Experiment 3

Image Processing Basics in OpenCV

Aim:

1) To implement the following basic functions on an image or a video in Open CV:

Convert to Gray Scale, implement image intensity transformations, Blur, Draw shapes and adding Text, Mask or Crop, Histogram, Thresholding, Image Addition and Image Subtraction.

Software/ Packages Used:

1. Pycharm IDE
2. Libraries used:
    - NumPy
    - opencv-python
    - matplotlib
    - scipy

Programs:

1. To implement the basic functions of image processing in openCV

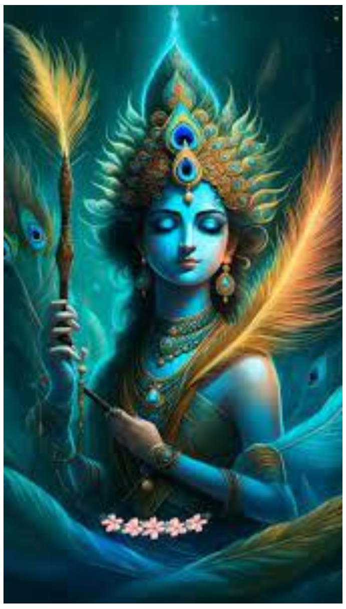
Image intensity transformations

1. Image Negative

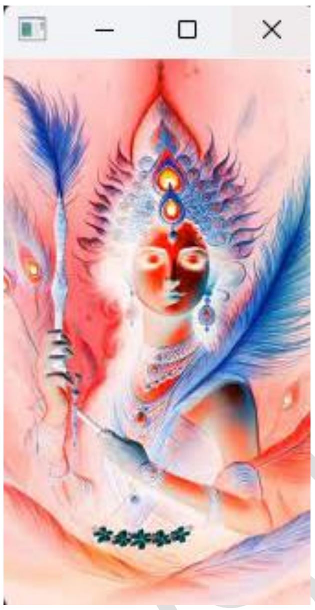
```
import cv2 as cv
import numpy as np
img=cv.imread("C:/Users/21r228\Downloads\download.jpg")
print("img.itype")
print("img")
img_res=255-img
cv.imshow("1",img)
cv.imshow("2",img_res)
```

IMAGE NEGATIVE:

INPUT:



OUTPUT:



PIXEL MATRIX:

IMAGE:

```
[[41 42  8]
  [41 42  8]
  [42 42  6]
  ...
  [35 34  6]
  [29 28  0]
  [29 28  0]]
```

IMAGE_REVERSE:

```
[[214 213 247]
  [214 213 247]
  [213 213 249]
  ...
  [220 221 249]
  [226 227 255]
  [226 227 255]]
```

```
    cv.waitKey(0)
    cv.destroyAllWindows()
```

2. Logarithmic Transformation

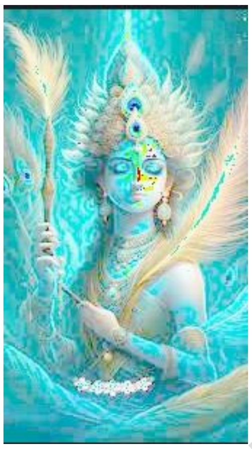
```
import cv2 as cv
import numpy as np

# Open the image.
img = cv.imread("C:/Users/21r228\Downloads\download.jpg")

# Apply log transform.
c = 255/(np.log(1 + np.max(img)))
log_transformed = c * np.log(1 + img)

# Specify the data type.
log_transformed = np.array(log_transformed, dtype = np.uint8)

# Save the output.
cv.imwrite('log_transformed.jpg', log_transformed)
cv.waitKey(0)
cv.destroyAllWindows()
```

3. Power Law (Gamma) Transformation

```
import cv2 as cv
import numpy as np

# Open the image.
img = cv.imread('C:/Users/21r228\Downloads\download.jpg')

# Trying 4 gamma values.
for gamma in [0.1, 0.5, 1.2, 2.2]:

   # Apply gamma correction.
   gamma_corrected = np.array(255 * (img / 255) ** gamma, dtype='uint8')

   # Save edited images.
   cv.imwrite('gamma_transformed' + str(gamma) + '.jpg', gamma_corrected)
cv.waitKey(0)
cv.destroyAllWindows()
```

