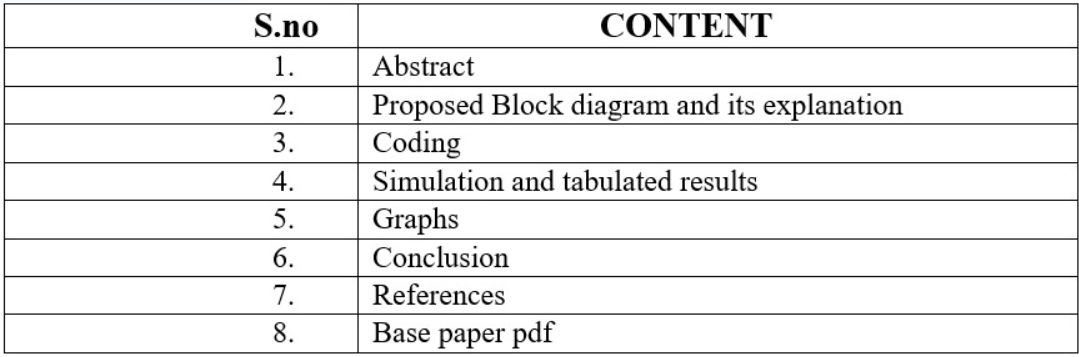


**Image Segmentation Using OpenCV**

**Buddepu Shiva -21BCE5692**

****

**Abstract:**

This study delves into advanced image segmentation methodologies employing OpenCV and Grab Cut algorithms. The first part utilizes OpenCV for edge segmentation, commencing with preprocessing techniques to enhance image quality, followed by Canny edge detection and contour approximation for receipt identification. Subsequently, perspective transformation is applied to rectify the image, and local thresholding is employed for binary image conversion. In contrast, the second segment explores Grab Cut algorithm-based region segmentation. Initially, a mask is created, and background/foreground models are defined. The Grab Cut algorithm is applied with initial segmentation using a predefined rectangle. The resulting mask is then refined to include probable background and foreground, and finally applied to the original image for segmentation. Through comparative analysis, the efficacy and applicability of these techniques are evaluated, offering insights into their respective strengths and limitations for diverse image segmentation tasks.

**Proposed Block diagram and its explanation:**

Image segmentation

Edge based segmentation

Region based segmentation

**Edge-based segmentation**: Edge-based segmentation involves identifying abrupt changes in pixel intensity or colour within an image to delineate object boundaries. Techniques like the Sobel operator, Canny edge detector, and Prewitt operator are commonly used to detect these edges. Once detected, edges can be further processed to form closed contours, defining the boundaries of objects. This approach is effective when objects have well-defined edges but may struggle with noisy or cluttered backgrounds.

**Region based Segmentation:** Region-based segmentation groups pixels with similar properties, such as intensity or colour, to form coherent regions or segments. Techniques like thresholding, clustering, and region growing are employed for this purpose. Region-based segmentation is robust to noise and clutter but may face challenges in distinguishing between regions with similar properties.

**Segmentation process:**

Input image

preprocessing

Segmentation algorithm

Post processing

Segmented image

**Input Image:** Represents the initial image that is input into the segmentation process.

**Preprocessing:** Includes noise reduction, smoothing, and colour space conversion steps to prepare the image for segmentation.

**Segmentation Algorithms:** Involves basic segmentation algorithms like thresholding and contour detection. In Edge based segmentation we used Canny operator and in region based we used Grab Cut algorithm

**Post-processing:** Includes refining segmentation results and object labelling.

**Output Image:** The final segmented image with distinct regions or objects delineated.

**Coding:**

**Edge Segmentation:**

import numpy as np

import cv2

import matplotlib.pyplot as plt

from PIL import Image

# Sample file out of the dataset

file\_name = '/content/pic7.jpg'

img = Image.open(file\_name)

img.thumbnail((800,800), Image.ANTIALIAS)

img

def opencv\_resize(image, ratio):

    width = int(image.shape[1] \* ratio)

    height = int(image.shape[0] \* ratio)

    dim = (width, height)

    return cv2.resize(image, dim, interpolation = cv2.INTER\_AREA)

