

EX.NO:1	FUNDAMENTAL OF COMPUTER VISION
DATE:07.07.23	

AIM:

To know the fundamental of computer vision.

FUNDAMENTAL OF COMPUTER VISION:

Computer Vision, often abbreviated as CV, is defined as a field of study that seeks to develop techniques to help computers “see” and understand the content of digital images such as photographs and videos.

The problem of computer vision appears simple because it is trivially solved by people, even very young children. Nevertheless, it largely remains an unsolved problem based both on the limited understanding of biological vision and because of the complexity of vision perception in a dynamic and nearly infinitely varying physical world.

In this post, you will discover a gentle introduction to the field of computer vision.

After reading this post, you will know:

- The goal of the field of computer vision and its distinctness from image processing.
- What makes the problem of computer vision challenging.
- Typical problems or tasks pursued in computer vision.

Computer vision is distinct from image processing.

Image processing is the process of creating a new image from an existing image, typically simplifying or enhancing the content in some way. It is a type of digital signal processing and is not concerned with understanding the content of an image.

A given computer vision system may require image processing to be applied to raw input, e.g. pre-processing images.

Examples of image processing include:

- Normalizing photometric properties of the image, such as brightness or color.
- Cropping the bounds of the image, such as centering an object in a photograph.
- Removing digital noise from an image, such as digital artifacts from low light levels.

The 2010 textbook on computer vision titled “Computer Vision: Algorithms and Applications” provides a list of some high-level problems where we have seen success with computer vision.

- Optical character recognition (OCR)
- Machine inspection
- Retail (e.g. automated checkouts)
- 3D model building (photogrammetry)
- Medical imaging
- Automotive safety
- Match move (e.g. merging CGI with live actors in movies)
- Motion capture (mocap)
- Surveillance
- Fingerprint recognition and biometrics

It is a broad area of study with many specialized tasks and techniques, as well as specializations to target application domains.

RESULT:

Thus, we know the fundamentals of computer vision.

EX. NO:2	INTRODUCTION TO OPENCV
DATE:14.07.23	

AIM:

To know about the library opencv for computer vision.

Introduction to OpenCV:

OpenCV was started at Intel in 1999 by Gary Bradsky, and the first release came out in 2000. Vadim Pisarevsky joined Gary Bradsky to manage Intel's Russian software OpenCV team. In 2005, OpenCV was used on Stanley, the vehicle that won the 2005 DARPA Grand Challenge. Later, its active development continued under the support of Willow Garage with Gary Bradsky and Vadim Pisarevsky leading the project. OpenCV now supports a multitude of algorithms related to Computer Vision and Machine Learning and is expanding day by day.

OpenCV supports a wide variety of programming languages such as C++, Python, Java, etc., and is available on different platforms including Windows, Linux, OS X, Android, and iOS. Interfaces for high-speed GPU operations based on CUDA and OpenCL are also under active development.

OpenCV-Python is the Python API for OpenCV, combining the best qualities of the OpenCV C++ API and the Python language.

Open cv in python:

OpenCV-Python is a library of Python bindings designed to solve computer vision problems.

Python is a general purpose programming language started by Guido van Rossum that became very popular very quickly, mainly because of its simplicity and code readability. It enables the programmer to express ideas in fewer lines of code without reducing readability.

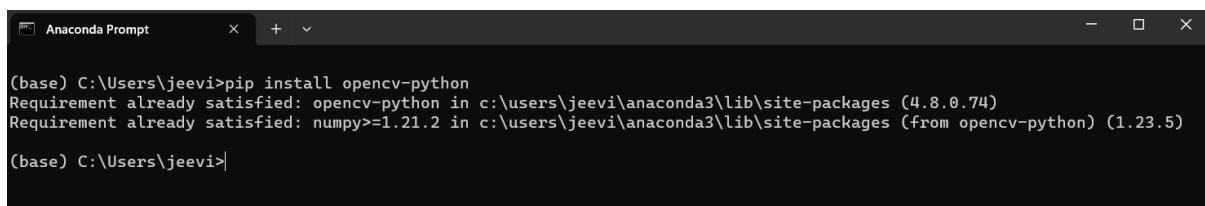
Compared to languages like C/C++, Python is slower. That said, Python can be easily extended with C/C++, which allows us to write computationally intensive code in C/C++ and create Python wrappers that can be used as Python modules. This gives us two advantages: first, the code is as fast as the original C/C++ code (since it is the actual C++ code working in

background) and second, it is easier to code in Python than C/C++. OpenCV-Python is a Python wrapper for the original OpenCV C++ implementation.

OpenCV-Python makes use of Numpy, which is a highly optimized library for numerical operations with a MATLAB-style syntax. All the OpenCV array structures are converted to and from Numpy arrays. This also makes it easier to integrate with other libraries that use Numpy such as SciPy and Matplotlib.

Step:1 Install the opencv python in anaconda prompt

Pip install opencv-python



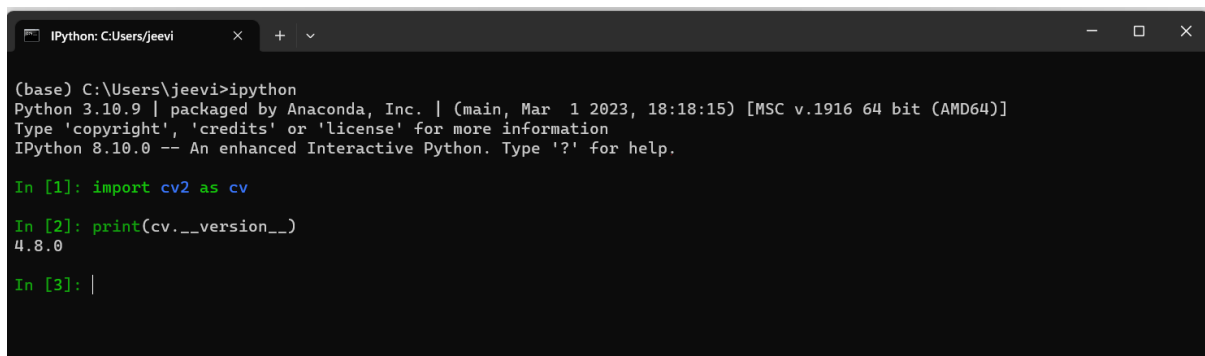
```
Anaconda Prompt
(base) C:\Users\jeevi>pip install opencv-python
Requirement already satisfied: opencv-python in c:\users\jeevi\anaconda3\lib\site-packages (4.8.0.74)
Requirement already satisfied: numpy>=1.21.2 in c:\users\jeevi\anaconda3\lib\site-packages (from opencv-python) (1.23.5)
(base) C:\Users\jeevi>
```

Open cv library is work only on ipython,shell,spyder.

If we run the opencv in jupyternotebook or colab the code will crash

Step:2 know the version of the opencv

Print(cv2.__version__)



```
IPython: C:\Users\jeevi
(base) C:\Users\jeevi>ipython
Python 3.10.9 | packaged by Anaconda, Inc. | (main, Mar 1 2023, 18:18:15) [MSC v.1916 64 bit (AMD64)]
Type 'copyright', 'credits' or 'license' for more information
IPython 8.10.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]: import cv2 as cv

In [2]: print(cv.__version__)
4.8.0

In [3]: |
```

RESULT:

We came to know about the opencv library in python and we installed the package and run it.

EX.NO:3	HOW TO READ, WRITE AND SHOW IMAGES IN OPENCV
DATE:28.07.23	

AIM:

To know how to read, write and show images in python using opencv.

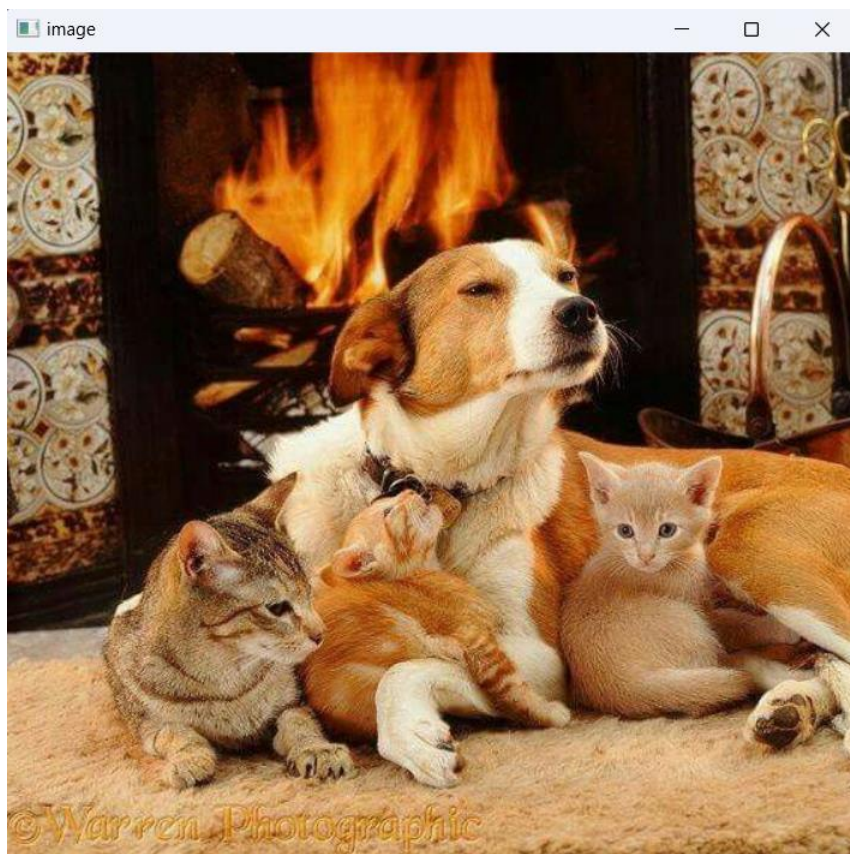
PROCEDURE:

Show the image in opencv:

Input:

```
In [6]: image=cv.imread(r"C:\Users\jeevi\Downloads\cat and dog.jpg")
...: cv.imshow("image", image)
...: cv.waitKey(0)
```

Output:



Write the image in opencv:

Input:

Here we crop the image and save the image

```
In [2]: import cv2 as cv

In [3]: image=cv.imread(r"C:\Users\jeevi\Downloads\cat and dog.jpg")

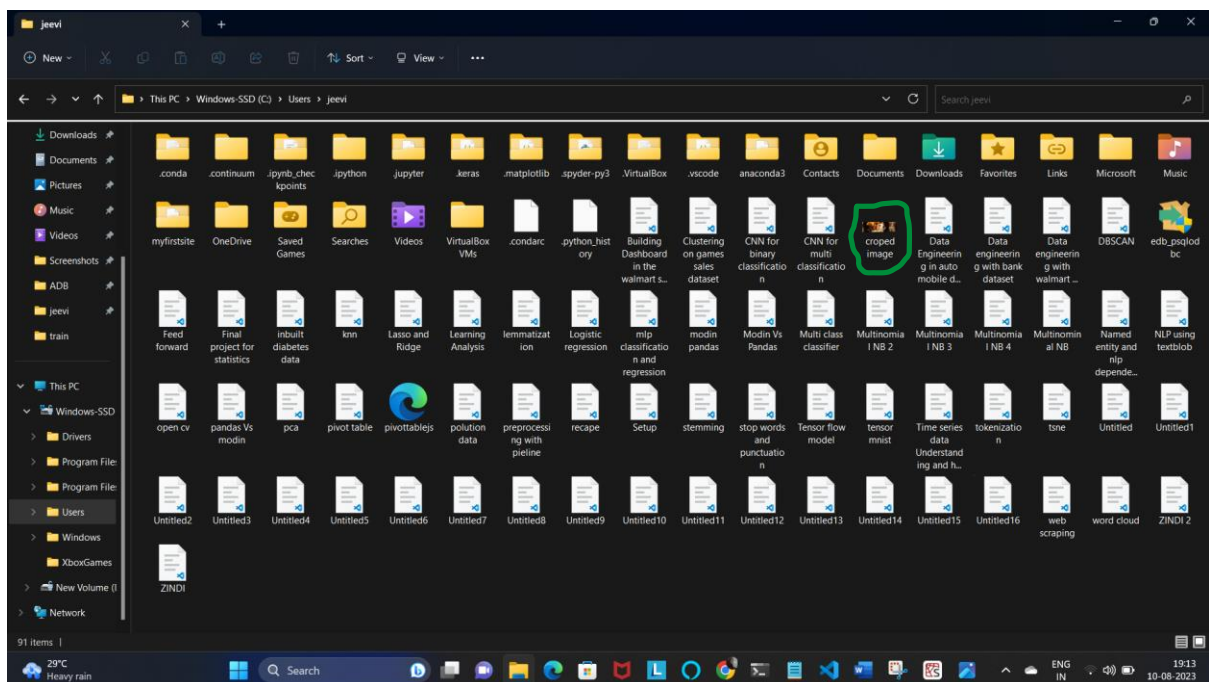
In [4]: crop_img = image[100:256, 50:824]
....: cv.imwrite("cropped image.jpg", crop_img)
....: cv.waitKey(0)
Out[4]: -1

In [5]:
```

Output:

Now we can see the image is written in our local system

C:\Users\jeevi\cropped image.jpg





Read the image using opencv:

Input:

```
Out[4]: -1  
  
In [5]: image=cv.imread(r"C:\Users\jeevi\Downloads\cat and dog.jpg")
```

Output:

```
In [6]: print(image)  
[[[ 4  43  87]  
  [ 27  66 110]  
  [ 60 100 142]  
  ...  
  [ 24  26  36]  
  [ 18  20  30]  
  [ 15  17  27]]  
  
  [[ 26  69 112]  
  [ 41  84 127]  
  [ 79 123 164]  
  ...  
  [ 28  30  40]  
  [ 23  25  35]  
  [ 14  16  26]]  
  
  [[ 25  74 118]  
  [ 15  65 107]  
  [ 24  76 116]  
  ...  
  [ 30  32  42]  
  [ 29  31  41]  
  [ 14  16  26]]  
  
  ...  
  
  [[110 170 224]  
  [ 98 158 212]  
  [100 160 214]  
  ...
```

RESULT:

We did the read, write and show in python using opencv.

EX.NO:4	IMAGE PROCESSING USING OPENCV
DATE:04.08.23	

AIM:

To do image processing using opencv in python.

PROCEDURE:

There are many steps in image processing in opencv here we use some technique of image processing.

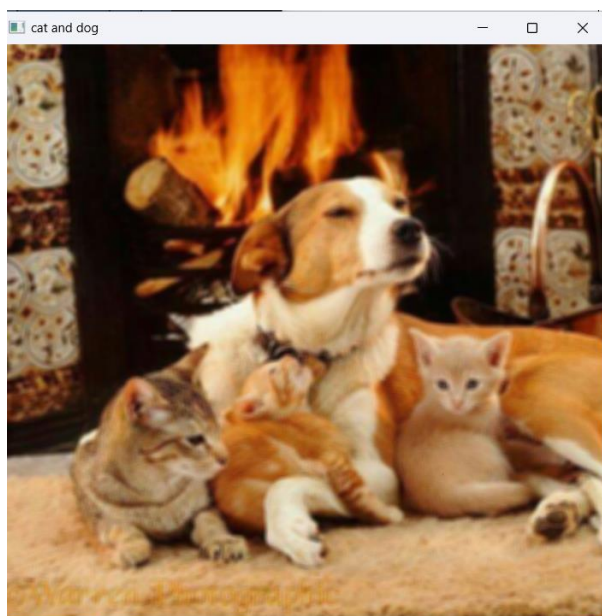
1. Blur the image:

Image Blurring This is done by convolving the image with a normalized box filter. It simply takes the average of all the pixels under kernel area and replaces the central element with this average.

Input:

```
In [10]: image=cv.imread(r"C:\Users\jeevi\Downloads\cat and dog.jpg")
...: blur = cv.blur(image,(5,5))
...: cv.imshow("cat and dog", blur)
...: cv.waitKey(0)
```

Output



2. Image resize:

We can resize the image that how we want the image

Input:

```
In [4]:
....: image=cv.imread(r"C:\Users\jeevi\Downloads\cat and dog.jpg")
....: dim=(200,300)
....: resized = cv.resize(image,dim, interpolation = cv.INTER_AREA)
....: cv.imshow("cat and dog", resized)
....: cv.waitKey(0)
....:
Out[4]: 13
In [5]:
```

Output:



3. Image threshold:

We need to change the image from RGB to grayscale image for our image processing.

Input:

```
In [7]: img = cv.imread(r"C:\Users\jeevi\Downloads\cat and dog.jpg", cv.IMREAD_GRAYSCALE)
....: ret,thresh1 = cv.threshold(img,127,255,cv.THRESH_BINARY)
....: cv.imshow("cat and dog", thresh1)
....: cv.waitKey(0)
```

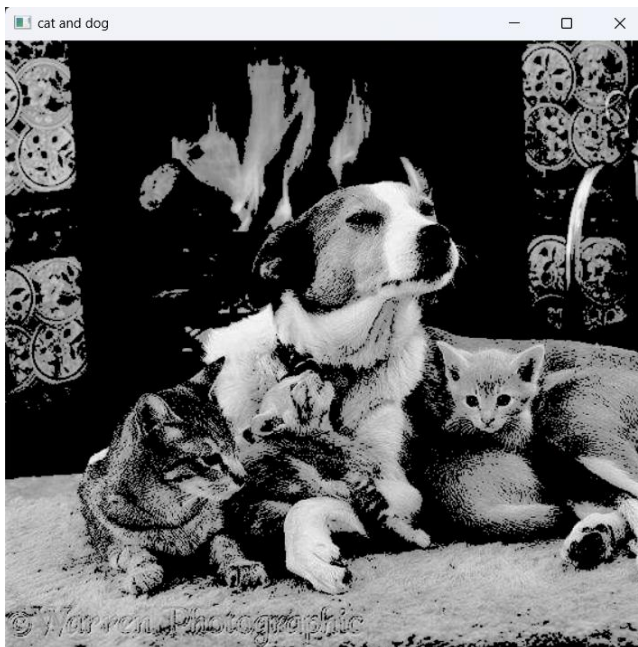
Output:



Input:

```
In [10]: img = cv.imread(r"C:\Users\jeevi\Downloads\cat and dog.jpg", cv.IMREAD_GRAYSCALE)
...: ret,thresh1 = cv.threshold(img,127,255,cv.THRESH_TOZERO)
...: cv.imshow("cat and dog", thresh1)
...: cv.waitKey(0)
```

Output:



RESULT:

We did some of the image processing using opencv in python

EX.NO: 5	IMAGE TRANSFORMATION I
DATE:18.08.23	

AIM:

To do the image transformation I in python using opencv.