LOGARITHMIC TRANSFORMATION

INPUT: **OUTPUT:**

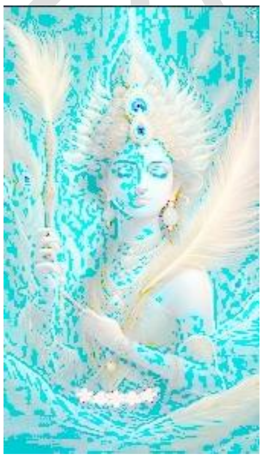


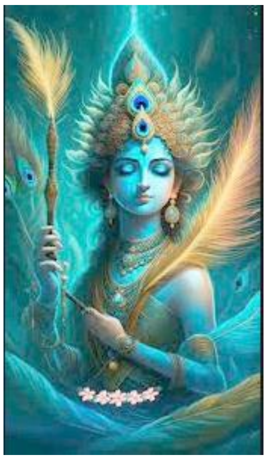
Power Law (Gamma) Transformation

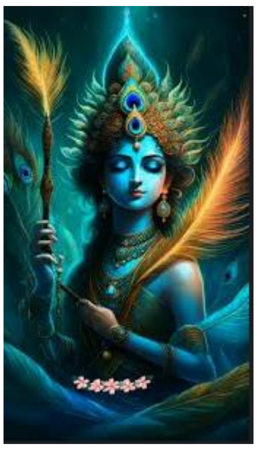
INPUT:

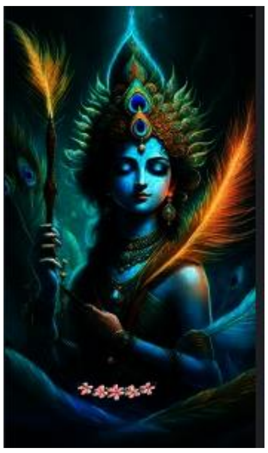
OUTPUT:

0.1 0.5 1.2 2.2







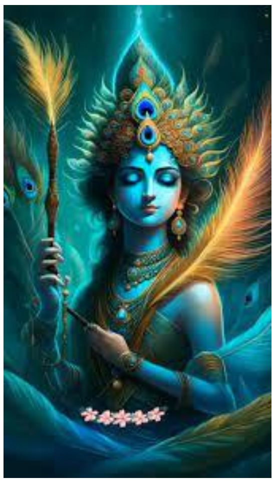


4. Contrast stretching

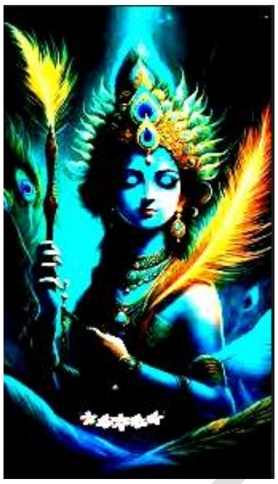
```
import cv2
import numpy as np

# Function to map each intensity level to output intensity level.
def pixelVal(pix, r1, s1, r2, s2):
    if (0 <= pix and pix <= r1):
        return (s1 / r1)*pix
    elif (r1 < pix and pix <= r2):
        return ((s2 - s1)/(r2 - r1)) * (pix - r1) + s1
    else:
        return ((255 - s2)/(255 - r2)) * (pix - r2) + s2

# Open the image.
img = cv2.imread('C:/Users/21r228\Downloads\download.jpg')

# Define parameters.
r1 = 70
s1 = 0
r2 = 140
s2 = 255

# Vectorize the function to apply it to each value in the Numpy array.
pixelVal_vec = np.vectorize(pixelVal)

# Apply contrast stretching.
contrast_stretched = pixelVal_vec(img, r1, s1, r2, s2)

# Save edited image.
cv2.imwrite('contrast_stretch.jpg', contrast_stretched)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

<u>CONTRAST STRETCHING</u>

INPUT:



OUTPUT:



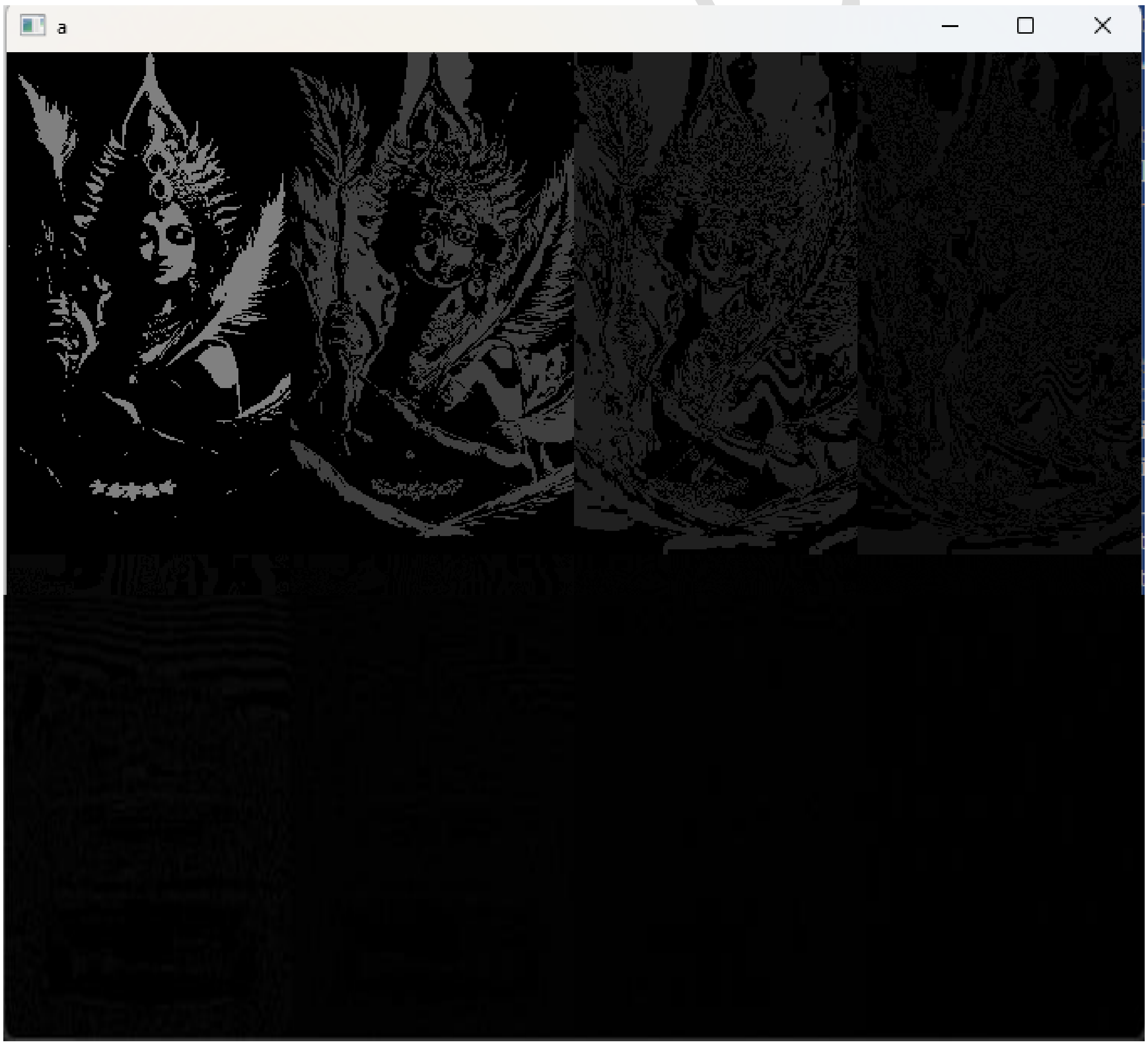
5. Bit plane slicing

