

import cv2

import numpy as np

def rotate\_image(image, angle):

    # Rotate the image around its center

    center = tuple(np.array(image.shape[1::-1]) / 2)

    rot\_mat = cv2.getRotationMatrix2D(center, angle, 1.0)

    rotated\_image = cv2.warpAffine(image, rot\_mat, image.shape[1::-1], flags=cv2.INTER\_LINEAR)

    return rotated\_image

# Read the main image

img\_rgb = cv2.imread('Industrial-software-example-for-Template-Matching.png')

# Convert it to grayscale

img\_gray = cv2.cvtColor(img\_rgb, cv2.COLOR\_BGR2GRAY)

# Read the template

template = cv2.imread('template\_image.png', 0)

# Define rotation angles

angles = range(0, 360, 10)

# Store match scores for each rotation

match\_scores = []

# Store the coordinates of matched areas with a score above threshold

matched\_points = []

# Iterate over each rotation angle

for angle in angles:

    # Rotate the template

    rotated\_template = rotate\_image(template, angle)

    # Perform match operations.

    res = cv2.matchTemplate(img\_gray, rotated\_template, cv2.TM\_CCOEFF\_NORMED)

    # Store the match scores and the rotated template

    match\_scores.append((res, rotated\_template))

# Get the width and height of the template

w, h = template.shape[::-1]

# Specify a threshold for match score

threshold = 0.8

# Iterate over the match scores and rotated templates

for res, rotated\_template in match\_scores:

    # Get the coordinates of matched areas with a score above threshold

    loc = np.where(res >= threshold)

    # Iterate over each matched point and store it

    for pt in zip(\*loc[::-1]):

        matched\_points.append((pt, pt[0] + w, pt[1] + h))

# Draw rectangles around the matched regions in the main image

for pt1, pt2, pt3 in matched\_points:

    cv2.rectangle(img\_rgb, pt1, (pt2, pt3), (0, 255, 255), 2)

# Show the final image with the matched areas.

cv2.imshow('Detected', img\_rgb)

cv2.waitKey(0)

cv2.destroyAllWindows()

**3. Demonstrate how to implement different template matching algorithms in Python using matrices to find a predefined sub-matrix within a larger matrix**

import cv2

import numpy as np

def template\_matching\_SSD(image, template):

    # Get dimensions of the image and template

    img\_h, img\_w = image.shape

    tpl\_h, tpl\_w = template.shape

    # Calculate the size of the result matrix

    result\_h = img\_h - tpl\_h + 1

    result\_w = img\_w - tpl\_w + 1

    # Initialize the result matrix

    result = np.zeros((result\_h, result\_w))

    # Perform template matching using SSD

    for y in range(result\_h):

        for x in range(result\_w):

            region = image[y:y+tpl\_h, x:x+tpl\_w]

            diff = np.sum((region - template) \*\* 2)

            result[y, x] = diff

    return result

def template\_matching\_NCC(image, template):

    # Get dimensions of the image and template

    img\_h, img\_w = image.shape

    tpl\_h, tpl\_w = template.shape

    # Calculate the size of the result matrix

    result\_h = img\_h - tpl\_h + 1

    result\_w = img\_w - tpl\_w + 1

    # Initialize the result matrix

    result = np.zeros((result\_h, result\_w))

    # Normalize the template and image

    norm\_tpl = (template - np.mean(template)) / np.std(template)

    norm\_img = (image - np.mean(image)) / np.std(image)

    # Perform template matching using NCC

    for y in range(result\_h):

        for x in range(result\_w):

            region = norm\_img[y:y+tpl\_h, x:x+tpl\_w]

            result[y, x] = np.sum(region \* norm\_tpl)

    return result

image = cv2.imread('main\_image.png', cv2.IMREAD\_GRAYSCALE)

template = cv2.imread('template\_image.jpg', cv2.IMREAD\_GRAYSCALE)

# Perform template matching using SSD

result\_SSD = template\_matching\_SSD(image, template)

# Perform template matching using NCC

result\_NCC = template\_matching\_NCC(image, template)

# Display the results

cv2.imshow('Result SSD', result\_SSD)