# Downscale image as finding receipt contour is more efficient on a small image

image = cv2.imread('/content/pic7.jpg')

resize\_ratio = 500 / image.shape[0]

original = image.copy()

image = opencv\_resize(image, resize\_ratio)

age):

    plt.figure(figsize=(16,10))

    return plt.imshow(image, cmap='Greys\_r')

# Convert to grayscale for further processing

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

plot\_gray(gray)

# Get rid of noise with Gaussian Blur filter

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

plot\_gray(blurred)

# Detect white regions

rectKernel = cv2.getStructuringElement(cv2.MORPH\_RECT, (9, 9))

dilated = cv2.dilate(blurred, rectKernel)

plot\_gray(dilated)

edged = cv2.Canny(dilated, 100, 200, apertureSize=3)

plot\_gray(edged)

def plot\_rgb(image):

    plt.figure(figsize=(16,10))

    return plt.imshow(cv2.cvtColor(image, cv2.COLOR\_BGR2RGB))

# Detect all contours in Canny-edged image

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

image\_with\_contours = cv2.drawContours(image.copy(), contours, -1, (0,255,0), 3)

plot\_rgb(image\_with\_contours)

# Get 10 largest contours

largest\_contours = sorted(contours, key = cv2.contourArea, reverse = True)[:10]

image\_with\_largest\_contours = cv2.drawContours(image.copy(), largest\_contours, -1, (0,255,0), 3)

plot\_rgb(image\_with\_largest\_contours)

# approximate the contour by a more primitive polygon shape

def approximate\_contour(contour):

    peri = cv2.arcLength(contour, True)

    return cv2.approxPolyDP(contour, 0.032 \* peri, True)

def get\_receipt\_contour(contours):

    # loop over the contours

    for c in contours:

        approx = approximate\_contour(c)

        # if our approximated contour has four points, we can assume it is receipt's rectangle

        if len(approx) == 4:

            return approx

receipt\_contour = get\_receipt\_contour(largest\_contours)

image\_with\_receipt\_contour = cv2.drawContours(image.copy(), [receipt\_contour], -1, (0, 255, 0), 2)

plot\_rgb(image\_with\_receipt\_contour)

def contour\_to\_rect(contour):

    pts = contour.reshape(4, 2)

    rect = np.zeros((4, 2), dtype = "float32")

    # top-left point has the smallest sum

    # bottom-right has the largest sum

    s = pts.sum(axis = 1)

    rect[0] = pts[np.argmin(s)]

    rect[2] = pts[np.argmax(s)]

    # compute the difference between the points:

    # the top-right will have the minumum difference

    # the bottom-left will have the maximum difference

    diff = np.diff(pts, axis = 1)

    rect[1] = pts[np.argmin(diff)]

    rect[3] = pts[np.argmax(diff)]

    return rect / resize\_ratio

def wrap\_perspective(img, rect):

    # unpack rectangle points: top left, top right, bottom right, bottom left

    (tl, tr, br, bl) = rect

    # compute the width of the new image

    widthA = np.sqrt(((br[0] - bl[0]) \*\* 2) + ((br[1] - bl[1]) \*\* 2))

    widthB = np.sqrt(((tr[0] - tl[0]) \*\* 2) + ((tr[1] - tl[1]) \*\* 2))

    # compute the height of the new image

    heightA = np.sqrt(((tr[0] - br[0]) \*\* 2) + ((tr[1] - br[1]) \*\* 2))

    heightB = np.sqrt(((tl[0] - bl[0]) \*\* 2) + ((tl[1] - bl[1]) \*\* 2))

    # take the maximum of the width and height values to reach

    # our final dimensions

    maxWidth = max(int(widthA), int(widthB))

    maxHeight = max(int(heightA), int(heightB))

    # destination points which will be used to map the screen to a "scanned" view

    dst = np.array([

        [0, 0],

        [maxWidth - 1, 0],

        [maxWidth - 1, maxHeight - 1],

        [0, maxHeight - 1]], dtype = "float32")

    # calculate the perspective transform matrix

    M = cv2.getPerspectiveTransform(rect, dst)

