# OPEN CV Libray :

> Welcome to my resume made by my a Student in Industrial and Automatic Computing engineering and a Computer Vision Enthusiast - Embark on a Journey into Computer Vision! In this extensive course, you'll delve into the intricacies of computer vision using the robust Python OpenCV library.
>
> Whether you're a novice or a seasoned programmer, this course will guide you through the essential principles and methodologies of computer vision, covering topics such as image processing, feature detection, object recognition, and more. Gain practical experience by working on real-world examples and projects, complemented by coding exercises to reinforce your newfound skills.
>
> Throughout the course, you'll harness the power of Python and OpenCV to craft sophisticated computer vision applications. From mastering face detection and object tracking to unraveling the complexities of image segmentation, you'll acquire hands-on expertise. Additionally, learn optimization techniques for enhancing code performance and explore integration with other popular tools and libraries.
>
> If you're eager to elevate your Python programming proficiency and venture into the realm of computer vision, this course is tailored for you! Join me , and let's embark on this exciting learning journey together.

**Course Highlights :**

- **Integrated Development Environment (IDE)** :PYCHARM
- **Libraries used**:CV2,NUMPY,MATPLOTLIB
- **Tasks covered:**

```
1 - Introduction to OpenCV
2 - How to Install OpenCV for Python on Windows 10
3 - How to Read, Write, Show Images in OpenCV
4 - How to Read, Write, Show Videos from Camera in OpenCV
5 - Draw geometric shapes on images using Python OpenCV
6 - Setting Camera Parameters in OpenCV Python
7 - Show Date and Time on Videos using OpenCV Python
8 - Handle Mouse Events in OpenCV
9 - More Mouse Event Examples in OpenCV Python
10 - cv.split, cv.merge, cv.resize, cv.add, cv.addWeighted, ROI
11- Bitwise Operations (bitwise AND, OR, NOT and XOR)
12 - How to Bind Trackbar To OpenCV Windows
13 - Object Detection and Object Tracking Using HSV Color Space
14 - Simple Image Thresholding
15 - Adaptive Thresholding
16 - matplotlib with OpenCV
```

```
17 - Morphological Transformations
18 - Smoothing Images | Blurring Images OpenCV
19 - Image Gradients and Edge Detection
20 - Canny Edge Detection in OpenCV
21 - Image Pyramids with Python and OpenCV
22 - Image Blending using Pyramids in OpenCV
22 - Image Blending using Pyramids in OpenCV
23 - Find and Draw Contours with OpenCV in Python
24 - Motion Detection and Tracking Using Opencv Contours
25 - Detect Simple Geometric Shapes using OpenCV in Python
26 - Understanding image Histograms using OpenCV Python
27 - Template matching using OpenCV in Python
28 - Hough Line Transform Theory
29 - Hough Line Transform using HoughLines method in OpenCV
30 - Probabilistic Hough Transform using HoughLinesP in OpenCV
31 - Road Lane Line Detection with OpenCV (Part 1)
32 - Road Lane Line Detection with OpenCV (Part 2)
33 - Road Lane Line Detection with OpenCV (Part 3)
34 - Circle Detection using OpenCV Hough Circle Transform
35 - Face Detection using Haar Cascade Classifiers
36 - Eye Detection Haar Feature based Cascade Classifiers
37 - Detect Corners with Harris Corner Detector in OpenCV
38 - Detect Corners with Shi Tomasi Corner Detector in OpenCV
39 - How to Use Background Subtraction Methods in OpenCV
40 - Mean Shift Object Tracking
41 - Object Tracking Camshift Method
```

# Digital Image:

**An image may be defined as a two-dimensional function f( x ,y) where x and y are spatial (plane) coordinates, and the amplitude of f at any pair of coordinates (x, y) is called the intensity or gray level of the image at that point. When x, y, and the intensity values of f are all finite, discrete quantities, we call the image a digital image. Note that a digital image is composed of a finite number of elements, each of which has a particular location and value. These elements are called picture elements, image elements, pels, and**

**pixels. Pixel is the term used most widely to denote the elements of a digital image**

# Image Data Type:

**Binary Images: 2-D arrays that assign one numerical value from the set 0 or 1 to each pixel in the image.**

**Intensity or grey-scale images are 2-D arrays that assign one numerical value to each pixel which is representative of the intensity at this point. As discussed previously, the pixel value range is bounded by the bit resolution of the image and such images are stored as N-bit integer images with a given format.The value is typically on a scale from 0 (black) to 255 (white), with shades of gray in between**

**Colored Image:3D array,each pixel is represented by a triplet of values (R, G, B).**

## *Image Enhancement:*

**This technique involves improving the visual quality of an image to make it more presentable and easier to interpret. It includes operations like contrast adjustment, noise reduction, and sharpening.**

## Image Filtering:

**Filtering is a crucial technique for removing unwanted noise and enhancing specific features in an image. Various filters, such as Gaussian blur, median filter, and edge detection filters, are employed for different purposes**

## Feature Extraction:

**Feature extraction involves identifying and quantifying relevant characteristics or patterns within an image. These features, such as edges, corners, or texture patterns, serve as the foundation for image recognition and classification tasks.**

**Pixel (Picture Element): A pixel is the smallest unit in a digital image. It is a single point in the image and can represent a specific color or shade.**

**Grid: The grid is a layout or structure that organizes these pixels in rows and columns. Each intersection of a row and column in the grid corresponds to a specific pixel in the image.**

**—An image with a resolution of 1920x1080 means it has 1920 pixels in width and 1080 pixels in height, arranged in a grid. The total number of pixels in the image is the product of these two dimensions (1920 * 1080 = 2,073,600 pixels).**

- `OPEN CV` **:image processing library created by intel.**
- **Digital image are typically stored into matrix.**
- `NUMPY:` **a highly optimized library for numerical operations.**
- **Digital images are 2D arrays of pixels**
- **All the OpenCV array structures are converted to and_from Numpy arrays.**

Invite de Commande Windows:

## How to Read, Write, Show Images in OpenCV:

-1. Accessing video sources: The primary purpose of `video.capture()` is to establish a connection to a video source, such as a webcam, IP camera, or a file containing a video. It provides a means to access the frames from these sources for further use in your program.

-Retrieving frames: Once the connection to the video source is established, `video.capture()` enables you to retrieve individual frames from the video stream. These frames are essentially snapshots of the video at a specific point in

time and can be utilized for various purposes

cap=cv2.videocapture()

-To save the video `:VideoWriter`

video_writer=cv2.VideoWriter(output_file,fourcc,fp,frame_size)

-5 - Draw geometric shapes on images using Python OpenCV

img=cv2.line(**img**, pt1: Point, pt2: Point, color: Scalar, thickness: int )

img=cv2.arrowedline(**img**, pt1: Point, pt2: Point, color: Scalar, thickness: int) img=cv2.rectangle(**img**, pt1: Point, pt2: Point, color: Scalar, thickness: int )
img=cv2.circle(
**img**, center: Point, radius: int, color: Scalar, thickness: int)

img=cv2.puttext(**img**, text: str, origine: Point, fontFace: int, fontScale(fontsize): float, color: Scalar, thickness: int

-https://github.com/ShawnHymel/computer-vision-with-embedded-machine-learning

- `cap.set()` :the video capture object will modify its properties according to the provided parameter, `cv2.CAP_PROP_FRAME_WIDTH` is the identifier for the frame width property, and `cv2.CAP_PROP_FRAME_HEIGHT` is the identifier for the frame height property.
- - Setting Camera Parameters in OpenCV Python
- - Show Date and Time on Videos using OpenCV Python

  import `datetime`

  dataset=str( `datetime.datetime.now()` )

```
import cv2

cap = cv2.VideoCapture(0)
output_file = 'output.mp4'
fourcc = cv2.VideoWriter_fourcc(*'XVID')
fps = 20
cap.set(3,1000)
cap.set(4,480)
frame_width = int(cap.get(3))
frame_height = int(cap.get(4))
frame_size = (frame_width, frame_height)
video_writer = cv2.VideoWriter(output_file, fourcc, fps, frame_size)

while cap.isOpened():
    ret, frame = cap.read()
    if ret:
        print(cap.get(3))
        print(cap.get(4))
```

```
        video_writer.write(frame)
        cv2.imshow('video', frame)

        if cv2.waitKey(1) == ord('q'):
            break
    else:
        break

cap.release()
cv2.destroyAllWindows()
```

```
-By default, if you do not manually set the frame size, it usually depends on
```

```
In video capture, there are several properties that can be set to control t
he video resolution, frame rate, etc. These properties are identified by nu
meric codes
```

- `Cap.set(3,x)` : `3` refers to the width property
-
`Cap.set(4,y)` : `4` refers to the height property

`-Cap.get(3)` :returns the width

`-Cap.get(4)` :return the height

- `dir(cv2)` : List all the classes,instances in cv2

- Draw geometric shapes on images using Python OpenCV

-The function `str()` in programming is used to convert a value into a string data type.

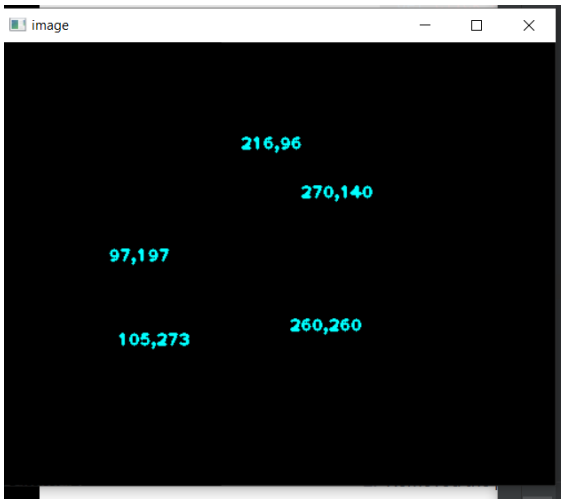
# Handle Mouse Events in OpenCV

— Various constants or flags related to mouse events in programming. Let's break it down and explain each one:

1. **EVENT_FLAG_ALTKEY**: This flag indicates that the Alt key was pressed during a mouse event.
2. **EVENT_FLAG_CTRLKEY:** This flag indicates that the Ctrl key was pressed during a mouse event.
3. **EVENT_FLAG_LBUTTON**: This flag indicates that the left mouse button was pressed during a mouse event.
4. ***EVENT_FLAG_MBUTTON***: This flag indicates that the middle mouse button was pressed during a mouse event.
5. **EVENT_FLAG_RBUTTON**: This flag indicates that the right mouse button was pressed during a mouse event.
6. **EVENT_FLAG_SHIFTKEY**: This flag indicates that the Shift key was pressed during a mouse event.

7. **EVENT_LBUTTONDBLCLK**: This constant represents a double-click event of the left mouse button.
8. **EVENT_LBUTTONDOWN:** This constant represents the pressing down event of the left mouse button.
9. EVENT_LBUTTONUP: This constant represents the releasing event of the left mouse button.
10. EVENT_MBUTTONDBLCLK: This constant represents a double-click event of the middle mouse button.
11. EVENT_MBUTTONDOWN: This constant represents the pressing down event of the middle mouse button.
12. EVENT_MBUTTONUP: This constant represents the releasing event of the middle mouse button.
13. EVENT_MOUSEHWHEEL: This constant represents a horizontal mouse wheel event.
14. EVENT_MOUSEMOVE: This constant represents the movement of the mouse.
15. EVENT_MOUSEWHEEL: This constant represents a vertical mouse wheel event.
16. EVENT_RBUTTONDBLCLK: This constant represents a double-click event of the right mouse button.
17. EVENT_RBUTTONDOWN: This constant represents the pressing down event of the right mouse button.
18. EVENT_RBUTTONUP: This constant represents the releasing event of the right mouse button.

```
import cv2
import numpy as np
events=[i for i in dir(cv2) if 'EVENT' in i]
print(events)

def click_event(event,x,y,flags,par):
    if event==cv2.EVENT_RBUTTONUP:
        print(x, ' , ', y)
        font=cv2.FONT_HERSHEY_PLAIN
        text=str(x)+','+str(y)
        cv2.putText(img,text,(x,y),font,1,(255,255,0),2)
        cv2.imshow('image',img)

img=np.zeros((400,500,3),np.uint8)
cv2.imshow('image',img)
cv2.setMouseCallback('image',click_event)
cv2.waitKey(0)
```




These constants are typically used in programming languages or frameworks that support mouse event handling. By checking these constants in event handling code, developers can identify which mouse button was pressed, released, or double-clicked, and whether any modifier keys (Alt, Ctrl, or Shift) were held during the event.

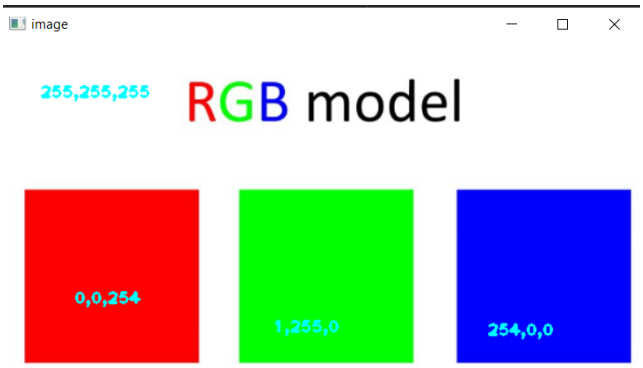
- img=`np.zeros`(n,m,3 ),np.uint8)

⇒Creating an image of size  n.m pixels with 3 channels (RGB) using the NumPy library.

- `img[y, x, 0]` : This **retrieves the blue channel intensity at the pixel** `(x, y)`. In OpenCV, the color channels are ordered as Blue, Green, and Red.
- `img[y, x, 1]` : This retrieves the green channel intensity at the pixel `(x, y)`.
- `img[y, x, 2]` : This retrieves the red channel intensity at the pixel `(x, y)`.

```
import cv2
import numpy as np
events=[i for i in dir(cv2) if 'EVENT' in i]
print(events)

def click_event(event,x,y,flags,par):
    if event==cv2.EVENT_RBUTTONUP:
        blue=img[y,x,0]
        green=img[y,x,1]
        red=img[y,x,2]
        print(x, ' , ', y)
        font=cv2.FONT_HERSHEY_PLAIN
        text=str(blue)+','+str(green)+','+str(red)
        cv2.putText(img,text,(x,y),font,1,(255,255,0),2)
        cv2.imshow('image',img)
path=r'C:\Users\T.M.S\OneDrive\Bureau\OPENCV tuto\chela.png'
img=cv2.imread(path)
cv2.imshow('image',img)
```

```
cv2.setMouseCallback('image',click_event)
cv2.waitKey(0)
```




- `img.shape`:it returns a tuple `(height, width, channels)`

height=img.shape[0]

width=img.shape[1]

- `img.size`: the total number of elements (pixels) in the imagethe total number of elements (pixels) in the image
- b, g, r = cv2.`split`(img)
- cv2.imshow('Blue Channel', b)
  cv2.imshow('Green Channel', g)
  cv2.imshow('Red Channel', r)
- `cv2.add`:

```
img = cv2.imread(path)
img1 = cv2.imread(path1)
img1 = cv2.resize(img1, (img.shape[1], img.shape[0]))

# Adding two images
new_image = cv2.add(img, img1)
cv2.imshow('added', new_image)
```

- `cv2.addWeighted()`:

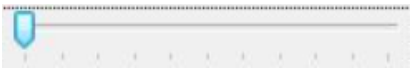
image2=cv2.addWeighted(img,90,img1,10,0)

`alpha`: Weight of the first image (`src1`)

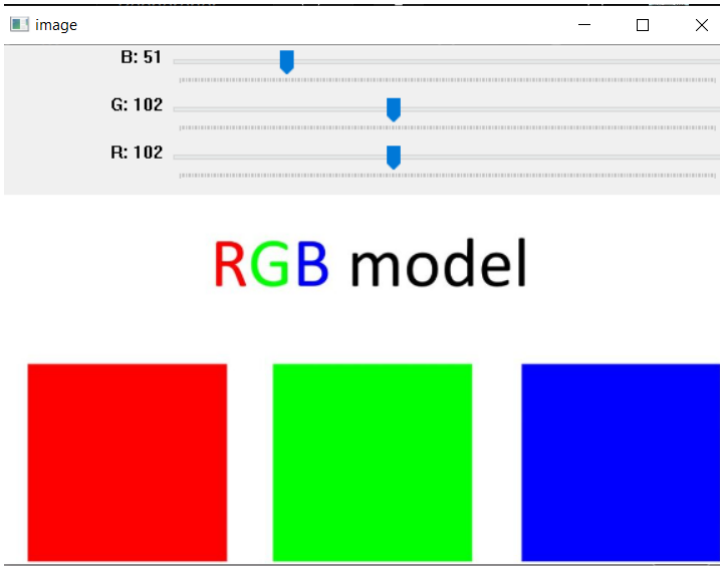
`beta`: Weight of the second image (`src2`)

-These weights are scalar values that typically range from 0 to 1, where 0 means no contribution from the respective image, and 1 means full contribution. Values between 0 and 1 represent a partial contribution.

