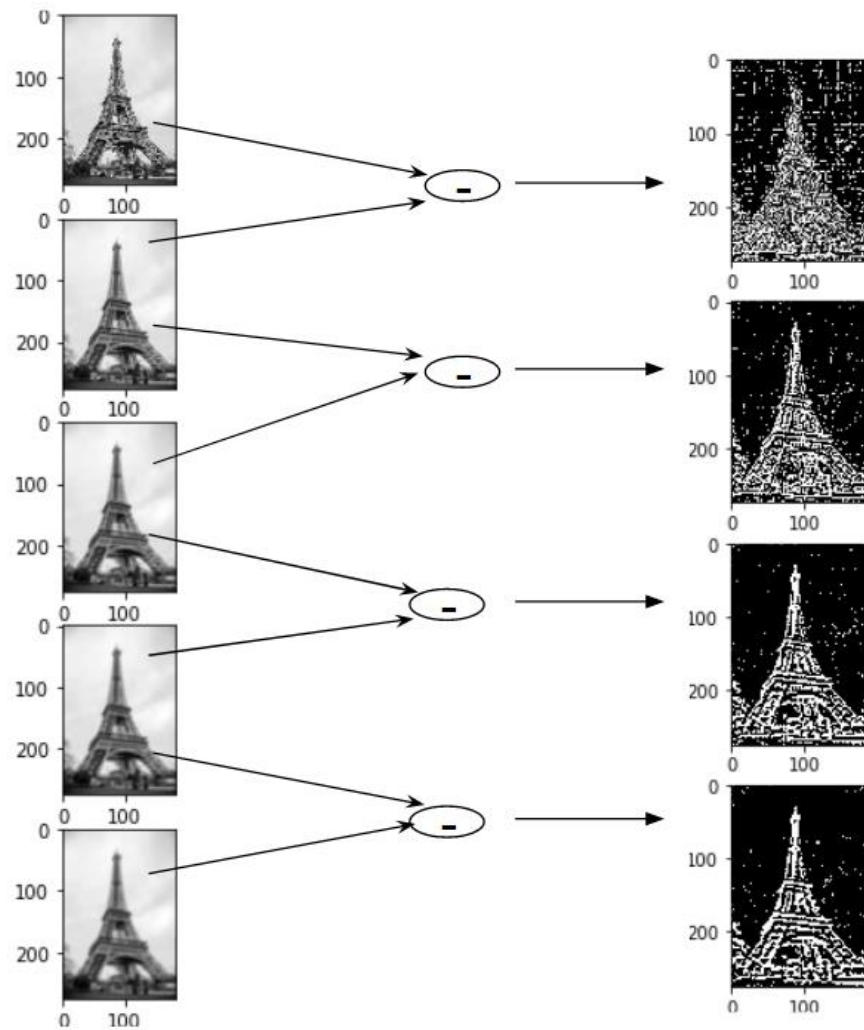
# Difference of Gaussian

- So far we have created images of multiple scales (often represented by  $\sigma$ ) and used Gaussian blur for each of them to reduce the noise in the image. Next, we will try to enhance the features using a technique called Difference of Gaussians or DoG.
- Difference of Gaussian is a feature enhancement algorithm that involves the subtraction of one blurred version of an original image from another, less blurred version of the original.
- DoG creates another set of images, for each octave, by subtracting every image from the previous image in the same scale. Here is a visual explanation of how DoG is implemented:





- Let us create the DoG for the images in scale space. Take a look at the below diagram. 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. The results are jaw-dropping!



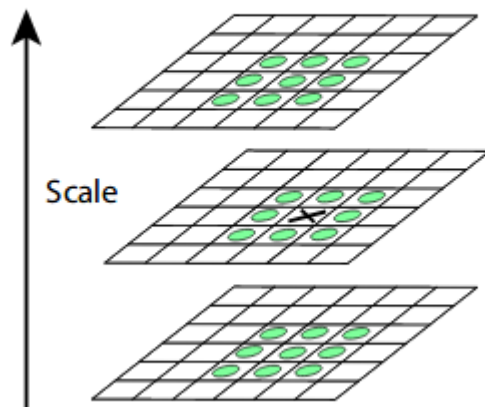
- We have enhanced features for each of these images. Note that here I am implementing it only for the first octave but the same process happens for all the octaves.
- Now that we have a new set of images, we are going to use this to find the important keypoints.