PROCEDURE:**Translation:**

Translation is the shifting of an object's location.

Input:

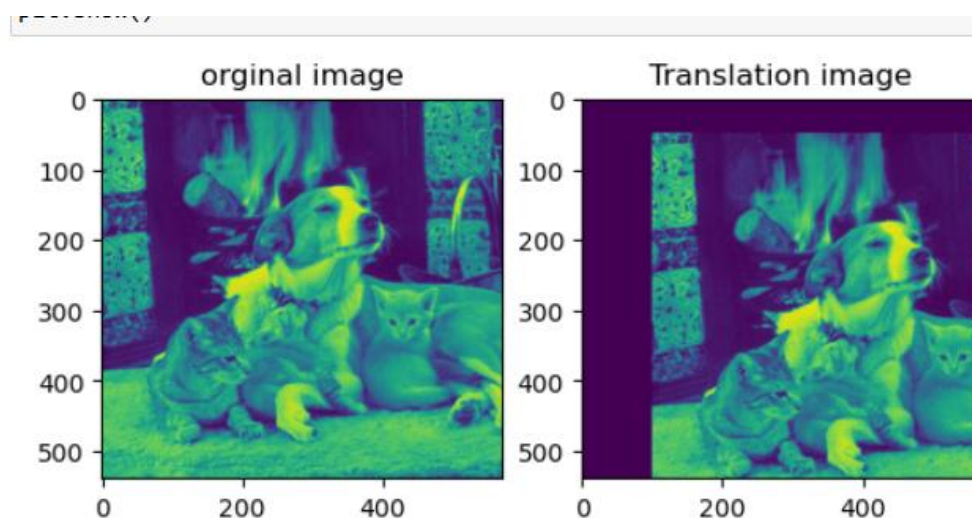
image translation

```
import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt
```

```
img = cv.imread(r"C:\Users\jeevi\Downloads\cat and dog.jpg", cv.IMREAD_GRAYSCALE)
rows,cols = img.shape
M = np.float32([[1,0,100],[0,1,50]])
dst = cv.warpAffine(img,M,(cols,rows))

plt.subplot(1,2,1),plt.imshow(img),plt.title("original image")
plt.subplot(1,2,2),plt.imshow(dst),plt.title("Translation image")
plt.show()
```

Output:



Rotation:

Rotating the image by transformation matrix.

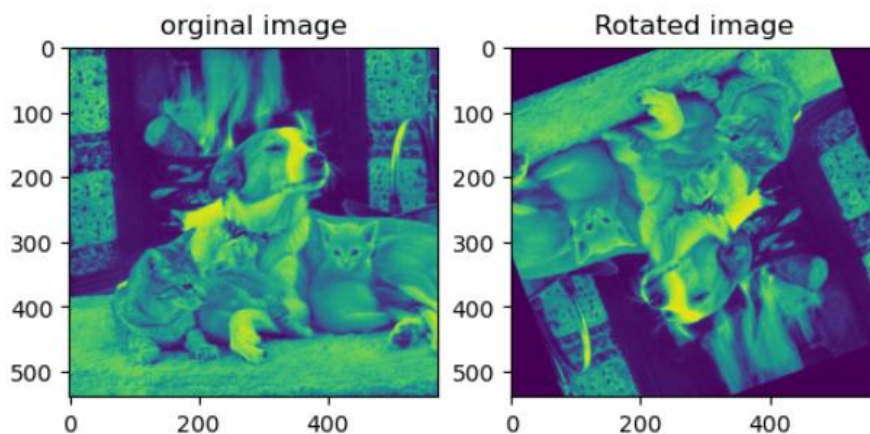
Input:

Rotation of image

```
] : img = cv.imread(r"C:\Users\jeevi\Downloads\cat and dog.jpg", cv.IMREAD_GRAYSCALE)
rows,cols = img.shape
M = cv.getRotationMatrix2D(((cols)/2.0,(rows)/2.0),200,1)
dst = cv.warpAffine(img,M,(cols,rows))

plt.subplot(1,2,1),plt.imshow(img),plt.title("original image")
plt.subplot(1,2,2),plt.imshow(dst),plt.title("Rotated image")
plt.show()
```

Output:



Perspective transformation:

For perspective transformation, you need a 3x3 transformation matrix. Straight lines will remain straight even after the transformation. To find this transformation matrix, you need 4 points on the input image and corresponding points on the output image. Among these 4 points, 3 of them should not be collinear. Then the transformation matrix can be found by the function [cv.getPerspectiveTransform](#). Then apply [cv.warpPerspective](#) with this 3x3 transformation matrix.

Input:

Perspective transformation

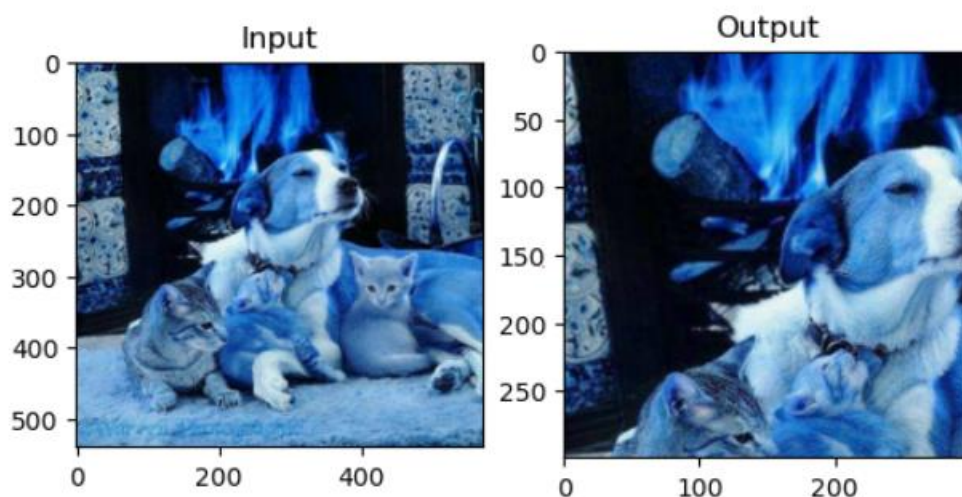
```
img = cv.imread(r"C:\Users\jeevi\Downloads\cat and dog.jpg")
rows,cols,ch = img.shape

pts1 = np.float32([[56,65],[368,52],[28,387],[389,390]])
pts2 = np.float32([[0,0],[300,0],[0,300],[300,300]])

M = cv.getPerspectiveTransform(pts1,pts2)
dst = cv.warpPerspective(img,M,(300,300))

plt.subplot(121),plt.imshow(img),plt.title('Input')
plt.subplot(122),plt.imshow(dst),plt.title('Output')
plt.show()
```

Output:



Scaling:

Scaling is just resizing of the image. OpenCV comes with a function [cv.resize\(\)](#) for this purpose. The size of the image can be specified manually, or you can specify the scaling factor. Different interpolation methods are used. Preferable interpolation methods are [cv.INTER_AREA](#) for shrinking and [cv.INTER_CUBIC](#) (slow) & [cv.INTER_LINEAR](#) for zooming. By default, the interpolation method [cv.INTER_LINEAR](#) is used for all resizing purposes.

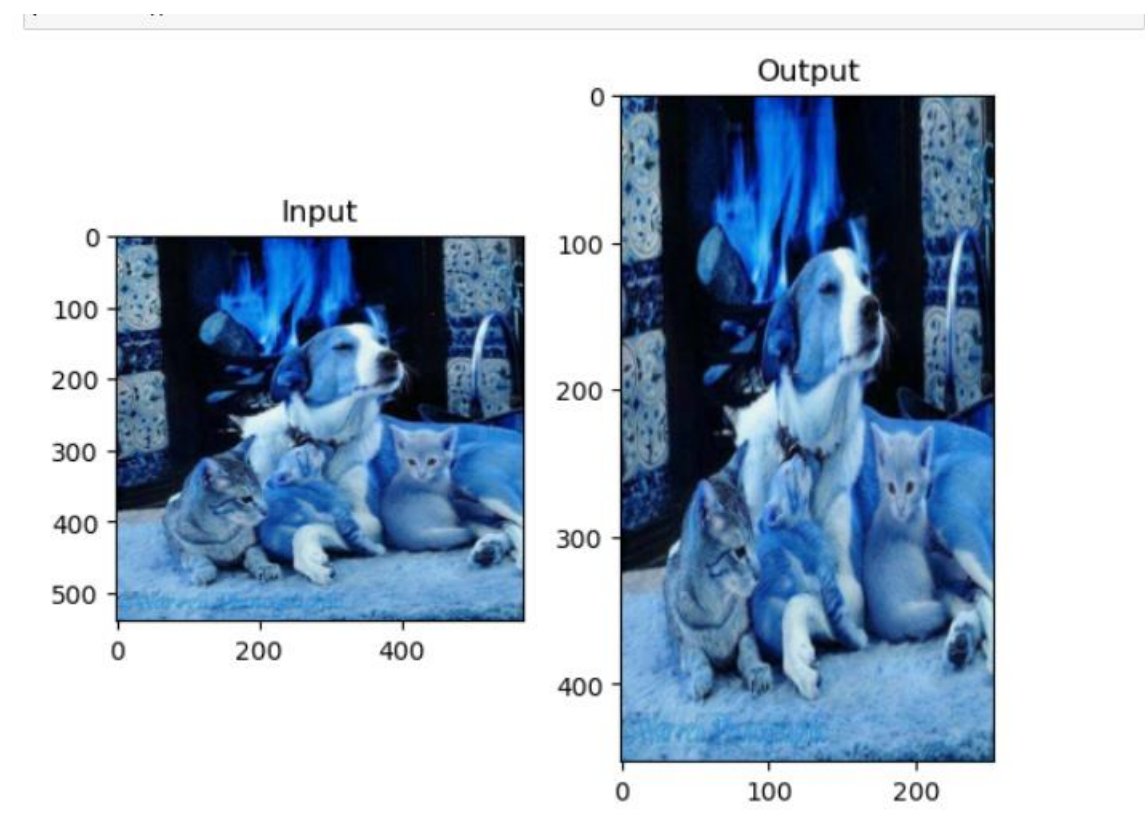
Input:

scaling

```
] : img = cv.imread(r"C:\Users\jeevi\Downloads\cat and dog.jpg")
    dim=(253,453)
    res=cv.resize(img,dim,interpolation=cv.INTER_AREA)

    plt.subplot(121),plt.imshow(img),plt.title('Input')
    plt.subplot(122),plt.imshow(res),plt.title('Output')
    plt.show()
```