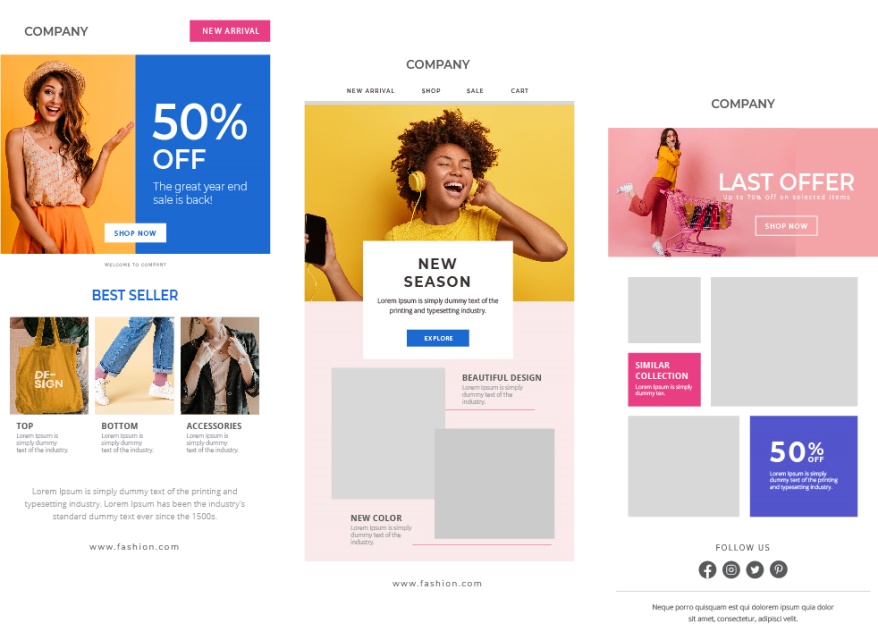
cv2.imshow('Result NCC', result\_NCC)

cv2.waitKey(0)

cv2.destroyAllWindows()

4. Implement template matching by using various algorithms to find a specific pattern in an image

Input Image:



Templates

# Python program to illustrate

# template matching

import cv2

import numpy as np

# Read the main image

img\_rgb = cv2.imread('main\_image.jpg')

# Convert it to grayscale

img\_gray = cv2.cvtColor(img\_rgb, cv2.COLOR\_BGR2GRAY)

# Read the template

template1= cv2.imread('template\_image\_1.png', 0)

template2 = cv2.imread('template\_image\_2.png', 0)

template3 = cv2.imread('template\_image\_3.png', 0)

# Store width and height of template in w and h

w, h = template1.shape[::-1]

# Perform match operations.

res1 = cv2.matchTemplate(img\_gray, template1, cv2.TM\_CCOEFF\_NORMED)

res2= cv2.matchTemplate(img\_gray, template2, cv2.TM\_CCOEFF\_NORMED)

res3 = cv2.matchTemplate(img\_gray, template3, cv2.TM\_CCOEFF\_NORMED)

# Specify a threshold

threshold = 0.8

# Store the coordinates of matched area in a numpy array

loc = np.where(res1 >= 0.6)

# Draw a rectangle around the matched region.

for pt in zip(\*loc[::-1]):

    cv2.rectangle(img\_rgb, pt, (pt[0] + w, pt[1] + h), (0, 255, 255), 2)

    # Store the coordinates of matched area in a numpy array

loc = np.where(res2 >= threshold)

# Draw a rectangle around the matched region.

for pt in zip(\*loc[::-1]):

    cv2.rectangle(img\_rgb, pt, (pt[0] + w, pt[1] + h), (255, 0, 255), 2)

    # Store the coordinates of matched area in a numpy array

loc = np.where(res3 >= threshold)

# Draw a rectangle around the matched region.

for pt in zip(\*loc[::-1]):

    cv2.rectangle(img\_rgb, pt, (pt[0] + w, pt[1] + h), (0, 0, 255), 2)

# Show the final image with the matched area.

cv2.imshow('Detected', img\_rgb)

cv2.waitKey(0)

Use Different Images incase of no output, probably because of the resolution breaking

1. Implement an edge detection algorithm in python that combines the canny edge detector with a custom algorithm for handling noisy images. Show all the 5 stages in canny edge detector for an image.

import cv2

import numpy as np

# Function to implement custom noise reduction algorithm

def custom\_noise\_reduction(image):

