COMPUTER VISION LAB

(COURSE CODE: 22UPCSC4E09)

A Laboratory record was Submitted to Periyar University, Salem. In partial fulfilment of the Requirements for the Degree of

MASTER OF SCIENCE
IN
DATA SCIENCE

BY

SRIRAM R
REG NO: U22PG507DTS023

DEPARTMENT OF COMPUTER SCIENCE

PERIYAR UNIVERSITY

PERIYAR PALKALAI NAGAR,
SALEM – 636011

CERTIFICATE

This is to certify that the Programming Laboratory entitled **"COMPUTER VISION LAB"** (22UPCSC4E09) is a bonafide record work done by **Mr. SRIRAM R** Register No. U22PG507DTS023 as partial fulfilment of the requirements for the degree of MASTER OF SCIENCE IN DATA SCIENCE in the Department of Computer Science, Periyar University, Salem, during the year 2022 – 2024.

Faculty In-Charge

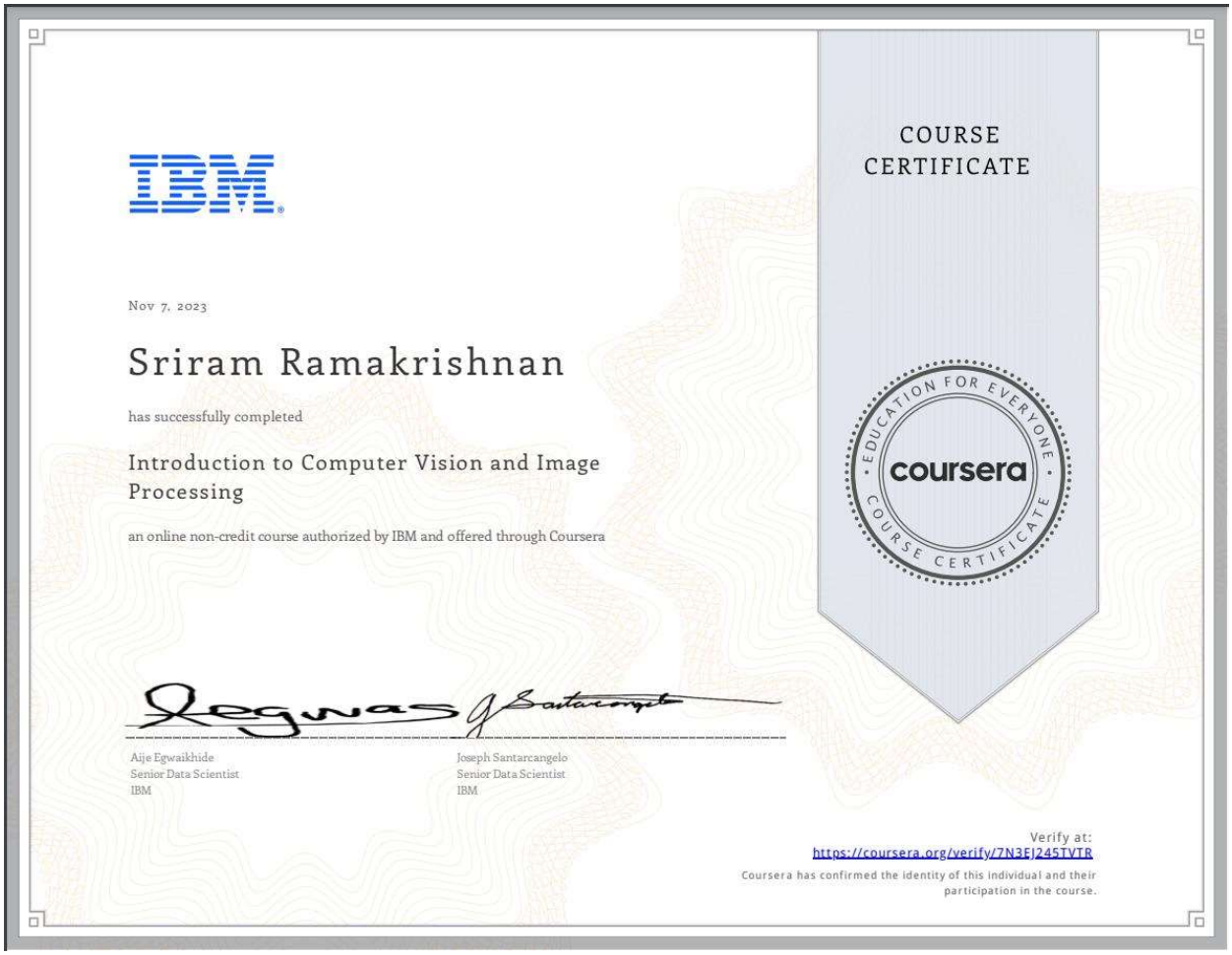
Head of the Department

Submitted for the practical examination held on ……. /……. /……

Marks Obtained

Internal Examiner

External Examiner



IBM

COURSE CERTIFICATE

Nov 7, 2023

Sriram Ramakrishnan

has successfully completed

Introduction to Computer Vision and Image Processing

an online non-credit course authorized by IBM and offered through Coursera

EDUCATION FOR EVERYONE · coursera · COURSE CERTIFICATE

Aije Egwaikhide
Senior Data Scientist
IBM

Joseph Santarcangelo
Senior Data Scientist
IBM

Verify at:
https://coursera.org/verify/7N3EJ245TVTR

Coursera has confirmed the identity of this individual and their participation in the course.

INDEX

S.No	DATE	TITLE	PAGE. No	SIGNATURE
1.	07/07/2023	Fundamental of computer vision	02	
2.	14/07/2023	Introduction to OpenCV	04	
3.	28/07/2023	How to Read, Write, Show Images in OpenCV	07	
4.	04/08/2023	Image processing using OpenCV	09	
5.	18/08/2023	Image transformation I	13	
6.	01/09/2023	Image transformation II	18	
7.	08/09/2023	Feature detection and description	23	
8.	29/09/2023	Feature matching and model fitting	26	
9.	06/10/2023	Colour fundamentals	29	
10.	13/10/2023	Clustering and classification	34	
11.	20/10/2023	Dimensional reduction and sparse representation	37	
12.	03/11/2023	Evaluating the learning models	42	

EX.NO: 1	FUNDAMENTAL OF COMPUTER VISION
DATE: 07/07/2023	

AIM:

Introduction to computer vision.

THEORY:

Computer vision is a field of study which enables computers to replicate the human visual system. It's a subset of artificial intelligence that collects information from digital images or videos and processes them to define the attributes. Origin of Computer Vision: Computer vision is not a new concept; in fact, it dates back to the 1960s. It all started with an MIT project - "Summer Vision Project" which analysed scenes to identify objects. David Marr, the celebrated neuroscientist, laid down the building blocks of computer vision, taking a cue from the functions of the cerebellum, hippocampus, and cortex of human perception. He has been dubbed the father of computer vision since, and the field has evolved to include much more complicated functionalities.

Computer Vision Tasks:

- **Image Classifications**

The Image Classification problem is the task of assigning an input image one label from a fixed set of categories. This is one of the core problems in a CV that, despite its simplicity, has a large variety of practical applications. Moreover, as we will see later, many other seemingly distinct CV tasks (such as object detection, segmentation) can be reduced to image classification.

- **Object Localizations**

This is a sort of intermediate task in between the other two ILSRVC tasks, image classification, and object detection. In image classification, you have to assign a single label to an image corresponding to the "main" object (eventually, the image can contain multiple objects). The classification + localization requires also localizing a single instance of this object, even if the image contains multiple instances of it. This task is also called "single-instance localization".

- **Object Detection**

Object detection is the process of finding instances of real-world objects such as faces, bicycles, and buildings in images or videos. Object detection algorithms typically use extracted features and learning algorithms to recognize instances of an object category. It is commonly used in applications such as image retrieval, security, surveillance, and automated vehicle parking systems.

- **Image Segmentation**

There are two kinds of segmentation tasks in a CV: Semantic Segmentation & Instance Segmentation. The difference between them is on Instance segmentation and semantic segmentation some other tasks in 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

Computer Vision Challenges:

Computer vision might have emerged as one of the top fields of machine learning, but there are still several obstacles in its way to becoming a leading technology. Human vision is a complicated and highly effective system that is difficult to replicate through technology. However, that's not to say that computer vision will not improve in the future.

Application of Computer Vision:

1. Retail: Customer Behaviour Tracking.
2. Agriculture: Detection of Wheat Rust.
3. Public Health: Image Segmentation of Scans.
4. Automotive Industry: Object Recognition and Classification in Traffic.
5. Fitness: Human Pose Estimation.
6. Medical Imaging: Computer vision helps in MRI reconstruction, automatic pathology, diagnosis, machine-aided surgeries, and more.
7. AR/VR: Object occlusion (dense depth estimation), outside-in tracking, inside-out tracking for virtual and augmented reality.
8. Smartphones: All the photo filters (including animation filters on social media), QR code scanners, panorama construction, Computational photography, face detectors, image detectors (Google Lens, Night Sight) that you use are computer vision applications.
9. Internet: Image search, geo localisation, image captioning, ariel imaging for maps, video categorization, and more.