## Trackbar(curseur):



```
import cv2
import  numpy as np
img=np.zeros((300,512,3),np.uint8)
path = r'C:\Users\T.M.S\OneDrive\Bureau\OPENCV tuto\capture.png'
img1=cv2.imread(path)
def nothing(x):
    print(x)
cv2.namedWindow('image')
cv2.createTrackbar('B','image',0,255,nothing)
cv2.createTrackbar('G','image',0,255,nothing)
cv2.createTrackbar('R','image',0,255,nothing)
cv2.imshow('image',img1)
cv2.waitKey(0)
cv2.destroyAllWindows()
```



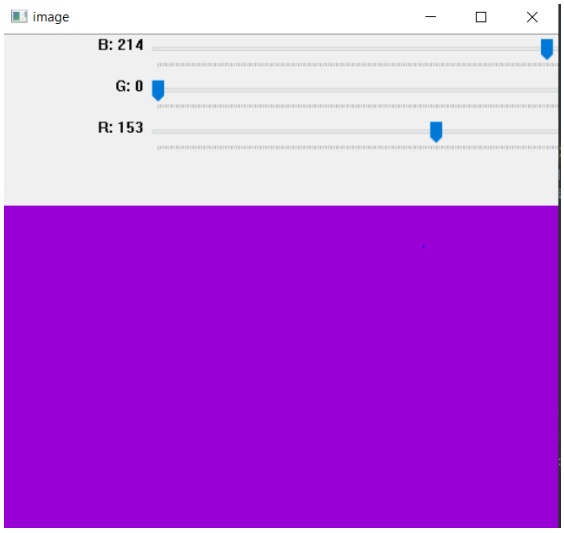
- b=cv2.`getTrackbarPos`('blue','imagee')

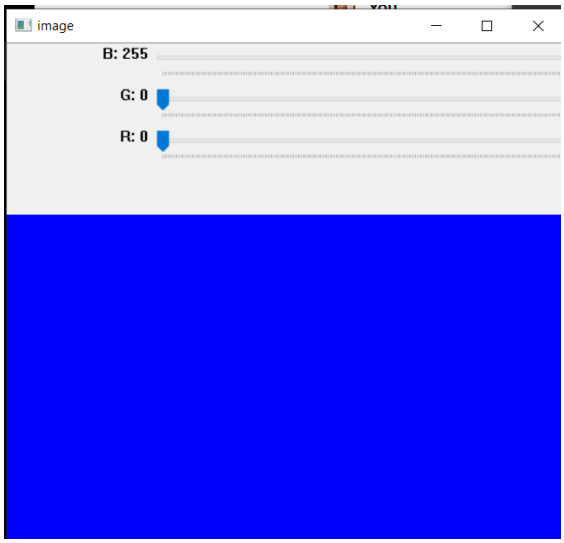
:Retrieve the current position (value) of the trackbar named 'blue' in the window named 'imagee’

-

```
import cv2
import numpy as np
from matplotlib   import pyplot as plt
img = np.zeros((300, 512, 3), np.uint8)

# Uncomment the following lines if you want to use an image
# path = r'C:\Users\T.M.S\OneDrive\Bureau\OPENCV tuto\capture.png'
# img1 = cv2.imread(path)

def nothing(x):
    print(x)

# Create a window and trackbars
cv2.namedWindow('image')

# Uncomment the following lines if you want to use an image
# cv2.imshow('image', img1)

# Create trackbars for B, G, and R channels
cv2.createTrackbar('B', 'image', 0, 255, nothing)
cv2.createTrackbar('G', 'image', 0, 255, nothing)
cv2.createTrackbar('R', 'image', 0, 255, nothing)

while True:
    cv2.imshow('image', img)

    # Get the current trackbar positions
    blue = cv2.getTrackbarPos('B', 'image')
    green = cv2.getTrackbarPos('G', 'image')
    red = cv2.getTrackbarPos('R', 'image')

    # Set the image color based on trackbar positions
    img[:] = [blue, green, red]

    key = cv2.waitKey(1) & 0xFF
    if key == 27:  # Break the loop when the 'Esc' key is pressed
        break
```

```
cv2.destroyAllWindows()
```




## *Matplotlib:*

`Matplotlib` is a 2D plotting library for the Python programming language. It allows you to create static, animated, and interactive visualizations in Python. Matplotlib is widely used for creating various types of plots and charts, making it a popular tool for data visualization in scientific computing, data analysis, machine learning, and more.

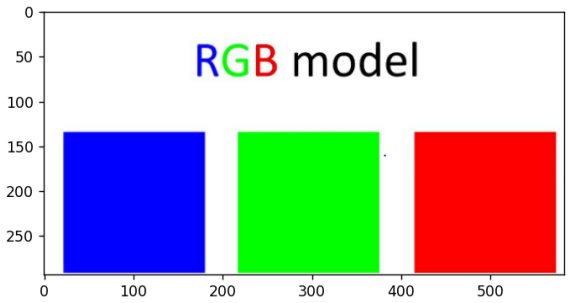
Key features of Matplotlib include:

1. **Support for Various Plot Types:**

- Line plots
- Scatter plots
- Bar plots
- Histograms
- Pie charts
- 3D plots, and more.

-Show an image using matplotlib library:

```
import cv2
from matplotlib   import pyplot as plt
path = r'C:\Users\T.M.S\OneDrive\Bureau\OPENCV tuto\capture.png'
img=cv2.imread(path)
cv2.imshow('imagr',img)
plt.imshow(img)
plt.show()
cv2.waitKey(0)
cv2.destroyAllWindows()
```

=⇒When OpenCV reads an image, it interprets the color channels in the order of Blue, Green, and Red (**BGR**),

=⇒when Matplotlib displays an image, it expects the color channels to be in the order of Red, Green, and Blue (**RGB**).

- `plt.xticks([]):` remove the axis ticks (labels) from the x-axis

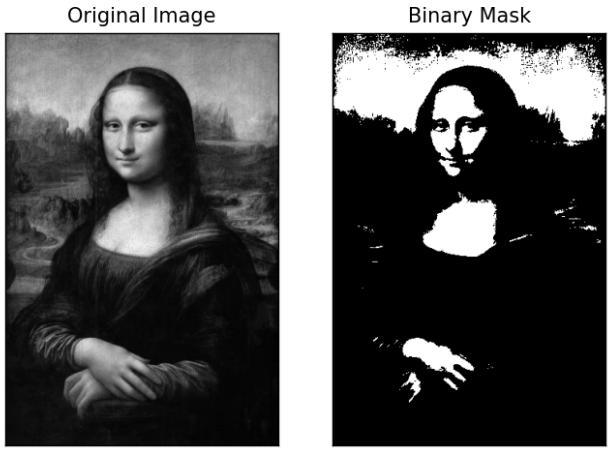
## Morphological Transformations:

-performed on binary images

-A kernel tells us how to change the value of any given  pixel by combining it with different amounts of the neighboring pixels

- **Binary Image:**
- In a binary image, each pixel is restricted to only two possible intensity values: usually 0 (black) or 255 (white). This means that each pixel is either fully off or fully on.
- Binary images are often used for tasks where only the presence or absence of information matters, such as in image segmentation or object detection.
- Binary images are commonly obtained through a process called **thresholding**, where a threshold value is chosen, and pixels with intensity values above the threshold are set to white, while those below are set to black

```
import cv2
from matplotlib import pyplot as plt

path = r'C:\Users\T.M.S\OneDrive\Bureau\OPENCV tuto\MONA_LISA.JPG'
img = cv2.imread(path, cv2.IMREAD_GRAYSCALE)  # Read the image in grayscale m
_, mask = cv2.threshold(img, 120, 255, cv2.THRESH_BINARY)

title = ['Grayscale Image', 'Binary Image']
images = [img, mask]

for i in range(2):
plt.subplot(1, 2, i + 1)
plt.imshow(images[i], cmap='gray')  # Moved cmap='gray' to plt.imshow()
plt.title(title[i])
plt.xticks([]), plt.yticks([])  # Hide axis ticks

plt.show()
```




- `kernel=np.ones((2,2),np.uint8):` creating a 2x2 matrix (kernel) filled with ones

```
import cv2
from matplotlib import pyplot as plt
import numpy as np
path = r'C:\Users\T.M.S\OneDrive\Bureau\OPENCV tuto\MONA_LISA.JPG'
img = cv2.imread(path, cv2.IMREAD_GRAYSCALE)  # Read the image in grayscale m
_, mask = cv2.threshold(img, 120, 255, cv2.THRESH_BINARY)
kernel=np.ones((2,2),np.uint8)
dilatation=cv2.dilate(mask,kernel)

title = ['Original Image', 'Binary Mask','dilatee']
images = [img, mask,dilatation]

for i in range(3):
    plt.subplot(1, 3, i + 1)
    plt.imshow(images[i], cmap='gray')  # Moved cmap='gray' to plt.imshow()
    plt.title(title[i])
    plt.xticks([]), plt.yticks([])  # Hide axis ticks

plt.show()
```


- kernel = `np.ones` ((3,3), np.uint8)

output: [ [1 1 1]
[1 1 1]
[1 1 1] ]

---

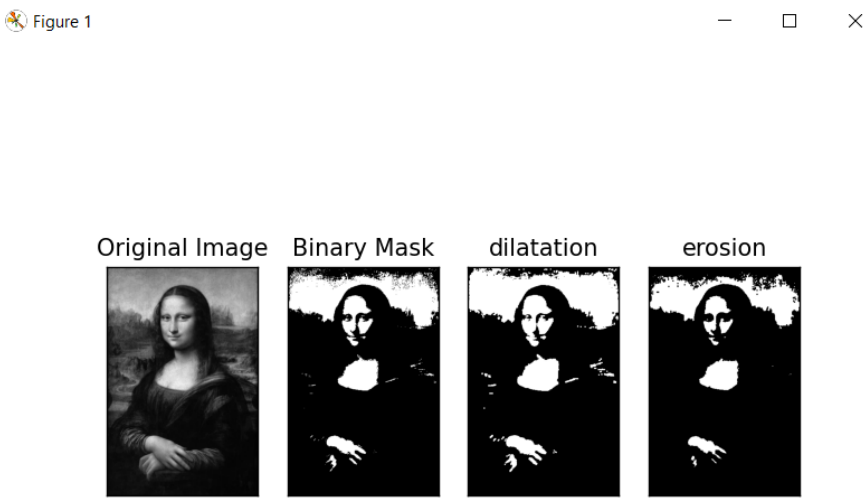
**Creating a Structuring Element (`kernel`):** The `np.ones((3,3), np.uint8)` creates a 3x3 matrix of ones. This matrix is used as a structuring element, which defines the neighborhood for the dilation operation. In this case, it's a simple 3x3 square-shaped structuring element.

**Dilation Operation:** The `cv2.dilate` function is then applied to the binary thresholded image (`mask`) using the specified structuring element (`kernel`). Dilation is a morphological operation that is commonly used to enhance the features of an image. It works by placing the structuring element at each pixel in the image and setting the pixel value to the maximum value within the neighborhood defined by the structuring element. This has the effect of thickening or **expanding bright regions in the image.**

- `erosion = cv2.erode(mask, kernel)`

---

**Erosion Operation:** Erosion is another morphological operation that is used to **shrink or erode the boundaries of bright regions in a binary image**. It works by placing the structuring element at each pixel in the image and setting the pixel value to the **minimum** value within the neighborhood defined by the structuring element.




- `opening = cv2.morphologyEx(mask, cv2.MORPH_OPEN, kernel)`

---

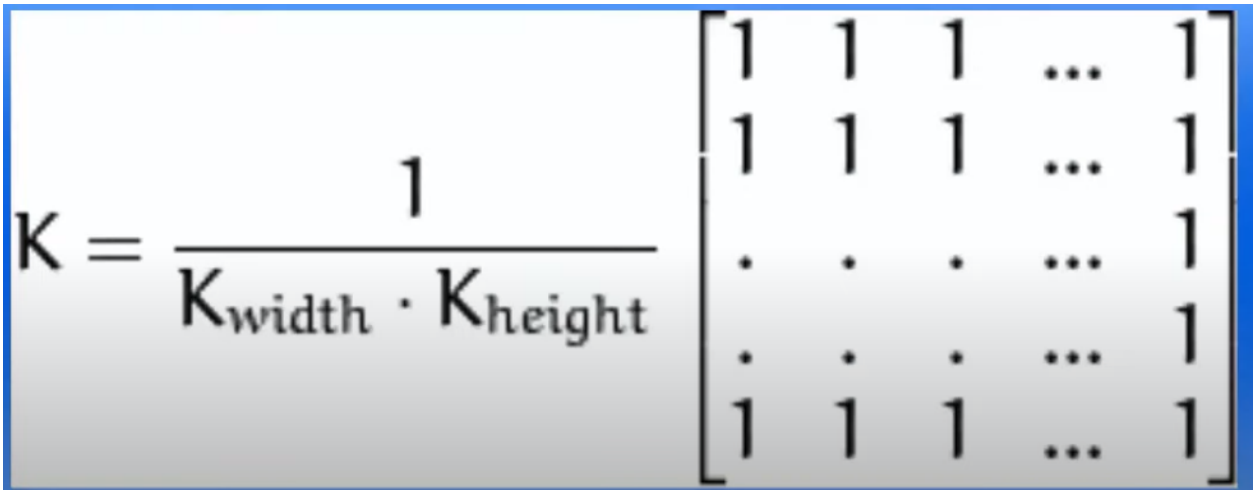
**Morphological Opening Operation:**

- Morphological opening is a combination of erosion followed by dilation. It is particularly useful in removing noise and small objects from binary images while preserving the overall structure of larger objects.

- `image1 = cv2.pyrDown(img)`

1. **Image Pyramids:**
   - Image pyramids are multi-scale representations of an image. They are commonly used in computer vision and image processing for tasks such as image blending, feature matching, and scale-invariant feature extraction.
2. **cv2.pyrDown:**
   - `cv2.pyrDown` is a function in the OpenCV library that performs downsampling (reducing the image size) by applying a Gaussian smoothing and then discarding every second pixel along each dimension. This process effectively reduces the image dimensions by half.

- In image processing,a kernel,convolution matrix,or mask is a small matrix,it is used for blurring ,sharpening ,embossing,edge detection, and more



## Depth:

—It is the number of bits used to represent the color of each pixel in an image. The depth determines the range of colors that can be represented, and it is often expressed in bits per channel.

- **8-bit depth**: Each color channel (Red, Green, Blue) is represented using 8 bits, allowing for 256 different intensity levels for each channel. This is common in images with a total of 24 bits per pixel (8 bits per channel).
- **16-bit depth**: Each color channel is represented using 16 bits, providing a larger range of possible intensity levels compared to 8-bit depth. This is often used in scientific or medical imaging.
- **32-bit depth**: Each color channel is represented using 32 bits, and this is often used in high dynamic range (HDR) images. The extra precision allows for a wider range of color value

## Kernel:

- In image processing, a kernel is a small matrix used for various operations, such as convolution. Convolution involves sliding the kernel over the image and performing a mathematical operation at each step. The choice of kernel determines the nature of the image processing operation being applied.

1. **Identity Kernel:**

```
[[0, 0, 0],
 [0, 1, 0],
 [0, 0, 0]]
```

   This kernel leaves the image unchanged. It's often used to demonstrate the concept of convolution.