    # warp the perspective to grab the screen

    return cv2.warpPerspective(img, M, (maxWidth, maxHeight))

scanned = wrap\_perspective(original.copy(), contour\_to\_rect(receipt\_contour))

plt.figure(figsize=(12,10))

plt.imshow(scanned)

from skimage.filters import threshold\_local

def bw\_scanner(image):

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

    T = threshold\_local(gray, 21, offset = 5, method = "gaussian")

    return (gray > T).astype("uint8") \* 255

result = bw\_scanner(scanned)

plot\_gray(result)

**Region Segmentation:**

import cv2

import numpy as np

import pandas as pd

from matplotlib import pyplot as plt

img = cv2.imread('/content/pic123.png')

# Create a mask

mask = np.zeros(img.shape[:2],np.uint8)

# Specify the background and foreground models

bgdModel = np.zeros((1,65),np.float64)

fgdModel = np.zeros((1,65),np.float64)

# Define the rectangle for initial segmentation (adjust as needed)

rect = (50,50,450,290)

# Apply GrabCut algorithm

cv2.grabCut(img,mask,rect,bgdModel,fgdModel,5,cv2.GC\_INIT\_WITH\_RECT)

# Modify the mask to include probable background and probable foreground

mask2 = np.where((mask==2)|(mask==0),0,1).astype('uint8')

# Apply the mask to the original image

segmented\_img = img \* mask2[:,:,np.newaxis]

# Display the original and segmented images

plt.subplot(121),plt.imshow(cv2.cvtColor(img, cv2.COLOR\_BGR2RGB))

plt.title('Original Image'), plt.xticks([]), plt.yticks([])

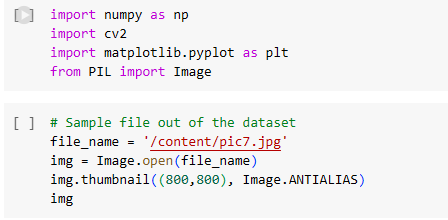
plt.subplot(122),plt.imshow(cv2.cvtColor(segmented\_img, cv2.COLOR\_BGR2RGB))

plt.title('Segmented Image'), plt.xticks([]), plt.yticks([])

plt.show()

**Simulation and tabulated results:**

**Edge Segmentation:**

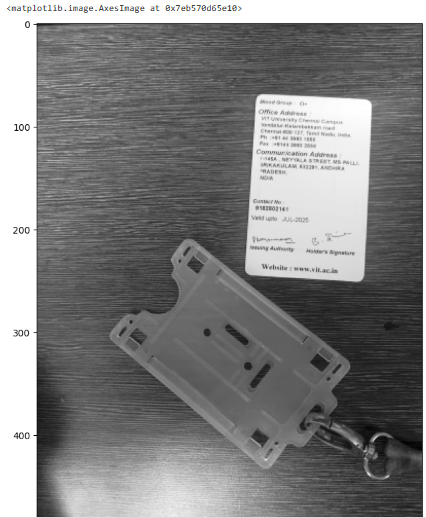
1. *****Import required libraries and give input image*

