**--Lab-6--Implement Video Operation Play a video using OpenCV—**

**import cv2**

**import numpy as np**

**# Create a VideoCapture object and read from input file**

**# If the input is the camera, pass 0 instead of the video file name**

**cap = cv2.VideoCapture('Guru.mp4')**

**# Check if camera opened successfully**

**if (cap.isOpened()== False):**

**print("Error opening video stream or file")**

**# Read until video is completed**

**while(cap.isOpened()):**

**# Capture frame-by-frame**

**ret, frame = cap.read()**

**if ret == True:**

**# Display the resulting frame**

**cv2.imshow('Frame',frame)**

**# Press Q on keyboard to exit**

**if cv2.waitKey(25) & 0xFF == ord('q'):**

**break**

**# Break the loop**

**else:**

**break**

**# When everything done, release the video capture object**

**cap.release()**

**# Closes all the frames**

**cv2.destroyAllWindows()**

**--Lab-7--To perform Hough transform on an image—**

**Code-1) # Detecting Lines with Hough Transform**

import cv2

import numpy as np

# Load image

img = cv2.imread("OCR.jpg")

gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)

# Edge detection

edges = cv2.Canny(gray, 50, 150, apertureSize=3)

# Hough Transform for lines

lines = cv2.HoughLines(edges, 1, np.pi/180, 100)

# Draw lines

if lines is not None:

for rho, theta in lines[:,0]:

a = np.cos(theta)

b = np.sin(theta)

x0 = a\*rho

y0 = b\*rho

x1 = int(x0 + 1000\*(-b))

y1 = int(y0 + 1000\*(a))

x2 = int(x0 - 1000\*(-b))

y2 = int(y0 - 1000\*(a))

cv2.line(img, (x1,y1), (x2,y2), (0,0,255), 2)

cv2.imshow("Hough Lines", img)

cv2.waitKey(0)

cv2.destroyAllWindows()

**Code-2) Detecting Circles with Hough Transform**

import cv2

import numpy as np

# Load image

img = cv2.imread("OCR.jpg")

gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)

# Blur to reduce noise

gray = cv2.medianBlur(gray, 5)

# Hough Transform for circles

circles = cv2.HoughCircles(

gray, cv2.HOUGH\_GRADIENT, dp=1.2, minDist=30,

param1=50, param2=30, minRadius=10, maxRadius=100

)

# Draw circles

if circles is not None:

circles = np.uint16(np.around(circles))

for x, y, r in circles[0, :]:

cv2.circle(img, (x, y), r, (0, 255, 0), 2)

cv2.circle(img, (x, y), 2, (0, 0, 255), 3)

cv2.imshow("Hough Circles", img)

cv2.waitKey(0)

cv2.destroyAllWindows()

**--Lab-8--To perform various pattern classification and analysis techniques on images—**

import cv2

import numpy as np

from sklearn.svm import SVC

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import accuracy\_score

# --- 1. Feature Extraction (using SIFT) ---

def extract\_sift\_features(image\_path):

image = cv2.imread(image\_path, cv2.IMREAD\_GRAYSCALE)

if image is None:

print(f"Error: Could not load image {image\_path}")

return None, None

sift = cv2.SIFT\_create()

keypoints, descriptors = sift.detectAndCompute(image, None)

return descriptors

# --- 2. Prepare Dataset (Example with dummy data) ---

# In a real application, you would load images from different classes

# and extract features for each.

# For demonstration, let's create some dummy feature vectors and labels.

# Imagine these are descriptors from different images.

features\_class\_a = [np.random.rand(10, 128) for \_ in range(20)] # 20 images, 10 keypoints each, 128-dim descriptor

features\_class\_b = [np.random.rand(15, 128) for \_ in range(20)] # 20 images, 15 keypoints each, 128-dim descriptor

# Combine all descriptors into a single list and create corresponding labels

all\_descriptors = []

labels = []

for desc in features\_class\_a:

if desc is not None:

all\_descriptors.extend(desc)

labels.extend([0] \* len(desc)) # Class 0

for desc in features\_class\_b:

if desc is not None:

all\_descriptors.extend(desc)

labels.extend([1] \* len(desc)) # Class 1

# Convert to NumPy arrays

X = np.array(all\_descriptors)

y = np.array(labels)