# Keypoint Localization

- Once the images have been created, the next step is to find the important keypoints from the image that can be used for feature matching. **The idea is to find the local maxima and minima for the images.** This part is divided into two steps:
  1. Find the local maxima and minima
  2. Remove low contrast keypoints (keypoint selection)

# Local Maxima and Local Minima

- To locate the local maxima and minima, we go through every pixel in the image and compare it with its neighboring pixels.
- When I say ‘neighboring’, this 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. 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. We will apply the last check over the selected keypoints to ensure that these are the most accurate keypoints to represent the image.

# Keypoint Selection

- So far we have successfully generated scale-invariant keypoints. But some of these keypoints may not be robust to noise. This is why we need to perform a final check to make sure that we have the most accurate keypoints to represent the image features.
- **Hence, we will eliminate the keypoints that have low contrast, or lie very close to the edge.**
- 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.
- So what do we do about the remaining keypoints? Well, we perform a check to identify the poorly located keypoints. These are the keypoints that are close to the edge and have a high edge response but may not be robust to a small amount of noise. A second-order Hessian matrix is used to identify such keypoints. You may go through the math behind this here.
- Now that we have performed both the contrast test and the edge test to reject the unstable keypoints, we will now assign an orientation value for each keypoint to make the rotation invariant.



# Orientation Assignment

- At this stage, we have a set of stable keypoints for the images. We will now assign an orientation to each of these keypoints so that they are invariant to rotation. We can again divide this step into two smaller steps:
  1. Calculate the magnitude and orientation
  2. Create a histogram for magnitude and orientation

# Calculate Magnitude and Orientation

- Consider the sample image shown below:
- Let's say we want to find the magnitude and orientation for the pixel value in red. For this, we will calculate the gradients in x and y directions by taking the difference between 55 & 46 and 56 & 42. This comes out to be  $G_x = 9$  and  $G_y = 14$  respectively.
- Once we have the gradients, we can find the magnitude and orientation using the following formulas:

- Magnitude =  $\sqrt{(G_x)^2 + (G_y)^2} = 16.64$

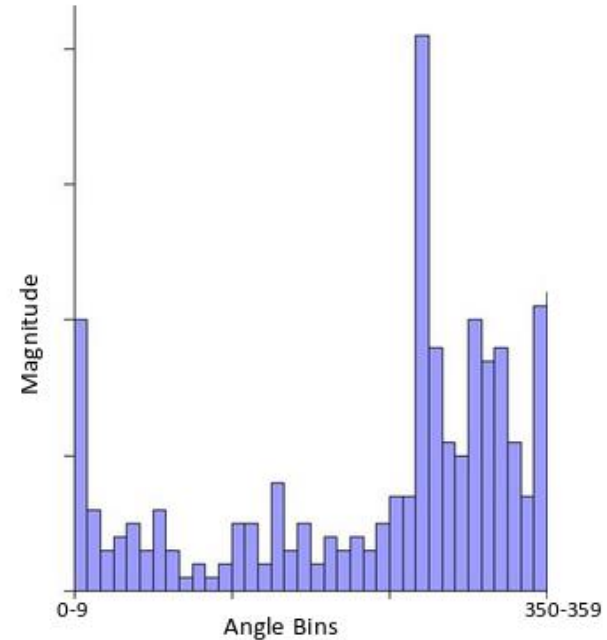
- $\Phi = \text{atan}(G_y / G_x) = \text{atan}(1.55) = 57.17$

35	40	41	45	50
40	40	42	46	52
42	46	50	55	55
48	52	56	58	60
56	60	65	70	75

- The magnitude represents the intensity of the pixel and the orientation gives the direction for the same.
- We can now create a histogram given that we have these magnitude and orientation values for the pixels.