RESULT:

Basics of computer vision discussed.

EX.NO: 2	INTRODUCTION TO OPENCV
DATE: 14/07/2023	

AIM:

Introduction to OpenCV library

THEORY:

OpenCV (Open-Source Computer Vision Library) is an open-source library that includes several hundreds of computer vision algorithms. OpenCV is one of the most popular computer vision libraries. If you want to start your journey in the field of computer vision, then a thorough understanding of the concepts of OpenCV is of paramount importance.

- Core functionality (core) - a compact module defining basic data structures, including the dense multi-dimensional array Mat and basic functions used by all other modules.
- Image Processing (imgproc) - an image processing module that includes linear and non-linear image filtering, geometrical image transformations (resize, affine and perspective warping, generic table-based remapping), color space conversion, histograms, and so on
- (video) - a video analysis module that includes motion estimation, background subtraction, and object tracking algorithms.
- Camera Calibration and 3D Reconstruction (calib3d) - basic multiple-view geometry algorithms, single and stereo camera calibration, object pose estimation, stereo correspondence algorithms, and elements of 3D reconstruction.
- 2D Features Framework (features2d) - salient feature detectors, descriptors, and descriptor matchers.
- Object Detection (objdetect) - detection of objects and instances of the predefined classes (for example, faces, eyes, mugs, people, cars, and so on).
- High-level GUI (highgui) - an easy-to-use interface to simple UI capabilities.
- Video I/O (videoio) - an easy-to-use interface to video capturing and video codecs.
- some other helper modules, such as FLANN and Google test wrappers, Python bindings, and others.

Topics:

1. Reading an image
2. Extracting the RGB values of a pixel
3. Extracting the Region of Interest (ROI)
4. Resizing the Image
5. Rotating the Image
6. Drawing a Rectangle
7. Displaying text

Image:

An Image is a spatial representation of a two dimensional or three-dimensional scene. It is an array or a matrix pixel (picture elements) arranged in columns and rows.....Digital Image is composed of picture elements, image elements, and pixels. A Pixel is most widely used to denote the elements of a Digital Image.

PROGRAM:

```
#import Library
import cv2
import matplotlib.pyplot as plt
#load the image
img = cv2.imread("/content/car.jpg")
#display the image
from google.colab.patches import cv2_imshow
cv2_imshow(img)
```

Extracting RGB Values from the image

```
img.shape
```

```
(457, 824, 3)
```

```
(B,G,R)=img[100,100]
print("B={},G={},R={}".format(B,G,R))
```

```
B=167,G=165,R=155
```

```
B=img[100,100,0]
print("B={}".format(B))

G=img[100,100,0]
print("G={}".format(G))

R=img[100,100,0]
print("R={}".format(R))
```

```
B=167
G=167
R=167
```

Extracting the Region of Interest (ROI)

```
roi=img[10:255,10:255]
cv2_imshow(roi)
```

RESULT:

Executed basic operations using OpenCV library.

EX.NO: 3	How to Read, Write, Show Images in OpenCV
DATE: 28/07/2023	

AIM:

To Create Read, Write, Show Images in OpenCv.

THEORY:

Reading, writing, and showing images are basic to image processing and computer vision. Even when cropping, resizing, rotating, or applying different filters to process images, you'll need to first read in the images. So, it's important that you need to know these basic operations.

OpenCV, the largest computer vision library in the world has these three built-in functions, let's find out what exactly each one does:

1. imread() helps us read an image

2. imshow() displays an image in a window

3. imwrite() writes an image into the file directory

PROCEDURE:

1. Import Libraries
2. Load Image()
3. Read Image()
4. Show Image() and
5. Write Image()

PROGRAM:

1. **Import cv2 Package:**

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

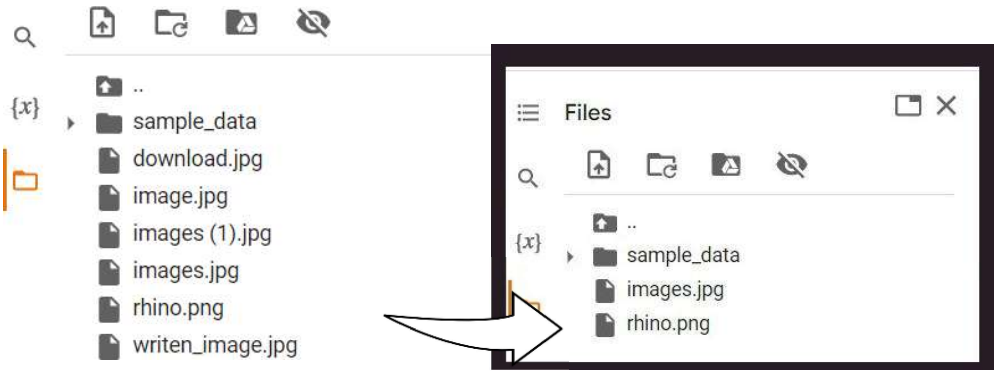
2. **Load** the Image to **read and show**:

```
df = cv2.imread("/content/car.jpg")
```

```
cv2_imshow(df)
```

3. Write the Image:

```
cv2.imwrite('writen_image.jpg',image)
```



RESULT:

Thus, above program has executed successfully.

EX.NO: 4	IMAGE PROCESSING USING OPEN CV
DATE: 04/08/2023	

AIM:

To write a basic image processing code using open cv.

THEORY:

OpenCV (Open-Source Computer Vision) is one of the most widely used tools for computer vision and image processing tasks. It is used in various applications such as face detection, video capturing, tracking moving objects, object disclosure, nowadays in Covid applications such as face mask detection, social distancing, and many more. If you want to know more about OpenCV. Taking pictures is just a matter of click so why playing around with it should be more than few lines of code. Seems not a case with python. There are quite a few good libraries available in python to process images such as open-cv, Pillow etc. In this article we'll be using Open CV, an open-source library for computer vision. It has C++, python and java interfaces available. It's highly optimized (written in C/C++) for real time applications in the domain of computer vision.

PROCEDURE:

Step 1: Importing necessary libraries
Step 2: Load dataset
Step 3: Applying image transformation techniques

1. Image resizing
2. Blurring the image
3. Making border to the image

Step 4: Displaying processed images.

PROGRAM:

1 Importing Libraries:

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

2. LOAD IMAGE:

```
df = cv2.imread("/content/car.jpg")
```

```
cv2_imshow(df)
```

3. IMAGE RESIZING:

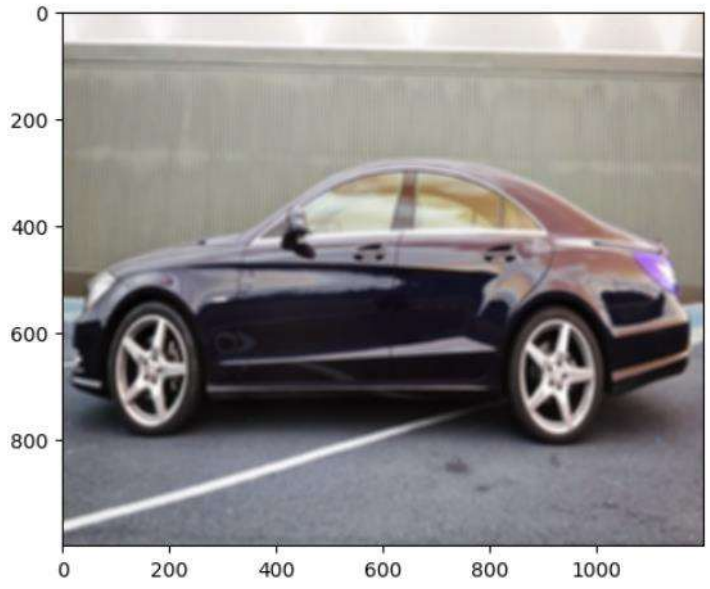
```
size=(1200,1000)
resized_image = cv.resize(df,size,interpolation=cv.INTER_AREA)
plt.imshow(resized_image)
plt.show
```

```
<function matplotlib.pyplot.show(close=None, block=None)>
```