# --- 3. Train a Classifier (Support Vector Machine) ---

# Split data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Initialize and train an SVM classifier

svm\_model = SVC(kernel='linear')

svm\_model.fit(X\_train, y\_train)

# --- 4. Evaluate the Classifier ---

y\_pred = svm\_model.predict(X\_test)

accuracy = accuracy\_score(y\_test, y\_pred)

print(f"Classifier Accuracy: {accuracy:.2f}")

**--Lab-9--To perform various pattern clustering and analysis techniques on images.—**

import cv2

import numpy as np

# Load the image

img = cv2.imread('CV\_Test.jpg')

# Reshape the image to a 2D array of pixels and their color values

# (height \* width, 3 for BGR channels)

Z = img.reshape((-1, 3))

Z = np.float32(Z)

# Define criteria for k-means (type, max\_iter, epsilon)

criteria = (cv2.TERM\_CRITERIA\_EPS + cv2.TERM\_CRITERIA\_MAX\_ITER, 10, 1.0)

# Number of clusters (K)

K = 8

# Apply k-means

compactness, labels, centers =

cv2.kmeans(Z, K, None, criteria, 10, cv2.KMEANS\_RANDOM\_CENTERS)

# Convert centers to uint8 and reshape back to the original image dimensions

centers = np.uint8(centers)

res = centers[labels.flatten()]

res2 = res.reshape((img.shape))

# Display the segmented/quantized image

cv2.imshow('Quantized Image', res2)

cv2.waitKey(0)

cv2.destroyAllWindows()

**--Lab-5- To perform Detect corner of an image using OpenCV--**

To **detect corners in an image using OpenCV**, one of the most common methods is **Harris Corner Detection** or **Shi-Tomasi Corner Detection**.

Below is a **Python example using OpenCV** for both methods.

**✅ 1. Harris Corner Detection (OpenCV)**

import cv2

import numpy as np

# Load image and convert to grayscale

img = cv2.imread('your\_image.jpg')

gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)

# Convert to float32

gray = np.float32(gray)

# Harris corner detection

dst = cv2.cornerHarris(gray, blockSize=2, ksize=3, k=0.04)

# Dilate to mark the corners

dst = cv2.dilate(dst, None)

# Threshold for an optimal value

img[dst > 0.01 \* dst.max()] = [0, 0, 255] # Mark corners in red

cv2.imshow('Harris Corners', img)

cv2.waitKey(0)

cv2.destroyAllWindows()

**✅ 2. Shi-Tomasi Corner Detection (cv2.goodFeaturesToTrack)**

import cv2

import numpy as np

# Load image and convert to grayscale

img = cv2.imread('your\_image.jpg')

gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)

# Detect corners using Shi-Tomasi method

corners = cv2.goodFeaturesToTrack(gray, maxCorners=100, qualityLevel=0.01, minDistance=10)

corners = np.int0(corners)

# Draw corners

for i in corners:

x, y = i.ravel()

cv2.circle(img, (x, y), 5, (0, 255, 0), -1)

cv2.imshow('Shi-Tomasi Corners', img)

cv2.waitKey(0)

cv2.destroyAllWindows()

**📝 Notes:**

* blockSize: Neighborhood size considered for corner detection.
* ksize: Aperture parameter for the Sobel operator.
* k: Harris detector free parameter in the equation.
* maxCorners: Maximum number of corners to return.
* qualityLevel: Minimum accepted quality of image corners.

**Lab-4 To perform Circle Detection using OpenCV | Python**

**Circle Detection using OpenCV in Python** is commonly done using the **Hough Circle Transform**, a feature extraction technique for detecting circles in images.

**🔹 What is Circle Detection?**

**Circle Detection** is a process in computer vision to find circles in digital images. It is especially useful in robotics, image analysis, and medical imaging.

**🔹 Purpose**

* To locate circular shapes in an image.
* Helps in object recognition, shape analysis, and tracking.