```
import numpy as np
import cv2

# Read the image in greyscale
img = cv2.imread('C:/Users/21r228\Downloads\download.jpg', 0)

# Iterate over each pixel and change pixel value to binary using np.binary_repr() and store
it in a list.
lst = []
for i in range(img.shape[0]):
    for j in range(img.shape[1]):
        lst.append(np.binary_repr(img[i][j], width=8))  # width = no. of bits

# We have a list of strings where each string represents binary pixel value. To extract bit
planes we need to iterate over the strings and store the characters corresponding to bit
planes into lists.
# Multiply with 2^(n-1) and reshape to reconstruct the bit image.
eight_bit_img = (np.array([int(i[0]) for i in lst], dtype=np.uint8) *
128).reshape(img.shape[0], img.shape[1])
seven_bit_img = (np.array([int(i[1]) for i in lst], dtype=np.uint8) *
64).reshape(img.shape[0], img.shape[1])
six_bit_img = (np.array([int(i[2]) for i in lst], dtype=np.uint8) * 32).reshape(img.shape[0],
img.shape[1])
five_bit_img = (np.array([int(i[3]) for i in lst], dtype=np.uint8) * 16).reshape(img.shape[0],
img.shape[1])
four_bit_img = (np.array([int(i[4]) for i in lst], dtype=np.uint8) * 8).reshape(img.shape[0],
img.shape[1])
three_bit_img = (np.array([int(i[5]) for i in lst], dtype=np.uint8) *
4).reshape(img.shape[0], img.shape[1])
two_bit_img = (np.array([int(i[6]) for i in lst], dtype=np.uint8) * 2).reshape(img.shape[0],
img.shape[1])
one_bit_img = (np.array([int(i[7]) for i in lst], dtype=np.uint8) * 1).reshape(img.shape[0],
img.shape[1])

# Concatenate these images for ease of display using cv2.hconcat()
finalr = cv2.hconcat([eight_bit_img, seven_bit_img, six_bit_img, five_bit_img])
finalv = cv2.hconcat([four_bit_img, three_bit_img, two_bit_img, one_bit_img])