****

*2.Resize the image to required ratio:* ****

*3.Convert image into grey scale image*

****

****

*4.clear noise with Gaussian Blur filter*

****

****

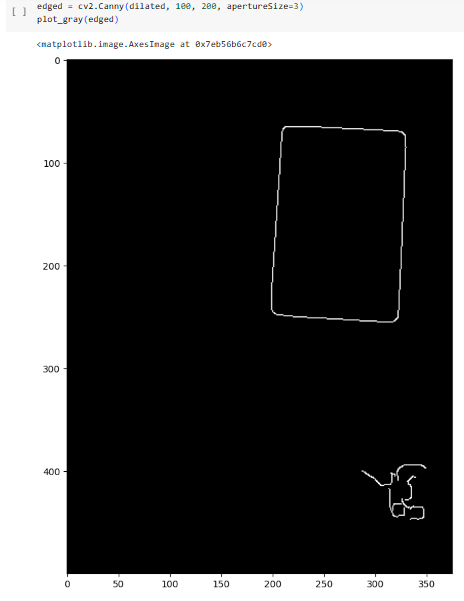
***5.Detect white regions:***

****

****

6.get edges of image

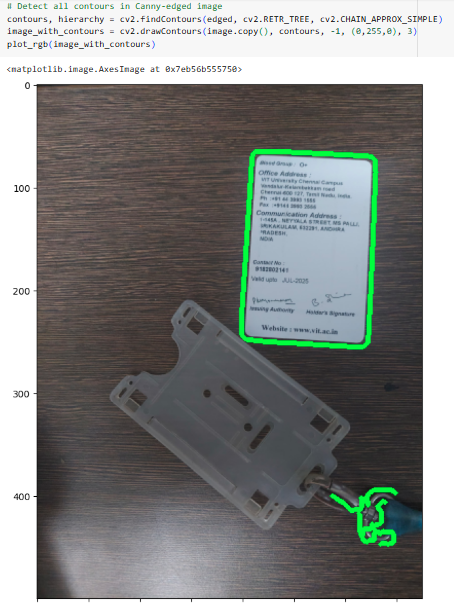
****

****

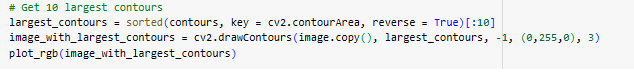
7.convert grey scale image to colour image