Output:



RESULT:

We did some transformation technique for image transformation like scaling, rotating, perspective transformation.

EX.NO: 6	IMAGE TRANSFORMATION II
DATE:01.09.23	

AIM:

To do the image transformation using opencv in python.

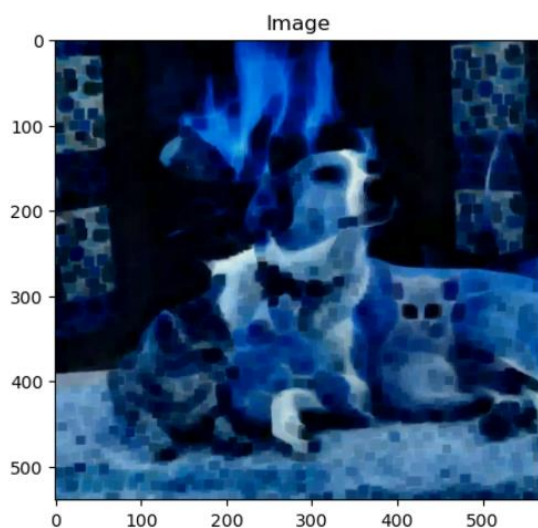
PROCEDURE:**1.Erosion:**

The basic idea of erosion is just like soil erosion only, it erodes away the boundaries of foreground object (Always try to keep foreground in white). So what it does? The kernel slides through the image (as in 2D convolution). A pixel in the original image (either 1 or 0) will be considered 1 only if all the pixels under the kernel is 1, otherwise it is eroded (made to zero).

So what happens is that, all the pixels near boundary will be discarded depending upon the size of kernel. So the thickness or size of the foreground object decreases or simply white region decreases in the image. It is useful for removing small white noises (as we have seen in colorspace chapter), detach two connected objects etc.

Erosion

```
img = cv.imread(r"C:\Users\jeevi\Downloads\cat and dog.jpg")
kernel = np.ones((10,10),np.uint8)
erosion = cv.erode(img,kernel,iterations = 1)
plt.imshow(erosion),plt.title("Image")
(<matplotlib.image.AxesImage at 0x2638e315660>, Text(0.5, 1.0, 'Image'))
```

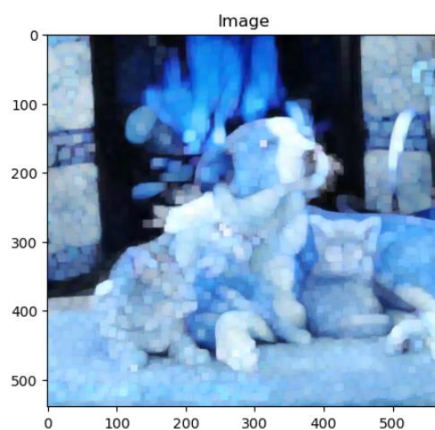


2.Dilation:

It is just opposite of erosion. Here, a pixel element is '1' if at least one pixel under the kernel is '1'. So it increases the white region in the image or size of foreground object increases. Normally, in cases like noise removal, erosion is followed by dilation. Because, erosion removes white noises, but it also shrinks our object. So we dilate it. Since noise is gone, they won't come back, but our object area increases. It is also useful in joining broken parts of an object.

Dilation

```
In [26]: img = cv.imread(r"C:\Users\jeevi\Downloads\cat and dog.jpg")
        kernel = np.ones((10,10),np.uint8)
        dilation = cv.dilate(img,kernel,iterations = 1)
        plt.imshow(dilation),plt.title("Image")
Out[26]: (<matplotlib.image.AxesImage at 0x2638e4c48e0>, Text(0.5, 1.0, 'Image'))
```

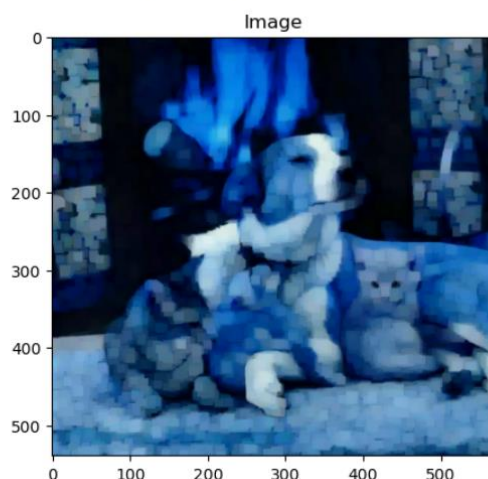


3.Opening:

Opening is just another name of erosion followed by dilation.

Opening

```
In [31]: img = cv.imread(r"C:\Users\jeevi\Downloads\cat and dog.jpg")
        kernel = np.ones((10,10),np.uint8)
        opening = cv.morphologyEx(img,cv.MORPH_OPEN,kernel)
        plt.imshow(opening),plt.title("Image")
Out[31]: (<matplotlib.image.AxesImage at 0x2638db9cc70>, Text(0.5, 1.0, 'Image'))
```



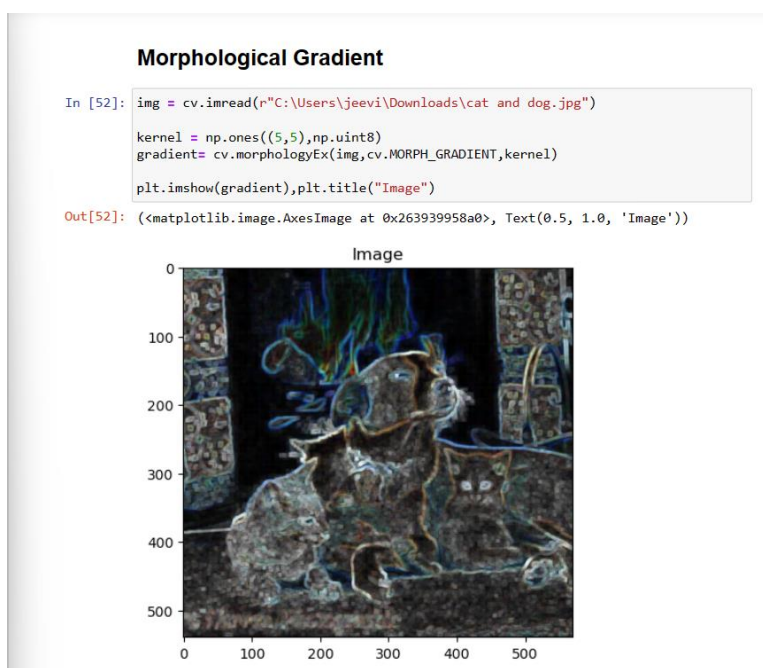
4.Closing:

Closing is reverse of Opening, Dilation followed by Erosion. It is useful in closing small holes inside the foreground objects, or small black points on the object.



5.Morphological Gradient:

It is the difference between dilation and erosion of an image. The result will look like the outline of the object.



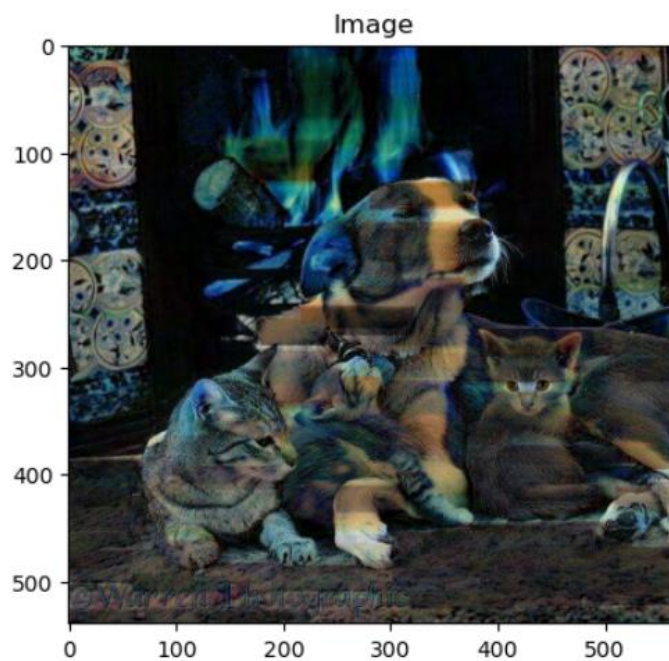
6. Top Hat:

It is the difference between input image and Opening of the image.