2. **Box Blur Kernel:**

```
[[1/9, 1/9, 1/9],
 [1/9, 1/9, 1/9],
 [1/9, 1/9, 1/9]]
```

This kernel is used for simple blurring. It calculates the average of the neighboring pixels.

3. **Edge Detection Kernels:**
   - Sobel Kernel for horizontal edges:

     ```
     [[-1, -2, -1],
      [0, 0, 0],
      [1, 2, 1]]
     ```

   - Sobel Kernel for vertical edges:

     ```
     [[-1, 0, 1],
      [-2, 0, 2],
      [-1, 0, 1]]
     ```

   These kernels are used for edge detection, emphasizing changes in intensity.

4. **Sharpening Kernel:**

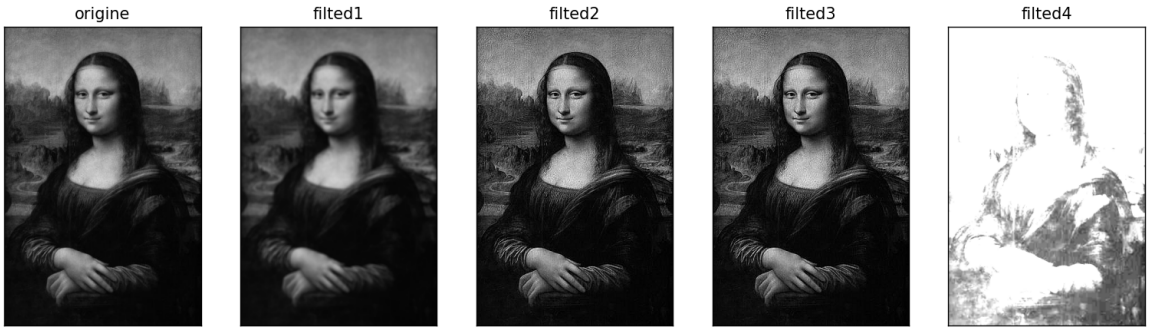
   ```
   [[0, -1, 0],
    [-1, 5, -1],
    [0, -1, 0]]
   ```

   This kernel enhances edges, making the image appear sharper.

5. **Gaussian Blur Kernel:**
   The Gaussian blur is commonly used for smoothing or blurring. The kernel size and values depend on the desired level of blurring.

   ```
   [[1, 2, 1],
    [2, 4, 2],
    [1, 2, 1]]
   ```

```
img1 = cv2.filter2D(img,-1,identity_kernel)
img2 = cv2.filter2D(img,-1,Sharpening_Kernel)
img3 = cv2.filter2D(img,-2,Sharpening_Kernel)
img4=cv2.filter2D(img,-1,gaussien_kernel)
```




- As in one dimensional signals,images also can be filtered with various **low_ pass filters(LPF): removing noise,high pass filters(HPF):finding edges in the image**
- `img_blur = cv2.blur(img, (1, 1))`

-Blurring, also known as smoothing, is a common image processing operation that reduces the sharpness of edges or fine details in an image. It's often used for various purposes, such as noise reduction, image preprocessing

-The averaging algorithm you mentioned is a simple blurring algorithm. The basic idea is to replace each pixel value in the image with the average value of its neighboring pixels. This process helps to reduce high-frequency components, resulting in a smoother image.

## Gaussien Filter:

`img6=cv2.GaussianBlur(img , ( 1,1 ) ,0 )`

-0:This parameter represents the standard deviation of the Gaussian kernel along the x-axis. In your example, it's set to 0, which means that OpenCV will automatically calculate the standard deviation based on the kernel size. You can also provide a specific value for the standard deviation if needed.

-(1,1):This is the kernel size. The kernel size determines the extent of the blur

-**Gaussian blur** is a popular image processing technique used to reduce noise and detail in an image. It's named after **the Gaussian distribution** because it involves convolving the image with **a Gaussian filter**, which has a bell-shaped curve when plotted. **The Gaussian filter is a weighted average of neighboring pixel values, with the weights determined by the Gaussian function.**

The Gaussian blur is characterized by the standard deviation parameter (σ), which controls the spread or width of the Gaussian curve. A larger standard deviation results in a wider, smoother blur, while a smaller standard

deviation produces a narrower blur.

The formula for a 2D Gaussian function in image processing is as follows:

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

-Applying a Gaussian blur to an image involves convolving the image with a Gaussian kernel. The larger the kernel (determined by the standard deviation), the more the high-frequency components (fine details, noise) are suppressed, resulting in a smoother appearance.

-The Gaussian blur is characterized by **the standard deviation parameter (σ)**, which controls the spread or width of the Gaussian curve. A larger standard deviation results in a wider, smoother blur, while a smaller standard deviation produces a narrower blur.

- `img_blur = cv2.medianBlur(img, 5)`

-Median blur is a nonlinear operation that replaces each pixel value with the median value of its neighborhood. It is particularly effective at removing salt-and-pepper noise from images while preserving edges. This makes it a good choice for denoising images with impulsive noise.

# Edge Detection:

## Image Laplacien Gradient:

-Directional change in the intensity or color in an image

```
lap = cv2.Laplacian(img, cv2.CV_64F)
lap = np.uint8(np.absolute(lap))
```

⇒ Compute the Laplacian of an image. It applies a **2D Laplace operator** to the image, which is a derivative operator used for edge detection.

- `cv2.CV_64F` : This is the data type of the output image. The Laplacian result may contain negative values, so a floating-point data type is often used to represent both positive and negative values. `cv2.CV_64F` corresponds to a 64-bit floating-point data type.

-

- `np.absolute` : This NumPy function is used to compute the absolute values of the Laplacian result, as you are interested in the magnitude of the gradient rather than the direction.
- `np.uint8` : This is then used to convert the result to an unsigned 8-bit integer. The Laplacian values are scaled and converted to the range [0, 255] suitable for display as an image.

## The Sobel operator:

-The Sobel operator is a convolution-based edge detection operator used in image processing and computer vision. It is designed to highlight edges in an image by computing the gradient of the image intensity. The Sobel

operator uses convolution with small, simple filters (kernels) to approximate the derivative of the image intensity in the x and y directions.

- `img1 = cv2.Sobel(img, cv2.CV_64F, 1, 0)`

-

For a 3x3 Sobel operator, the filters for the x-direction (Sobel X) and y-direction (Sobel Y) are as follows:

Sobel X=[−101−202−101]Sobel X=⌊⌈−1−2−1000121⌋⌉

Sobel Y=[−1−2−1000121]Sobel Y=⌊⌈−101−202−101⌋⌉

The Sobel operator calculates the gradient of the image by convolving the image with these kernels. The result of the convolution in the x-direction highlights edges running vertically, and in the y-direction, it highlights edges running horizontally.

## The canny edge detection algorithm:

- The canny edge detection algorithm is composed of 5 steps:

— Noise reduction

—gradient calculation

—non maximum_suppression

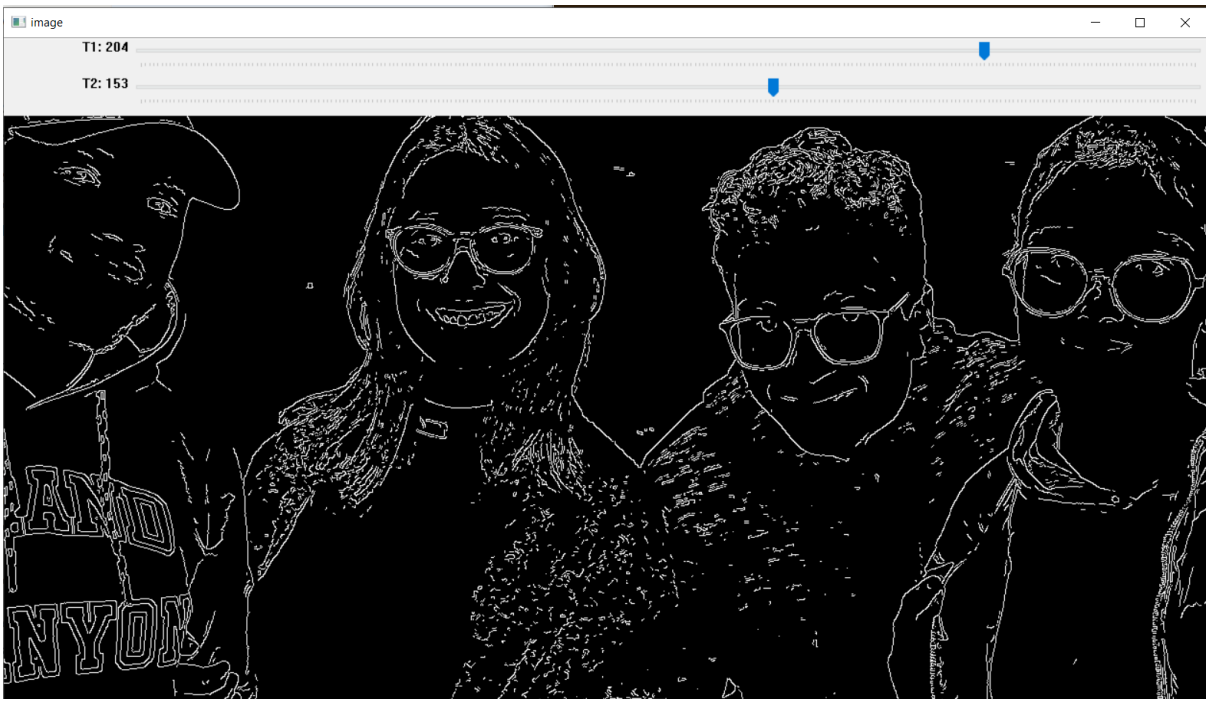
—double treshold

—edge tracking by hysteresis

```
    canny = cv2.Canny(img,treshold1,treshold2)
```

```
import cv2
import numpy as np

path = r'C:\Users\T.M.S\OneDrive\Bureau\OPENCV tuto\chela.PNG'
img = cv2.imread(path,0)

def nothing(x):
    pass

cv2.namedWindow('image')
cv2.createTrackbar('T1','image',0,255,nothing)
cv2.createTrackbar('T2','image',0,255,nothing)

while True:
```

```
    treshold1 = cv2.getTrackbarPos('T1','image')
    treshold2 = cv2.getTrackbarPos('T2','image')

    canny = cv2.Canny(img,treshold1,treshold2)

    # Display the Canny edge detection result
    cv2.imshow('image',canny)

    key = cv2.waitKey(1) &0xFF
if key ==27:  # Break the loop when the 'Esc' key is pressed
        break
```



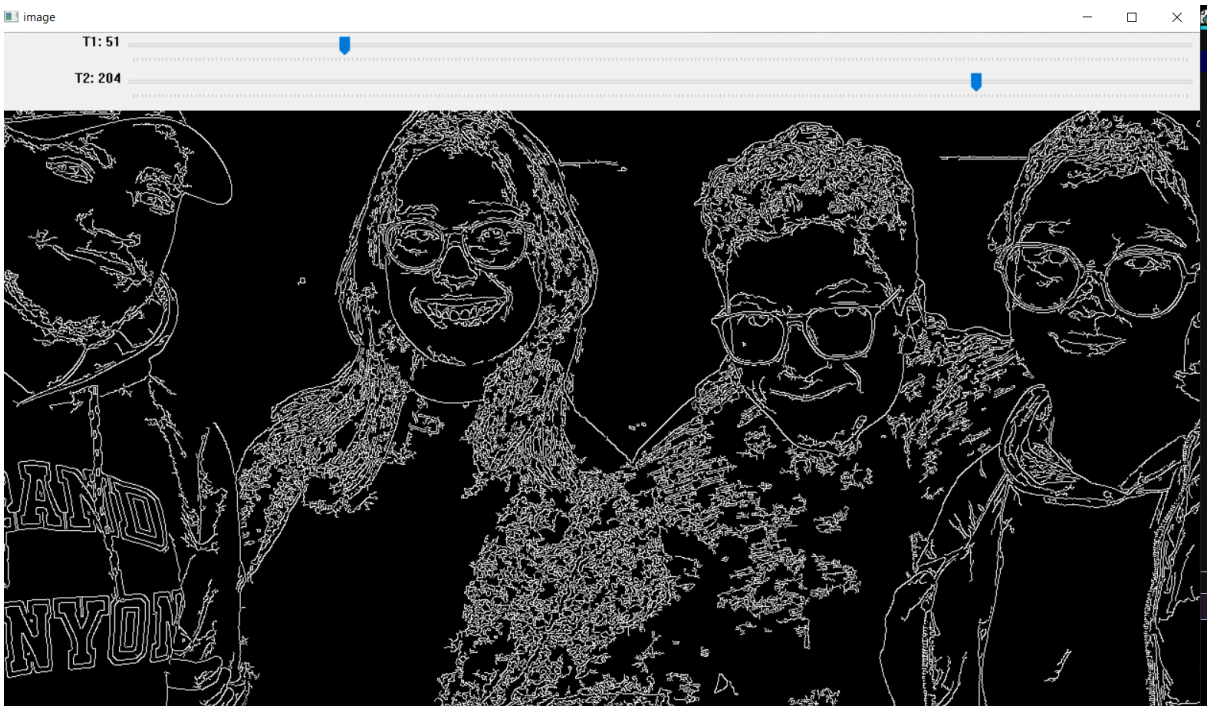
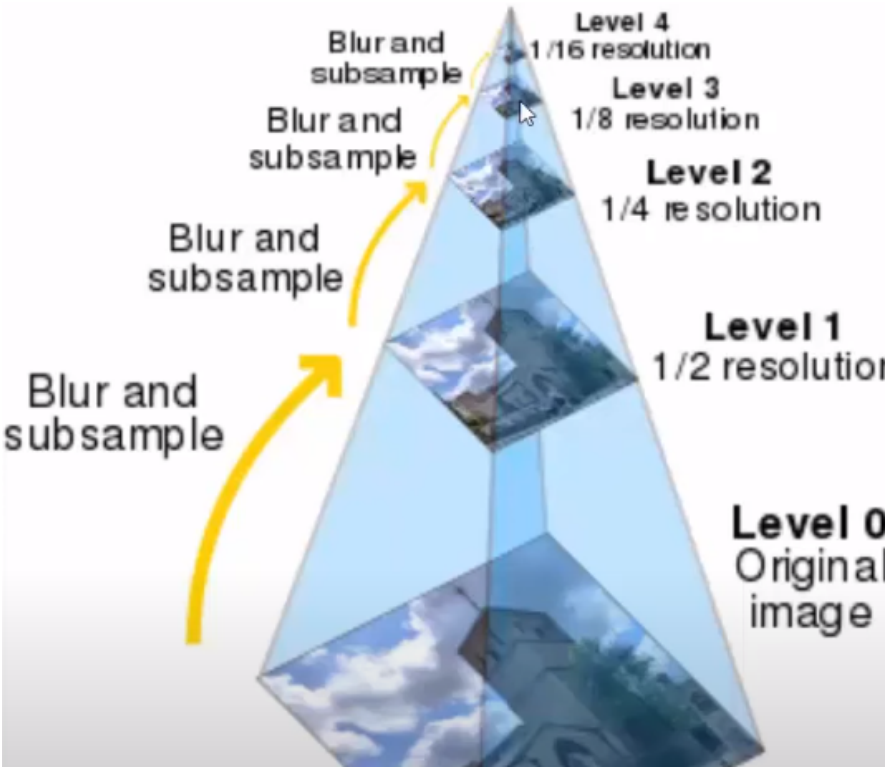


image
T1: 51
T2: 204

-

## Pyramids:


Blur and subsample
Blur and subsample
Blur and subsample
Blur and subsample
Level 4
1/16 resolution
Level 3
1/8 resolution
Level 2
1/4 resolution
Level 1
1/2 resolution
Level 0
Original image

-`cv2.pyrDown()` function reduces the size of the input image by half. It works by smoothing and then subsampling the original image, creating a lower-resolution version of it. This process is often used in computer vision and image processing applications where lower resolution is acceptable for certain tasks, such as feature detection or object recognition.

- `combined = np.hstack((img[:, :256], img1[:, :256]))`

  -This line is concatenating the left part of the first image (`img[:, :256]`) with the left part of the second image (`img1[:, :256]`) horizontally, creating a new image where the content of both images is displayed side by side. This is often done for visualization purposes or as a step in certain image processing tasks.