**🔹 Applications**

* Coin recognition systems
* Eye (iris/pupil) detection
* Ball tracking in sports
* Traffic sign recognition
* Industrial automation

**🔹 OpenCV Function Used**

cv2.HoughCircles()

**✅ Example Code: Circle Detection using OpenCV**

import cv2

import numpy as np

# Load image

image = cv2.imread('circles.png') # Replace with your image path

output = image.copy()

gray = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)

# Apply Gaussian Blur to reduce noise

gray\_blurred = cv2.GaussianBlur(gray, (9, 9), 2)

# Detect circles using HoughCircles

circles = cv2.HoughCircles(gray\_blurred,

cv2.HOUGH\_GRADIENT,

dp=1.2,

minDist=20,

param1=50,

param2=30,

minRadius=0,

maxRadius=0)

# If some circles are detected, draw them

if circles is not None:

circles = np.uint16(np.around(circles))

for i in circles[0, :]:

# Draw the outer circle

cv2.circle(output, (i[0], i[1]), i[2], (0, 255, 0), 2)

# Draw the center of the circle

cv2.circle(output, (i[0], i[1]), 2, (0, 0, 255), 3)

# Show results

cv2.imshow('Detected Circles', output)

cv2.waitKey(0)

cv2.destroyAllWindows()

**🔹 Parameters Explained**

| **Parameter** | **Meaning** |
| --- | --- |
| dp | Inverse ratio of accumulator resolution to image resolution. |
| minDist | Minimum distance between detected centers. |
| param1 | Higher threshold for Canny edge detector. |
| param2 | Accumulator threshold for circle detection. |
| minRadius | Minimum radius of circles to detect. |
| maxRadius | Maximum radius of circles to detect. |

**Lab-3: To perform Image Smoothing and Blurring using Python OpenCV**

Image smoothing and blurring are common techniques used in image processing to reduce noise and detail in an image. OpenCV provides several methods for this in Python.

* Image noise refers to random variations in brightness or color information within an image, often appearing as grainy or speckled patterns.

**✅ Common Methods in OpenCV for Smoothing/Blurring:**

1. **Averaging (Mean Filter)**
2. **Gaussian Blur**
3. **Median Blur**
4. **Bilateral Filter**

**📌 Required Library:**

pip install opencv-python

**✅ Example Code: Image Smoothing and Blurring in Python (Using OpenCV)**

**🔍 Explanation:**

| **Method** | **Description** | **Use Case** |
| --- | --- | --- |
| cv2.blur() | Averages pixels in a kernel area (simple smoothing) | Basic denoising |
| cv2.GaussianBlur() | Uses Gaussian kernel for weighted smoothing | Better for natural blurring |
| cv2.medianBlur() | Takes the median value of surrounding pixels | Good for removing salt-and-pepper noise |
| cv2.bilateralFilter() | Smooths image while keeping edges sharp | Best for face smoothing, edge-preserving |

1. **Gaussian Blur Example-**

import cv2

import numpy as np

# Load the image

image = cv2.imread('your\_image.jpg')

# Check if image loaded successfully

if image is None:

print("Error: Could not load image.")

else:

# Apply Gaussian blurring with a 5x5 kernel and sigmaX=0 (auto-calculated)

blurred\_gaussian = cv2.GaussianBlur(image, (5, 5), 0)

# Display the original and blurred images

cv2.imshow('Original Image', image)

cv2.imshow('Gaussian Blurred Image', blurred\_gaussian)

cv2.waitKey(0)

cv2.destroyAllWindows()

**2. To perform Image Transformations using OpenCV in Python**

**Image Transformations** in computer vision refer to techniques used to **modify or manipulate an image** in terms of its **geometry, orientation, size, or pixel values** without altering its essential content.

These transformations are crucial for tasks like **image preprocessing, data augmentation, image registration, or computer vision model training**.

**🔄 Types of Image Transformations:**

**1. Geometric Transformations**

Change the spatial relationship of pixels.

* **Translation** – Moves the image in x/y direction.
* **Rotation** – Rotates the image by a certain angle.
* **Scaling** – Resizes the image (zoom in/out).
* **Shearing** – Tilts the image.
* **Flipping** – Mirror the image horizontally or vertically.