4. BLURRING THE IMAGE:

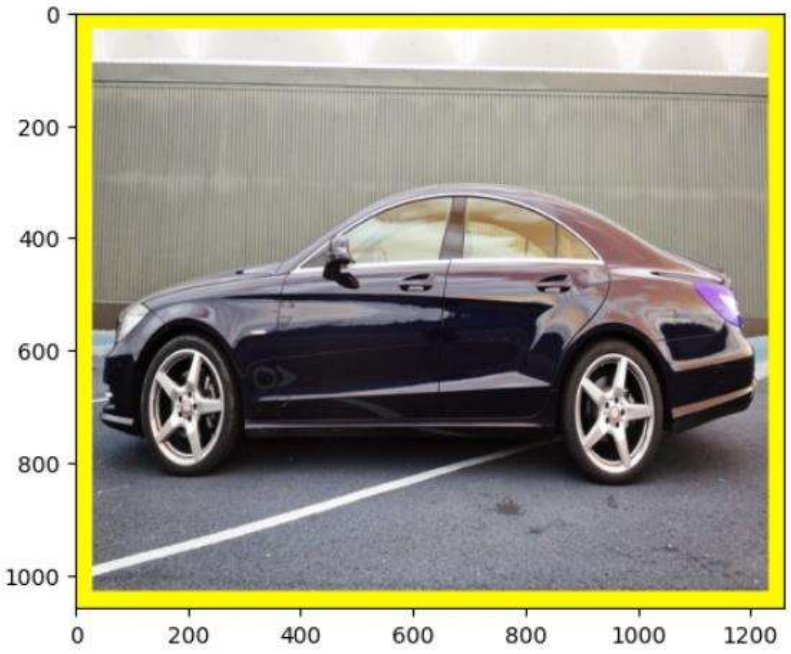
```
blur_image= cv.blur(resized_image,(6,6))
plt.imshow(blur_image)
plt.show()
```


```
blur_image= cv.blur(resized_image,(10,10))
plt.imshow(blur_image)
plt.show()
```



5. MAKING BORDER TO IMAGE:

```
border=cv.copyMakeBorder(resized_image,30,30,30,30, cv.BORDER_CONSTANT,None,value=[255,250,2])
plt.imshow(border)
plt.show()
```



RESULT:

Thus, the above program has executed successfully.

EX.NO: 5	IMAGE TRANSFORMATION- I
DATE: 18/08/2023	

AIM:

To write a basic image transformation code using open cv.

THEORY:

In image processing, image transformation can be defined as having control on its dimensional points to edit the images by moving them into three dimensional or two-dimensional space. Next in the article will perform some basic transformations of 2D images. Before going into the implementation of image transformation, let's see what Euclidean transformation is a type of geometric transformationthat causes changes in the dimensions and angles without causing the change in the basic structure-like area. Image Transformation works based on Euclidean transformation, where the image can look shifted or rotated, but the structure of pixels and matrix will remain the same.

Roughly we can divide image transformation into two sections:

- Affine transformation.
- Projective transformation.

PROCEDURE:

Step 1: Importing necessary libraries.
Step 2: Load image.
Step 3: Applying transformation.
Step 4: Displaying images.

PROGRAM:

1.IMPORTING LIBRARIES:

```
[1]  import pandas as pd
     import numpy as np
     import cv2 as cv
     import matplotlib.pyplot as plt
```

2 .LOAD IMAGE:

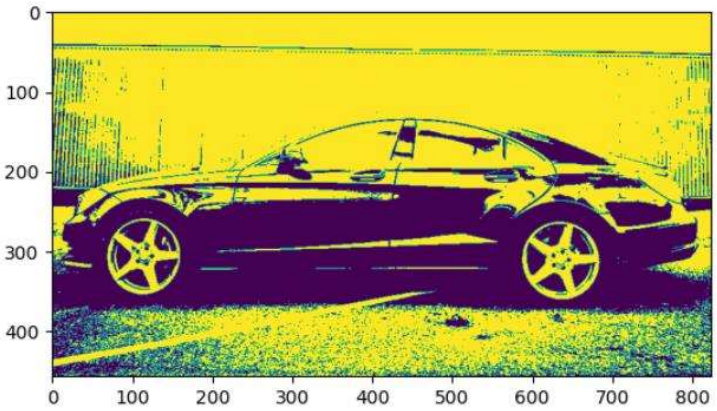
```
df = cv2.imread("/content/car.jpg")
```

```
cv2_imshow(df)
```

3. IMAGE TRANSLATION:

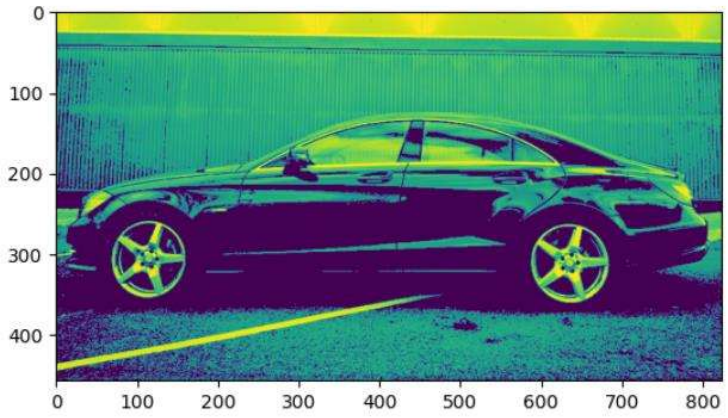
a).Different threshold Technique's:

```
gray_img=cv.imread("/content/car.jpg",cv.IMREAD_GRAYSCALE)
ret,th1 = cv.threshold(gray_img,125,225,cv.THRESH_BINARY)
plt.imshow(th1)
plt.show()
```



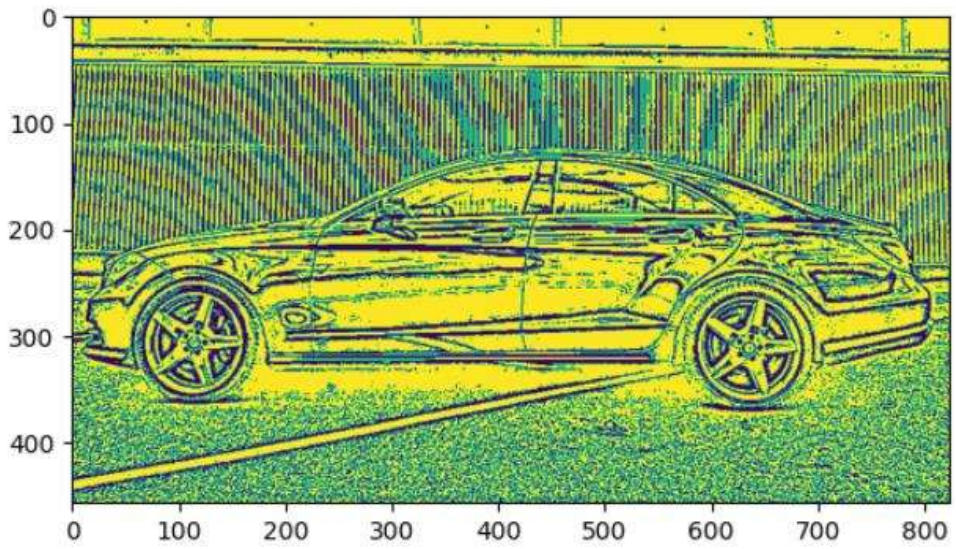
THRES_TOSERO:

```
ret,th1=cv.threshold(gray_img,125,225,cv.THRESH_TOZERO)
plt.imshow(th1)
plt.show()
```



ADAPTIVE_THRESOLDING

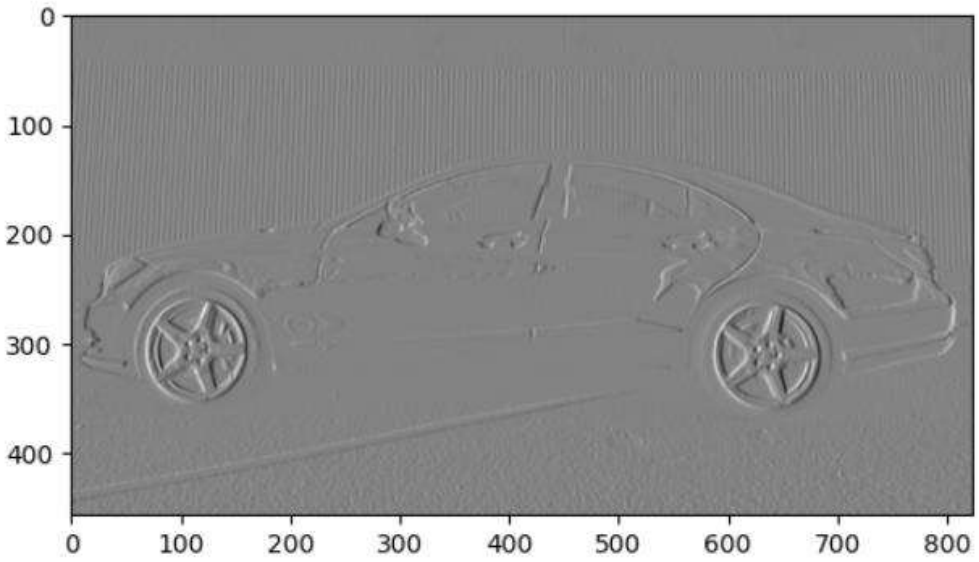
```
th3 = cv.adaptiveThreshold(gray_img,255,cv.ADAPTIVE_THRESH_GAUSSIAN_C,cv.THRESH_BINARY,11,4)
plt.imshow(th3)
plt.show()
```



a) Image Gradient:

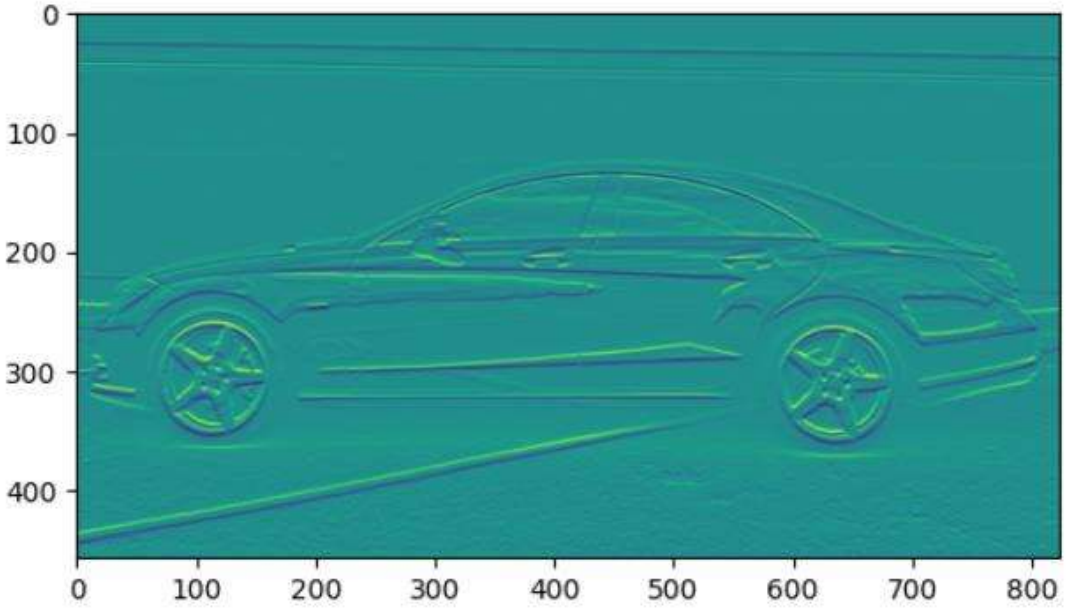
Soble_in_vertical_direction

```
sobelx = cv.Sobel(gray_img, cv.CV_64F, 1, 0, ksize=5)
plt.imshow(sobelx,cmap="gray")
plt.show()
```



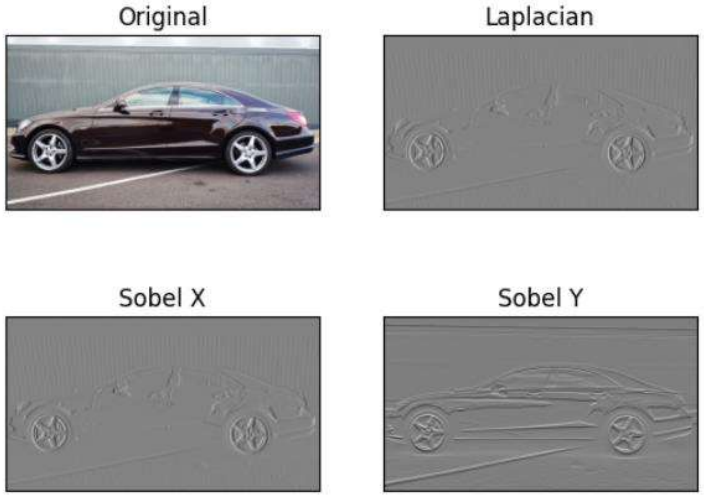
Sobel_in_horizontal_direction:

```
sobely = cv.Sobel(gray_img, cv.CV_64F, 0, 1, ksize=5)
plt.imshow(sobely)
plt.show()
```



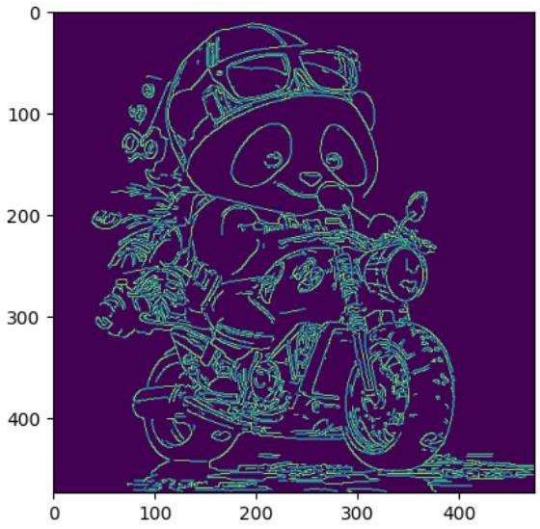
```
plt.subplot(2,2,1),plt.imshow(car_plt,cmap='gray')
plt.title('Original'),plt.xticks([]),plt.yticks([])
plt.subplot(2,2,2),plt.imshow(sobelx,cmap='gray')
plt.title('Laplacian'),plt.xticks([]),plt.yticks([])
plt.subplot(2,2,3),plt.imshow(sobelx,cmap='gray')
plt.title('Sobel X'),plt.xticks([]),plt.yticks([])
plt.subplot(2,2,4),plt.imshow(sobely,cmap = 'gray')
plt.title('Sobel Y'),plt.xticks([]),plt.yticks([])
```

```
(Text(0.5, 1.0, 'Sobel Y'), ([], []), ([], []))
```



C) CANNY_EDGE_DETECTION:

```
edges_150=cv.Canny(gray_img,150,100)
plt.imshow(edges_150)
plt.show()
```



RESULT:

Thus, the above program has executed successfully.

EX.NO: 6	IMAGE PROCESSING - II
DATE: 01/09/2023	

AIM:

To write a basic image transformation code using open cv.

THEORY:

OpenCV provides two transformation functions, **cv2.warpAffine** and **cv2.warpPerspective,** with which you can have all kinds of transformations. **cv2.warpAffine** takes a 2x3 transformation matrix while **cv2.warpPerspective** takes a 3x3 transformation matrix as input.

SCALING:

Scaling is just resizing of the image. OpenCV comes with a function **cv2.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 **cv2.INTER_AREA** for shrinking and **cv2.INTER_CUBIC** (slow) & **cv2.INTER_LINEAR** for zooming. By default, interpolation method used is **cv2.INTER_LINEAR** for all resizing purposes.

- Affine transformation.
- Projective transformation

PROCEDURE:

Step 1: Importing necessary libraries
Step 2: Load image
Step 3: Applying transformations
Step 4: Showing transformed image

PROGRAM:

1. IMPORTING LIBRARIES:

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

2. LOAD IMAGE:

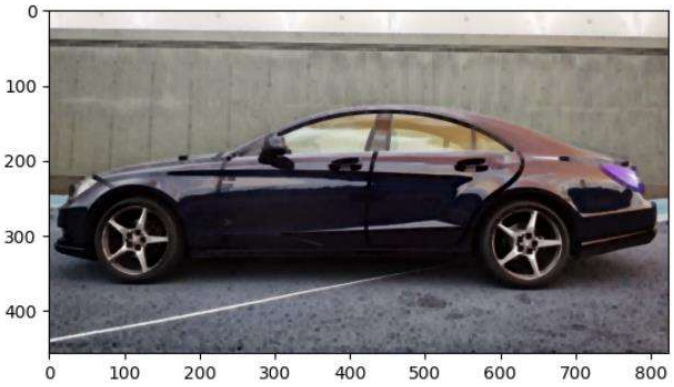
```
car = cv.imread("/content/car.jpg")
car_plt = plt.imread("/content/car.jpg")
```

```
plt.imshow(car_plt)
plt.show()
```

3.IMAGE TRANSFORMATION:

MORPHOLOGICAL TRANSFORMATION-EROSIO:

```
Kernel = np.ones((5,5),np.uint8)
erosion = cv.erode(car,Kernel,iterations=1)
plt.imshow(erosion)
plt.show()
```



```
gray_img=cv.imread("/content/car.jpg",cv.IMREAD_GRAYSCALE)
```

```
plt.subplot(121),plt.imshow(gray_img,cmap = 'gray')
plt.title('Original Image'),plt.xticks([]),plt.yticks([])
plt.subplot(122),plt.imshow(erosion,cmap = 'gray')
plt.title('Erosion Image'),plt.xticks([]),plt.yticks([])
plt.show()
```

DILATION:

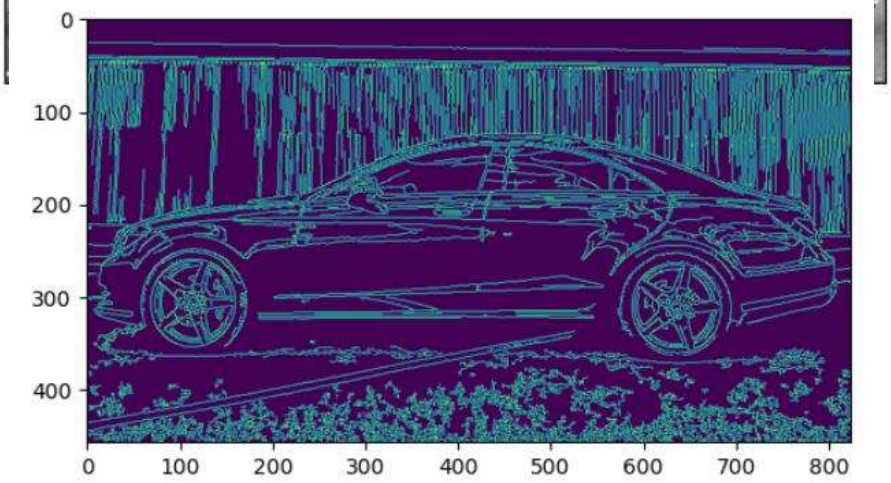
```
dilation = cv.dilate(gray_img,Kernel,iterations=1)
plt.imshow(dilation)
plt.show()
```

```
plt.subplot(121),plt.imshow(gray_img,cmap = 'gray')
plt.title("Original Image"),plt.xticks([]),plt.yticks([])
plt.subplot(122),plt.imshow(dilation,cmap = 'gray')
plt.title('Dilation image'),plt.xticks([]),plt.yticks([])
plt.show()
```

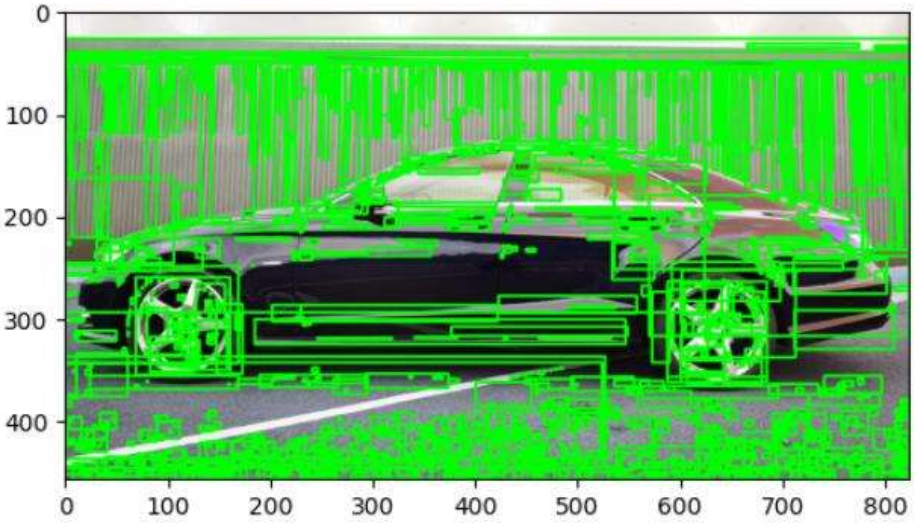
A) Counters:

So be fire finding counters, apply threshold or canny edge detection.

```
image = cv.imread("/content/car.jpg")
img_gray = cv.cvtColor(image,cv.COLOR_BGR2GRAY)
gray = cv.bilateralFilter(img_gray,11,17,17)
edged = cv.Canny(gray,20,90)
plt.imshow(edged)
plt.show()
```



```
contours, hierarchy = cv.findContours(edged,cv.RETR_TREE, cv.CHAIN_APPROX_SIMPLE)
x,y,w,h = cv.boundingRect(contours[0])

for i in range(len(contours)):
  area = cv.contourArea(contours[i])
  x,y,w,h = cv.boundingRect(contours[i])
  imgrect = cv.rectangle(image,(x,y),(x+w,y+h),(0,255,0),2)
  outfile = ('%s.jpg'% i)
  cv.imwrite(outfile,imgrect)
plt.imshow(imgrect)
plt.show()
cv.destroyAllWindows()
```



RESULT:

Thus, the above program has executed successfully.

EX.NO: 7	FEATURE DETECTION AND DESCRIPTION
DATE: 08/09/2023	

AIM:

Feature detection and feature description of image using different algorithm with OpenCV library.

THEORY:

computer vision is what allows computers to see and process visual data just like humans. Computer vision involves analysing images to produce useful information. The clues which are used to identify or recognize an image are called features of an image. In the same way, computer functions, to detect various features in an image.

1. Feature Detection Algorithms

1. Harris Corner Detection

Harris corner detection algorithm is used to detect corners in an input image. This algorithm has three main steps.

1. Determine which part of the image has a large variation in intensity as corners have large variations in intensities. It does this by moving a sliding window throughout the image.
2. For each window identified, compute a score value R.
3. Apply threshold to the score and mark the corners.

1. Scale-Invariant Feature Transform (SIFT)

SIFT is used to detect corners, blobs, circles, and so on. It is also used for scaling an image. It can detect features from the image irrespective of its size and orientation.

2. FAST Algorithm

Fast algorithm is a one-shot facial recognition algorithm. It is currently being used in your mobile phones and apps like Google photos in which you group the people stab you see the images are grouped according to the people. This algorithm does not require any kind of major computations. It does not require GPU. It works on key point matching. Key point matching of distinctive regions in an image like the intensity variations.

PROCEDURE:

Step 1: Importing necessary libraries
Step 2: Load image
Step 3: Apply feature detection algorithms
Step 4: Apply feature descriptor algorithms
Step 5: Display the resultant images

PROGRAM:

Feature Detection Algorithms

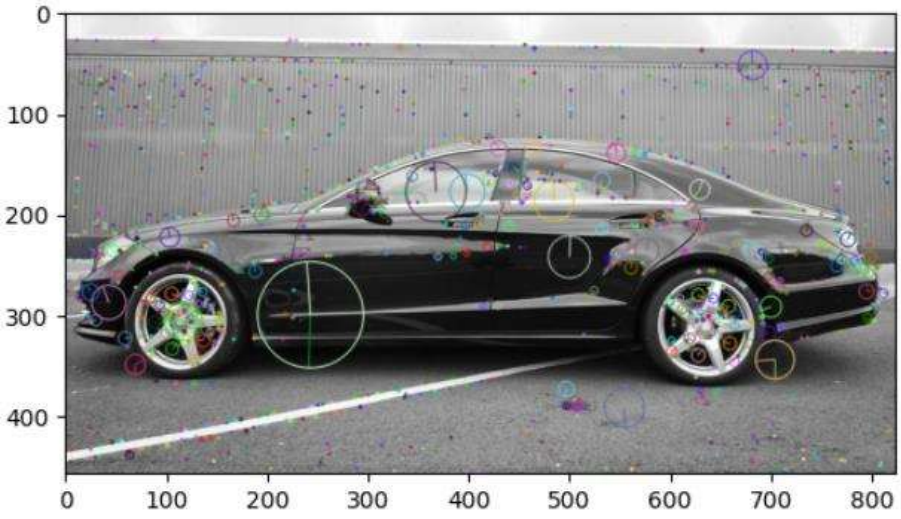
```
import matplotlib.pyplot as plt
import numpy as np
import cv2
import matplotlib.pyplot as plt
```

```
img = cv2.imread("/content/car.jpg")
```

```
plt.imshow(img)
plt.show()
```


```
gray=cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
sift=cv2.SIFT_create()
kp,des=sift.detectAndCompute(gray,None)
img=cv2.drawKeypoints(gray,kp,img,flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
```

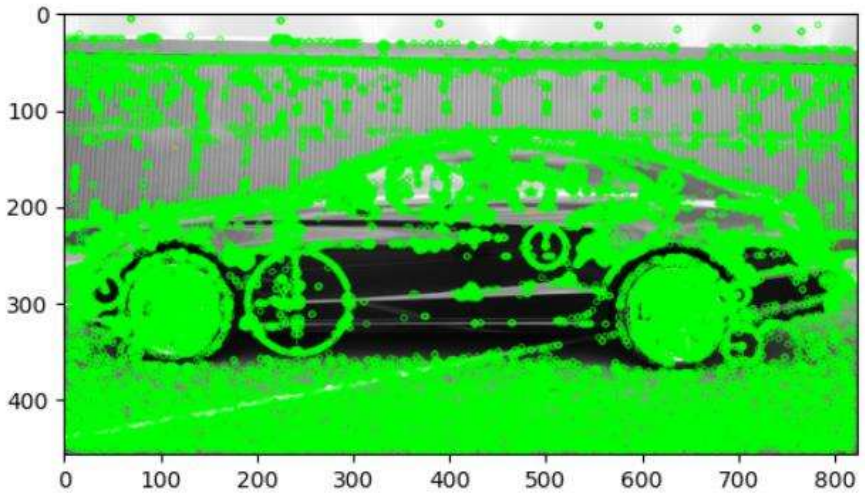
```
plt.imshow(img)
if cv2.waitKey(0)& 0xff== 27:
  cv2.destroyAllWindows()
```



FAST Algorithm

```
fast=cv2.FastFeatureDetector_create()
fast.setNonmaxSuppression(False)
kp=fast.detect(img,None)
kp_image=cv2.drawKeypoints(img,kp,None,color=(0,255,0))
plt.imshow(kp_image)
```

```
<matplotlib.image.AxesImage at 0x7d57b1be3cd0>
```



RESULT:

Execute different feature detection and description algorithms.