- Note:

```
img=cv2.imread(path)   :colored image
gray=cv2.cvtColor(img,cv2.COLOR_BGR2GRAY) :grayscale image
ret,thresh=cv2.threshold(gray,127,255,0)  ;binary image
```

## Find and Draw Contours :

-Contours is a python **List** of all the contours in an image ,each individual contour of the list is a **numpy array of (x,y) coordinates of boundary points of the object .**

print(contours[0])

==⇒ [[[0 0]]

[[0 1]]

[[0 2]]

...

[[3 0]]

[[2 0]]

[[1 0]]]

-`cv2.findContours()`**: a** function is typically applied to **binary or grayscale images**, where contours can be easily identified based on differences in pixel intensities. When working with colored images, it's common practice to convert them to grayscale first before applying the contour detection.

```
import cv2
import numpy as np

# Read a colored image
img = cv2.imread('path/to/your/image.jpg')

# Convert the image to grayscale
```

```
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# Apply thresholding to create a binary image
ret, thresh = cv2.threshold(gray, 127, 255, 0)

# Find contours in the binary image
contours, hierarchy = cv2.findContours(thresh, cv2.RETR_TREE, cv2.CHAIN_AP

# Draw the contours on the original image
cv2.drawContours(img, contours, -1, (0, 255, 0), 2)

# Display the result
cv2.imshow('Contours', img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

- `cv2.RETR_TREE`: This is the retrieval mode for the contours. `cv2.RETR_TREE` retrieves all of the contours and reconstructs a full hierarchy of nested contours. Other retrieval modes are available, such as `cv2.RETR_EXTERNAL` (only retrieves the extreme outer contours) or `cv2.RETR_LIST` (retrieves all contours without reconstructing a hierarchy).

- `cv2.CHAIN_APPROX_NONE`: This is the contour approximation method. In this case, `cv2.CHAIN_APPROX_NONE` means that all the contour points are stored, without any approximation. Other options include `cv2.CHAIN_APPROX_SIMPLE` (compresses horizontal, vertical, and diagonal segments and leaves only their end points) and `cv2.CHAIN_APPROX_TC89_L1` or `cv2.CHAIN_APPROX_TC89_KCOS` (applies the Teh-Chin chain approximation algorithm)

- `cv2.drawContours`():Draw contours on original image, find contours in grayscale image

```
contours,hierarchy=cv2.findContours(thresh,cv2.RETR_TREE,cv2.CHAIN_APPROX_
cv2.drawContours(img,contours,-1,(0,255,0),2)
-1:All the contours
2:thickness
img:original image
```

https://labs.cognitiveclass.ai/v2/tools/jupyterlab?ulid=ulid-f7c86ff2267614a9d8031ab610e228307f3cf478

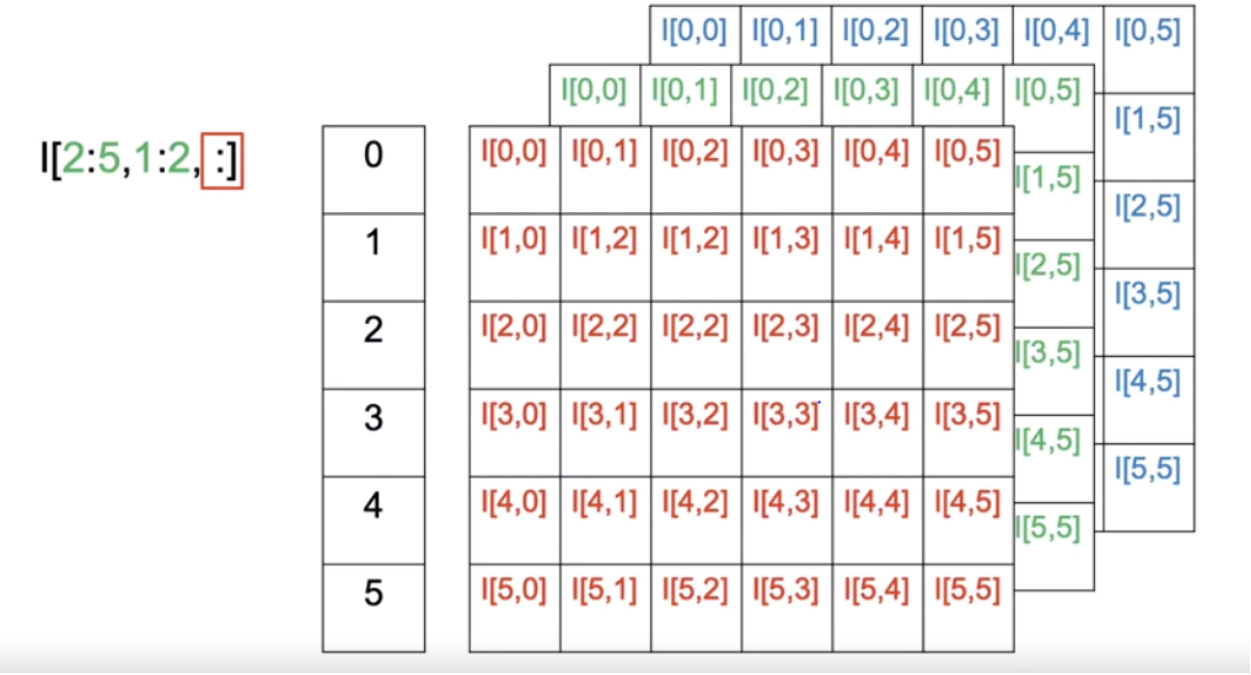
- Cropping:  It  is "cutting out" the part of the image and throwing out the rest

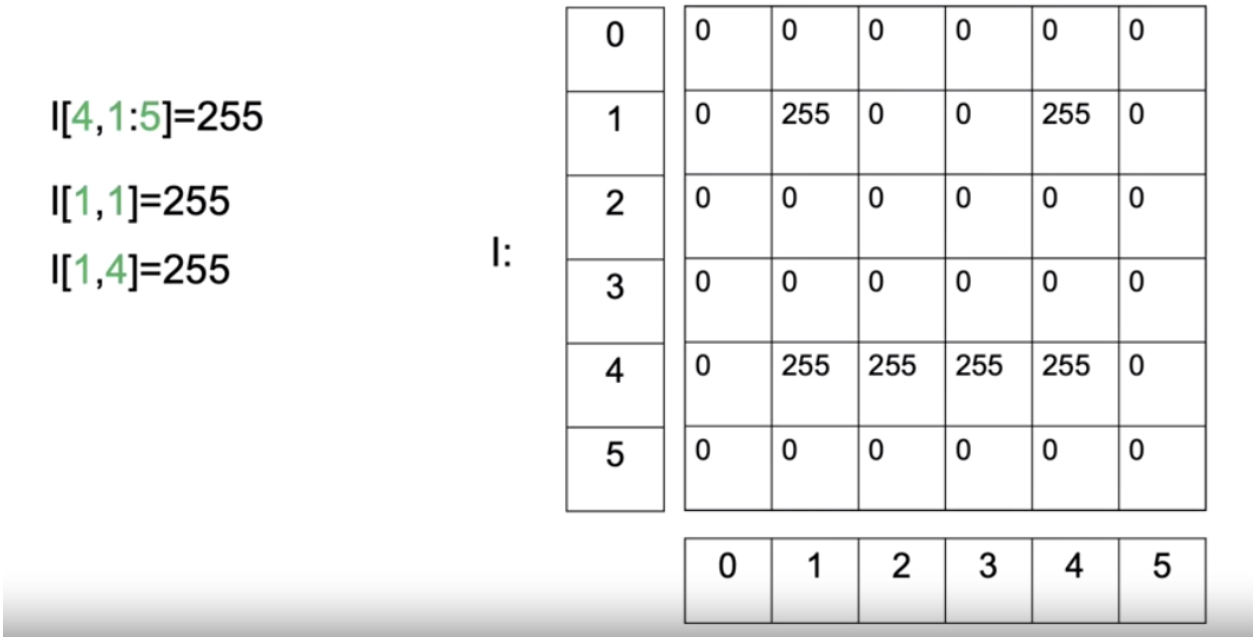
I[2:5,:]

| | | | | | | |
|---|---|---|---|---|---|---|
| 0 | I[0,0] | I[0,1] | I[0,2] | I[0,3] | I[0,4] | I[0,5] |
| 1 | I[1,0] | I[1,2] | I[1,2] | I[1,3] | I[1,4] | I[1,5] |
| 2 | I[2,0] | I[2,2] | I[2,2] | I[2,3] | I[2,4] | I[2,5] |
| 3 | I[3,0] | I[3,1] | I[3,2] | I[3,3] | I[3,4] | I[3,5] |
| 4 | I[4,0] | I[4,1] | I[4,2] | I[4,3] | I[4,4] | I[4,5] |
| 5 | I[5,0] | I[5,1] | I[5,2] | I[5,3] | I[5,4] | I[5,5] |
| | 0 | 1 | 2 | 3 | 4 | 5 |

I[2:5,1:2]

| | | | | | | |
|---|---|---|---|---|---|---|
| 0 | I[0,0] | I[0,1] | I[0,2] | I[0,3] | I[0,4] | I[0,5] |
| 1 | I[1,0] | I[1,2] | I[1,2] | I[1,3] | I[1,4] | I[1,5] |
| 2 | I[2,0] | I[2,2] | I[2,2] | I[2,3] | I[2,4] | I[2,5] |
| 3 | I[3,0] | I[3,1] | I[3,2] | I[3,3] | I[3,4] | I[3,5] |
| 4 | I[4,0] | I[4,1] | I[4,2] | I[4,3] | I[4,4] | I[4,5] |
| 5 | I[5,0] | I[5,1] | I[5,2] | I[5,3] | I[5,4] | I[5,5] |
| | 0 | 1 | 2 | 3 | 4 | 5 |



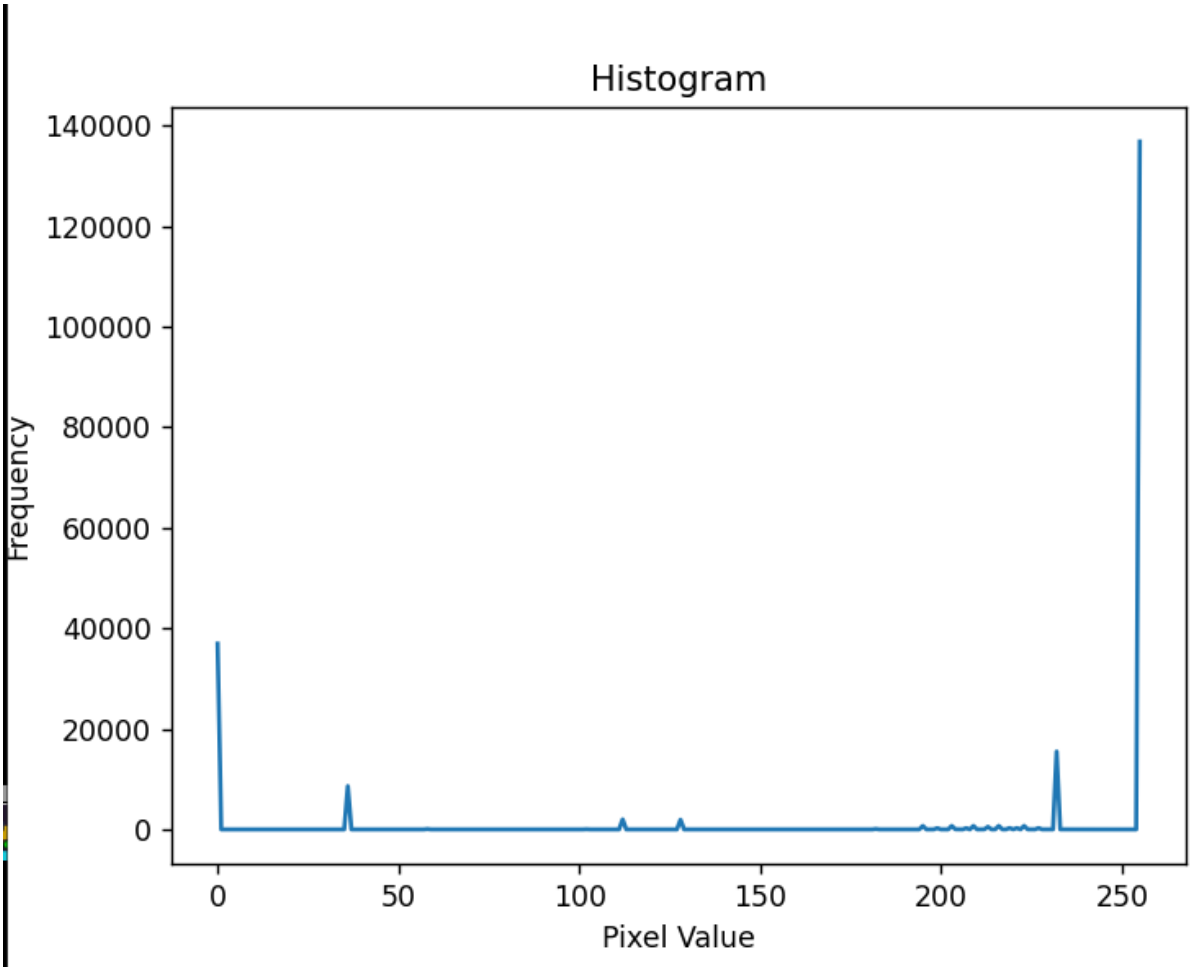




## Histogram:

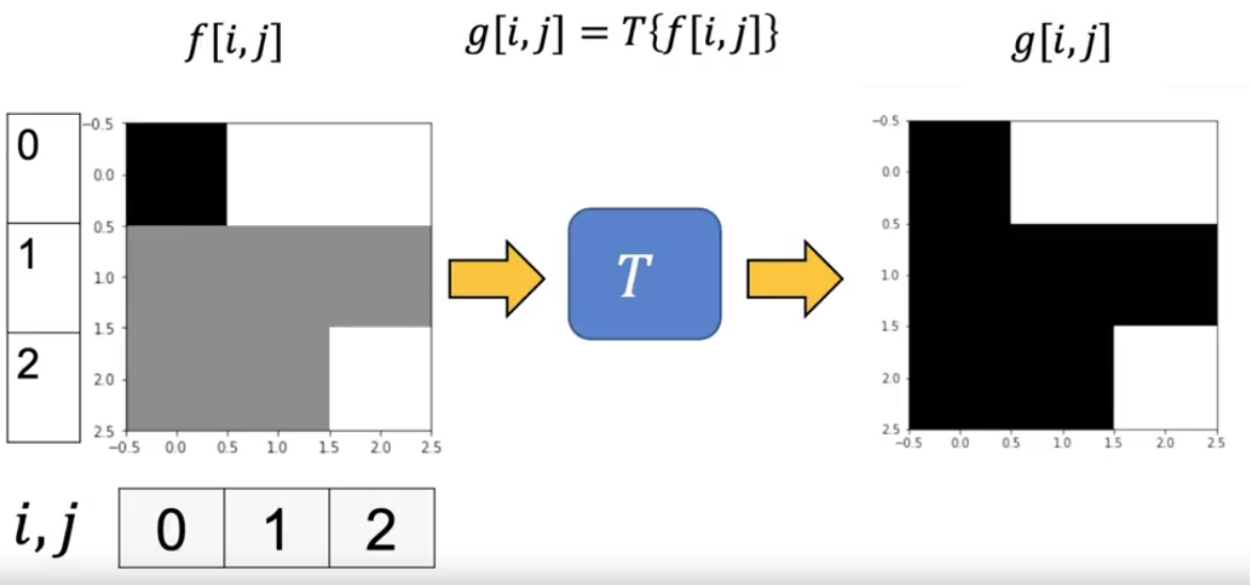
-A histogram is an accurate graphical representation of the distribution of numerical data.

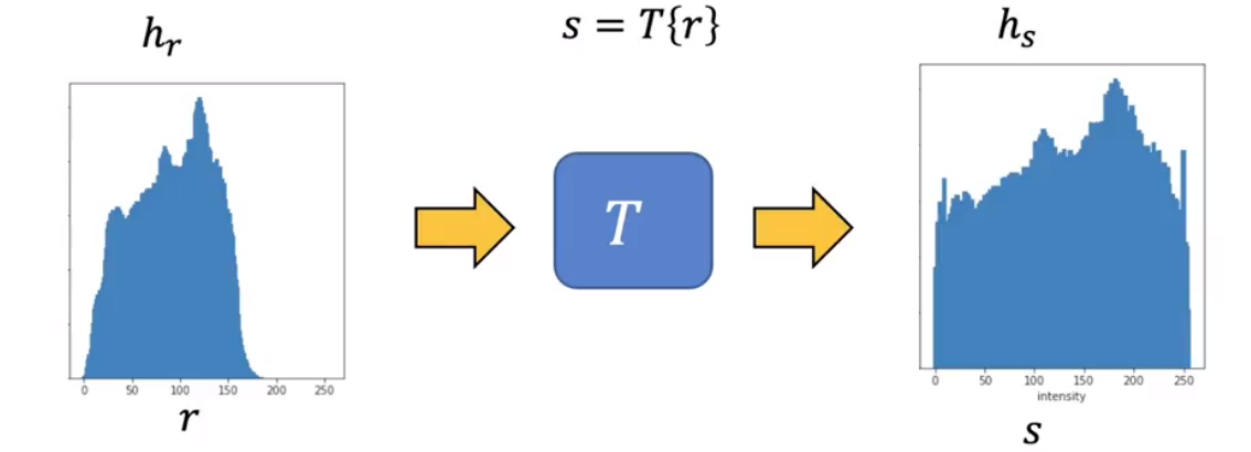
-To construct a histogram, the first step is to “bin” the range of values—that is, divide the entire range of values into a series of intervals—and then count how many values fall into each interval.