# Vertically concatenate
final = cv2.vconcat([finalr, finalv])

# Display the images
cv2.imshow('a', final)
cv2.waitKey(0)
```

Bit plane slicing

INPUT:

OUTPUT:



6. Paint and Draw on an image

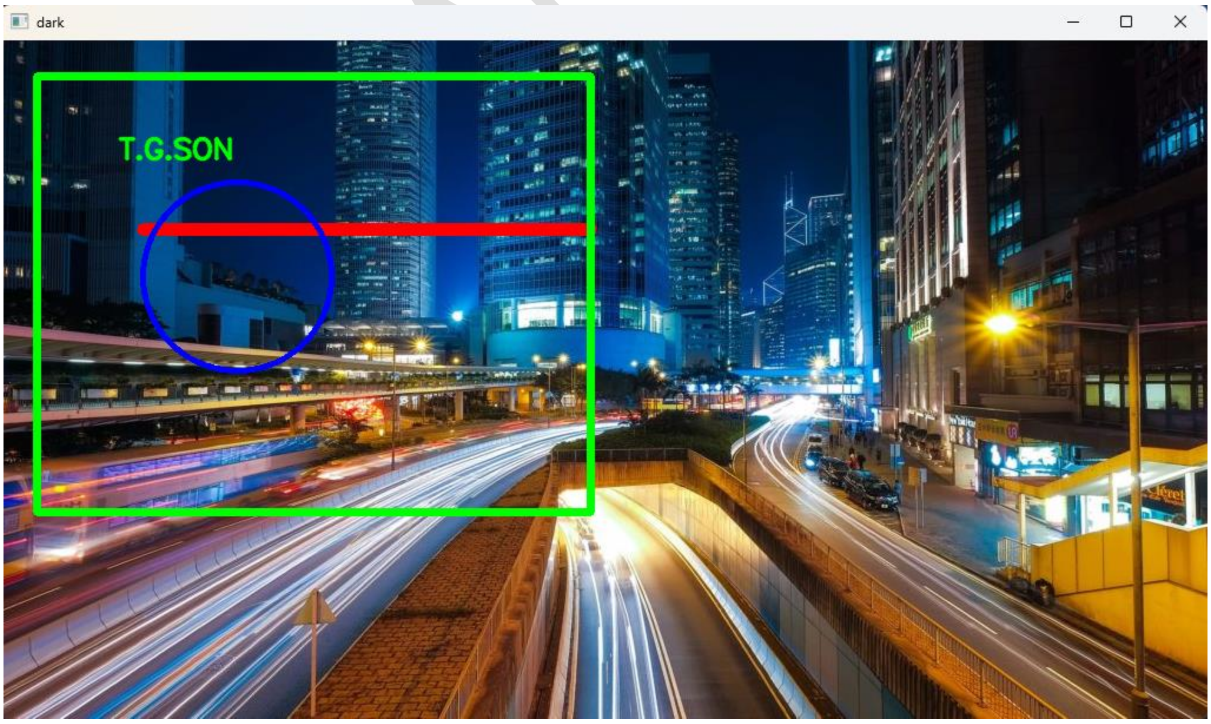
# 1. Paint the image with certain color
# 2. Draw a Rectangle
# 3. Draw a circle
# 4. Draw a line
# 5. Write text

```
# Python3 program to draw solid-colored
# image using numpy.zeroes() function
import numpy as np
import cv2

# Creating a black image with 3 channels
# RGB and unsigned int datatype
img = cv2.imread("C:/Users/21r228\Downloads\long-exposure-photo-1024x576.jpg")
# Creating line
cv2.line(img, (120, 160), (500, 160), (0, 0, 255), 10)

# Creating rectangle
cv2.rectangle(img, (30, 30), (500, 400), (0, 255, 0), 5)

# Creating circle
cv2.circle(img, (200, 200), 80, (255, 0, 0), 3)

# writing text
font = cv2.FONT_HERSHEY_SIMPLEX
cv2.putText(img, 'T.G.SON', (100, 100),
        font, 0.8, (0, 255, 0), 2, cv2.LINE_AA)

cv2.imshow('dark', img)

# Allows us to see image
# until closed forcefully
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Paint and Draw on an image

INPUT

OUTPUT



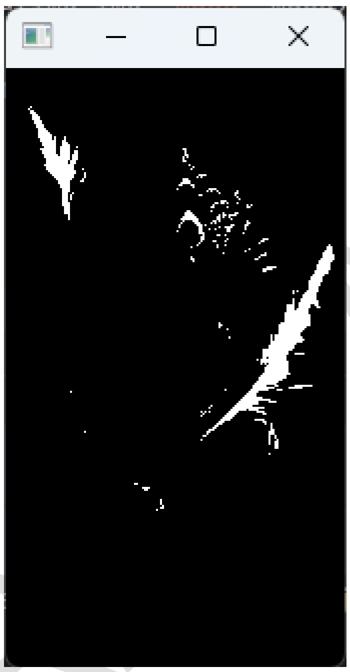
7. IMAGE MASKING:

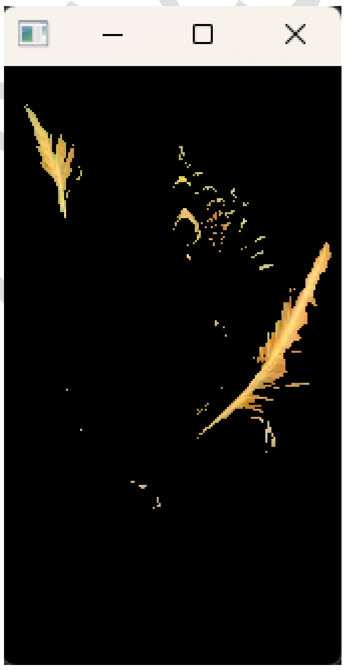
```
# import required libraries
import cv2
import numpy as np

# read input image
img = cv2.imread('C:/Users/21r228\Downloads\download.jpg')

# Convert BGR to HSV
hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)

# define range of blue color in HSV
lower_yellow = np.array([15,50,180])
upper_yellow = np.array([40,255,255])

# Create a mask. Threshold the HSV image to get only yellow colors
mask = cv2.inRange(hsv, lower_yellow, upper_yellow)

# Bitwise-AND mask and original image
result = cv2.bitwise_and(img,img, mask= mask)

# display the mask and masked image
cv2.imshow('Mask',mask)
cv2.waitKey(0)
cv2.imshow('Masked Image',result)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

MASKING IMAGE:

INPUT:

OUTPUT:



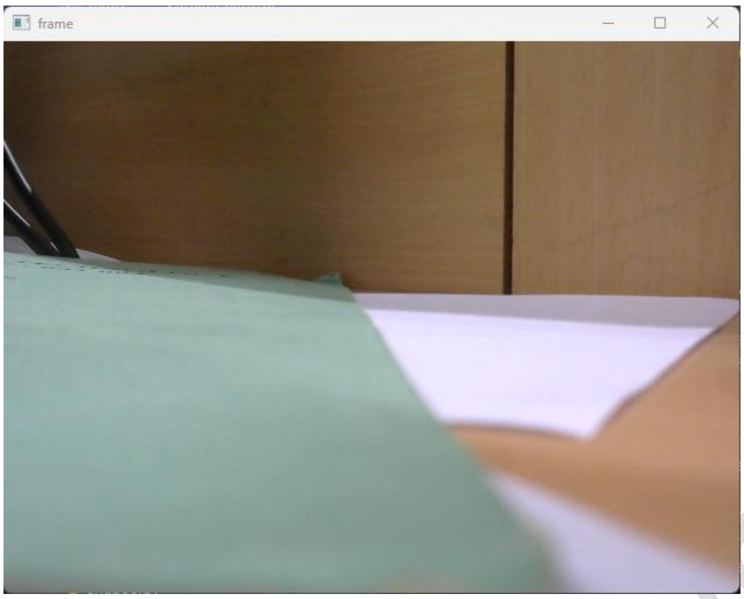


8. MASKING THE IMAGE VIDEO

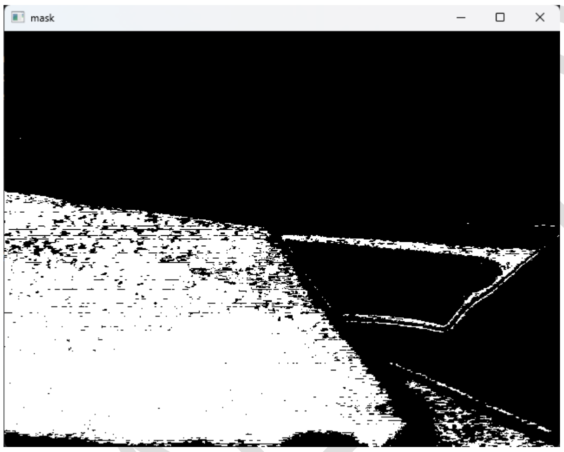
```
import cv2
import numpy as np

cap = cv2.VideoCapture(0)

while (1):
    _, frame = cap.read()
    # It converts the BGR color space of image to HSV color space
    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

    # Threshold of blue in HSV space
    lower_blue = np.array([60, 35, 140])
    upper_blue = np.array([180, 255, 255])

    # preparing the mask to overlay
    mask = cv2.inRange(hsv, lower_blue, upper_blue)

    # The black region in the mask has the value of 0,
    # so when multiplied with original image removes all non-blue regions
    result = cv2.bitwise_and(frame, frame, mask=mask)

    cv2.imshow('frame', frame)
    cv2.imshow('mask', mask)
    cv2.imshow('result', result)

    cv2.waitKey(0)