Here's a step-by-step guide with **examples** for each:

**✅ 1. Import Required Libraries**

import cv2

import numpy as np

**📷 2. Read the Image**

img = cv2.imread('your\_image.jpg')

cv2.imshow('Original Image', img)

cv2.waitKey(0)

cv2.destroyAllWindows()

**🔀 3. Translation (Moving the image)**

# Move image by 100px right and 50px down

rows, cols = img.shape[:2]

M = np.float32([[1, 0, 100], [0, 1, 50]])

translated\_img = cv2.warpAffine(img, M, (cols, rows))

cv2.imshow('Translated Image', translated\_img)

cv2.waitKey(0)

cv2.destroyAllWindows()

**🔄 4. Rotation**

# Rotate by 45 degrees around the center

center = (cols // 2, rows // 2)

M = cv2.getRotationMatrix2D(center, 45, 1.0)

rotated\_img = cv2.warpAffine(img, M, (cols, rows))

cv2.imshow('Rotated Image', rotated\_img)

cv2.waitKey(0)

cv2.destroyAllWindows()

**🔍 5. Resizing**

resized\_img = cv2.resize(img, (300, 300), interpolation=cv2.INTER\_LINEAR)

cv2.imshow('Resized Image', resized\_img)

cv2.waitKey(0)

cv2.destroyAllWindows()

**↔️ 6. Flipping**

# Flip vertically (0), horizontally (1), both (-1)

flipped\_img = cv2.flip(img, 1)

cv2.imshow('Flipped Image', flipped\_img)

cv2.waitKey(0)

cv2.destroyAllWindows()

**🧭 7. Affine Transformation**

pts1 = np.float32([[50, 50], [200, 50], [50, 200]])

pts2 = np.float32([[10, 100], [200, 50], [100, 250]])

M = cv2.getAffineTransform(pts1, pts2)

affine\_img = cv2.warpAffine(img, M, (cols, rows))

cv2.imshow('Affine Transform', affine\_img)

cv2.waitKey(0)

cv2.destroyAllWindows()

**🔳 8. Perspective Transformation**

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

pts2 = np.float32([[0,0], [300,0], [0,300], [300,300]])

M = cv2.getPerspectiveTransform(pts1, pts2)

perspective\_img = cv2.warpPerspective(img, M, (300, 300))

cv2.imshow('Perspective Transform', perspective\_img)

cv2.waitKey(0)

cv2.destroyAllWindows()

**💡 Notes:**

* Always call cv2.waitKey(0) and cv2.destroyAllWindows() after displaying an image.
* Replace 'your\_image.jpg' with the actual path to your image.

Would you like this in **.py file format** or need **plots using matplotlib** as well?

-----------------------------------------------------------------------------------------

1. **Translation (Moving Image) in OpenCV**

**Translation** means shifting an image along the X and/or Y axis. In computer vision, translation is commonly used for image augmentation or shifting focus within an image.

**🧠 Concept**

We use a **transformation matrix** to move the image:

Translation Matrix=[10tx01ty]\text{Translation Matrix} = \begin{bmatrix} 1 & 0 & t\_x \\ 0 & 1 & t\_y \end{bmatrix}

Where:

* t\_x = shift along X-axis
* t\_y = shift along Y-axis

**🧾 Python Code using OpenCV**

import cv2

import numpy as np

# Load the image

image = cv2.imread('your\_image.jpg')

# Define translation values

tx = 100 # Shift 100 pixels to the right

ty = 50 # Shift 50 pixels down

# Create the translation matrix

M = np.float32([[1, 0, tx],[0, 1, ty]])

# Apply the translation

translated\_image = cv2.warpAffine(image, M, (image.shape[1], image.shape[0]))

# Display the result

cv2.imshow('Original Image', image)

cv2.imshow('Translated Image', translated\_image)

cv2.waitKey(0)

cv2.destroyAllWindows()

**🔍 Important Functions Used**

* cv2.imread() – Reads an image.
* np.float32() – Defines the matrix with 32-bit float values.
* cv2.warpAffine() – Applies the affine transformation.
* cv2.imshow() – Displays the image.

2. **🔄 Rotation in OpenCV (Python) with Code**

In OpenCV, rotating an image involves using an **affine transformation matrix**. This is commonly done using cv2.getRotationMatrix2D() and cv2.warpAffine().