```
histogram = cv2.calcHist([img], [channels], None, [hist_bins], [hist_range])
```




## Intensity Transformations:



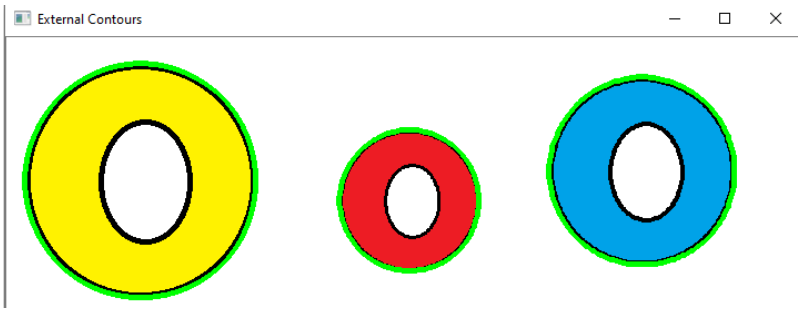



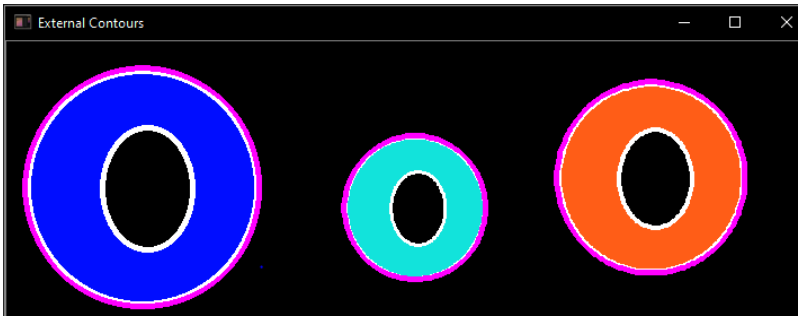

## Image Negative:

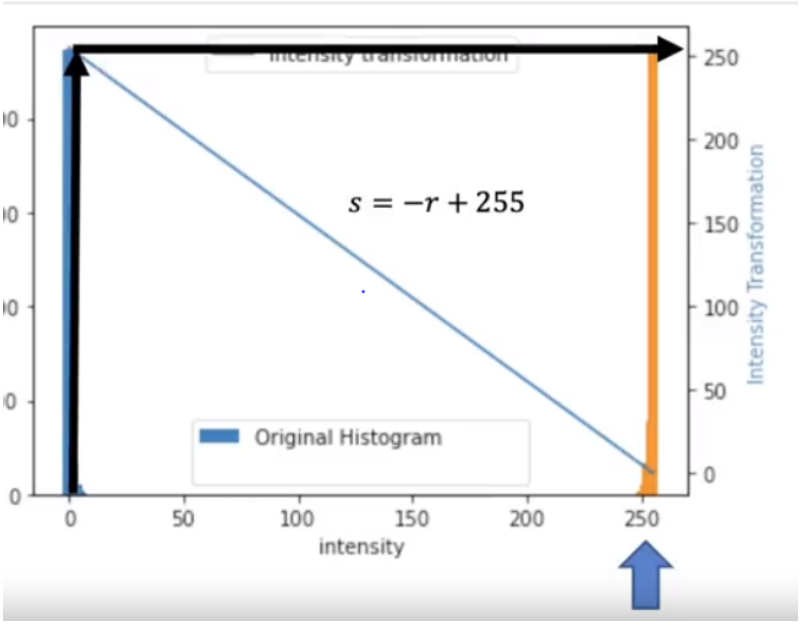
`img3=-img+ 255`

-Subtracting each pixel value in the image ( `img` ) from 255 and then negating the result. This effectively creates a negative of the image and shifts the pixel values so that the darkest pixels become the brightest and vice versa.

$$g[i,j] = -1f[i,j] + 255$$




## Brightness and Contrast Adjustments:

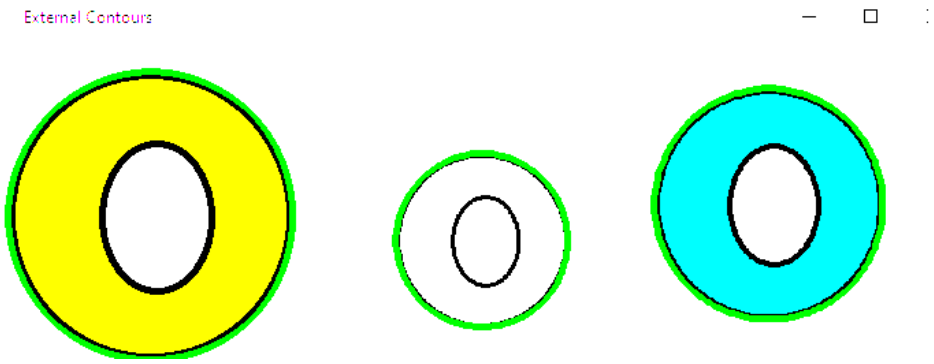
$$g[i,j] = \alpha f[i,j] + \beta$$

```
alpha = 1 # Simple contrast control
beta = 100   # Simple brightness control
```

$$g[i,j] = 1f[i,j] + 100$$

```
img4 = cv2.convertScaleAbs(img, -1, 255):
```

- This line uses `cv2.convertScaleAbs()` to perform a **linear transformation** on the input image ( `img` ). The function scales and shifts the input pixel values according to the specified alpha and beta parameters.




## Histogram Equalization:

The `cv2.equalizeHist()` function in OpenCV is used for histogram equalization of grayscale images. Histogram equalization is a technique to enhance the contrast of an image by spreading out the intensity values across the entire range. It is particularly useful when an image has a limited dynamic range, and certain intensity values dominate the image.

Here's a brief explanation of the role of `cv2.equalizeHist()` :

1. **Enhancing Contrast:**
    - Histogram equalization redistributes the intensity values of an image so that the full range of pixel values (usually 0 to 255 in an 8-bit image) is utilized. This helps in enhancing the overall contrast of the image.
2. **Improving Visibility:**
    - In images where the pixel values are concentrated in a specific range, details may be lost or difficult to distinguish. Histogram equalization aims to spread out the pixel values, making the image more visually appealing and improving the visibility of details.
3. **Adaptive Brightness Adjustment:**
    - It acts as a form of adaptive brightness adjustment. It can be particularly effective in images where the lighting conditions vary across different regions.

```
img_equalized = cv2.equalizeHist(img)
```

## Tresholding and simple Segmentation:

```
def thresholding(input_img,threshold,max_value=255, min_value=0):
    N,M=input_img.shape
    image_out=np.zeros((N,M),dtype=np.uint8)

    for i  in range(N):
        for j in range(M):
            if input_img[i,j]> threshold:
                image_out[i,j]=max_value
            else:
                image_out[i,j]=min_value

    return image_out
```

## Histogram:

```
import matplotlib.pyplot as plt
import cv2
import numpy as np
  def plot_image   (image_1, image_2,title_1="Orignal", title_2="New Image"):
    plt.figure(figsize=(10,10))
    plt.subplot(1, 2, 1)
    plt.imshow(image_1,cmap="gray")
    plt.title(title_1)
    plt.subplot(1, 2, 2)
    plt.imshow(image_2,cmap="gray")
    plt.title(title_2)
    plt.show()
 def   plot_hist(old_image, new_image,title_old="Orignal", title_new="New Ima
    intensity_values=np.array([x for x in range(256)])
    plt.subplot(1, 2, 1)
    plt.bar(intensity_values, cv2.calcHist([old_image],[0],None,[256],[0,256]
    plt.title(title_old)
    plt.xlabel('intensity')
    plt.subplot(1, 2, 2)
    plt.bar(intensity_values, cv2.calcHist([new_image],[0],None,[256],[0,256]
    plt.title(title_new)
```

```
    plt.xlabel('intensity')
    plt.show()
```

`hist[:, 0]` extracts all the values in the first column of the histogram array. In the context of a histogram, each row corresponds to a specific intensity level, and the value in the first column of that row represents the frequency or count of pixels with that intensity level in the image.

```
hist = cv2.calcHist([goldhill],[0], None, [256], [0,256])
intensity_values = np.array([x for x in range(hist.shape[0])])
plt.bar(intensity_values, hist[:,0], width = 5)
plt.title("Bar histogram")
plt.show()
```

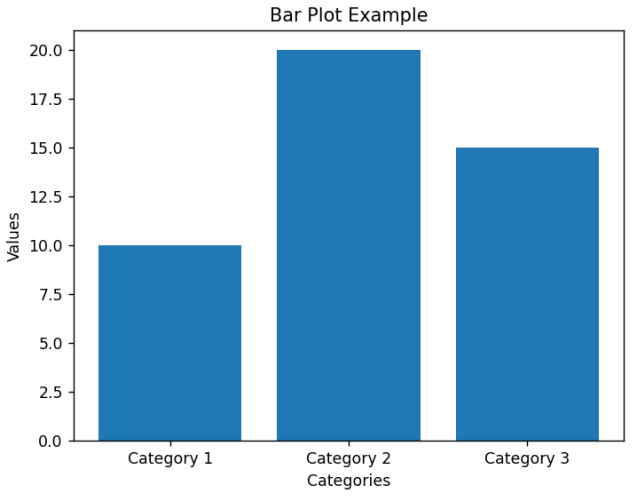
`hist.shape[0]==histSize` **:** it returns the number of rows/bins/levels in the NumPy array `hist`. In the context of a histogram, each row corresponds to a bin or a specific intensity level.

```
import cv2
import matplotlib.pyplot as plt

# Load an image
image = cv2.imread('example.jpg', cv2.IMREAD_GRAYSCALE)

# Calculate histogram
hist = cv2.calcHist([image], [0], None, [256], [0,256])

# Plot the histogram
plt.plot(hist)
plt.title('Histogram')
plt.xlabel('Pixel Value')
plt.ylabel('Frequency')
plt.show()
```

```
hist=cv2.calcHist(images, channels, mask, histSize, ranges[, hist[, accumulat
```

- `images`: This is the source image (or list of images). It should be in square brackets, even if you're using a single image.
- `channels`: This is the channel or channels for which you calculate the histogram. It is also specified in square brackets. For **grayscale images, it is [0]**, and for **color images, it can be [0], [1], [2] to represent the Blue, Green, and Red channels, respectively.**
- `mask`: An optional mask. If a mask is provided, the histogram will only be computed for the masked pixels.

- `histSize`: This represents **the number of bins or levels in the histogram**. For example, [256] indicates 256 bins for a grayscale image.
- `ranges`: This is **the range of pixel values** to be considered. Typically, it is **[0,256] for the full range of intensity values in an 8-bit image.**
- `hist`: This is the output histogram. It is an optional parameter.
- `accumulate`: This is also an optional parameter. If it is set, the function does not clear the output histogram in the beginning. It is useful in the case of multiple images.




Untitled

## Geometric transformations:

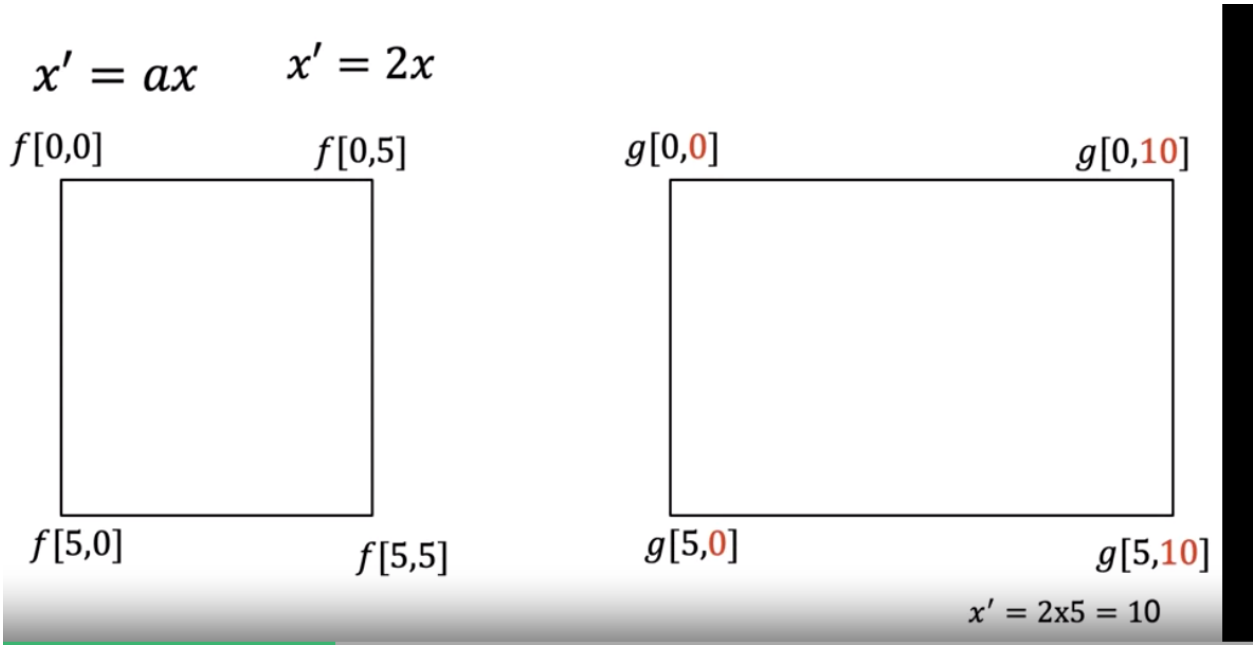
⇒Shrink or Expand the image

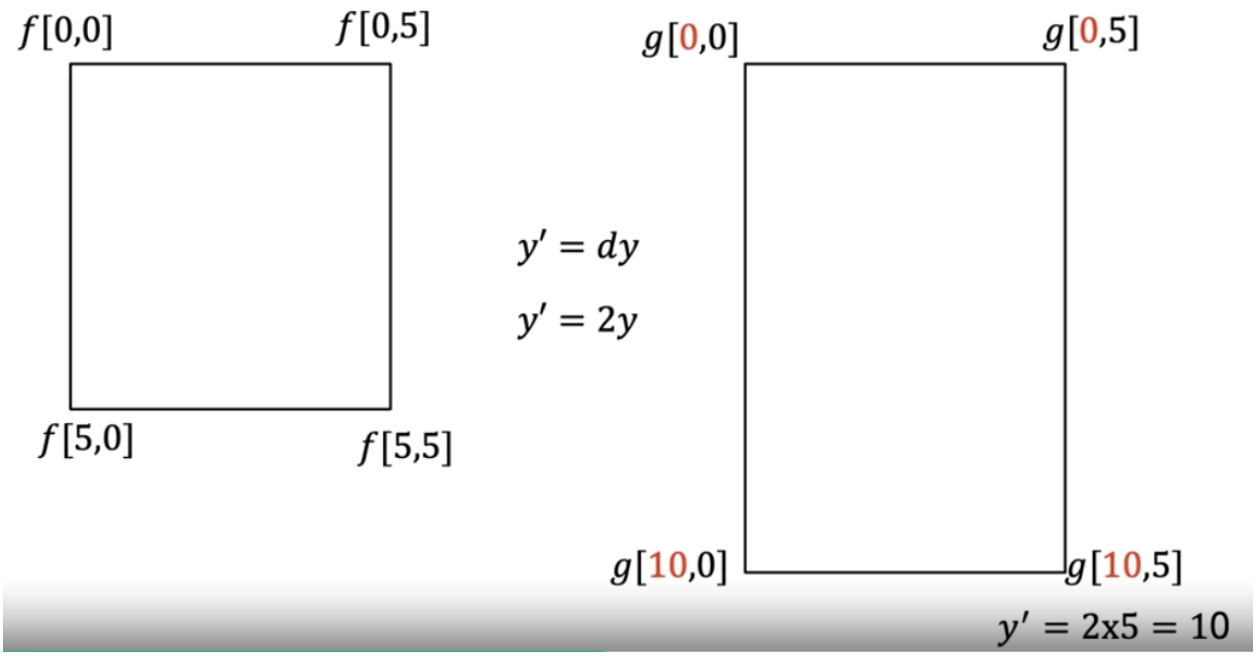
-Scaling

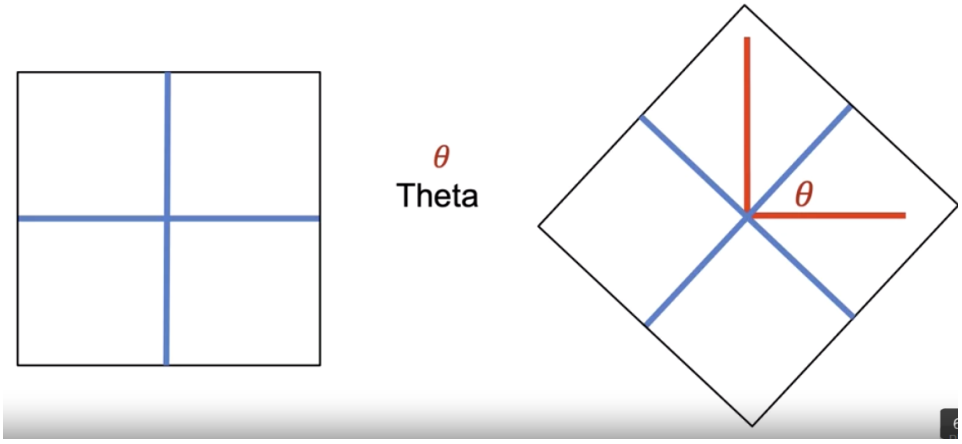
-Rotation

-translation


$f(y,x)$
$y,x$
Geometric Transformation
$g(y',x')$
$y',x'$


$x' = ax$
$x' = 2x$
$f[0,0]$
$f[0,5]$
$f[5,0]$
$f[5,5]$
$g[0,0]$
$g[0,10]$
$g[5,0]$
$g[5,10]$
$x' = 2x5 = 10$