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

    return blurred

# Function to implement Canny edge detection

def custom\_canny\_edge\_detection(image):

    # Apply custom noise reduction algorithm

    image = custom\_noise\_reduction(image)

    # Convert image to grayscale

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

    # Apply Sobel operator to find gradients

    grad\_x = cv2.Sobel(gray, cv2.CV\_64F, 1, 0, ksize=3)

    grad\_y = cv2.Sobel(gray, cv2.CV\_64F, 0, 1, ksize=3)

    # Calculate gradient magnitude and direction

    mag = np.sqrt(grad\_x\*\*2 + grad\_y\*\*2)

    mag = np.uint8(mag)

    theta = np.arctan2(grad\_y, grad\_x)

    # Quantize gradient direction to four angles

    theta\_quantized = np.round(theta / (np.pi/4)) % 4

    # Apply non-maximum suppression

    suppressed = np.zeros\_like(mag)

    for i in range(1, mag.shape[0]-1):

        for j in range(1, mag.shape[1]-1):

            direction = theta\_quantized[i, j]

            if direction == 0:

                neighbors = [mag[i, j-1], mag[i, j], mag[i, j+1]]

            elif direction == 1:

                neighbors = [mag[i-1, j+1], mag[i, j], mag[i+1, j-1]]

            elif direction == 2:

                neighbors = [mag[i-1, j], mag[i, j], mag[i+1, j]]

            else:

                neighbors = [mag[i-1, j-1], mag[i, j], mag[i+1, j+1]]

            if mag[i, j] == max(neighbors):

                suppressed[i, j] = mag[i, j]

    # Apply double thresholding and edge tracking

    high\_threshold\_ratio = 0.3

    low\_threshold\_ratio = 0.1

    high\_threshold = np.max(suppressed) \* high\_threshold\_ratio

    low\_threshold = high\_threshold \* low\_threshold\_ratio

    strong\_edges = (suppressed >= high\_threshold)

    weak\_edges = (suppressed >= low\_threshold) & (suppressed < high\_threshold)

    # Perform edge tracking

    for i in range(1, suppressed.shape[0]-1):

        for j in range(1, suppressed.shape[1]-1):

            if weak\_edges[i, j]:

                if np.any(strong\_edges[i-1:i+2, j-1:j+2]):

                    strong\_edges[i, j] = True

    return strong\_edges.astype(np.uint8) \* 255

# Load the input image

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

# Perform Canny edge detection

edges = custom\_canny\_edge\_detection(image)

# Show the 5 stages in Canny edge detection

cv2.imshow('Original Image', image)

cv2.imshow('Custom Noise Reduction', custom\_noise\_reduction(image))

# Convert grayscale image to 3D BGR

gray\_bgr = cv2.cvtColor(cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY), cv2.COLOR\_GRAY2BGR)

# Stack the grayscale and edge images horizontally

cv2.imshow('Gradients and Directions', np.hstack((gray\_bgr, cv2.cvtColor(edges, cv2.COLOR\_GRAY2BGR))))

cv2.imshow('Non-Maximum Suppression', edges)

cv2.imshow('Edge Tracking', edges)

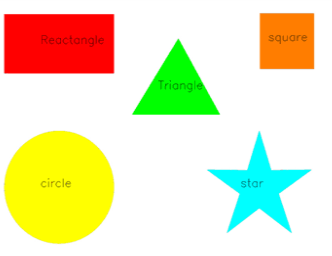
cv2.waitKey(0)

cv2.destroyAllWindows()

7. A.Write a function in Python that accepts a decimal number and returns the equivalent binary number. To make this simple, the decimal number will always be less than 1,024, so the binary number returned will always be less than ten digits long.

b. Find the shapes in the given input image and show the output given.