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

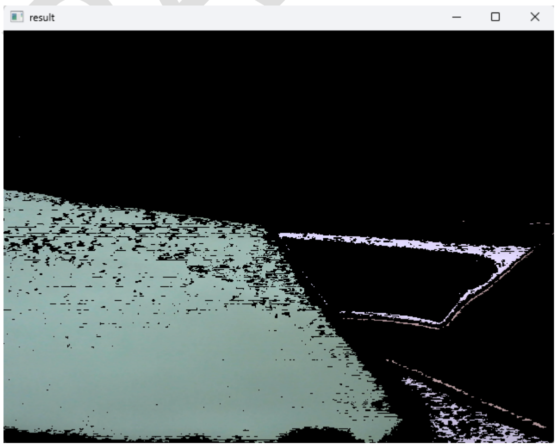
MASKING THE IMAGE VIDEO

INPUT:



OUTPUT:





9. Histogram

```
# importing required libraries of opencv
import cv2

# importing library for plotting
from matplotlib import pyplot as plt

# reads an input image
img = cv2.imread("C:/Users\RAGHUL\Downloads\Green and Black Dynamic Geometric Sport
Instagram Post.png",0)

# find frequency of pixels in range 0-255
histr = cv2.calcHist([img],[0],None,[256],[0,256])

# show the plotting graph of an image
plt.plot(histr)
plt.show()
```

10. Histogram Equalization

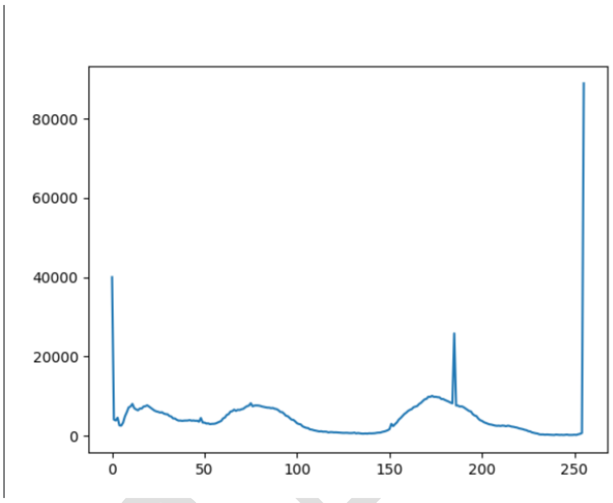
```
# import Opencv
import cv2

# import Numpy
import numpy as np

# read a image using imread
img = cv2.imread(\'F:\\do_nawab.png\', 0)

# creating a Histograms Equalization
# of a image using cv2.equalizeHist()
equ = cv2.equalizeHist(img)

# stacking images side-by-side
res = np.hstack((img, equ))

# show image input vs output
cv2.imshow(\'image\', res)

cv2.waitKey(0)
cv2.destroyAllWindows()
```

Histogram

Input:

PSG COLLEGE OF TECHNOLOGY
DEPARTMENT OF PHYSICAL EDUCATION
FIT INDIA
PHASE - 1
REGISTER NOW
INTRAMURALS
2023 - 2024
Registration Dates : 11th to 13th December 2023
Venue : Physical Education Department
psg_tech_sports

Output:



Histogram Equalization

Input:

PSG COLLEGE OF TECHNOLOGY
DEPARTMENT OF PHYSICAL EDUCATION
FIT INDIA
PHASE - 1
REGISTER NOW
INTRAMURALS
2023 - 2024
Registration Dates : 11th to 13th December 2023
Venue : Physical Education Department
psg_tech_sports

Output:

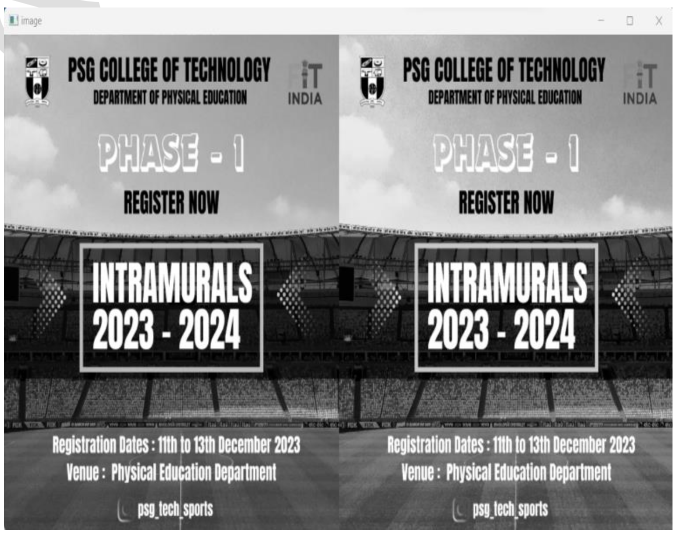


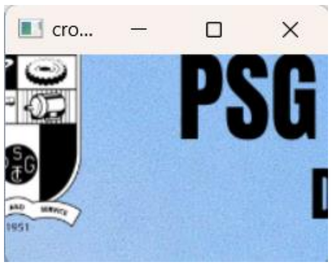
image
PSG COLLEGE OF TECHNOLOGY
DEPARTMENT OF PHYSICAL EDUCATION
INDIA
PHASE - 1
REGISTER NOW
INTRAMURALS
2023 - 2024
Registration Dates : 11th to 13th December 2023
Venue : Physical Education Department
psg_tech_sports