**✅ Code to Rotate an Image**

import cv2

import numpy as np

# Load the image

image = cv2.imread('your\_image.jpg')

# Get the image dimensions

(h, w) = image.shape[:2]

# Define the center of the image (around which rotation happens)

center = (w // 2, h // 2)

# Define the angle of rotation (positive = counterclockwise)

angle = 45 # rotate by 45 degrees

# Define the scale of the image

scale = 1.0 # keep original size

# Get the rotation matrix

rotation\_matrix = cv2.getRotationMatrix2D(center, angle, scale)

# Perform the rotation using warpAffine

rotated\_image = cv2.warpAffine(image, rotation\_matrix, (w, h))

# Display the rotated image

cv2.imshow('Rotated Image', rotated\_image)

cv2.waitKey(0)

cv2.destroyAllWindows()

**🧠 Explanation**

* cv2.getRotationMatrix2D(center, angle, scale):
  + Returns a 2x3 affine transform matrix for rotation.
* cv2.warpAffine(image, matrix, (w, h)):

Let me know if you want this in a Jupyter notebook, with sample images, or rotation via a GUI slider.

**3. Image Resizing-**

3. In OpenCV, **resizing** an image refers to changing its dimensions (width and height) using the cv2.resize() function.

**✅ Syntax:**

resized\_image = cv2.resize(src, dsize, interpolation)

* src: Original image.
* dsize: Desired size in the form (width, height).
* interpolation: (Optional) Interpolation method like cv2.INTER\_LINEAR, cv2.INTER\_AREA, etc.

**📌 Common Interpolation Methods:**

* cv2.INTER\_LINEAR: Default, good for upscaling.
* cv2.INTER\_AREA: Preferred for downscaling.
* cv2.INTER\_NEAREST: Fastest, lower quality.

**✅ Example Code:**

import cv2

# Load the original image

image = cv2.imread('example.jpg')

# Resize to a specific width and height

resized = cv2.resize(image, (300, 200)) # width=300, height=200

# Resize by scale (e.g., half the size)

resized\_half = cv2.resize(image, (0, 0), fx=0.5, fy=0.5)

# Show images

cv2.imshow('Original', image)

cv2.imshow('Resized (300x200)', resized)

cv2.imshow('Resized Half', resized\_half)

cv2.waitKey(0)

cv2.destroyAllWindows()

**📝 Notes:**

* If you're using **scaling factors** (fx, fy), set dsize to (0, 0).
* Always use cv2.waitKey() and cv2.destroyAllWindows() to display and then close image windows properly.

Let me know if you want to resize using aspect ratio or with GUI sliders.

* **Lab-1** **Image Enhancement Techniques using OpenCV – Python—**

**IMAGE ENHANCEMENT**

It is the process of image manipulation to make it more suitable for specific use. It provides better contrast and a more detailed image and is used to enhance medical images, images captured in remote sensing, images from satellite etc.

Image enhancement is the process of improving the quality and appearance of an image. It can be used to correct flaws or defects in an image, or to simply make an image more visually appealing. Image enhancement techniques can be applied to a wide range of images, including photographs, scans, and digital images. Some common goals of image enhancement include increasing contrast, sharpness, and colorfulness; reducing noise and blur; and correcting distortion and other defects. Image enhancement techniques can be applied manually using image editing software, or automatically using algorithms and computer programs such as OpenCV.   
In this article, we will explore a variety of image enhancement techniques that can be performed using OpenCV and Python. OpenCV is a powerful, open-source computer vision library that provides a wide range of image processing and computer vision algorithms. By combining the capabilities of OpenCV with the versatility of Python, we can easily implement a variety of image enhancement techniques to improve the quality and appearance of our images.

**Getting started**

Prerequisites for running the code are:

1. Python
2. Python-OpenCV Library
3. Jupiter Notebook

**Image Enhancement can be done in:**

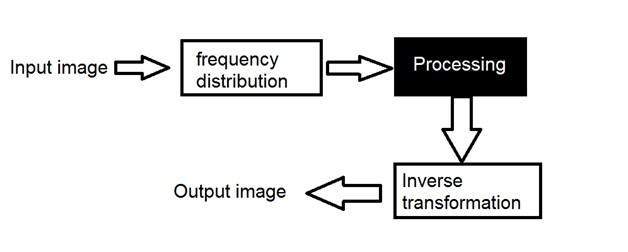
* **Spatial Domain:**

There is a direct manipulation of a pixel in an image (on the image plane)



* **Frequency Domain:**

Processing the image is based on modifying the Fourier transform of an image.