INPUT OUTPUT



import cv2

import numpy as np

import random

#7A

a=int(input("Enter the Decimal number : "))

print(" "+bin(a).replace("0b"," "))

# Load the image

image = cv2.imread('shapes.png')

original\_image = image.copy()

# Convert the image to grayscale

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

# Threshold the image

\_, thresh = cv2.threshold(gray, 240, 255, cv2.THRESH\_BINARY\_INV)

# Find contours in the thresholded image

contours, \_ = cv2.findContours(thresh, cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)

# Dictionary to map number of sides to shape names

shape\_names = {

    3: 'Triangle',

    4: 'Rectangle',

    5: 'Pentagon',

    6: 'Hexagon',

    10:'star'

    # Add more shape names as needed

}

# Iterate through contours

for contour in contours:

    # Approximate the contour to reduce the number of points

    approx = cv2.approxPolyDP(contour, 0.04 \* cv2.arcLength(contour, True), True)

    # Get the number of sides

    sides = len(approx)

    # Get the shape name from the dictionary

    shape\_name = shape\_names.get(sides, 'Circle')

    # Get a random color for the mask

    color = (random.randint(0, 255), random.randint(0, 255), random.randint(0, 255))

    # Draw filled contour with random color

    cv2.drawContours(image, [contour], -1, color, -1)

    # Get the center of the contour

    M = cv2.moments(contour)

    if M["m00"] != 0:

        cX = int(M["m10"] / M["m00"])

        cY = int(M["m01"] / M["m00"])

    else:

        cX, cY = 0, 0

    # Draw shape name

    cv2.putText(image, shape\_name, (cX - 20, cY), cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (255, 255, 255), 2)

# Display the image

cv2.imshow('Identified Shapes', image)

cv2.waitKey(0)

cv2.destroyAllWindows()

8. Convolution Perfoming for the given image and given kernel

import numpy as np

# Function to perform convolution operation

def convolution(image, kernel):

    image\_height, image\_width = image.shape

    kernel\_height, kernel\_width = kernel.shape

    output = np.zeros((image\_height, image\_width))

    # Pad the image to handle border pixels

    padded\_image = np.pad(image, ((1, 1), (1, 1)), mode='constant')

    # Apply convolution operation

    for i in range(1, image\_height + 1):

        for j in range(1, image\_width + 1):

            # Extract the region of interest from the padded image

            region = padded\_image[i - 1:i + 2, j - 1:j + 2]

            # Apply kernel

            output[i - 1, j - 1] = np.sum(region \* kernel)

    return output.astype(np.uint8)

# Input image and kernel

image = np.array([[0, 0, 0, 1, 0],

                  [0, 1, 1, 1, 2],

                  [1, 1, 1, 1, 1],

                  [2, 1, 1, 2, 1],

                  [2, 2, 1, 0, 0]])

kernel = np.array([[1, 0, 1],

                   [0, 1, 0],

                   [1, 0, 1]])

# Perform convolution operation

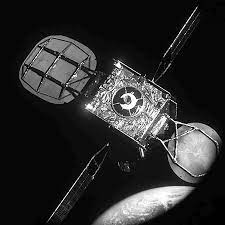
output = convolution(image, kernel)

# Print the output

print("Result of Convolution:")

print(output)

1. a. Remove the glare from the image and display the enhanced image.



b. Apply a box filter without using in-built function on any image and show the difference on applying a median filter on the same.

import cv2

# Load the image

image = cv2.imread('image\_with\_glare.jpg',0)

# Apply histogram equalization

enhanced\_image = cv2.equalizeHist(image)

# Display the enhanced image

cv2.imshow('Enhanced Image', enhanced\_image)

cv2.waitKey(0)

cv2.destroyAllWindows()

B

import cv2

import numpy as np

# Function to apply a box filter manually

def box\_filter(image, kernel\_size):

    kernel = np.ones((kernel\_size, kernel\_size), dtype=np.float32) / (kernel\_size \* kernel\_size)

    output = np.zeros\_like(image, dtype=np.float32)

    rows, cols = image.shape

    for i in range(rows - kernel\_size + 1):

        for j in range(cols - kernel\_size + 1):

            output[i:i+kernel\_size, j:j+kernel\_size] += image[i:i+kernel\_size, j:j+kernel\_size] \* kernel

    return output.astype(np.uint8)

# Load the image

image = cv2.imread('main\_image.jpg', cv2.IMREAD\_GRAYSCALE)

# Apply box filter

filtered\_image = box\_filter(image, kernel\_size=5)

# Apply median filter

median\_filtered\_image = cv2.medianBlur(image, 5)

# Display the original, box-filtered, and median-filtered images

cv2.imshow('Original Image', image)

