

**NATIONAL UNIVERSITY OF SCIENCES AND TECHNOLOGY**

**School of Electrical Engineering and Computer Sciences**

**Department of Computing**

**Digital Image Processing**

**Submitted to: Sir Asad Khan**

**Submitted By:**

**Qudsia Tariq (368134)**

**Mariam Babar Khan (375059)**

**Muhammad Uneeb Bin Aftab (370175)**

**Class / Section: BSCS-11 / B**

**Date of Submission: 27th September, 2023**

**GitHub Repository:** [**https://github.com/mariambabarkhan/Digital-Image-Processing-1**](https://github.com/mariambabarkhan/Digital-Image-Processing-1)

**TASK:1**

**Image Resizing**

**Implementation Methodology:**

We have used OpenCV’s resize() function to resize the image, with the new dimensions as 256x256. The interpolation parameter specifies the method used for interpolation during resizing, in this case, cubic interpolation. The resized image is stored in imgResize, after which the resized as well as original image is displayed using imshow() function.

**Code:**

import cv2

img = cv2.imread('walle.jpg', 1)

imgResize = cv2.resize(img,(256, 256), interpolation = cv2.INTER\_CUBIC)

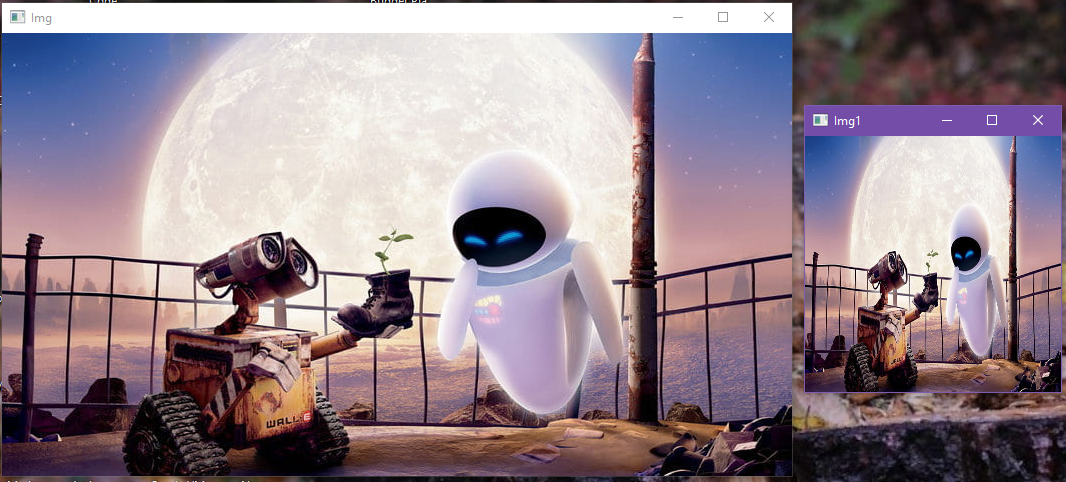
cv2.imshow('Img', img)

cv2.imshow('Img1', imgResize)

cv2.waitKey(0)

cv2.destroyAllWindows()

**Output:**



Resized Image:

**TASK: 2**

**RGB to Grayscale Conversion**

**Implementation Methodology:**

We use OpenCV's cvtColor() function to convert an RGB image to grayscale. This function takes two arguments: the input image and the conversion code (COLOR\_BGR2GRAY). The conversion code specifies the color space transformation to be applied, in this case, from BGR to grayscale.