Top hat

```
In [39]: img = cv.imread(r"C:\Users\jeevi\Downloads\cat and dog.jpg")  
  
kernel = np.ones((14,54),np.uint8)  
tophat = cv.morphologyEx(img,cv.MORPH_TOPHAT,kernel)  
  
plt.imshow(tophat),plt.title("Image")
```

```
Out[39]: (<matplotlib.image.AxesImage at 0x26391f05690>, Text(0.5, 1.0, 'Image'))
```



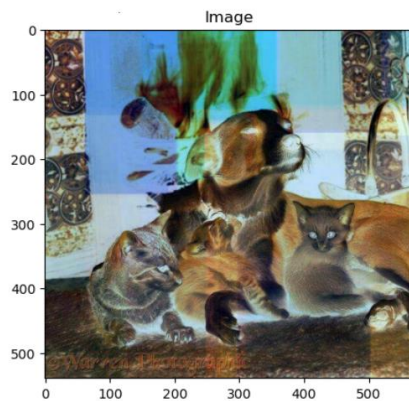
7. Black Hat:

It is the difference between the closing of the input image and input image.

Black Hat

```
In [44]: img = cv.imread(r"C:\Users\jeevi\Downloads\cat and dog.jpg")
        kernel = np.ones((233,123),np.uint8)
        blackhat = cv.morphologyEx(img,cv.MORPH_BLACKHAT,kernel)
        plt.imshow(blackhat),plt.title("Image")

Out[44]: (matplotlib.image.AxesImage at 0x26393127d90, Text(0.5, 1.0, 'Image'))
```



8. Image gradient:

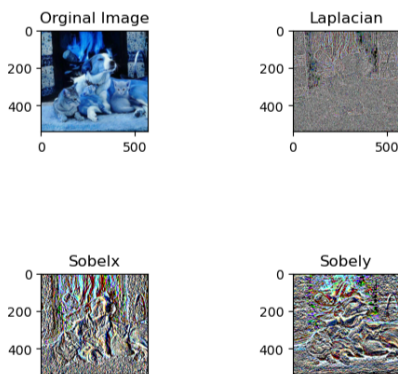
Image gradient

```
In [51]: #Image Gradients - these are filters (Laplacian,sobelx,sobely) for horizontal or vertical
img = cv.imread(r"C:\Users\jeevi\Downloads\cat and dog.jpg")

laplacian = cv.Laplacian(img,cv.CV_64F)
sobelx = cv.Sobel(img,cv.CV_64F,1,0,ksize=5)
sobely = cv.Sobel(img,cv.CV_64F,0,1,ksize=5)

plt.subplot(3,2,1),plt.imshow(img),plt.title("Original Image")
plt.subplot(3,2,2),plt.imshow(laplacian),plt.title("Laplacian")
plt.subplot(3,2,3),plt.imshow(sobelx),plt.title("Sobelx")
plt.subplot(3,2,4),plt.imshow(sobely),plt.title("Sobely")
plt.show()
```

Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).



RESULT:

We have explored some transformation technique like opening, dilation, image gradient, closing, top hat, black hat.

EX.NO: 7

DATE:08.09.23

FEATURE DETECTION AND DESCRIPTION

AIM:

To do feature detection and description of an image with OpenCV.

PROCEDURE:

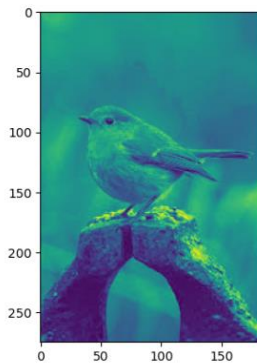
Harris Corner Detector:

```
In [15]: import cv2 as cv
import matplotlib.pyplot as plt
import numpy as np
```

```
In [41]: #convert to grayscale image
bird=cv.imread(r"C:\Users\jeevi\Downloads\bird.jpg")
gray=cv.cvtColor(bird,cv.COLOR_BGR2GRAY)
```

```
In [42]: plt.imshow(gray)
```

```
Out[42]: <matplotlib.image.AxesImage at 0x219797cdb70>
```



Harris Corner Detector in open cv

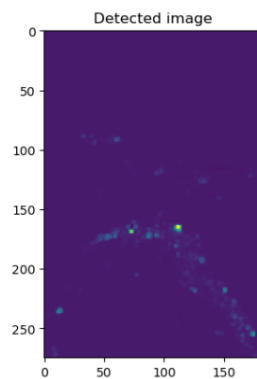
```
In [44]: gray = np.float32(gray)
dst = cv.cornerHarris(gray,2,3,0.04)

#result is dilated for marking the corners, not important
dst = cv.dilate(dst,None)

# Threshold for an optimal value, it may vary depending on the image.
bird[dst>0.01*dst.max()]=[0,0,255]

plt.imshow(dst),plt.title("Detected image")
```

```
Out[44]: (<matplotlib.image.AxesImage at 0x219798122c0>,
Text(0.5, 1.0, 'Detected image'))
```



SIFT (Scale Invariant Feature Transform):

Input:

SIFT (Scale-Invariant Feature Transform)

Harris corner detector is not good enough when scale of image changes. Lowe developed a breakthrough method to find scale-invariant features and it is called SIFT

```
[56]: #read the image
img = cv2.imread(r"C:\Users\jeevi\Downloads\bird.jpg", cv2.IMREAD_GRAYSCALE)

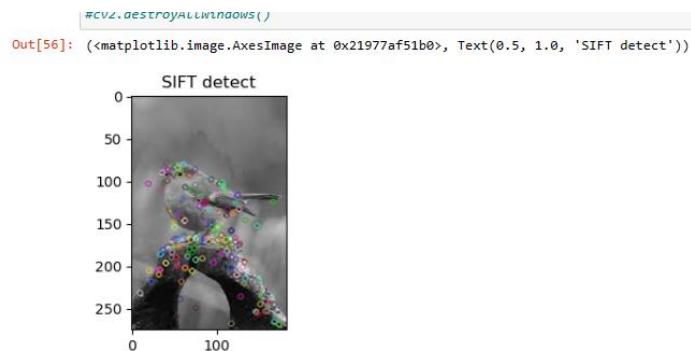
#create the sift creator
sift = cv2.SIFT_create()

#sift.detectAndCompute function is used to detect the keypoints
#and compute the descriptor about the dataset
keypoints_sift, descriptors = sift.detectAndCompute(img, None)

# draw a keypoints in a circle
img = cv2.drawKeypoints(img, keypoints_sift, None)

#show the detected image
plt.figure(figsize=(3,3))
plt.imshow(img),plt.title("SIFT detect")
#cv2.waitKey(0)
#cv2.destroyAllWindows()
```

Output:



FAST Algorithm for corner detection:

FAST Algorithm for Corner Detection

All the above feature detection methods are good in some way. But they are not fast enough to work in real-time applications like SLAM. There comes the FAST algorithm, which is really "FAST".

```
[72]: import numpy as np
import cv2 as cv
from matplotlib import pyplot as plt

#read the image
img = cv.imread(r"C:\Users\jeevi\Downloads\bird.jpg", cv.IMREAD_GRAYSCALE)

# Initiate FAST object with default values
fast = cv.FastFeatureDetector_create()

# find and draw the keypoints
kp = fast.detect(img, None)
img2 = cv.drawKeypoints(img, kp, None, color=(255,0,0))

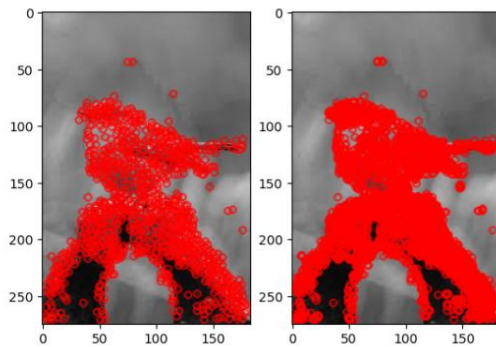
# Print all default params
print( "Threshold: {}".format(fast.getThreshold()) )
print( "nonmaxSuppression: {}".format(fast.getNonmaxSuppression()) )
print( "neighborhood: {}".format(fast.getN()) )
print( "Total Keypoints with nonmaxSuppression: {}".format(len(kp)) )
plt.subplot(1,2,1)
plt.imshow(img2)

# Disable nonmaxSuppression
fast.setNonmaxSuppression(0)
kp = fast.detect(img, None)
print( "Total Keypoints without nonmaxSuppression: {}".format(len(kp)) )
img3 = cv.drawKeypoints(img, kp, None, color=(255,0,0))
plt.subplot(1,2,2)
plt.imshow(img3)
```

```

Threshold: 10
nonmaxSuppression: True
neighborhood: 2
Total Keypoints with nonmaxSuppression: 1082
Total Keypoints without nonmaxSuppression: 3739
Out[72]: <matplotlib.image.AxesImage at 0x2197a0f9a80>

```



BRIEF (Binary Robust Independent Elementary Features):

SIFT uses a feature descriptor with 128 floating point numbers. Consider thousands of such features. It takes lots of memory and more time for matching. We can compress it to make it faster. But still we have to calculate it first. There comes BRIEF which gives the shortcut to find binary descriptors with less memory, faster matching, still higher recognition rate.