EX.NO: 8	FEATURE MATCHING AND MODEL FITTING
DATE: 29/09/2023	

AIM:

Find features of an image using feature detection algorithms with OpenCV library.

THEORY:

Feature matching is like comparing the features of two images which may be different in orientations, perspective, lightening, or even differ in sizes and colours. Let's see its implementation.

When using OpenCV for image processing and analysis, there will be scenarios that require curve fitting and circle fitting. There are ready-made functions in OpenCV to implement circle fitting and curve fitting. It is fitted with a straight line, and it will also tell you what the radius of the fitted circle is. It is super convenient. Another commonly used scene is curve fitting. The common one is based on polynomial fitting, which can be based on the set polynomial power. Sub-generation of polynomial equations, and then generate a series of points according to the equations to form a complete curve. This is especially useful in scenarios such as lane line detection and contour curve fitting. Here are two simple examples to learn about the application of curve fitting and circle fitting.

Curve fitting and application

Based on the implementation of the polyfit function of the Numpy package, the three parameters supported are the set of x points, the set of y points, and the power of the polynomial. After the polynomial equation is obtained, the curve can be completely fitted.

PROCEDURE:

Step 1: Importing necessary libraries
Step 2: Load image
Step 3: Apply Feature matching algorithm

1. ORB detector
2. SIFT detector

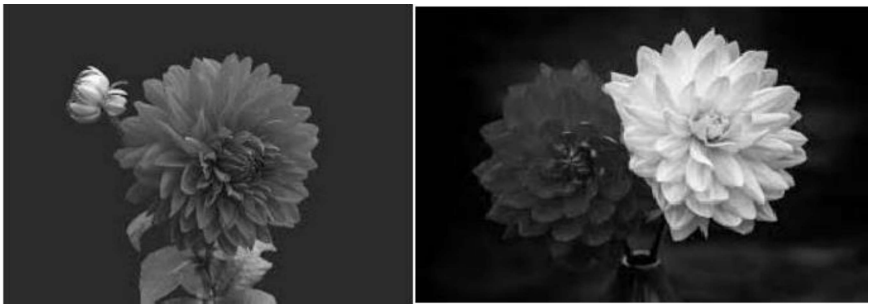
Step 4: Model fitting
Step 5: Display images with matches

PROGRAM:

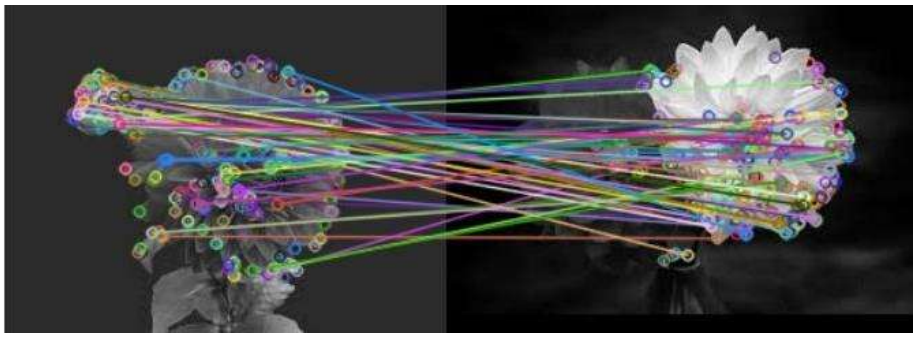
1. ORB detector

```
#import libraries
import cv2
from google.colab.patches import cv2_imshow
img1=cv2.imread('/content/download (1).jpg',0)
img2=cv2.imread('/content/download (2).jpg',0)
#Initiate ORB detector
orb=cv2.ORB_create(nfeatures=500)
#find the keypoints and descriptors with ORB
kp1,des1=orb.detectAndCompute(img1,None)
kp2,des2=orb.detectAndCompute(img2,None)
#create BFMatcher object
bf=cv2.BFMatcher(cv2.NORM_HAMMING,crossCheck=True)
#Match descriptors
matches=bf.match(des1,des2)
#sort them in the order of their distance
matches=sorted(matches,key=lambda x: x.distance)
#Draw first 50 matches
match_img=cv2.drawMatches(img1,kp1,img2,kp2,matches[:50],None)
```

```
cv2_imshow(img1)
cv2_imshow(img2)
```



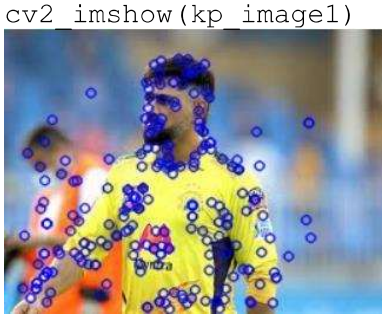
```
cv2_imshow(match_img)

cv2.waitKey()
```



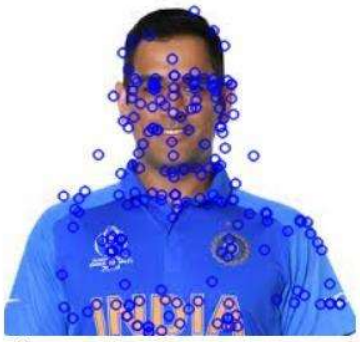
2. SIFT detector

```
#Import libraries
import numpy as np
import cv2
```

```
import matplotlib.pyplot as plt
from google.colab.patches import cv2_imshow


#load images
MIN_MATCH_COUNT=10
img1=cv2.imread('/content/images (2).jpg')
img2=cv2.imread('/content/images (3).jpg')
img3=cv2.imread('/content/images (4).jpg')


#Initiate SIFT detector
sift=cv2.SIFT_create()
#find the keypoints and descriptors with SIFT
kp1,des1=sift.detectAndCompute(img1,None)
kp2,des2=sift.detectAndCompute(img2,None)


kp_image1=cv2.drawKeypoints(img1,kp1,None,color=(255,0,0))
cv2_imshow(kp_image1)
```



```
kp_image2=cv2.drawKeypoints(img2,kp2,None,color=(255,0,0))
cv2_imshow(kp_image2)
cv2.waitKey()
```



```
-1
```

RESULT:

Found out matched features of two images using ORB and SIFT detectors.

EX.NO: 9	COLOUR FUNDAMENTALS
DATE: 06/10/2023	

AIM:

To write a code for colour fundamentals using open cv.

THEORY:

HSV (Hue, Saturation, Value):

Just like the most common colour spaces we use to represent an image like RGB and BGR, HSV too is a colour space in which H denotes hue, S denotes Saturation and V denotes value. As we already know what hue is let us see about saturation and value.

Saturation:

Saturation gives the purity of a colour. A pure colour has no gray mixed in it. Greater the amount of gray mixed in a colour, lesser the saturation. The saturation value is usually measured from 0 to 100% but in OpenCV the scale for saturation is from 0 to 255.

Value:

This is a measure of the brightness of a colour. When the brightness value is maximum, the colour turns white and when the brightness value is minimum, the colour turns black. This is usually measured from 0 to 100% but in OpenCV the scale for value is from 0 to 255.

PROCEDURE:

Step 1: Import the required libraries and read the image
Step 2: Convert the image bgr to hsv the set the lower bounds and upper bounds
Step 3: Convert the image into different formats
Step 4: Showing the image

PROGRAM:

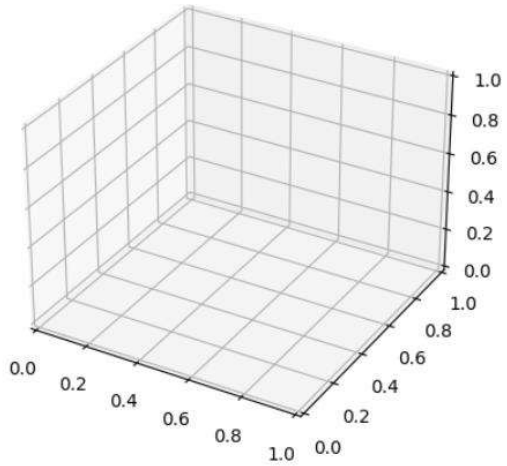
```
import matplotlib.pyplot as plt
import numpy as np
import cv2
import matplotlib.pyplot as plt
```

```
car = cv2.imread("/content/car.jpg")
```


```
car = cv2.cvtColor(car, cv2.COLOR_BGR2RGB)
plt.imshow(car)
plt.show()
```


```
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm
from matplotlib import colors
```