```
new_dimensions = (new_width, new_height)

resized_image = cv2.resize(image, new_dimensions)
resized_image = cv2.resize(image, None, fx=2, fy=1)
fx:width
fy:height
```

$$x' = ax + t_x$$
$$y' = dy + t_y$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & 0 \\ 0 & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

$$\begin{bmatrix} a & 0 & t_x \\ 0 & b & t_y \end{bmatrix}$$

- The `cv2.warpAffine()` function in OpenCV is used for applying **an affine transformation** to an image. Affine transformations include operations such as translation, rotation, scaling, and shearing. This function takes an input image and a 2x3 transformation matrix as parameters and produces the transformed image.

```
import cv2
import numpy as np

# Load an image
image = cv2.imread('your_image.jpg')

# Define the translation matrix (tx, ty) - shift by 100 pixels to the right a
tx, ty = 100, 50
translation_matrix = np.float32([[1, 0, tx], [0, 1, ty]])

# Apply the affine transformation using cv2.warpAffine
transformed_image = cv2.warpAffine(image, translation_matrix, (image.shape[1]

# Display the original and transformed images
cv2.imshow('Original Image', image)
cv2.imshow('Transformed Image', transformed_image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

— `image.shape[1], image.shape[0])` extracts the width and height of the original image, and these values are used as the size of the output image.

— `cv2.getRotationMatrix2D()` is a function in the OpenCV library that is used to generate a 2D affine rotation matrix. This matrix can be used with `cv2.warpAffine()` to perform a rotation on an image.

- `center`: The rotation center, specified as a tuple (x, y).
- `angle`: The rotation angle in degrees.

- `scale`: An optional scaling factor.

```
import cv2
import numpy as np

# Load an image
image = cv2.imread('your_image.jpg')

# Define the rotation center (center of the image)
center = (image.shape[1] // 2, image.shape[0] // 2)

# Define the rotation angle (clockwise)
angle = 45

# Define the scaling factor (optional)
scale = 1.0

# Get the rotation matrix
rotation_matrix = cv2.getRotationMatrix2D(center, angle, scale)

# Apply the affine transformation using cv2.warpAffine
rotated_image = cv2.warpAffine(image, rotation_matrix, (image.shape[1], image

# Display the original and rotated images
cv2.imshow('Original Image', image)
cv2.imshow('Rotated Image', rotated_image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

## Spatial Operations: Convolution:Linear filtering

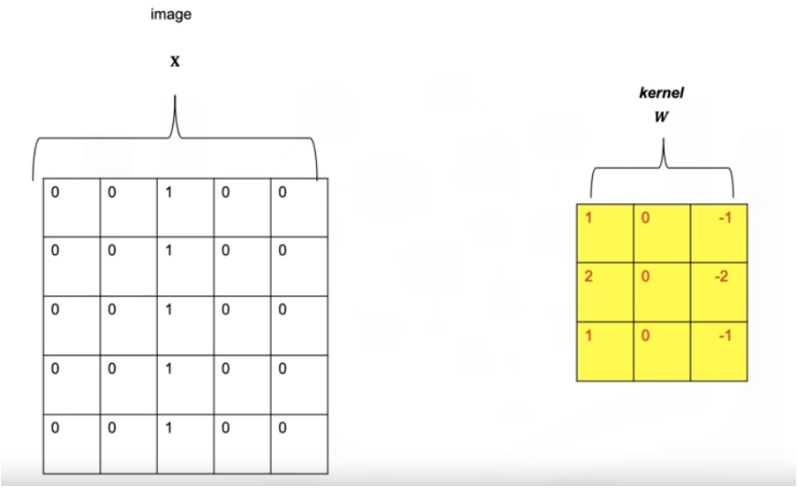
```
cv2.filter2D(src, ddepth, kernel[, dst[, anchor[, delta[, borderType]]]])
```

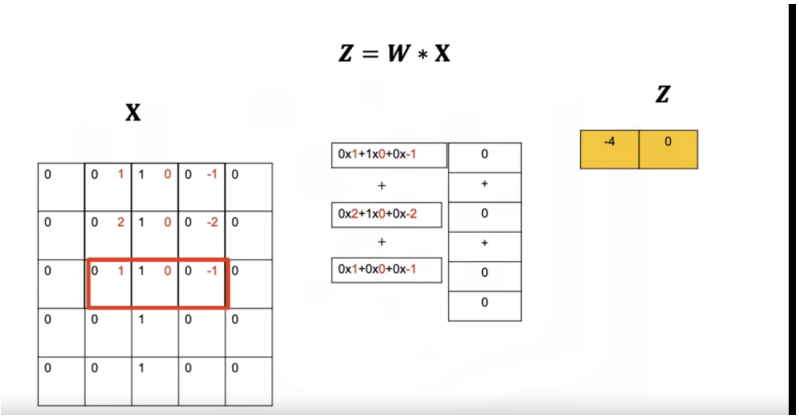
# What convolution

$$z = wx$$

$$Z = W * X$$

kernel



$$Z$$

| | | |
|---|---|---|
| -4 | 0 | 4 |
| -4 | 0 | 4 |
| -4 | 0 | 4 |

- Edge Detection:

```
ddepth: Desired depth of the destination image (use -1 for the same depth as
# Load an image
image = cv2.imread('your_image.jpg')