****

8.Detect all Counters in image

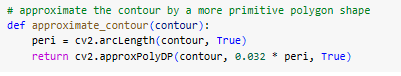
****

9.Get 10 largest counters

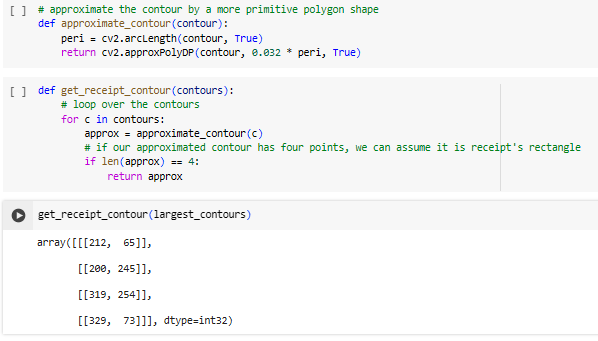
****

****

***10.*** *approximate the contour by a more primitive polygon shape*

****

***11.****Get largest contour value*

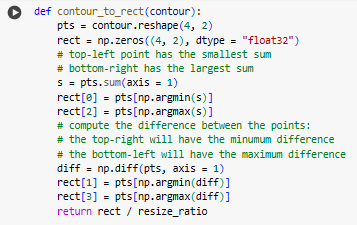
****

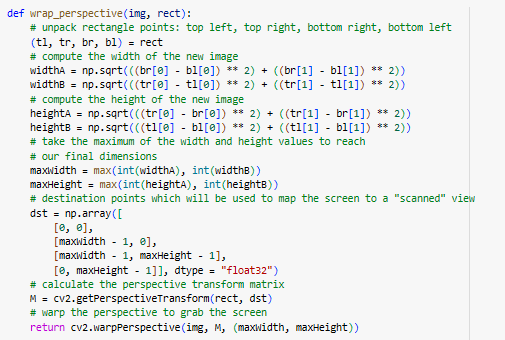
*12. identify largest contour*

****

****

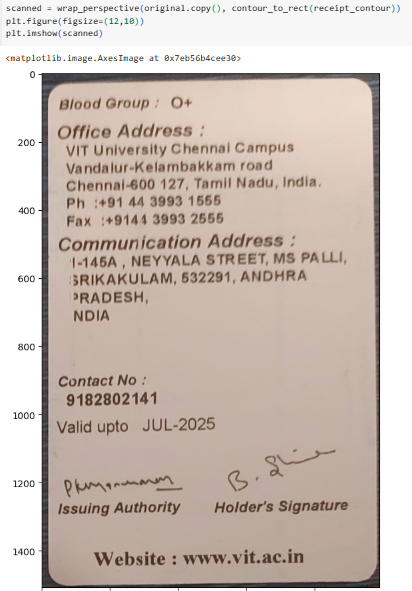
*13.Segment largest contour*

****

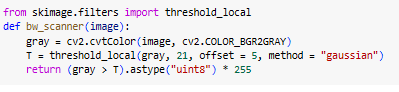
****

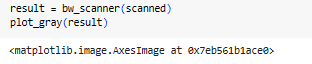
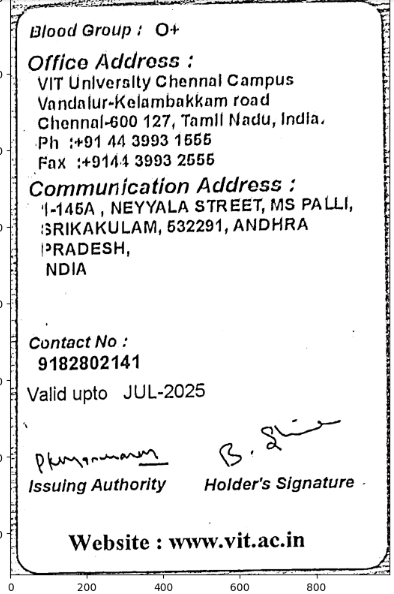
*14.Get segmented image*

****

****

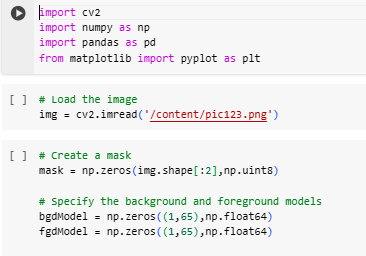
*15.Scanned Copy of segmented image*

****

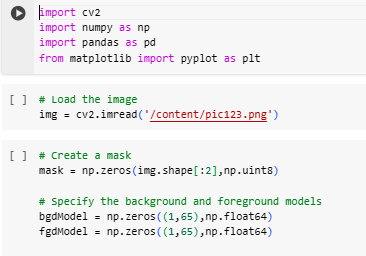
**** ****

**Region Segmentation**:

**1.Import required librarires and take input image**



**2.create mask**



**3.Apply GrabCut algorithm**



**4.Display original image**



**5.Display segmented image**



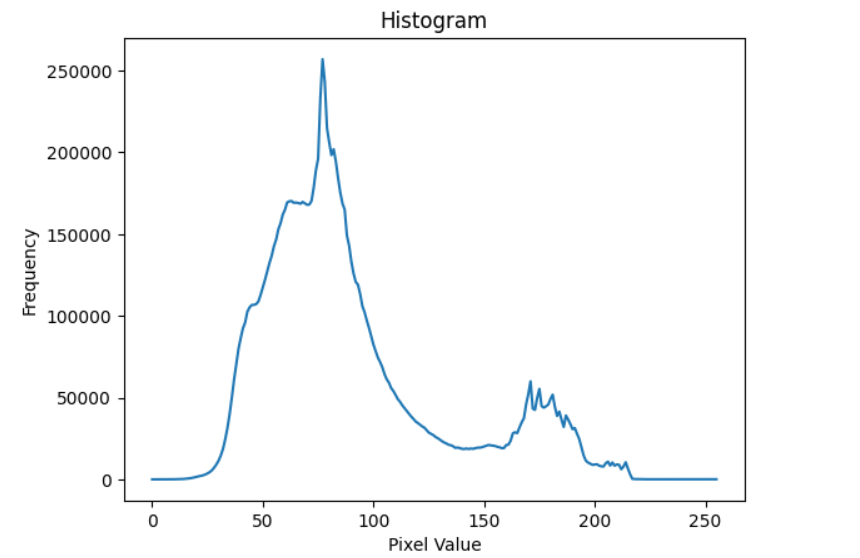
**TABULATED RESULTS**

|  |  |  |
| --- | --- | --- |
| **EVALUATION METRICS** | **Edge Based** | **Region Based** |
| Peak Signal-to-Noise ratio  (PSNR) | 8.100528064795272 | 6.200688379976057 |
| Mean Squared Error  (MSE) | 99.50198937191283 | 93.30836762688615 |
| Structural Similarity Index (SSIM) | 0.21397901875801584 | 0.01380347306687828 |
| Mean Absolute Error  (MAE) | 107.01073015427113 | 119.1846622085048 |