Input:

```

In [13]: import numpy as np
import cv2 as cv
from matplotlib import pyplot as plt

#read the image
img = cv.imread(r"C:\Users\jeevi\Downloads\bird.jpg", cv.IMREAD_GRAYSCALE)

# Initiate FAST detector
star = cv.xfeatures2d.StarDetector_create()

# Initiate BRIEF extractor
brief = cv.xfeatures2d.BriefDescriptorExtractor_create()

# find the keypoints with STAR
kp = star.detect(img, None)

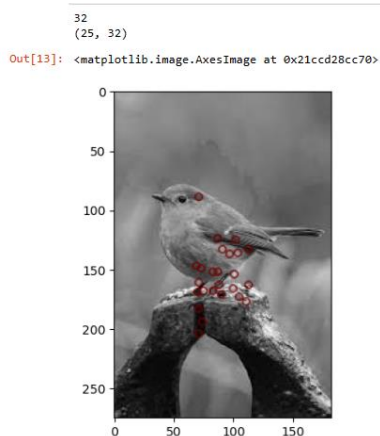
# compute the descriptors with BRIEF
kp, des = brief.compute(img, kp)

print( brief.descriptorSize() )
print( des.shape )

img2=cv.drawKeypoints(img,kp,None,color=(107,0,0))
plt.imshow(img2)

```

Output:



RESULT:

Thus, we created some of the technique for feature detection and description.

EX.NO: 8	FEATURE MATCHING AND MODEL FITTING
DATE:29.09.23	

AIM:

To do feature matching and model fitting.

PROCEDURE:

Feature Matching:

Basic of Brute-Force Matcher:

Brute-Force matcher is simple. It takes the descriptor of one feature in first set and is matched with all other features in second set using some distance calculation. And the closest one is returned.

Step 1: load the necessary packages

```
In [1]: import numpy as np
import cv2 as cv
import matplotlib.pyplot as plt
```

Step 2: load the images

```
In [10]: img1 = cv.imread(r"C:\Users\jeevi\Downloads\cropped book.jpeg",cv.IMREAD_GRAYSCALE) # queryImage
img2 = cv.imread(r"C:\Users\jeevi\Downloads\book in hand.jpeg",cv.IMREAD_GRAYSCALE) # trainImage
```

Step 3: create the ORB Detector

```
In [11]: # Initiate ORB detector (oriented fast and rotated brief)
orb = cv.ORB_create()
```

Step 4: find the key points and descriptors with ORB

```
In [12]: # find the keypoints and descriptors with ORB
kp1, des1 = orb.detectAndCompute(img1,None)
kp2, des2 = orb.detectAndCompute(img2,None)
```

Step 5: create the BFMatcher object

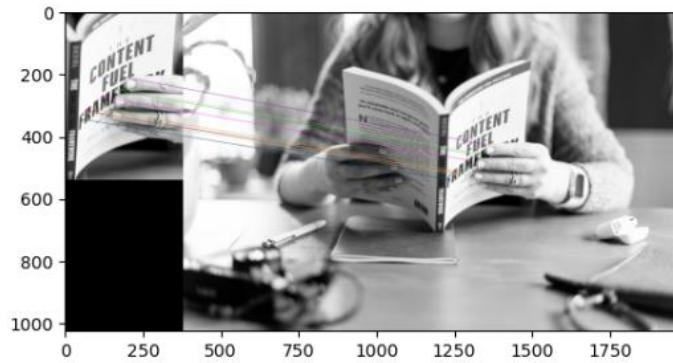
```
In [13]: # create BFMatcher object
bf = cv.BFMatcher(cv.NORM_HAMMING, crossCheck=True)
```

```
In [14]: # Match descriptors.
matches = bf.match(des1,des2)
```

```
In [15]: # Sort them in the order of their distance.
matches = sorted(matches, key = lambda x:x.distance)
```


Step 6: draw the first 10 matches

```
In [16]: # Draw first 10 matches.  
img3 = cv.drawMatches(img1,kp1,img2,kp2,matches[:10],None,flags=cv.DrawMatchesFlags_NOT_DRAW_SINGLE_POINTS)  
  
In [17]: plt.imshow(img3),plt.show()
```



```
Out[17]: (matplotlib.figure.Figure at 0x28b416f300): None
```

RESULT:

Thus, we did feature matching and model fitting for the images.

EX.NO: 9	COLOUR FUNDAMENTALS
DATE:06.10.23	

AIM:

To know the colour fundamentals.

PROCEDURE:**Import the necessary packages:**

```
In [7]: import cv2
import matplotlib.pyplot as plt
import numpy as np
```

RGB Colour Model:

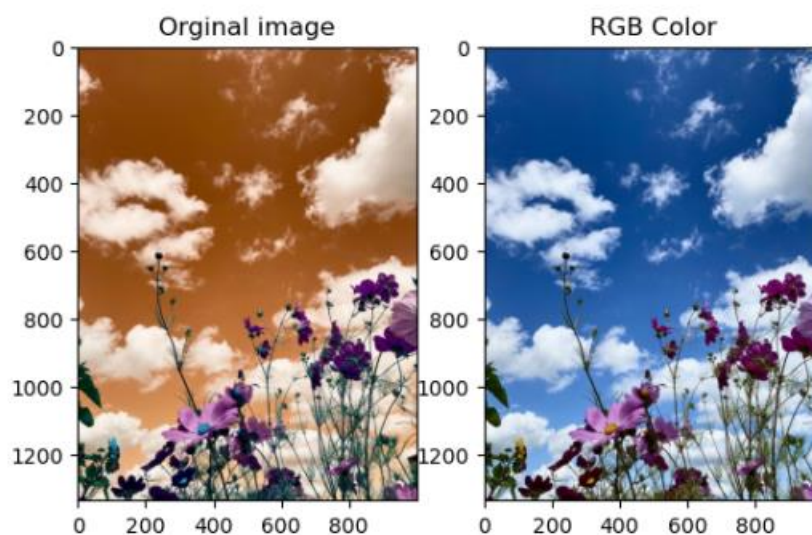
The RGB color model is often represented in the form of BGR (Blue, Green, Red), which is the default color order used in many image and video processing functions within the library. This is because in many computer vision and image processing contexts, the BGR order is used as the standard, although RGB is more common in general usage.

Input:

```
In [8]: # Read an image in BGR format
image = cv2.imread(r"C:\Users\jeevi\Downloads\flower.jpg")

# Convert from BGR to RGB for display
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

plt.subplot(1,2,1)
plt.imshow(image),plt.title("Original image")
plt.subplot(1,2,2)
plt.imshow(image_rgb),plt.title("RGB Color")
plt.show()
```

Output:

XYZ Colour Model:

The XYZ color model is not natively supported as a color space conversion function. OpenCV primarily provides functions for working with common color spaces like BGR, RGB, HSV, Lab, etc. However, you can convert between XYZ and other color spaces, such as BGR, using a combination of OpenCV and other libraries like NumPy.

Input:

```
In [15]: # Load an image in BGR format
image_bgr = cv2.imread(r"C:\Users\jeevi\Downloads\flower.jpg")

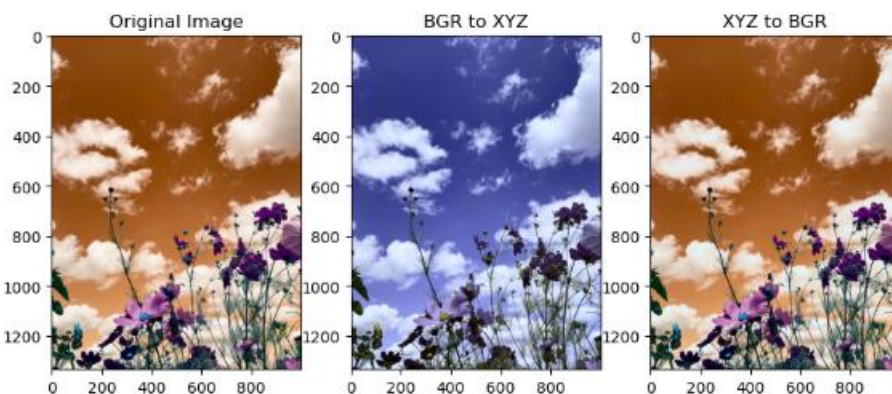
# Convert BGR to XYZ
image_xyz = cv2.cvtColor(image_bgr, cv2.COLOR_BGR2XYZ)

# Convert XYZ to BGR
image_bgr_again = cv2.cvtColor(image_xyz, cv2.COLOR_XYZ2BGR)

plt.figure(figsize=(10,10))
plt.subplot(1,3,1),plt.imshow(image_bgr),plt.title("Original Image")
plt.subplot(1,3,2),plt.imshow(image_xyz),plt.title("BGR to XYZ")
plt.subplot(1,3,3),plt.imshow(image_bgr_again),plt.title('XYZ to BGR')
```

Output:

```
Out[15]: (<Axes: title={'center': 'XYZ to BGR'}>,
<matplotlib.image.AxesImage at 0x240d0456fb0>,
Text(0.5, 1.0, 'XYZ to BGR'))
```



YUV Colour Model:

The YUV color model is commonly used in digital video and image compression. In this model, "Y" represents the luma (brightness) component, and "U" and "V" represent the chroma (color) components.