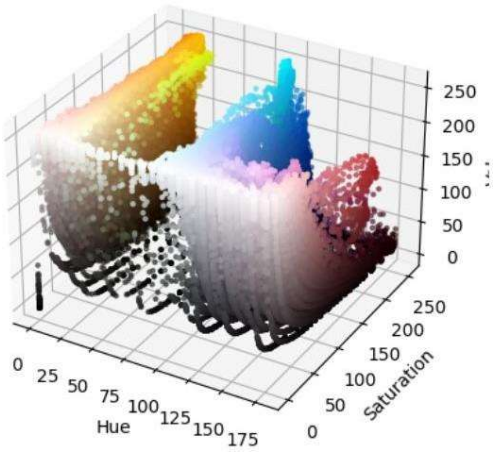
```
r, g, b = cv2.split(car)
fig = plt.figure()
axis = fig.add_subplot(1, 1, 1, projection="3d")
```



```
# Visualizing Nemo in HSV Color Space
```

```
hsv_cute = cv2.cvtColor(cute, cv2.COLOR_RGB2HSV)
```

```
h, s, v = cv2.split(hsv_cute)
fig = plt.figure()
axis = fig.add_subplot(1, 1, 1, projection="3d")

axis.scatter(h.flatten(), s.flatten(), v.flatten(), facecolors=pixel_colors, marker=".")
axis.set_xlabel("Hue")
axis.set_ylabel("Saturation")
axis.set_zlabel("Value")
plt.show()
```

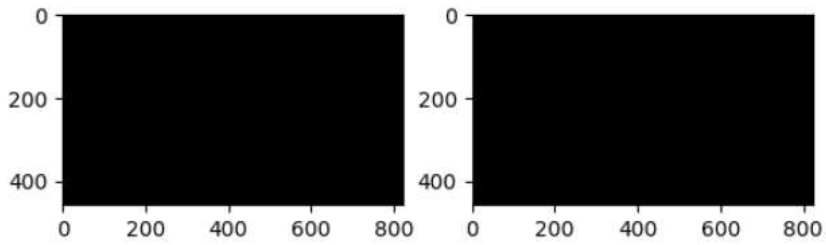


```
# picking out range

light_orange = (1, 190, 200)
dark_orange = (18, 255, 255)
```

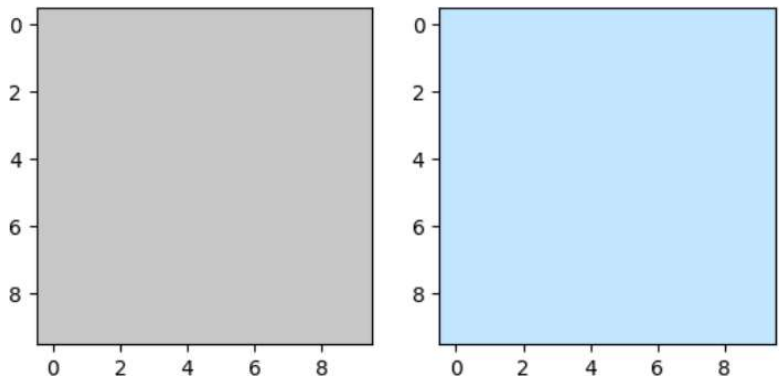
```
mask = cv2.inRange(hsv_cute, light_orange, dark_orange)
```

```
result = cv2.bitwise_and(cute, cute, mask=mask)
```

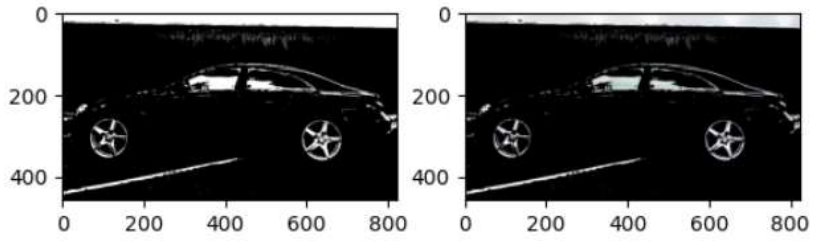
```
plt.subplot(1, 2, 1)
plt.imshow(mask, cmap="gray")
plt.subplot(1, 2, 2)
plt.imshow(result)
plt.show()
```



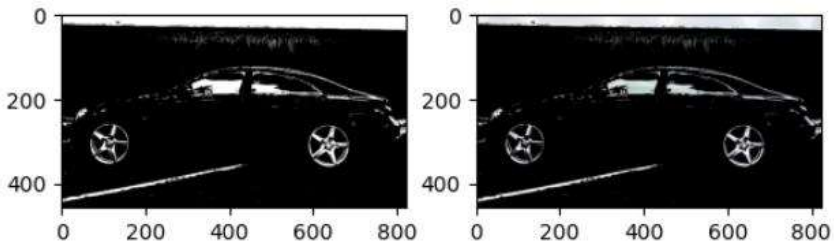

```
from matplotlib.colors import hsv_to_rgb

light_white = (0, 0, 200)
dark_white = (145, 60, 255)


lw_square = np.full((10, 10, 3), light_white, dtype=np.uint8) / 255.0
dw_square = np.full((10, 10, 3), dark_white, dtype=np.uint8) / 255.0

plt.subplot(1, 2, 1)
plt.imshow(hsv_to_rgb(lw_square))
plt.subplot(1, 2, 2)
plt.imshow(hsv_to_rgb(dw_square))
plt.show()
```



```
mask_white = cv2.inRange(hsv_car, light_white, dark_white)
result_white = cv2.bitwise_and(car, car, mask=mask_white)

plt.subplot(1, 2, 1)
plt.imshow(mask_white, cmap="gray")
plt.subplot(1, 2, 2)
plt.imshow(result_white)
plt.show()
```




```
final_mask = mask + mask_white

final_result = cv2.bitwise_and(cute, cute, mask=final_mask)
plt.subplot(1, 2, 1)
plt.imshow(final_mask, cmap="gray")
plt.subplot(1, 2, 2)
plt.imshow(final_result)
plt.show()
```



RESULT:

Executed several image filters on the image. Finally, we have successfully separated the red colour from the image and turned the remaining image to grayscale.

EX.NO: 10	CLUSTERING OF IMAGES
DATE: 13/10/2023	

AIM:

Cluster the image using K - means Clustering algorithm

THEORY:

It is basically a type of unsupervised learning method. An unsupervised learning method is a method in which we draw references from datasets consisting of input data without labelled responses. Generally, it is used as a process to find meaningful structure, explanatory underlying processes, generative features, and groupings inherent in a set of examples.

Clustering is the task of dividing the population or data points into a number of groups such that data points in the same groups are more similar to other data points in the same group and dissimilar to the data points in other groups. It is basically a collection of objects on the basis of similarity and dissimilarity between them.

Clustering on images perform Image Segmentation of pixel-wise segmentation. In this type of segmentation, we try to cluster the pixels that are together.

PROCEDURE:

Step 1: Importing necessary libraries
Step 2: Load image
Step 3: Reshaping of image
Step 4: Applying Kmeans clustering algorithm
Step 5: Showing the new image with 3 number of colours.

PROGRAM:

IMPORTING LIBRARIES:

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

```
# load image from images directory
image = cv2.imread('/content/car.jpg')
```

```
from google.colab.patches import cv2_imshow
cv2_imshow(image)
```

```
# load image from images directory
image = cv2.imread('/content/car.jpg')

# 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)
```

```
#criteria
criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 100, 0.85)

# Choosing number of cluster
k = 20

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.imshow(segmented_image)
```

```
<matplotlib.image.AxesImage at 0x7d57b1a0ca90>
```

RESULT:

Applied Kmeans clustering on an image dataset and got new image with number of clusters is three. That means the image is represented in three colours.

EX.NO: 11	DIMENSIONALITY REDUCTION
DATE: 20/10/2023	

AIM:

Reduce the dimension of an image dataset using dimensionality reduction techniques, Principle Component Analysis (PCA), Singular Value Decomposition (SVD) and Non negative Matrix Factorization (NMF).

THEORY:

Dimensionality reduction is the process of reducing the dimension of a dataset. Since the dataset contains large number of features, the computation will be complex. In order to reduce the computational complexity and difficulty dimensionality reduction is a useful method.

PCA

It is a feature extraction method in which it will create a new component called principle component which is a combination of other dimensions.

It is an orthogonal transformation in which the large variation by projection of data points taken as the first principle component and second largest is considered as second principle component and so on.

First it will find the Correlation matrix then it finds the eigen vectors corresponding to the eigen values. The unit eigen vector corresponds to large eigen value is the first principle component.

Eigen value, $| A - \lambda I | x = 0$, I is the identity matrix, λ is the eigen value and x is the eigen vector.

As we increase the number of components, it will increase the clarity of the image.

SVD

In SVD we are representing the matrix as the combination of three matrices.

$A=USV^T$

U is the unit eigen vector corresponds to AA^T

V is the unit vector corresponds to A^TA. And

S is the singular value which is the square root of eigen values.

AA^T and A^TA will be symmetrical matrices with same eigen values and the eigen values are positive always.

Combining these three matrices we will get the image in a lower dimension.

As we increase the number of components, it will increase the clarity of the image.

NMF

It is an advanced dimensionality reduction method where a matrix is represented by two lower ranking matrices W and H with non-negative elements.