Image Cropping:

Input:

PSG COLLEGE OF TECHNOLOGY
DEPARTMENT OF PHYSICAL EDUCATION
FIT INDIA
PHASE - 1
REGISTER NOW
INTRAMURALS
2023 - 2024
Registration Dates : 11th to 13th December 2023
Venue : Physical Education Department
psg_tech_sports

Output:



11. Image Cropping:

```
import cv2

img = cv2.imread("test.jpeg")
print(type(img))

# Shape of the image
print("Shape of the image", img.shape)

# [rows, columns]
crop = img[50:180, 100:300]

cv2.imshow('original', img)
cv2.imshow('cropped', crop)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Post Lab Questions:

1. What are the applications of masking and cropping in image processing?
2. Write a program to apply a circular mask on an image
3. What are the different functions available in OpenCV and Numpy packages to calculate the histogram and histogram equalization of an image?
4. Write a program to find the coordinates of an image
5. What impact does masking a picture in a histogram have?

Answers :

1. Masking and cropping are common techniques used in image processing for various applications. Here's an overview of their applications:

- Object Segmentation:
  Masking: Masks are used to isolate specific regions or objects within an image. By creating a binary mask, where the desired object is highlighted and the rest is masked out, object segmentation becomes possible. This is crucial in computer vision tasks like object detection and recognition.

  Cropping: Once an object is identified using masking, cropping can be applied to extract and focus on that particular region of interest (ROI). This is useful in applications where the precise location and boundaries of an object need to be determined.

- Image Editing:

Masking: In image editing software, masking is commonly used to selectively apply changes to specific parts of an image. For example, you can apply filters, adjustments, or effects only to the regions defined by the mask, leaving the rest of the image unchanged.

Cropping: Cropping is a fundamental tool for framing and composition. It allows you to remove unwanted elements from an image and focus on the essential components. This is commonly used in photography and graphic design to improve the visual appeal of the image.

- Background Removal:
  Masking: Masks are often employed for background removal, where the foreground object is separated from its background. This is particularly useful in e-commerce, product photography, and creating images with transparent backgrounds.

  Cropping: Cropping can also be used for background removal, especially when the background is simple and the main object is centrally located. However, masking provides more flexibility in handling complex backgrounds.

- Image Annotation and Augmentation:
  Masking: Masks are used for annotating objects in images, such as in computer vision datasets. They provide a precise delineation of object boundaries, aiding in the training of machine learning models.

  Cropping: Cropping can be part of data augmentation strategies during training machine learning models. It helps in exposing the model to different scales and perspectives of the same object, enhancing its ability to generalize.

- Privacy Protection:
  Masking: Masks can be applied to blur or hide specific regions of an image to protect privacy, such as faces or license plates. This is commonly used in applications like street view imagery.

  Cropping: Cropping can be used to exclude sensitive information from an image, but masking is often preferred for more precise control over the areas to be protected.

In summary, masking and cropping are versatile techniques in image processing, providing solutions for object isolation, image editing, background removal, annotation, data augmentation, and privacy protection. Their applications extend across various fields, including computer vision, photography, graphic design, and machine learning.

2.

```
import cv2
import numpy as np

def apply_circular_mask(image_path, output_path, center, radius):
    # Read the input image
    img = cv2.imread(image_path)
    # Create a black mask with the same size as the input image
    mask = np.zeros_like(img)

    # Generate a circular mask
    cv2.circle(mask, center, radius, (255, 255, 255), thickness=-1)

    # Apply the circular mask to the input image
    result = cv2.bitwise_and(img, mask)

    # Save the result to the output path
    cv2.imwrite(output_path, result)

if __name__ == "__main__":
    # Input and output file paths
    input_image_path = "input_image.jpg"
    output_image_path = "output_image_with_circular_mask.jpg"

    # Center and radius of the circular mask
    mask_center = (150, 150)  # Example values; adjust as needed
    mask_radius = 100  # Example value; adjust as needed

    # Apply circular mask
    apply_circular_mask(input_image_path, output_image_path, mask_center, mask_radius)

    print(f"Circular mask applied. Result saved to {output_image_path}")
```

3.

Both OpenCV and NumPy provide functions to calculate histograms and perform histogram equalization on images.

OpenCV Functions:

1. Histogram Calculation:
   - `cv2.calcHist(images, channels, mask, histSize, ranges[, hist[, accumulate]])`: Calculates the histogram of a set of images. Returns a dense or sparse histogram.

Example:

```
import cv2
import matplotlib.pyplot as plt
image = cv2.imread('image.jpg', cv2.IMREAD_GRAYSCALE)
hist = cv2.calcHist([image], [0], None, [256], [0, 256])
plt.plot(hist)
plt.title('Histogram')
plt.show()
```

2. Histogram Equalization:
   cv2.equalizeHist(src[, dst])`: Equalizes the histogram of a grayscale image.

Example:
```
import cv2
image = cv2.imread('image.jpg', cv2.IMREAD_GRAYSCALE)
equalized_image = cv2.equalizeHist(image)
```

NumPy Functions:

1. Histogram Calculation
   - `numpy.histogram(a, bins=10, range=None, normed=False, weights=None, density=None)`: Computes the histogram of a set of data.

Example:
```
import numpy as np
import matplotlib.pyplot as plt
image = cv2.imread('image.jpg', cv2.IMREAD_GRAYSCALE)
hist, bins = np.histogram(image.flatten(), bins=256, range=[0, 256])

plt.plot(hist)
plt.title('Histogram')
plt.show()
```

2. Histogram Equalization:
   - NumPy itself doesn't have a built-in function for histogram equalization. However, you can use OpenCV in conjunction with NumPy:

Example:
```
import cv2
import numpy as np
image = cv2.imread('image.jpg', cv2.IMREAD_GRAYSCALE)
equalized_image = cv2.equalizeHist(image)
```

These functions help in visualizing and enhancing the contrast of an image by redistributing pixel intensities. Adjust the parameters as needed based on your specific use case.

4.

```
import cv2
# Global variables to store coordinates
coordinates = []

def mouse_callback(event, x, y, flags, param):
    if event == cv2.EVENT_LBUTTONDOWN:
        coordinates.append((x, y))
        print(f"Coordinate clicked: ({x}, {y})")

if __name__ == "__main__":
    # Read the image
```

```
image_path = "image.jpg"  # Replace with the path to your image
img = cv2.imread(image_path)

if img is None:
    print(f"Error: Unable to read the image at {image_path}")
else:
    # Create a window and set the mouse callback function
    cv2.namedWindow("Image")
    cv2.setMouseCallback("Image", mouse_callback)

    # Display the image
    while True:
        cv2.imshow("Image", img)
        key = cv2.waitKey(1) & 0xFF

        # Break the loop if the 'esc' key is pressed
        if key == 27:
            break

    # Close the window
    cv2.destroyAllWindows()

    # Print the final coordinates
    print("Final Coordinates:", coordinates)
```

5.

Masking a picture in a histogram involves selecting a specific region or subset of pixels within an image and analyzing the histogram only for that particular region. The impact of masking on a histogram can be significant, and it provides insights into the distribution of pixel intensities within the masked area. Here are some effects and considerations:

- **Isolation of Specific Regions:**
  Impact: The primary purpose of masking in a histogram is to isolate specific regions or objects within an image. This allows you to analyze the pixel intensity distribution of only the selected area.
  Use Case: Useful for focusing on certain objects or areas of interest, especially in complex images where different regions may have distinct characteristics.

- **Enhanced Contrast Analysis:**
  Impact: Masking can help in enhancing contrast analysis by limiting the histogram computation to a specific region. This is beneficial when you want to observe the pixel intensity distribution in a localized part of the image.
  Use Case: Useful for improving visibility and understanding the details within a specific region without being affected by the overall intensity distribution of the entire image.

- **Selective Image Editing:**
  Impact: When editing an image, applying changes selectively to a masked region

based on its histogram can lead to more controlled and targeted adjustments.
Use Case: Allows for localized image enhancement, such as adjusting brightness, contrast, or color balance, in specific areas without affecting the entire image uniformly.

- **Object Detection and Recognition:**
  Impact: Masking in histograms is relevant for computer vision tasks like object detection. It helps in understanding the pixel intensity distribution of objects of interest.
  Use Case: Useful when training machine learning models or developing algorithms that rely on the analysis of specific regions within images.

- **Background Removal:**
  Impact: When removing the background from an image, masking and analyzing the histogram within the masked area can help ensure accurate selection and removal.
  Use Case: Commonly employed in applications where the extraction of foreground objects is required, such as in image segmentation.

In summary, masking in a histogram has a localized impact, providing more focused insights into the pixel intensity distribution of specific regions within an image. This can be valuable for various image processing tasks, allowing for more precise analysis, editing, and understanding of localized features within an image.

Department of RAE			
Criteria	Excellent (75% - 100%)	Good (50 – 75%)	Poor (<50%)
Preparation (30)			
Performance (30)			
Evaluation (20)			
Report (20)			
Sign:		**Total (100)**	

Result:

Thus the basic image processing concepts were learnt using OpenCV in Pycharm IDE.