Input:

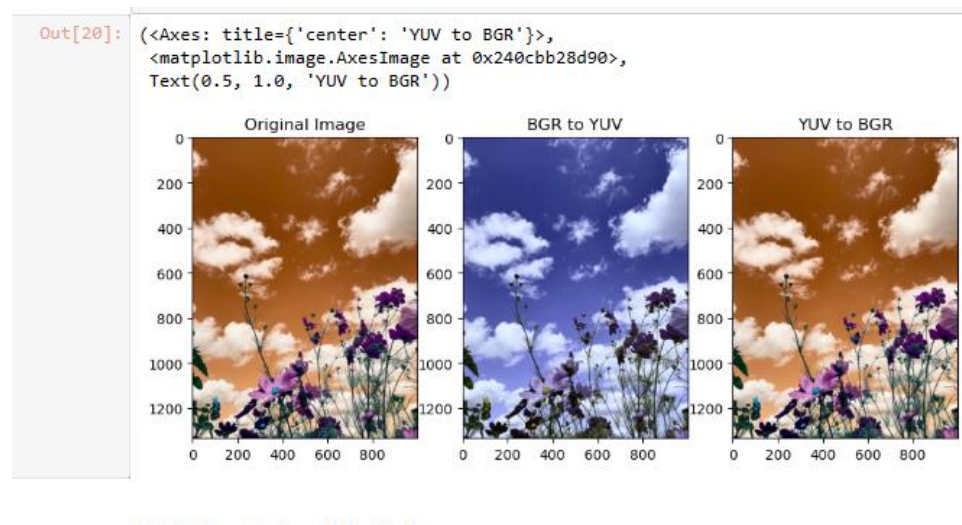
```
In [20]: # Load an image in BGR format (OpenCV's default)
image_bgr = cv2.imread(r"C:\Users\jeevi\Downloads\flower.jpg")

# Convert BGR to YUV
image_yuv = cv2.cvtColor(image_bgr, cv2.COLOR_BGR2YUV)

# Convert YUV to BGR
image_bgr_again = cv2.cvtColor(image_yuv, cv2.COLOR_YUV2BGR)

plt.figure(figsize=(10,10))
plt.subplot(1,3,1),plt.imshow(image_bgr),plt.title("Original Image")
plt.subplot(1,3,2),plt.imshow(image_yuv),plt.title("BGR to YUV")
plt.subplot(1,3,3),plt.imshow(image_bgr_again),plt.title('YUV to BGR')
```

Output:



YCbCr Colour Model:

The YCbCr color model is often used in digital image and video processing. In this model, "Y" represents the luma (brightness) component, while "Cb" and "Cr" represent the chroma (color difference) components. To work with the YCbCr color model in Python, you can use libraries like OpenCV, Pillow, or scikit-image, depending on your preferences.

Input:

```
In [21]: # Load an image in BGR format (OpenCV's default)
image_bgr = cv2.imread(r"C:\Users\jeevi\Downloads\flower.jpg")

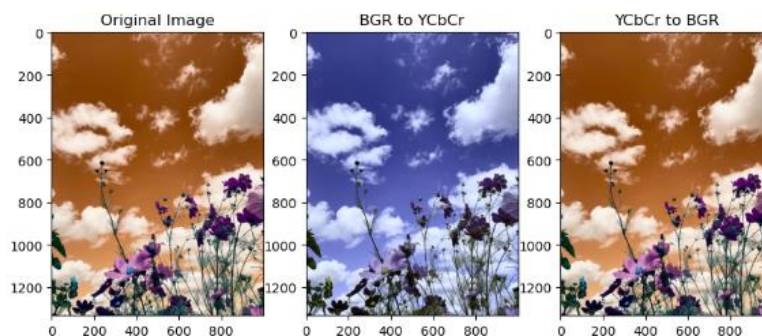
# Convert BGR to YCbCr
image_ycbcr = cv2.cvtColor(image_bgr, cv2.COLOR_BGR2YCrCb)

# Convert YCbCr to BGR
image_bgr_again = cv2.cvtColor(image_ycbcr, cv2.COLOR_YCrCb2BGR)

plt.figure(figsize=(10,10))
plt.subplot(1,3,1),plt.imshow(image_bgr),plt.title("Original Image")
plt.subplot(1,3,2),plt.imshow(image_ycbcr),plt.title("BGR to YCbCr")
plt.subplot(1,3,3),plt.imshow(image_bgr_again),plt.title("YCbCr to BGR")
```

Output:

```
Out[21]: (<Axes: title={'center': 'YCbCr to BGR'}>,
<matplotlib.image.AxesImage at 0x240cbfd160>,
Text(0.5, 1.0, 'YCbCr to BGR'))
```



HSV Colour Model:

The HSV (Hue, Saturation, Value) color model is widely used in computer vision and image processing for tasks like color detection and segmentation. To work with the HSV color model in Python, you can use various libraries, such as OpenCV or scikit-image.

Input:

```
In [22]: # Load an image in BGR format (OpenCV's default)
image_bgr = cv2.imread(r"C:\Users\jeevi\Downloads\flower.jpg")

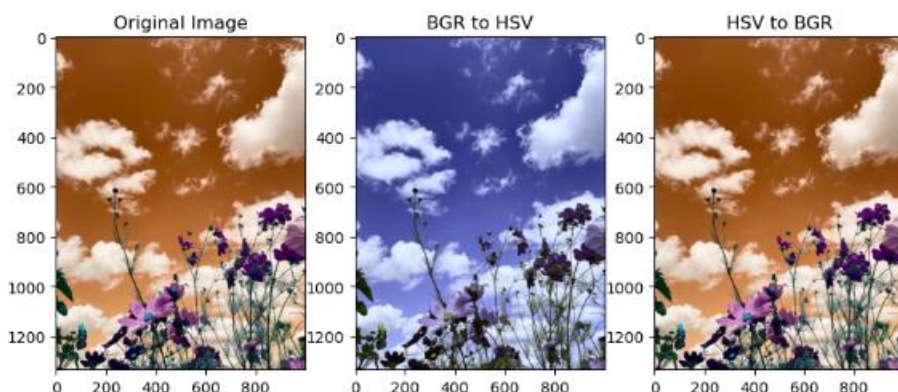
# Convert BGR to HSV
image_hsv = cv2.cvtColor(image_bgr, cv2.COLOR_BGR2HSV)

# Convert HSV to BGR
image_bgr_again = cv2.cvtColor(image_hsv, cv2.COLOR_HSV2BGR)

plt.figure(figsize=(10,10))
plt.subplot(1,3,1),plt.imshow(image_bgr),plt.title("Original Image")
plt.subplot(1,3,2),plt.imshow(image_hsv),plt.title("BGR to HSV")
plt.subplot(1,3,3),plt.imshow(image_bgr_again),plt.title('HSV to BGR')
```

Output:

```
Out[22]: (<Axes: title={'center': 'HSV to BGR'}>,
<matplotlib.image.AxesImage at 0x240d17d2dd0>,
Text(0.5, 1.0, 'HSV to BGR'))
```



You can also extract specific color ranges using HSV:

Input:

```
In [24]: # You can also extract specific color ranges using HSV.

# Define the lower and upper bounds for blue in HSV
lower_blue = np.array([110, 50, 50])
upper_blue = np.array([130, 255, 255])

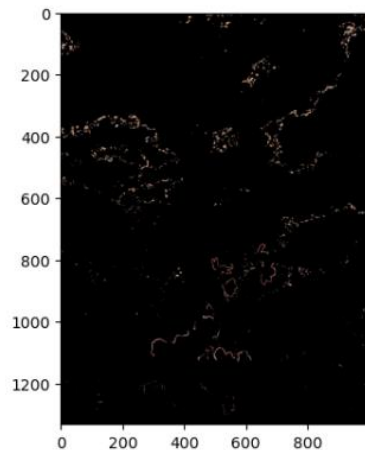
# Create a mask to isolate blue pixels
mask = cv2.inRange(image_hsv, lower_blue, upper_blue)

# Extract only the blue regions from the original image
blue_objects = cv2.bitwise_and(image_bgr, image_bgr, mask=mask)

plt.imshow(blue_objects)
```


Output:

Out[24]: <matplotlib.image.AxesImage at 0x240d4abf8b0>



HSL Colour Model:

The HSL (Hue, Saturation, Lightness) color model is another widely used color space for image processing and manipulation. You can work with the HSL color model in Python using libraries such as OpenCV or scikit-image.