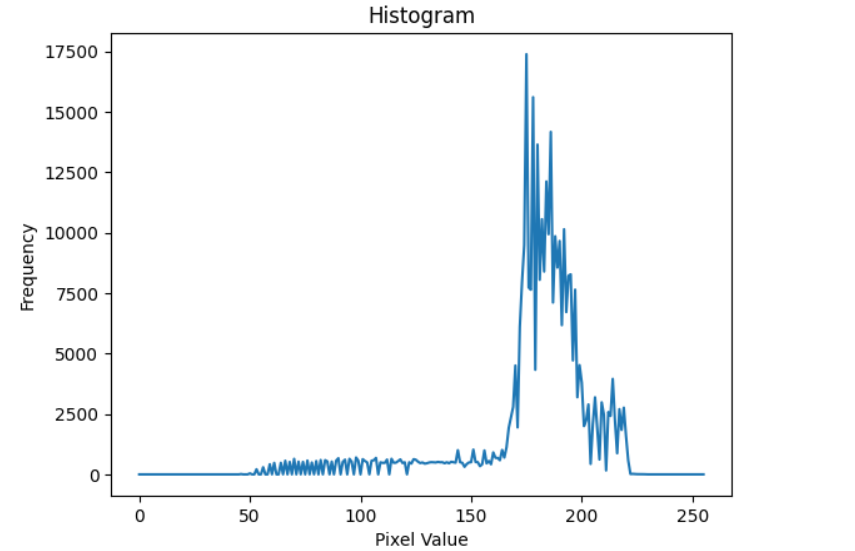
**Graphs:**

***Edge Based***

Original Image:

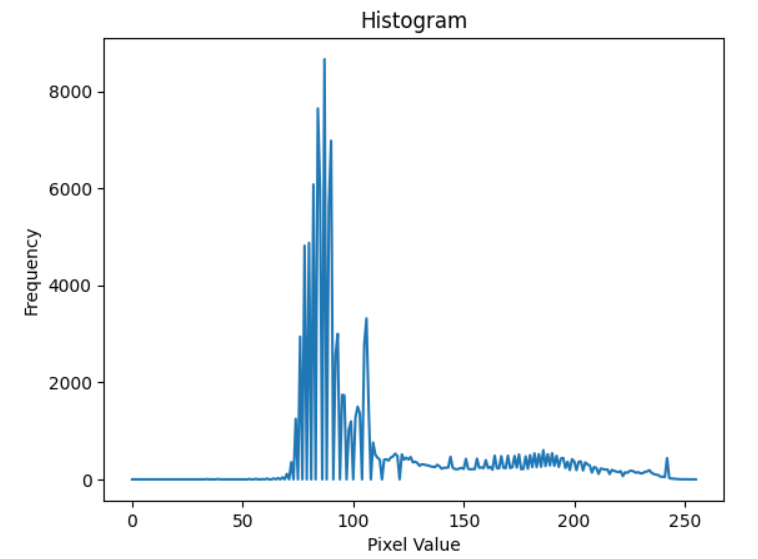
******

Segmented Image:

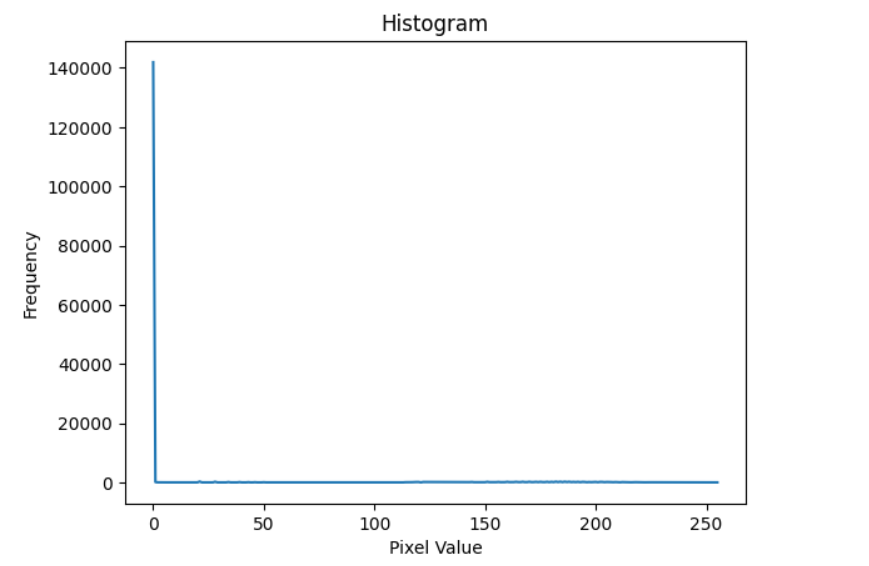
******

***Region Based***

Original Image:



Segmented Image:



**Conclusion:**

In conclusion, this paper has provided a comprehensive overview of image segmentation techniques using OpenCV, a powerful library for computer vision applications. We explored various preprocessing methods to enhance image quality, followed by an in-depth analysis of segmentation algorithms, including thresholding, contour detection, and clustering techniques. Furthermore, advanced segmentation methods such as watershed segmentation and graph-based segmentation, as well as integration with machine learning models like deep learning-based semantic segmentation, were examined for their effectiveness in segmenting complex images. Practical considerations such as parameter tuning, handling diverse image types, and computational efficiency were addressed throughout the paper. Experimental results and comparisons demonstrated the efficacy and performance of the proposed approach in segmenting images accurately and efficiently. Overall, this paper provides insights and methodologies to enhance image segmentation using OpenCV. Future research could focus on exploring novel techniques and advancements in segmentation algorithms to further improve segmentation accuracy and efficiency.

**References:**

[1]Guobo Xie and Wen Lu ,Image Edge Detection Based On Opencv, International Journal of Electronics and Electrical Engineering Vol. 1, No. 2, June 2013

[2] Udit Malik, International Journal for Research in Applied Science & Engineering Technology (IJRASET) ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538, Volume 10, Issue VI. June 2022

[3] Kaijan Khan, IMAGE DETECTION USING OPENCV PYTHON, International Research Journal of Modernization in Engineering Technology and Science, Volume:05, Issue:10, October-2023

[4] Ashraf A. Aly , Safaai Bin Deris, Nazar Zaki,RESEARCH REVIEW FOR DIGITAL IMAGE SEGMENTATION TECHNIQUES, International Journal of Computer Science & Information Technology (IJCSIT) Vol 3, No 5, Oct 2011 ,DOI : 10.5121,ijcsit.2011.3509 99

[5] A. BURDI, A. A. KHAN, S. AWAN, H. A. SHAH, and F. A. ABBASI, “Image Segmentation Approach Using Python OpenCV to Detect Tuberculosis”, *Sindh. Uni. Res. J. SS*, vol. 51, no. 1, Apr. 2020.

[6] Nishant Sharma, Parveen Sultana H, Rahul Singh, Shriniwas Patil, Sumit Pareek,Image Segmentation in Constrained IoT, INTERNATIONAL CONFERENCE ON RECENT TRENDS IN ADVANCED COMPUTING,2019

# [7] Prabu, S. & Gnanasekar, J.M. (2021). A Study on Image Segmentation Method for Image Processing. 10.3233/APC210223.

# [8] Nida M. Zaitoun, Musbah J. Aqel,Survey on Image Segmentation Techniques, Procedia Computer Science,Volume 65,2015.

# [9] S. Sahu, H. Sarma and D. Jyoti Bora, "Image Segmentation and its Different Techniques: An In-Depth Analysis," 2018 International Conference on Research in Intelligent and Computing in Engineering (RICE), San Salvador, El Salvador, 2018, pp. 1-7, doi: 10.1109/RICE.2018.8509038.

# [10] A. M. Khan, Ravi. S, Image Segmentation Methods: A Comparative Study, International Journal of Soft Computing and Engineering (IJSCE) ISSN: 2231-2307, Volume-3, Issue-4, September 2013

# Base Paper:

# S. Saini and K. Arora, "A Study Analysis on the Different Image Segmentation Techniques," Int. J. Inf. Comput. Technol., vol. 4, no. 14, pp. 1445-1452, 2014.