**Why is it done?**

It is done for the following reasons:

* Highlighting interesting detail in images.
* Removing noise from images.
* Making images more visually appealing.

The transformation function is given as follows:

**S = T(r)**

Where, r is the pixels of the input image. s is the pixels of the output image. T is a transformation function that maps each value to r to each value of s.

It can be achieved through grey level transformations.

1. **Adjusting brightness and contrast**

There are several ways to adjust the brightness and contrast of an image using OpenCV and Python. One common method is to use the **cv2.addWeighted()** function, which allows you to adjust the brightness by adding a scalar value to each pixel in the image, and the contrast by scaling the pixel values.

#Import the necessary libraries

import cv2

import matplotlib.pyplot as plt

import numpy as np

# Load the image

image = cv2.imread('GFG.jpeg')

#Plot the original image

plt.subplot(1, 2, 1)

plt.title("Original")

plt.imshow(image)

# Adjust the brightness and contrast

# Adjusts the brightness by adding 10 to each pixel value

brightness = 10

# Adjusts the contrast by scaling the pixel values by 2.3

contrast = 2.3

image2 = cv2.addWeighted(image, contrast, np.zeros(image.shape, image.dtype), 0, brightness)

#Save the image

cv2.imwrite('modified\_image.jpg', image2)

#Plot the contrast image

plt.subplot(1, 2, 2)

plt.title("Brightness & contrast")

plt.imshow(image2)

plt.show()

1. **Sharpening images**

Sharpening is the process of enhancing the edges and fine details in an image to make it appear sharper and more defined. It is important because it can help to bring out the details and features in an image, making it more visually appealing and easier to understand.

One common method for sharpening images using OpenCV and Python is to use the **cv2.filter2D()** function, which convolves the image with a kernel. The kernel can be designed to enhance the edges in the image, resulting in a sharper image.

#Import the necessary libraries

import cv2

import matplotlib.pyplot as plt

import numpy as np

# Load the image

image = cv2.imread('GFG.jpeg')

#Plot the original image

plt.subplot(1, 2, 1)

plt.title("Original")

plt.imshow(image)

# Create the sharpening kernel

kernel = np.array([[0, -1, 0], [-1, 5, -1], [0, -1, 0]])

# Sharpen the image

sharpened\_image = cv2.filter2D(image, -1, kernel)

#Save the image

cv2.imwrite('sharpened\_image.jpg', sharpened\_image)

#Plot the sharpened image

plt.subplot(1, 2, 2)

plt.title("Sharpening")

plt.imshow(sharpened\_image)

plt.show()

1. **Enhancing color in images**

Color enhancement is adjusting the colors in an image to make them more vibrant, balanced, or natural.

There are several ways to enhance the colors in an image using OpenCV and Python. One common method is to use the **cv2.cvtColor()** function, which allows you to convert the image from one color space to another. This can be useful for adjusting the color balance or saturation of the image.  
Here is an example of how to enhance the colors in an image using the **cv2.cvtColor()** function:

#Import the necessary libraries

import cv2

import matplotlib.pyplot as plt

import numpy as np

# Load the image

image = cv2.imread('GFG.jpeg')

#Plot the original image

plt.subplot(1, 2, 1)

plt.title("Original")

plt.imshow(image)

# Convert the image from BGR to HSV color space

image = cv2.cvtColor(image, cv2.COLOR\_RGB2HSV)

# Adjust the hue, saturation, and value of the image

# Adjusts the hue by multiplying it by 0.7

image[:, :, 0] = image[:, :, 0] \* 0.7

# Adjusts the saturation by multiplying it by 1.5

image[:, :, 1] = image[:, :, 1] \* 1.5

# Adjusts the value by multiplying it by 0.5

image[:, :, 2] = image[:, :, 2] \* 0.5

# Convert the image back to BGR color space

image2 = cv2.cvtColor(image, cv2.COLOR\_HSV2BGR)

#Save the image

cv2.imwrite('enhanced coloured.jpg', image2)

#Plot the enhanced image

plt.subplot(1, 2, 2)

plt.title("enhanced coloured")

plt.imshow(image2)

plt.show()