# Define a 3x3 kernel for a simple blur
kernel = np.ones((3, 3), np.float32) / 9  # averaging kernel

# Apply the convolution using cv2.filter2D
blurred_image = cv2.filter2D(image, -1, kernel)
blurred_image = cv2.GaussianBlur(image, (5, 5), 0)
```

**Gaussian Blur ():**

- **Purpose:** Primarily used for blurring or smoothing an image.
- **Kernel:** Employs a Gaussian kernel, which is a bell-shaped curve. The standard deviation (`sigma`) parameter controls the width of the curve, influencing the extent of blurring.

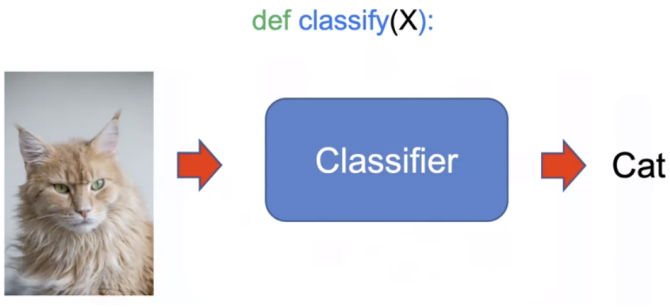
**Filter2D ():**

- **Purpose:** General-purpose convolution operation that allows you to apply a custom convolution kernel to an image. It can be used for various image processing tasks, including blurring, sharpening, edge detection, etc.
- **Kernel:** Any 2D convolution kernel can be used. The user can define a matrix that specifies how the neighborhood of each pixel should be weighted.

## Image Classification:

**-Image classification is the process of taking an image or picture and getting a computer to automatically classify it.**

**-Computers can't understand images but they can understand the intensity values of a digital image. We will use the intensity values to classify the image.**




- **Challenges:**

—change in view point

—illumination

—Deformation

—Occlusion

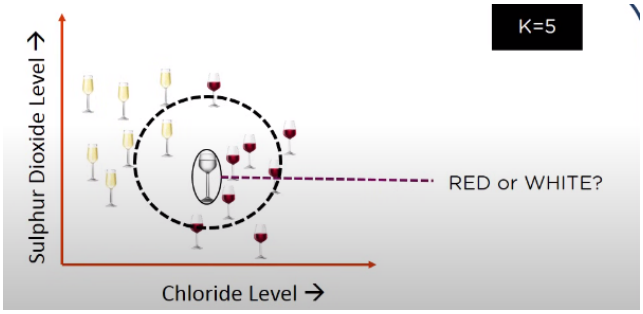
—Background Clutter

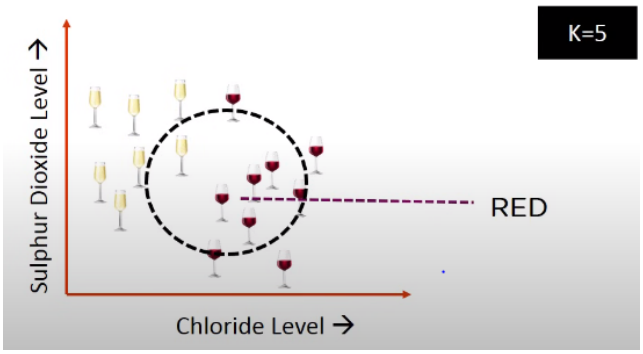
- **K-Nearest Neighbors**
- **Feature Extraction**
- **Linear classifiers**

## K-NN:

- **K- nearest neighbor. It is one of the simplest classification algorithms. KNN classifies the unknown data points by finding the most common classes in the k- nearest examples**
- **K in KNN is a parameter that refers to the number of nearest neighbors to include in the majority voting process**

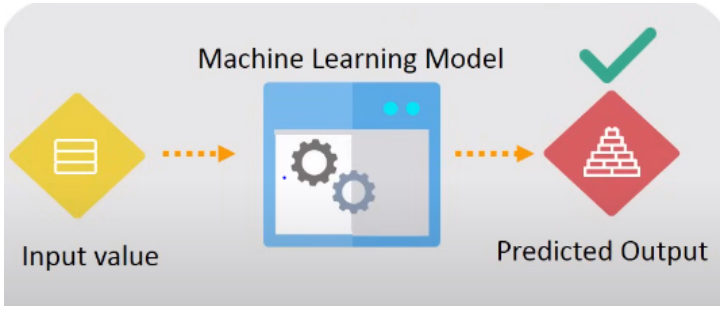
Exemple :

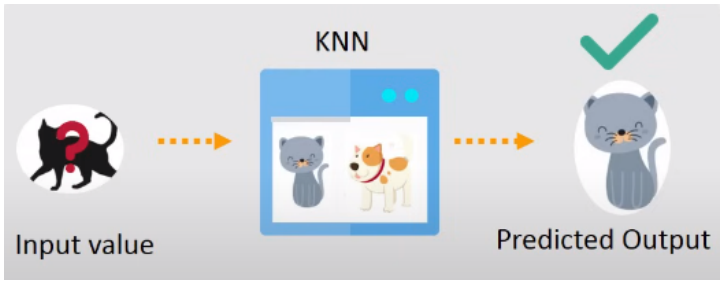





Here, the unknown point would be classified as red, since 4 out of 5 neighbors are red

- **A data point is classified by majority votes from its k nearest neighbors**
- **Machine Learning Model: make predictions by learning from past data available .**



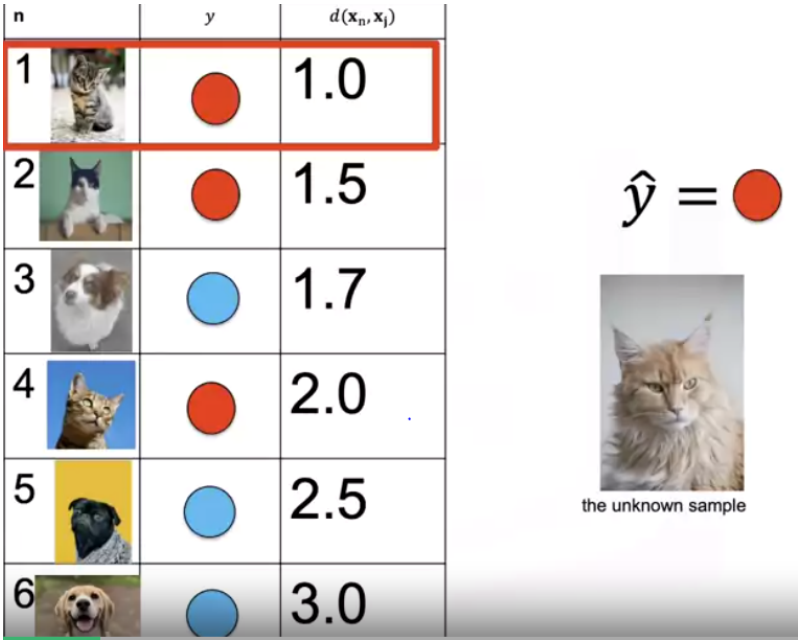

-To choose a value of k:

- sqrt(n):n is the total number of data points
- odd value of k is selected to avoid confusion between two classes of data

-The Euclidean Distance formula :

$$dist(d) = \sqrt{(x - a)^2 + (y - b)^2}$$

- A positive integar K is specified ,along with a new sample
- we select the k entries in our database which are closest to the new sample
- **we find the most common classification of these entries**
- **this is the classification we give to the new sample**


- We have a set of images we will refer to them as **training samples**
- Separating data into **training and testing sets** is an important part of model evaluation. We use the test data to get an idea how our model will perform in the real world When we split a data set, usually the larger portion of data is used for training and a smaller part is used for testing.we use a training set to build a model, we then use a testing set to evaluate model performance

- we have also the **unknown sample**
- We will use the euclidean  distance to predict the label of the unknown class. As this is **prediction** of the class we use y hat,the hat means it's **an estimate**, We calculate the distance from our unknown sample, we find **the nearest point ( the minimum  euclidean distance)** or nearest neighbour we assign the label to the unknown sample
- **Accuracy** (**précision)**shows us how good our method works i,e the average number of times our model got it correct

| Samples: n | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| $y$ | 1 | 0 | 1 | 0 |
| $\hat{y}$ | 1 | 0 | 0 | 1 |
| *Correct* | 1 | 1 | 0 | 0 |

$$= \frac{1}{4}(1+1+0+0) = 0.5$$

y: The actual class Label

y^:The predicted class Label  determined by the model

accuracy =0.5

- To select **k ,**we use a subset called the validation data to determine the best K, this is called a hyper parameter. To select **the Hyperparameter** we **split our data set into three parts**, the **training set, validation set, and test set**
- We select the hyperparameter K that **maximizes accuracy on the validation set.**
- We use the training set for different hyperparameters
- We use the test data to see how the model will perform on the real world

## Linear Classifier:

- Learnable parameters:

z=wX+b

X=sample

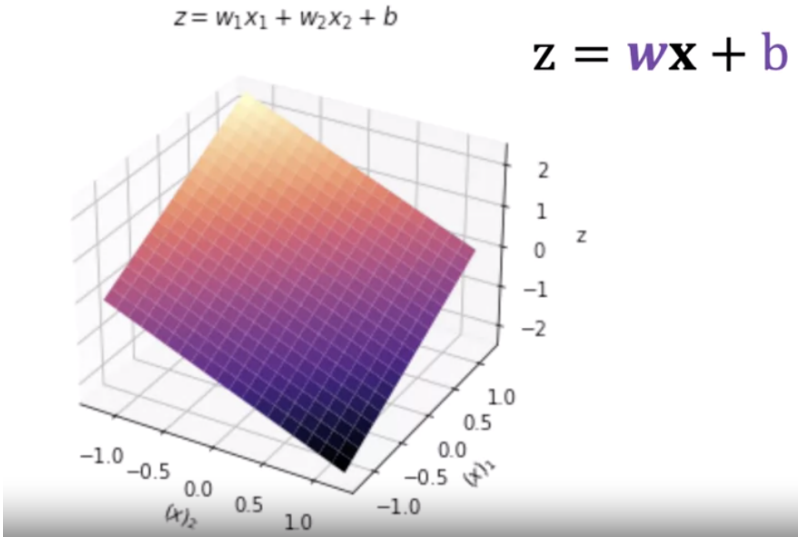
z=w1x1+w2x2+……+b

x1=algebra

—w represents the **weight term** and b represents the **bias term**

—For arbitrary dimensions, this equation generalizes to a **hyperplane**. You can represent this equation as a dot product of **row vector w** and **image x**. This is called **the decision plane**

—In two dimensions we can plot the equation as a plane.




- **Image Feature:**

—Features are measurements taken from the image that help with classification

—Classifying an image involves the relationship between pixels; a slight change in the image affects this relationship

—One way too overcome this problem, to split image into sub-images and calculate the histogram for each sub-image

—Histogram of oriented gradients – H.O.G., is one of many types of features that have been developed over the years

— Linear classifiers are commonly used in both machine learning and computer vision tasks. A linear classifier is a type of algorithm that makes predictions by combining linear combinations of input features. The decision boundary, which separates different classes in the input space, is a hyperplane.

Here are a few examples of linear classifiers and their applications in both machine learning and computer vision:

1. **Logistic Regression:**
    - **Machine Learning:** Logistic regression is a popular linear classifier used for binary classification tasks. It models the probability that an instance belongs to a particular class.
    - **Computer Vision:** Logistic regression can be used for tasks such as image classification, where the goal is to categorize images into different classes.
2. **Support Vector Machines (SVM):**
    - **Machine Learning:** SVM is a versatile linear classifier that can be used for both binary and multiclass classification. It aims to find a hyperplane that maximally separates different classes.

- **Computer Vision:** SVMs are commonly used in computer vision tasks such as image classification and object detection.

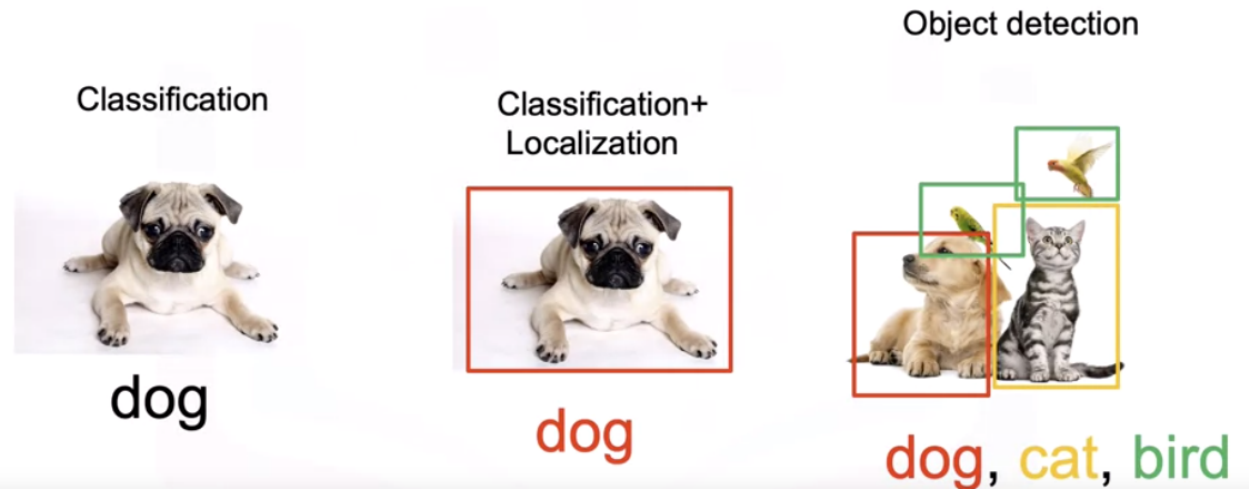
3. **Perceptron:**
   - **Machine Learning:** The perceptron is a basic linear classifier that can be used for binary classification. It learns a linear decision boundary and updates its weights to correctly classify instances.
   - **Computer Vision:** Perceptrons can be used for simple image classification tasks, although more complex models like neural networks are often preferred for more challenging computer vision problems.
4. **Linear Support Vector Machine (Linear SVM):**
   - **Machine Learning:** Similar to traditional SVM, linear SVM focuses on linear decision boundaries. It is used for classification tasks and aims to find the hyperplane with the maximum margin between classes.
   - **Computer Vision:** Linear SVM can be applied to image classification problems where the input features are linearly separable.
5. **Linear Discriminant Analysis (LDA):**
   - **Machine Learning:** LDA is a linear classifier used for both binary and multiclass classification. It models the distribution of classes and computes linear combinations of features to make predictions.
   - **Computer Vision:** LDA can be applied to tasks such as face recognition and object recognition in images.

While linear classifiers are effective for certain tasks, they may struggle with complex, nonlinear relationships in data. In such cases, more advanced models like non-linear classifiers or deep neural networks might be more suitable for machine learning and computer vision applications.

**Next Step** : —**Machine Learning in Computer Vision**

**—Deep Learning in Computer Vision**

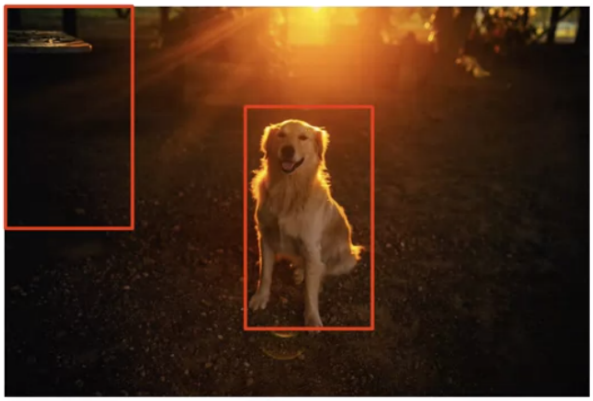
## Object Detection:



—In object detection problems, we generally have to find all the possible objects in the image

- **Sliding window detection Algorithm :**

—If we want to detect a dog we consider a **fixed window size**, if chosen properly, the dog will occupy most of the window, this is essentially a sub image that we would like to classify as a dog the other sub images would be classified as background each image that does not contain the dog would be considered a **background class**

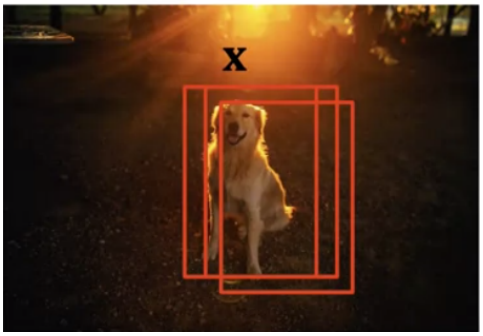



—The sliding window algorithm is a more systematic approach We start in one region in the image, classify that Sub-image We then shift the window and classify the next sub-image We repeat the process When we get to the horizontal border We move a few pixels down in the Vertical direction and repeat the process When the object occupies most of the window, it will be **classified as a dog**

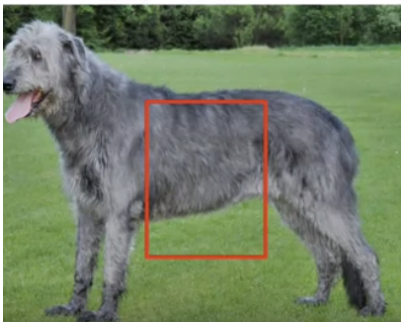


- **Problems of Sliding Window:**

- Overlapping Boxes



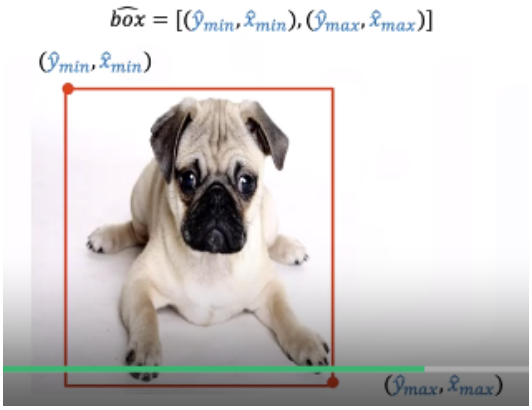

- Object sizes



- Overlapping objects

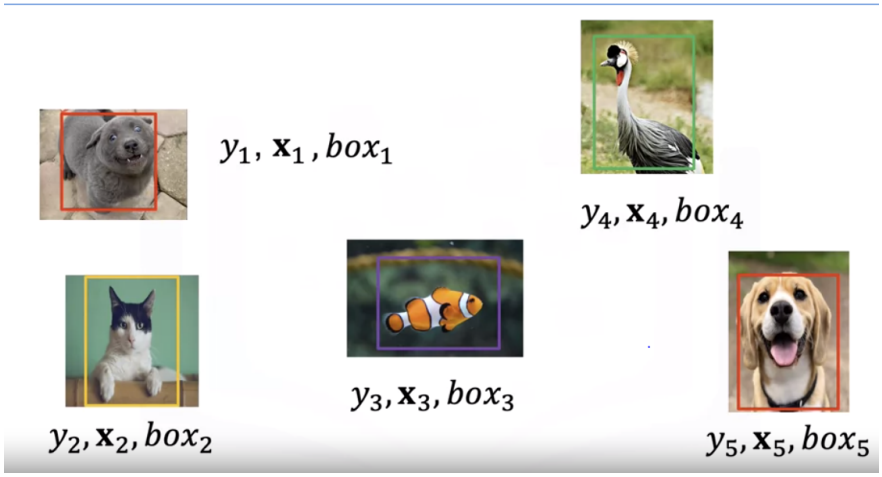
- **Bounding Box algorithm:**

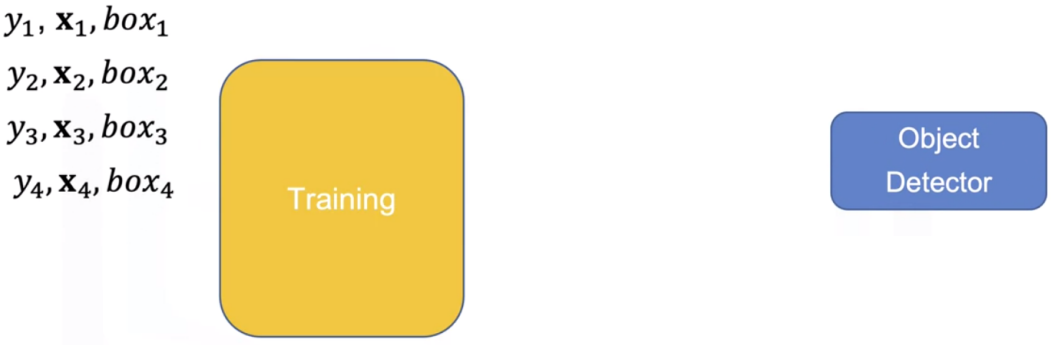
—The goal of object detection is to predict these points, so we add a “hat” to indicate it’s a prediction




- **The Bounding Box Pipeline:**

—It's  Like classification, we have the class y and x. We also have the bounding box, just like classification we have a dataset of Classes and their Bounding boxes Similar to classification, we use the dataset to train the model, we include the box coordinates, the result is an object detector with updated learning parameters




- **Haar Classifier:**

-It’s a machine learning method

-Trained on positive and negative images

-After millions of training images are fed to the system, the classifier begins by extracting features from each image. Haar wavelets are convolution kernels used to extract features

Roadmap In Computer Vision:

## Motion Detection and Tracking Using Opencv Contours:

```
(x,y,z,t)=cv2.boundingRect(contour)
```

=⇒ Find the bounding rectangle around a given contour. It calculates the coordinates of the top-left corner (x, y) and the dimensions (width, height) of the rectangle that encloses the contour.

-This function is commonly used in object detection and tracking applications. Once you have the bounding rectangle, you can use it to draw a rectangle around the detected object or to extract a region of interest (ROI) from the image.

- Contours are regions of interest in an image, and their area indicates how much space they cover.
- The condition `cv2.contourArea(contour) < 700` is a filter to exclude small contours. If the area of a contour is less than 700 pixels, it is considered small.
- **Small contours** are often ignored or skipped because they might represent **noise**, small artifacts, or objects that are not of interest.
- The purpose is usually to focus on larger and more significant contours that correspond to meaningful objects or features in the image.

## Detect Simple Geometric Shapes using OpenCV in Python

`_, thresh = cv2.threshold(gray, 100, 255, cv2.THRESH_BINARY)`

- If a pixel in the `gray` image has a **value less than 100, it will be set to 0 (black).**
- If a pixel in the `gray` image has a **value greater than or equal to 100, it will be set to 255 (white).**

So, the resulting `thresh` image will be a **binary image** where pixels are either 0 or 255

— The variable `contours` will contain a list of contours, where each contour is represented as an array of points.