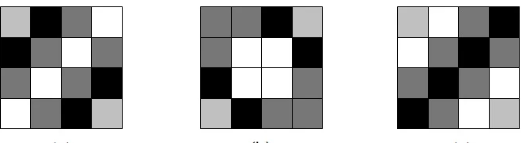
cv2.imshow('Box Filtered Image', filtered\_image)

cv2.imshow('Median Filtered Image', median\_filtered\_image)

cv2.waitKey(0)

cv2.destroyAllWindows()

1. Replicate the image using shapes (use rectangle function to create the below image) and perform histogram equalization



import cv2

import numpy as np

# Load the original image

original\_image = cv2.imread('main\_image.png')

# Create a blank canvas with the same dimensions as the original image

replicated\_image = np.zeros\_like(original\_image)

# Get the dimensions of the original image

height, width, \_ = original\_image.shape

# Define the number of rows and columns for the grid

rows = 10

columns = 10

# Calculate the width and height of each rectangle

rect\_width = width // columns

rect\_height = height // rows

# Draw rectangles on the blank canvas to replicate the original image

for i in range(rows):

    for j in range(columns):

        x1 = j \* rect\_width

        y1 = i \* rect\_height

        x2 = (j + 1) \* rect\_width

        y2 = (i + 1) \* rect\_height

        color = original\_image[y1:y2, x1:x2].mean(axis=(0, 1)).astype(int)  # Get average color within the rectangle

        color = tuple(map(int, color))  # Convert color to tuple of integers

        cv2.rectangle(replicated\_image, (x1, y1), (x2, y2), color, -1)  # Draw filled rectangle

# Perform histogram equalization on the replicated image

replicated\_image\_equalized = cv2.cvtColor(replicated\_image, cv2.COLOR\_BGR2YUV)

replicated\_image\_equalized[:,:,0] = cv2.equalizeHist(replicated\_image\_equalized[:,:,0])

replicated\_image\_equalized = cv2.cvtColor(replicated\_image\_equalized, cv2.COLOR\_YUV2BGR)

# Display the original and replicated images

cv2.imshow('Original Image', original\_image)

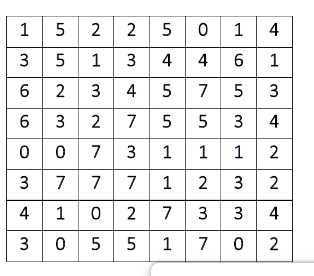
cv2.imshow('Replicated Image', replicated\_image)

cv2.imshow('Replicated Image with Histogram Equalization', replicated\_image\_equalized)

cv2.waitKey(0)

cv2.destroyAllWindows()

1. View the below pixel values in histogram and perform histogram equalization. Display the new pixel vales



import cv2

import numpy as np

import matplotlib.pyplot as plt

# Load the image

image = cv2.imread('main\_image.png', cv2.IMREAD\_GRAYSCALE)

# Calculate histogram

hist, bins = np.histogram(image.flatten(), 256, [0,256])

# Cumulative distribution function (CDF)

cdf = hist.cumsum()

cdf\_normalized = cdf \* hist.max() / cdf.max()

# Plot the original histogram

plt.figure(figsize=(8, 6))

plt.subplot(1, 2, 1)

plt.hist(image.flatten(), 256, [0,256], color='r')

plt.xlim([0,256])

plt.legend(('Original Histogram',), loc='upper right')

# Perform histogram equalization

cdf\_m = np.ma.masked\_equal(cdf, 0)

cdf\_m = (cdf\_m - cdf\_m.min()) \* 255 / (cdf\_m.max() - cdf\_m.min())

cdf = np.ma.filled(cdf\_m, 0).astype('uint8')

# Apply equalization

equalized\_image = cdf[image]

# Plot the equalized histogram

plt.subplot(1, 2, 2)

plt.hist(equalized\_image.flatten(), 256, [0,256], color='b')

plt.xlim([0,256])

plt.legend(('Equalized Histogram',), loc='upper right')

plt.show()

# Display the new pixel values

cv2.imshow('Original Image', image)

cv2.imshow('Equalized Image', equalized\_image)

cv2.waitKey(0)

cv2.destroyAllWindows()

1. Construct a 5\*5 matrix with elements starting with ASCII values of your name followed by ASCII values of your register number. The first letter of each word and initials should be in capital. Space should be treated as an element. The balance elements should be filled with the pattern, where x is the last digit of your register number. Perform thresholding for the above matrix.