# Creating a Histogram for Magnitude and Orientation

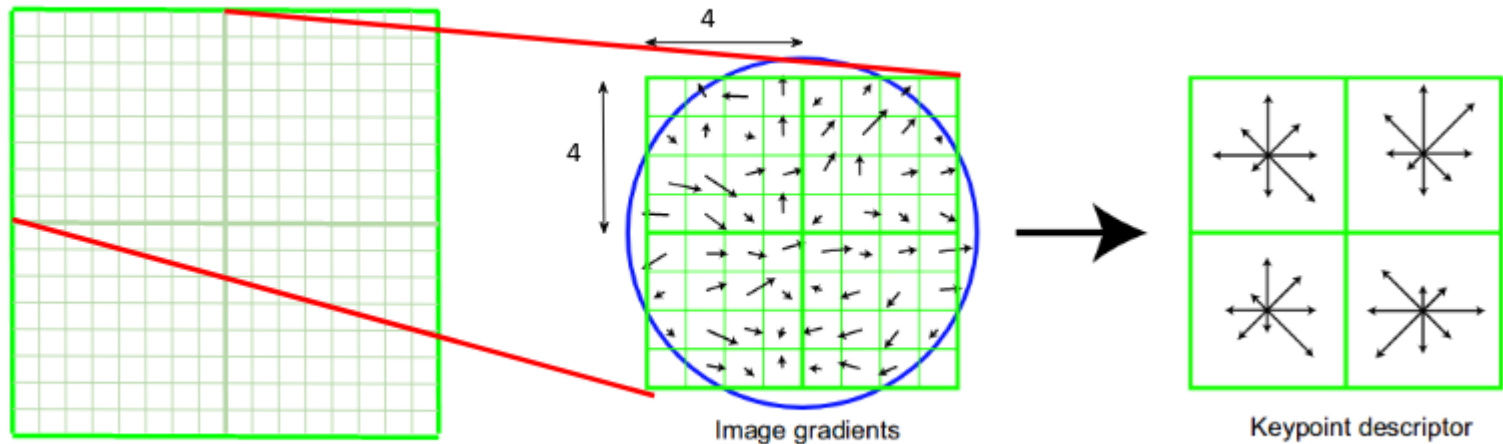
- On the x-axis, we will have bins for angle values, like 0-9, 10 – 19, 20-29, up to 360. Since our angle value is 57, it will fall in the 6th bin. The 6th bin value will be in proportion to the magnitude of the pixel, i.e. 16.64. We will do this for all the pixels around the keypoint.
- This is how we get the below histogram:



- This histogram would peak at some point. **The bin at which we see the peak will be the orientation for the keypoint.** Additionally, if there is another significant peak (seen between 80 – 100%), then another keypoint is generated with the magnitude and scale the same as the keypoint used to generate the histogram. And the angle or orientation will be equal to the new bin that has the peak.
- Effectively at this point, we can say that there can be a small increase in the number of keypoints.

# Keypoint Descriptor

- This is the final step for SIFT. So far, we have stable keypoints that are scale-invariant and rotation invariant. In this section, we will use the neighboring pixels, their orientations, and magnitude, to generate a unique fingerprint for this keypoint called a 'descriptor'.
- Additionally, since we use the surrounding pixels, the descriptors will be partially invariant to illumination or brightness of the images.
- We will first take a  $16 \times 16$  neighborhood around the keypoint. This  $16 \times 16$  block is further divided into  $4 \times 4$  sub-blocks and for each of these sub-blocks, we generate the histogram using magnitude and orientation.



- At this stage, the bin size is increased and we take only 8 bins (not 36). Each of these arrows represents the 8 bins and the length of the arrows define the magnitude. So, we will have a total of 128 bin values for every keypoint.