```
contours, hierarchy = cv2.findContours(image, mode, method, contours=None, of
```

- `image`**:** This is the input binary image (black and white). In your case, it's the thresholded image ( `thrash` in your code).
- `mode`**:** It specifies the retrieval mode of the contours. There are several modes, but the commonly used ones are:
  - `cv2.RETR_EXTERNAL`: Retrieves only the extreme outer contours.

- `cv2.RETR_LIST`: Retrieves all contours without any hierarchical information.
- `cv2.RETR_TREE`: Retrieves all contours and reconstructs a full hierarchy of nested contours.

- `method`**:** It specifies the contour approximation method. Common options include:
  - `cv2.CHAIN_APPROX_SIMPLE`: Removes all redundant points and compresses the contour, saving memory.
  - `cv2.CHAIN_APPROX_NONE`: Stores all the contour points. It may consume more memory.
- `contours`**:** (Optional) Output parameter that receives the contours. This is a list of contours, where each contour is represented as an array of points.
- `hierarchy`**:** (Optional) Output parameter that receives information about the hierarchical structure of the contours. It is useful when using hierarchical retrieval modes like `cv2.RETR_TREE`

⇒**The variable `contours` will contain a list of contours, where each contour is represented as an array of points**

```
approx = cv2.approxPolyDP(contour, 0.02 * cv2.arcLength(contour, True), True)
```

-The `cv2.approxPolyDP` function returns **a list of vertices** that represents **the approximated polygonal curve**. Each vertex is a **point (x, y)** in the image.

-The structure of `approx` is a **list where each element is a NumPy array** containing the (x, y) coordinates of a vertex in the approximated polygon. So, if `approx` has `n` vertices, it will be a list of `n` arrays.

- `len(approx)` gives you the **number of vertices** in that particular approximated polygon

```
    x=approx.ravel()[0]
    y=approx.ravel()[1]
```

- `ravel()`**, it flattens the array, meaning it transforms a multi-dimensional array into a one-dimensional array**. In the case of `approx`, each vertex is represented by a pair of coordinates (x, y) in the flattened array.

-We Find contours only  in the binary image

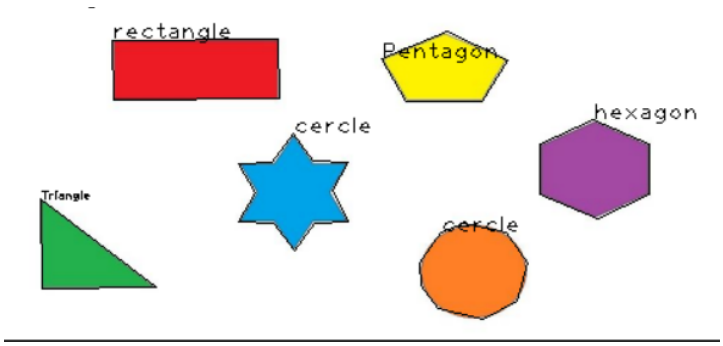
```
contours, _ = cv2.findContours(thresh, cv2.RETR_TREE, cv2.CHAIN_APPROX_NONE)
```

1. **Approximate Contours:**

```
# Loop through each contour and approximate the polygon
for contour in contours:
    epsilon = 0.02 * cv2.arcLength(contour, True)
    approx = cv2.approxPolyDP(contour, epsilon, True
```

2. **Shape Identification:**

```
# Based on the number of vertices in the approximated polygon, classify
the shape
if len(approx) == 3:
    # Triangle
elif len(approx) == 4:
    # Rectangle or Square based on aspect ratio
elif len(approx) == 5:
    # Pentagon
# Add more conditions for other shapes as needed
```




## Understanding image Histograms using OpenCV Python

- Creating a 2D array of zeros:

```
zeros_2d = np.zeros((3, 4), dtype=int)
print(zeros_2d)
```

Output:

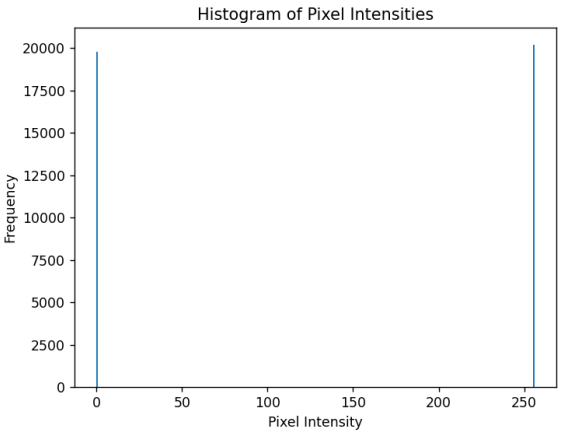
```
[[0 0 0 0]
 [0 0 0 0]
 [0 0 0 0]]
```

```
plt.hist(img_flattened, bins=256, range=[0, 256], color='black', alpha=0.7
```

x-axis: pixel's intensity

y-axis:frequency:number of pixels

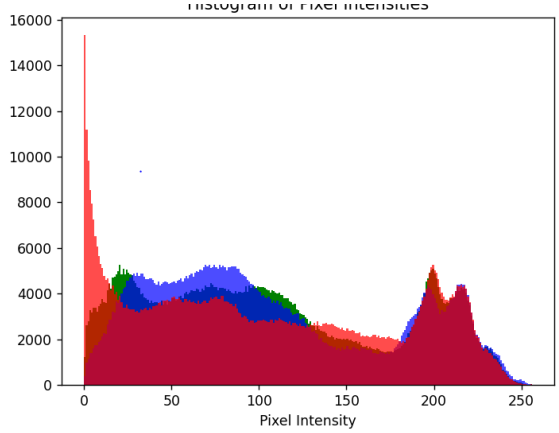
```
img=np.zeros((200,200),np.uint8)
cv2.rectangle(img,(0,0),(200,100),255,-1)
plt.hist(img.ravel(),255,[0,256])
```



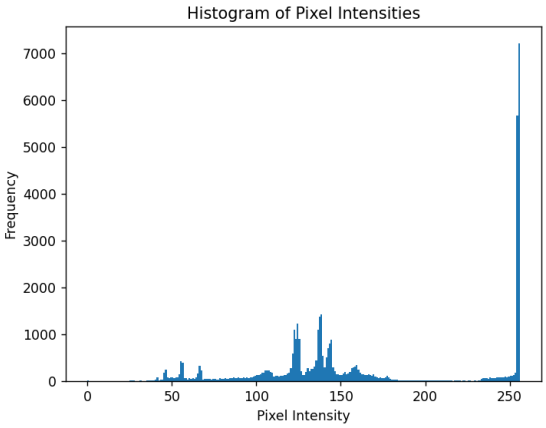

```
b, g, r = cv2.split(image)
```

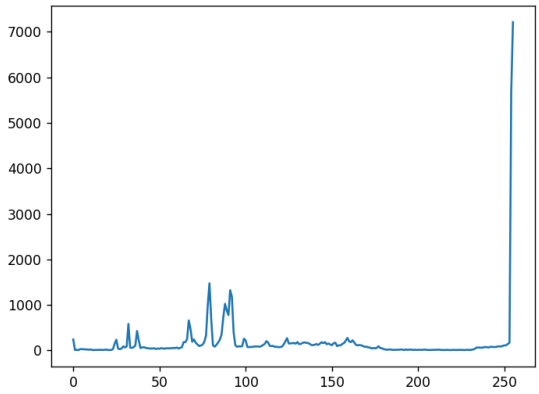
-This function returns three separate matrices `b`, `g`, and `r`, representing **the blue, green, and red channels**, respectively. Each of these matrices will be a **2D array** with the same size as the input image but containing the intensity values for only one color channel.

```
b, g, r = cv2.split(img)

# Plot histograms for each channel
plt.hist(g.ravel(), bins=256, range=[0, 256], color='green')
plt.hist(b.ravel(), bins=256, range=[0, 256], color='blue', alpha=0.7)
plt.hist(r.ravel(), bins=256, range=[0, 256], color='red', alpha=0.7)
```




```
hist=cv2.calcHist([img],[0],None,[256],[0,256])

x_axis:pixel's intensity :[0,255]
y_axis:frequency or number ofpixels corresponding to each pixel's intensit
 plt.hist(gray.ravel(),bins=256,range=[0,256])
=>Create a histogram: x _axis: pixel's intensity :[0,255]
                      y_axis:frequency
==>
```






`cv2.calcHist(images, channels, mask, histSize, ranges[, hist[, accumulate]])`

- `cv2.calcHist()` is to calculate the histogram of an image or a set of images. The histogram is a representation of the distribution of pixel intensities in the image

- `channels` : This is the channel or channels for which you want to calculate the histogram. For grayscale images, it is [0]. For color images, you can pass [0], [1], [2] to calculate the histogram of blue, green, or red channel respectively.

- `histSize` : This represents the number of bins in the histogram. For full scale, use [256].

`ranges` : This is the range of intensity values. For grayscale images, it is usually [0, 256].

```
plt.hist(x, bins=None, range=None)
```

- `x` : The input data. This is the data for which the histogram will be calculated.
- `bins` : The number of bins (intervals) to use in the histogram. It can be an integer specifying the number of bins, or a sequence specifying the bin edges.
- `range` : The lower and upper range of the bins. Values outside this range are ignor

## Template matching using OpenCV in Python

```
cv2.matchTemplate(image, templ, method[, result[, mask]])
```

- `image` : The input image where the search will be conducted.
- `templ` : The template image that you want to find in the input image. It must be smaller than the input image.
- `method` : The matching method. It specifies the way the template is compared with the image. Common methods include `cv2.TM_SQDIFF` , `cv2.TM_SQDIFF_NORMED` , `cv2.TM_CCORR` , `cv2.TM_CCORR_NORMED` , `cv2.TM_CCOEFF` , and `cv2.TM_CCOEFF_NORMED` .

- `result` **(optional)**: The output map of comparison results. It is a single-channel, **floating-point array.**
- The result of the `cv2.matchTemplate()` function is a grayscale image where each pixel corresponds to the result of the template matching at that location.
- The elements of the floating-point array obtained from `cv2.matchTemplate()` represent the match scores at each position in the result matrix. These match scores indicate how well the template matches the corresponding sub-region in the input image.
- **Higher Score:** A higher match score at a particular position indicates a stronger correlation between the template and the image at that location.
- **Lower Score:** Conversely, a lower match score means a weaker correlation.

The interpretation of the match scores depends on the matching method used, which is specified by the `method` parameter in `cv2.matchTemplate()`. Common methods include:

- `cv2.TM_SQDIFF`: The closer the match, the smaller the score.
- `cv2.TM_CCORR`: The closer the match, the larger the score.
- `cv2.TM_CCOEFF`: The score can be positive or negative, with a larger positive value indicating a better match.

## Open CV Projects :(Important  to check)

https://www.geeksforgeeks.org/template-matching-using-opencv-in-python/

```
ar=np.array([1,2,3,4,5])
res=np.where(ar>2)
Output:
(array([2, 3, 4], dtype=int64),) as ar[2]=3>2 and ar[3]=4>2
=>The np.where() function in NumPy is used to return the indices of elemen
```

## Haar Classifier:

- they are trained using lots of positive and negative images

1. **Positive Images:**
    - **Contain the Object of Interest:** Positive images are examples that contain the object or class you want to detect.
    - **Labeled Examples:** Each positive image is labeled to indicate the presence of the object of interest.
    - **Varied Poses and Conditions:** Positive images should cover a range of poses, orientations, lighting conditions, and backgrounds relevant to the application.
2. **Negative Images:**
    - **Do Not Contain the Object of Interest:** Negative images are examples that do not contain the object or class you want to detect.
    - **Labeled as Absence:** Negative images are labeled to indicate the absence of the target object.

```
face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcasca
faces = face_cascade.detectMultiScale(gray, scaleFactor=1.3, minNeighbo
for (x, y, w, h) in faces:
        cv2.rectangle(frame, (x, y), (x+w, y+h), (0, 255, 0), 3)
```

- **Face Detection :**

-Initializes a `CascadeClassifier` object in OpenCV using a pre-trained Haar Cascade classifier for frontal face detection.

-The `detectMultiScale` method returns a list of rectangles (represented as (x, y, width, height)) where faces are detected in the image. Each rectangle corresponds to a detected face, and the list ( `faces` ) contains all the detected faces in the image.

- **Harris Corner Detector:**

-Corner :Point where two edges meet ,rapid changes  of image intensity in two directions within a small region

1.determine which windows produce very large variations in intensity when moved in both X and Y directions.

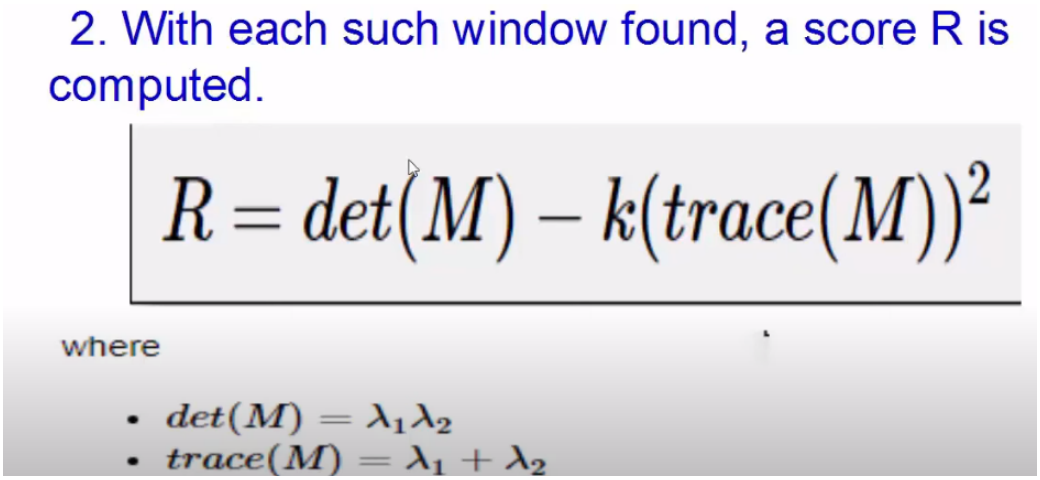
$$E(u,v) \cong [u,v] \quad M \quad \begin{bmatrix} u \\ v \end{bmatrix}$$

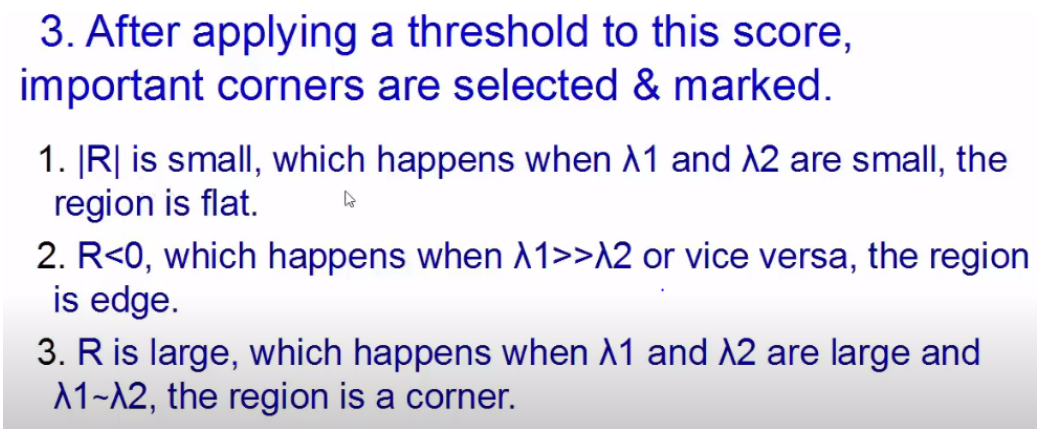
where $M$ is a 2×2 matrix computed from image derivatives:

$$M = \sum_{x,y} w(x,y) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}$$

Windowing function - computing a weighted sum (simplest case, w=1)

Note: these are just products of components of the gradient, Ix, Iy




- Background segmentation is a common task in computer vision that involves separating foreground objects from the background in a video sequence

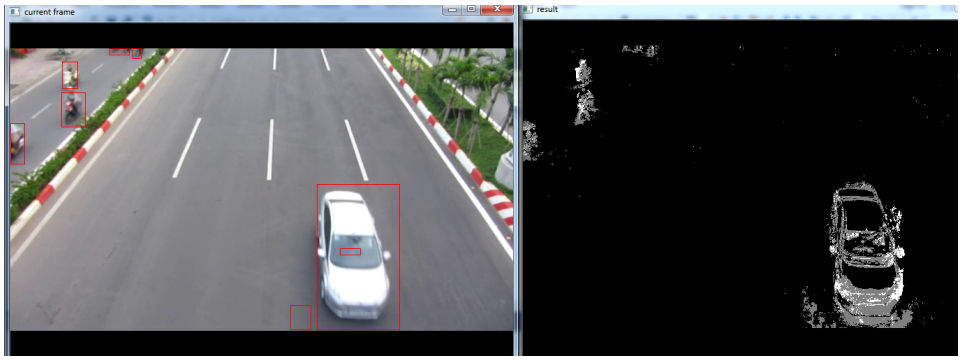
## Background segmentation

—It involves separating foreground objects from the background in a video sequence.

- **BackgroundSubtractorMOG:**

This algorithm is based on a mixture of Gaussians (MOG) model. It models each pixel as a mixture of several Gaussian distributions to represent both the background and foreground.

```
bg_subtractor = cv2.bgsegm.createBackgroundSubtractorMOG()
```


current frame
result