Input:

```
In [25]: # Load an image in BGR format (OpenCV's default)
image_bgr = cv2.imread(r"C:\Users\jeevi\Downloads\flower.jpg")

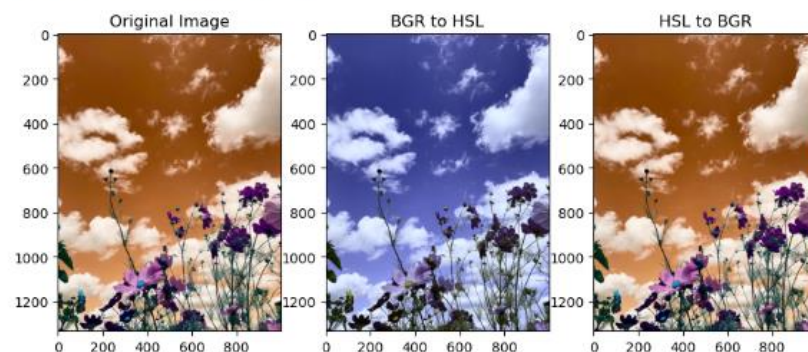
# Convert BGR to HSL
image_hls = cv2.cvtColor(image_bgr, cv2.COLOR_BGR2HLS)

# Convert HSL to BGR
image_bgr_again = cv2.cvtColor(image_hls, cv2.COLOR_HLS2BGR)

plt.figure(figsize=(10,10))
plt.subplot(1,3,1),plt.imshow(image_bgr),plt.title("Original Image")
plt.subplot(1,3,2),plt.imshow(image_hls),plt.title("BGR to HSL")
plt.subplot(1,3,3),plt.imshow(image_bgr_again),plt.title('HSL to BGR')
```

Output:

Out[25]: (<Axes: title=['center': 'HSL to BGR']>, <matplotlib.image.AxesImage at 0x240d4df76d0>, Text(0.5, 1.0, 'HSL to BGR'))



RESULT:

Thus, we know the fundamentals of colour in open cv.

EX.NO: 10	CLUSTERING AND CLASSIFICATION
DATE:13.10.23	

AIM:

To do clustering and classification in open cv.

PROCEDURE:

Import the necessary library:

```
In [9]: import cv2
import numpy as np
import matplotlib.pyplot as plt
```

Step: 1 Load the image and do the preprocess:

Input:

```
In [13]: # Load image from images directory
image = cv2.imread(r"C:\Users\jeevi\Downloads\images.png")

# Change color to RGB (from BGR)
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

# Reshaping the image into a 2D array of pixels and 3 color values (RGB)
pixel_vals = image.reshape((-1,3)) # numpy reshape operation -1 unspecified

# Convert to float type only for supporting cv2.kmean
pixel_vals = np.float32(pixel_vals)
```

```
In [16]: #criteria
criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 100, 0.85)

# Choosing number of cluster
k = 5

retval, labels, centers = cv2.kmeans(pixel_vals, k, None, criteria, 10, cv2.KMEANS_RANDOM_CENTERS)

# convert data into 8-bit values
centers = np.uint8(centers)

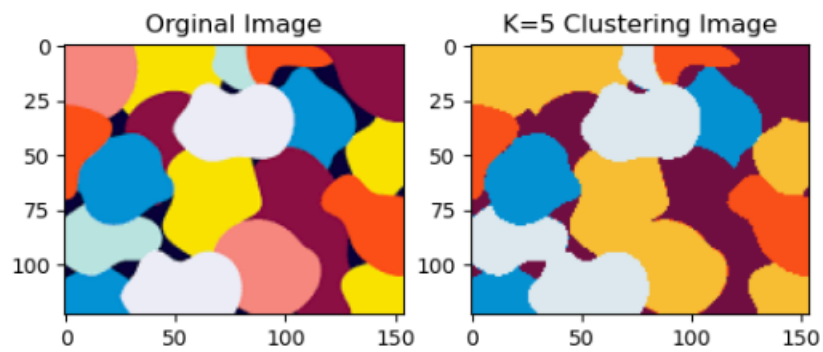
segmented_data = centers[labels.flatten()] # Mapping labels to center points( RGB Value)

# reshape data into the original image dimensions
segmented_image = segmented_data.reshape((image.shape))

plt.subplot(1,2,1),plt.imshow(image),plt.title("Original Image")
plt.subplot(1,2,2),plt.imshow(segmented_image),plt.title("K=5 Clustering Image")
```

Output:

```
Out[16]: (<Axes: title={'center': 'K=5 Clustering Image'}>,  
<matplotlib.image.AxesImage at 0x16e118c3a30>,  
Text(0.5, 1.0, 'K=5 Clustering Image'))
```



RESULT:

Thus, we did k means clustering using Open CV.

EX.NO: 11	DIMENSIONAL REDUCTION AND SPARSE REPRESENTATION
DATE:20.10.23	

AIM:

To do Dimensional reduction and sparse representation.

PROCEDURE:

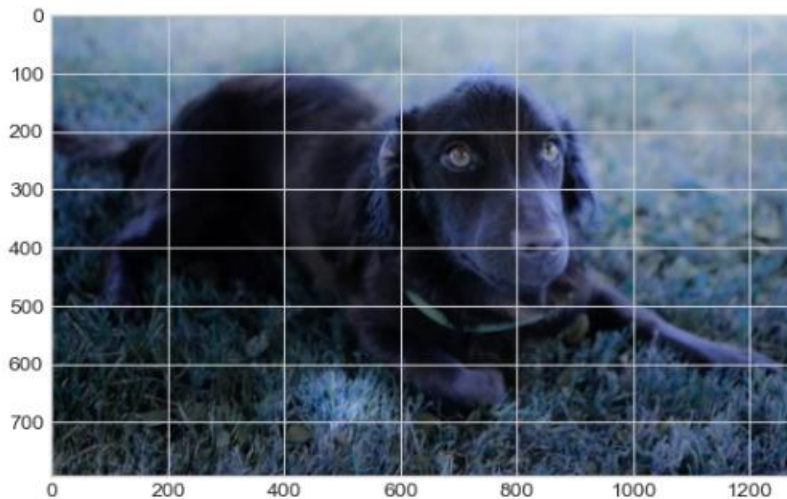
Step1: load the necessary packages:

```
In [7]: # Importing required libraries
import cv2
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
```

Step 2: load the image:

```
In [14]: # Loading the image
img = cv2.imread(r"C:\Users\jeevi\Downloads\sample dog.jpeg") #you can use
plt.imshow(img)
```

Out[14]: <matplotlib.image.AxesImage at 0x16fdb01baf0>



Step 3: Do dimensionality reduction:

```
In [15]: # Splitting the image in R,G,B arrays.

blue,green,red = cv2.split(img)
#it will split the original image into Blue, Green and Red arrays

In [16]: #initialize PCA with first 20 principal components
pca = PCA(20)

#Applying to red channel and then applying inverse transform to transformed array.
red_transformed = pca.fit_transform(red)
red_inverted = pca.inverse_transform(red_transformed)

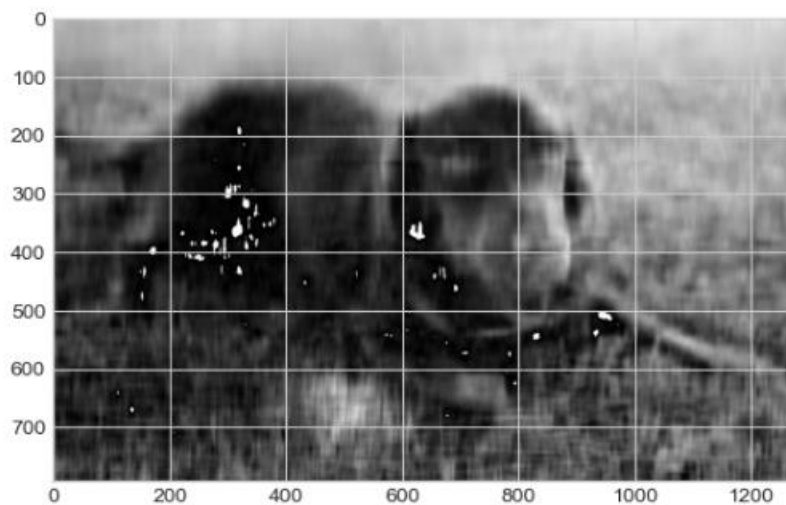
#Applying to Green channel and then applying inverse transform to transformed array.
green_transformed = pca.fit_transform(green)
green_inverted = pca.inverse_transform(green_transformed)

#Applying to Blue channel and then applying inverse transform to transformed array.
blue_transformed = pca.fit_transform(blue)
blue_inverted = pca.inverse_transform(blue_transformed)

In [17]: img_compressed = (np.dstack((red_inverted, green_inverted, blue_inverted))).astype(np.uint8)
```

```
In [18]: #viewing the compressed image
plt.imshow(img_compressed)
```

```
Out[18]: <matplotlib.image.AxesImage at 0x16fdb0f56f0>
```



RESULT:

Thus, we did the dimensionality reduction in open cv.

EX.NO: 12	EVALUATING THE LEARNING MODELS
DATE:03.11.23	

AIM:

To know about the Evaluating the learning models.

PROCEDURE:**Mean Average Precision (mAP) in Object Detection:**

Mean Average Precision (mAP) is a performance metric used for evaluating machine learning models. It is the most popular metric that is used by benchmark challenges such as PASCAL VOC, COCO, ImageNET challenge, Google Open Image Challenge, etc. Mean Average Precision has different meanings on various platforms. Hence, oftentimes it can be confusing. Continue reading the article for a thorough explanation of mAP.

Mean Average Precision (mAP) in Object Detection [TL;DR] :

1. Building blocks of Mean Average Precision. The model evaluation helper metrics – IoU, Confusion Matrix, Precision, and Recall.
2. Calculate class-specific Precision and Recall.
3. Plot Precision-Recall curve.
4. Calculate Average Precision (AP) using the PASCAL VOC 11-point interpolation method.
5. Calculate the Average Precision for all the classes.
6. Find Mean Average Precision (mAP) by averaging APs.

You will learn how mAP evolved over time from PASCAL VOC to MS COCO. Finally, we will also walk through a bit of the history of some important datasets and their significance.

RESULT:

Thus, we know the evaluating the learning models.