A= W+H

W is the NMF base like the basic information on the image and

H is the coefficient matrix which is having intensity values.

As we increase the number of components, it will increase the clarity of the image.

PROCEDURE:

Step 1: Importing necessary libraries
Step 2: Load image dataset
Step 3: Pre-processing the dataset
Step 4: Applying PCA, SVD and NMF

PROGRAM

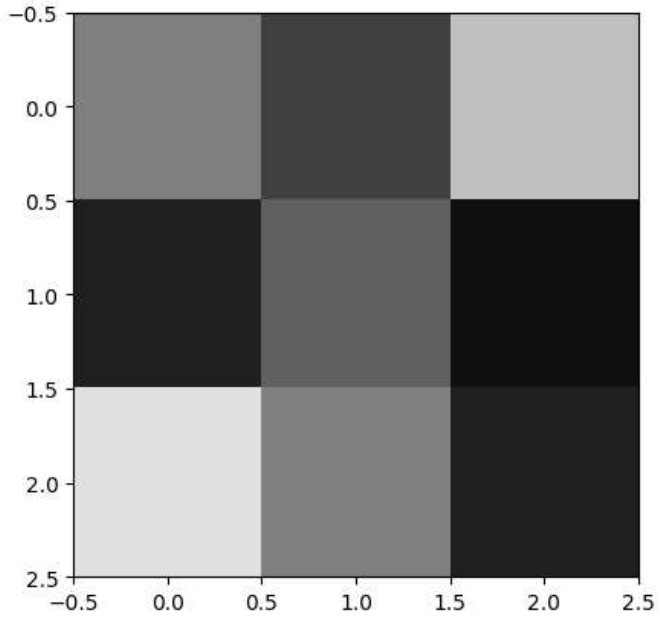
PCA

```
#import libraries
import cv2
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA

#Read and display image
img=cv2.imread("/content/car.jpg")
from google.colab.patches import cv2_imshow
cv2_imshow(img)
```

```
#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.
```

```
green_inverted = np.array([[128, 64, 192], [32, 96, 16], [224, 128, 32]])
img_compressed = (np.dstack((green_inverted,green_inverted,green_inverted))).astype(np.uint8)
plt.imshow(img_compressed)
```

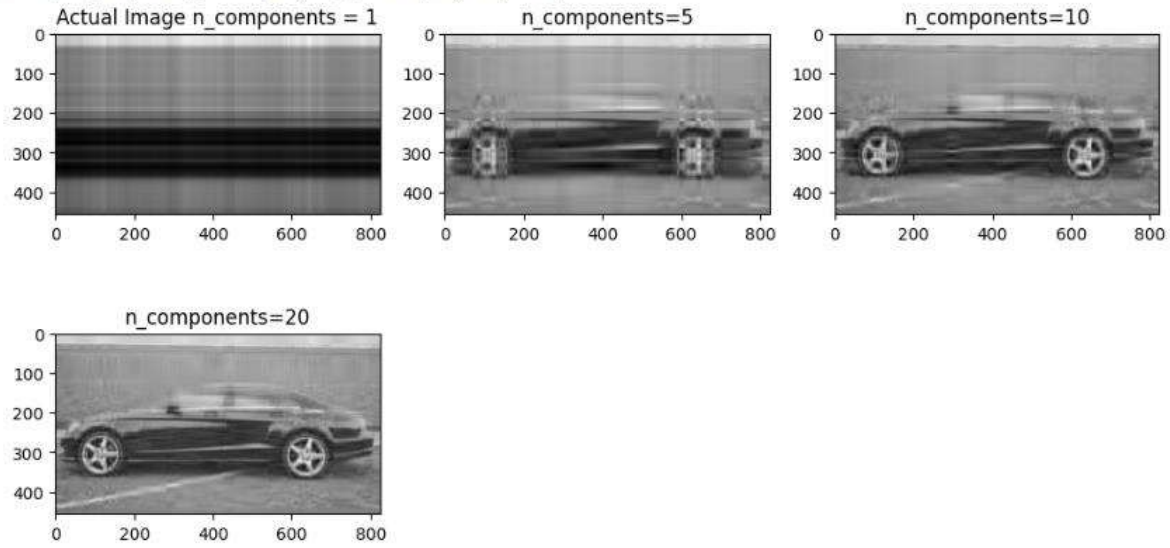
```
<matplotlib.image.AxesImage at 0x7ece87e90c40>
```



SVD

```
#import module import requests
import cv2
import numpy as np
import matplotlib.pyplot as plt

img=cv2.imread("/content/car.jpg") #Converting the image into gray scale for faster
gray_image=cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)

#Calculating the svd
u,s,v=np.linalg.svd(gray_image)

#inspect shapes of the matrices
print(f'u.shape:{u.shape},s.shape:{s.shape},v.shape:{v.shape}')

#plot images with different number of components
comps=[1,5,10,20]
plt.figure(figsize=(12,6))
for i in range(len(comps)):
  low_rank=u[:,:comps[i]]@np.diag(s[:comps[i]])@v[:comps[i],:]
  if(i==0):
    plt.subplot(2,3,i+1), plt.imshow(low_rank,cmap='gray'),
    plt.title(f'Actual Image n_components = {comps[i]}')
  else:
    plt.subplot(2,3,i+1),
    plt.imshow(low_rank,cmap='gray'),
    plt.title(f'n_components={comps[i]}')
```

```
u.shape:(457, 457),s.shape:(457,),v.shape:(824, 824)
```



NMF

```
from sklearn import decomposition

img=cv2.imread('/content/car.jpg')

#converting original image to gray image
gray_image=cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
cv2_imshow(gray_image)

estimator=decomposition.NMF(n_components=3,init='random',tol=5e-3)
W=estimator.fit_transform(gray_image)
H=estimator.components_
new_img=np.dot(W,H)
cv2_imshow(new_img)
plt.show()
```

RESULT:

Dimensions of the dataset is reduced using PCA, SVD and NMF and showed the lower dimensional images with different number of components.

It shows that PCA is more efficient because it extracts most relevant features and leave less significant features.

EX.NO: 12	EVALUATING THE LEARNING MODELS
DATE: 03/11/2023	

AIM:

To write about evaluating the learning models.

THEORY:

Evaluating machine learning models in computer vision (CV) is essential to assess their performance and determine how well they generalize to unseen data. The process of evaluation involves using various metrics and techniques to understand how effectively a model is solving a particular CV problem. Here are some commonmethods and metrics for evaluating learning models in computer vision:

1. Train-Test Split:

- The simplest way to evaluate a model is to split your dataset into a training set and a testing set. The model is trained on the training set and evaluated on the testing set. The accuracy of the model on the testing set is a common metric for classification tasks.

2. Cross-Validation:

- Cross-validation is a more robust method for evaluating models. It involves splitting the dataset into multiple subsets or "folds." The model is trained and tested on different combinations of these folds to ensure that it generalizes well across different data partitions.

3. Confusion Matrix:

- For classification tasks, the confusion matrix provides a detailed view of the model's performance. It shows the number of true positives, true negatives, false positives, and false negatives. From the confusion matrix, you can calculate metrics like accuracy, precision, recall, F1-score, and more.

4. ROC and AUC:

- Receiver Operating Characteristic (ROC) curves are used to evaluate binary classifiers. ROC curves illustrate the trade-off between the true positive rate and false positive rate as you vary the decision threshold. The Area Under the ROC Curve (AUC) is a single metric that quantifies the model's overall performance.

5. Mean Average Precision (mAP):

- mAP is commonly used in object detection and image segmentation tasks. It measures the precision-recall curve's average precision across different confidence thresholds, providing a comprehensive evaluation of model accuracy.

6. Intersection over Union (IoU):

- IoU is used to evaluate object detection and image segmentation models. It measures the overlap between the predicted and ground truth bounding boxes or segmentation masks. A higher IoU indicates a better model.

7. Root Mean Squared Error (RMSE):

- RMSE is a common metric for regression tasks. It measures the average error between predicted and actual values. Smaller RMSE values indicate better model performance.

8. Custom Metrics:

- Depending on your specific CV problem, you may need to define custom evaluation metrics. For instance, in facial recognition, you could use face verification metrics like False Acceptance Rate (FAR) or False Rejection Rate (FRR).

9. Visual Inspection:

- In many CV tasks, it's essential to visually inspect the model's outputs. This is especially true in tasks like image denoising, super-resolution, and style transfer, where qualitative assessment is crucial.

10. Validation on Real-world Data:

- Real-world data validation is essential to ensure that a model performs well in the actual environment where it will be deployed. This includes assessing the model's performance under various lighting conditions, angles, and backgrounds.

It's important to choose evaluation metrics and techniques that are relevant to your specific computer vision problem. Additionally, you may want to combine multiple metrics to gain a comprehensive understanding of your model's performance. Regular evaluation and validation are crucial for ensuring that your CV model works effectively and reliably in real-world scenarios.

RESULT:

Therefore, evaluating the learning models are successfully.