**Code:**

import cv2

# Load an RGB image

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

# Convert the RGB image to grayscale

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

# Display or save the grayscale image

cv2.imshow('Grayscale Image', gray\_image)

cv2.waitKey(0)

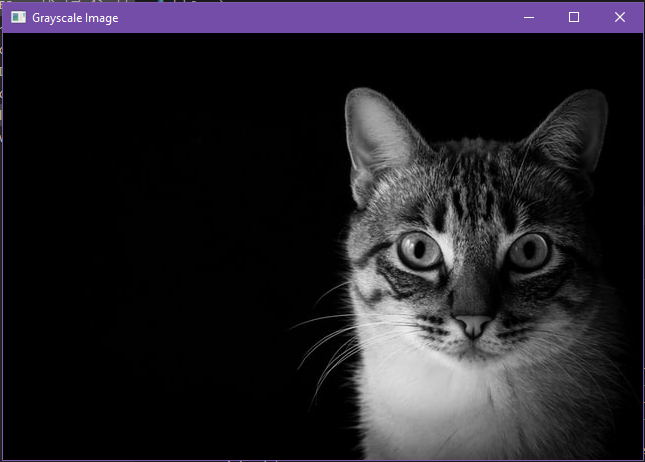
cv2.destroyAllWindows()

**Output:**

Original Image:



Gray Scaled:



**TASK: 3(a)**

**RGB to Grayscale Conversion**

**Implementation Methodology:**

This is the same method used in task 2.

**TASK: 3(b)**

**RGB to Binary Conversion**

**Implementation Methodology:**

In this code, the grayscale image is first loaded. Then, a threshold value of 128 is set. The threshold() function is applied to the grayscale image, which converts it into a binary image. Pixels with intensity values greater than 128 become white, while those below become black.

**Code:**

import cv2

# Load a grayscale image

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

# Set a threshold value (you can adjust this value as needed)

threshold\_value = 128

# Apply binary thresholding

\_, binary\_image = cv2.threshold(gray\_image, threshold\_value, 255, cv2.THRESH\_BINARY)

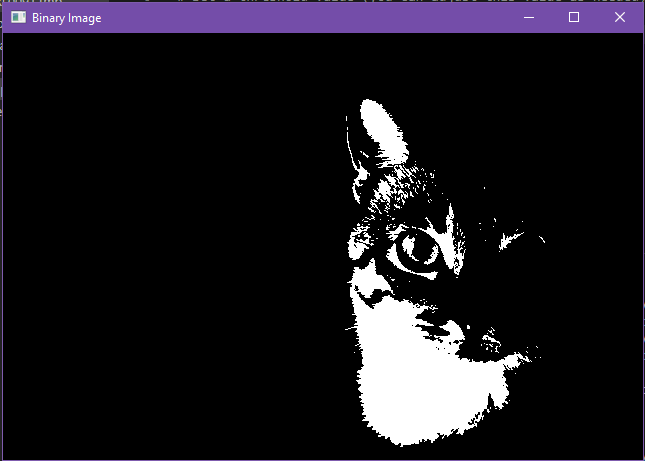
# Display or save the binary image

cv2.imshow('Binary Image', binary\_image)

cv2.waitKey(0)

cv2.destroyAllWindows()

**Output:**



**TASK: 4**

**Coin Count and Segmenting**

**Implementation Methodology:**

We start with image enhancement by converting the image from RGB to grayscale to binary, as it simplifies the image and makes it easier to identify objects. In this specific context, it helps isolate the coins from the background, which will aid in their detection.

Once we have the binary image, we use contour detection to find the boundaries of the coins. By analyzing these contours, we can estimate the number of coins present in the image. Finally, the loop over the contours serves the purpose of filtering out contours that are too large or too small to be considered as coins.

Final contours are displayed along with printing total number of estimated coins.

**Code:**

import cv2

image = cv2.imread("Capture.png")

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

threshold = 120

maxValue = 255

th, dst = cv2.threshold(gray, threshold, maxValue, cv2.THRESH\_BINARY)

cv2.imshow("Thresholded Image", dst)

cv2.waitKey(0)

#contours around black objects

contours, hierarchy = cv2.findContours(dst, cv2.RETR\_TREE, cv2.CHAIN\_APPROX\_SIMPLE)

cv2.drawContours(dst, contours, -1, (0, 0 , 255), 20)

cv2.imshow("Contours", dst)

cv2.waitKey(0)

count = 0

for contour in contours:

    area = cv2.contourArea(contour)

    if area > 100 and area < 100000:

        count += 1

print("Number of coins: ", count)

**Output:**