import cv2

import numpy as np

# Function to get ASCII values of a string

def ascii\_values(string):

    return [ord(char) for char in string]

# Replace 'YourName' and 'YourRegisterNumber' with your actual name and register number

name = "Vinothgirubha K"  # Example name

register\_number = "21R253"  # Example register number

# Get ASCII values of name and register number

name\_ascii = ascii\_values(name)

register\_ascii = ascii\_values(register\_number)

# Construct the 5x5 matrix

matrix = np.zeros((5, 5), dtype=np.uint8)

# Fill the matrix according to the specified criteria

for i in range(5):

    for j in range(5):

        if i == 0 and j == 0:

            matrix[i, j] = name\_ascii[0]

        elif i == 1 and j == 0:

            matrix[i, j] = name\_ascii[0]

        elif i == 0 and j == 2:

            matrix[i, j] = name\_ascii[1]

        elif i == 1 and j == 2:

            matrix[i, j] = name\_ascii[1]

        elif i == 0 and j == 4:

            matrix[i, j] = name\_ascii[2]

        elif i == 1 and j == 4:

            matrix[i, j] = name\_ascii[2]

        elif i == 2 and j == 0:

            matrix[i, j] = register\_ascii[0]

        elif i == 3 and j == 0:

            matrix[i, j] = register\_ascii[0]

        elif i == 2 and j == 2:

            matrix[i, j] = register\_ascii[1]

        elif i == 3 and j == 2:

            matrix[i, j] = register\_ascii[1]

        elif i == 2 and j == 4:

            matrix[i, j] = register\_ascii[2]

        elif i == 3 and j == 4:

            matrix[i, j] = register\_ascii[2]

        elif i == 4 and j == 0:

            matrix[i, j] = register\_ascii[3]

        elif i == 4 and j == 2:

            matrix[i, j] = register\_ascii[4]

        else:

            last\_digit = int(register\_number[-1])  # Get the last digit of the register number

            matrix[i, j] = 2 \* last\_digit

# Perform thresholding on the matrix

\_, thresholded\_matrix = cv2.threshold(matrix, 127, 255, cv2.THRESH\_BINARY)

# Display the original and thresholded matrices

print("Original Matrix:")

print(matrix)

print("\nThresholded Matrix:")

print(thresholded\_matrix)

# Uncomment below lines if you want to display the matrices using OpenCV

# cv2.imshow('Original Matrix', matrix)

# cv2.imshow('Thresholded Matrix', thresholded\_matrix)

# cv2.waitKey(0)

# cv2.destroyAllWindows()

6.Pixel values:

180 160 160 140 120

110 110 120 140 120

110 140 120 120 140

120 160 160 170 170

170 120 110 140 110

For the above pixel values perform Image intensity transformations (3 members)

import cv2

import numpy as np

# Provided pixel values

pixel\_values = np.array([[180, 160, 160, 140, 120],

                         [110, 110, 120, 140, 120],

                         [110, 140, 120, 120, 140],

                         [120, 160, 160, 170, 170],

                         [170, 120, 110, 140, 110]])

# Convert pixel values to uint8 (required for OpenCV operations)

pixel\_values\_uint8 = pixel\_values.astype(np.uint8)

# Method 1: Brightness adjustment

brightness\_factor = 50

brightness\_transformed = cv2.add(pixel\_values\_uint8, brightness\_factor)

# Method 2: Contrast adjustment

contrast\_factor = 1.5

contrast\_transformed = cv2.multiply(pixel\_values\_uint8, contrast\_factor)

# Method 3: Gamma correction

gamma = 1.5

gamma\_transformed = np.uint8(np.power(pixel\_values\_uint8 / 255.0, gamma) \* 255)

# Display original and transformed pixel values

print("Original Pixel Values:")

print(pixel\_values)

print("\nBrightness Adjusted Pixel Values:")

print(brightness\_transformed)

print("\nContrast Adjusted Pixel Values:")

print(contrast\_transformed)

print("\nGamma Corrected Pixel Values:")

print(gamma\_transformed)

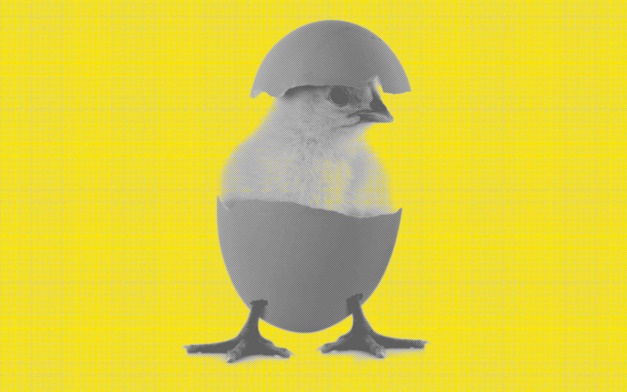
17 .a. Demonstrate the masking concepts try to mask the boy and the bowl in the given image

b. Mask the green color in the given image and change to yellow color with the text of your choice

. 

c. Crop the chick in the image and change the cropped part into red color with your own text



C

import cv2

import numpy as np

# Load the image

image = cv2.imread(r'D:\Sample images\main.png')

# Convert the image from BGR to HSV color space

hsv\_image = cv2.cvtColor(image, cv2.COLOR\_BGR2HSV)

# Define lower and upper bounds of the green color in HSV

lower\_green = np.array([40, 40, 40])

upper\_green = np.array([80, 255, 255])

# Create a mask to isolate the green areas in the image

mask = cv2.inRange(hsv\_image, lower\_green, upper\_green)

# Replace green areas with yellow color

image[mask > 0] = [0, 255, 255]  # BGR value for yellow color

# Add text to the modified image

text = "Your text here"

cv2.putText(image, text, (50, 50), cv2.FONT\_HERSHEY\_SIMPLEX, 1, (0, 0, 0), 2)

# Display the modified image

cv2.imshow('Modified Image', image)

cv2.waitKey(0)

cv2.destroyAllWindows()

B

import cv2

import numpy as np

# Load the image

image = cv2.imread(r"C:\Users\LENOVO\OneDrive\Desktop\chick.jpg")

# Define coordinates of the bounding box around the chick

x, y, w, h = 200, 100, 300, 400  # Example coordinates (x, y, width, height)

# Crop the chick using the defined coordinates

cropped\_chick = image[y:y+h, x:x+w]

# Create a red background image of the same size as the cropped chick

red\_background = np.ones\_like(cropped\_chick, dtype=np.uint8) \* (0, 0, 255)  # Red color in BGR format

# Add text to the red background image

text = "hi"

text\_size = cv2.getTextSize(text, cv2.FONT\_HERSHEY\_SIMPLEX, 1, 2)[0]

text\_x = (red\_background.shape[1] - text\_size[0]) // 2

text\_y = (red\_background.shape[0] + text\_size[1]) // 2

cv2.putText(red\_background, text, (text\_x, text\_y), cv2.FONT\_HERSHEY\_SIMPLEX, 1, (255, 255, 255), 2)

# Convert the cropped chick to float32 for blending

cropped\_chick\_float32 = cropped\_chick.astype(np.float32) / 255.0

# Convert the red background to float32 for blending

red\_background\_float32 = red\_background.astype(np.float32) / 255.0

# Blend the cropped chick with the red background

blended\_image\_float32 = cv2.addWeighted(cropped\_chick\_float32, 1.0, red\_background\_float32, 0.5, 0)

# Convert the blended image back to uint8

blended\_image = (blended\_image\_float32 \* 255).astype(np.uint8)

# Display the modified image

cv2.imshow('Modified Image', blended\_image)

cv2.waitKey(0)

cv2.destroyAllWindows()

A

import cv2

import numpy as np

# Load the image

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

# Convert image to grayscale

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

# Threshold to create a binary mask

\_, mask = cv2.threshold(gray, 1, 200, cv2.THRESH\_BINARY)

# Invert the binary mask

mask\_inv = cv2.bitwise\_not(mask)

# Create a green mask

green\_mask = np.zeros\_like(image, np.uint8)

green\_mask[:] = (0, 255, 0)  # Green color

# Apply mask to the original image

masked\_image = cv2.bitwise\_and(image, image, mask=mask\_inv)

# Add green mask to the masked region

final\_image = cv2.add(masked\_image, green\_mask)

# Display the result

cv2.imshow('Result', final\_image)

cv2.waitKey(0)

cv2.destroyAllWindows()