Pattern Recognition-using image processing library OpenCV

Submitted by Manish Bafna
Student Id: 19655
Instructor: Dr. Henry Chang
GIT: **Solve Maze using Depth First**

Table of Content

Introduction

Pattern recognition:

- Definition: Pattern recognition is the process of identifying regularities, similarities, or relationships within data or information.
- Importance: It plays a crucial role in various fields like computer science, psychology, biology, finance, and more.

Applications:

- Computer Vision: Autonomous vehicles, medical imaging, security systems.
- Natural Language Processing (NLP): Sentiment analysis, language translation, chatbots.
- Biometrics: Fingerprint recognition, iris scanning, voice identification.
- Finance: Stock market prediction, fraud detection.
- Medicine: Disease diagnosis, medical image analysis.

Introduction

Types of Patterns

- Visual Patterns
- Examples: Recognizing faces, shapes, symbols, and objects.
- Applications: Facial recognition, image processing, object detection.

- Speech and Audio Patterns
- Examples: Identifying spoken words, sounds, and tones.
- Applications: Speech recognition, voice assistants, music analysis.

- Sequential Patterns
- Examples: Analyzing time series data, sequences of events.
- Applications: Stock market prediction, weather forecasting, DNA sequence analysis.

Overview

The problem statement is for Pattern Recognition - using image processing library OpenCV.

Pattern recognition is a process in which a system or algorithm identifies and classifies patterns within data. In the context of image processing, pattern recognition involves identifying specific features or objects within an image based on certain criteria. Let's break down the steps involved in pattern recognition using the given scenario:

Gray-level Image and Object Digitization:

The initial image is in gray-scale, representing the original picture with two distinct objects – a rectangle and a circle. Gray-level images represent pixel intensities with varying shades of gray, usually ranging from 0 (black) to 255 (white). The first step is to convert the image into a digital format, where each pixel's intensity is represented by a numerical value.

Overview

Binary Image after Histogram Analysis and Thresholding:

In order to simplify the image and separate the objects from the background, histogram analysis and thresholding are applied. The histogram shows the frequency distribution of pixel intensities in the image. Thresholding involves selecting a value that separates pixels into two categories – those below the threshold (background) and those above (objects). This process results in a binary image, where pixels are either 0 (background) or 1 (object).

Connectivity Analysis and Region Labeling:

After thresholding, you have a binary image with isolated objects represented as connected groups of foreground pixels (1s). Connectivity analysis is applied to identify these connected regions. One common approach is to use techniques like connected component labeling or contour tracing to segment the image into separate regions. Each region is assigned a unique label or number.

Overview

Compute Attributes and Recognize Objects:

Once regions are labeled and isolated, various attributes or features are computed for each region. These attributes might include:

Area: Number of pixels within the region.

Perimeter: Sum of the lengths of the boundary pixels.

Centroid: The center of mass of the region.

Circularity: A measure of how closely the shape resembles a circle.

After computing these attributes, the next step is to classify the objects based on the attributes. By comparing the computed attributes against predefined criteria, we can determine whether a region corresponds to a circle or a square

Overview

Convolution with a Gaussian filter is a common technique used in image processing to perform smoothing or blurring operations on an image. In the context of our scenario, we want to use a Gaussian filter to remove salt and pepper noise from a JPEG image containing objects (rectangle and circle). Here's how we can do it step by step:

Salt and Pepper Noise:

Salt and pepper noise is a type of noise that randomly turns some pixels in the image to either the maximum (salt) or minimum (pepper) pixel intensity values. This creates isolated bright and dark pixels that do not correspond to the actual image content.

Gaussian Filter:

A Gaussian filter is a type of linear filter that's used for blurring or smoothing an image. It's characterized by its Gaussian distribution, which assigns more weight to pixels closer to the center and less weight to pixels farther away. The Gaussian filter helps to average out pixel values and reduce high-frequency noise.

Overview

The convolution operation involves sliding the Gaussian filter over the image, pixel by pixel, and computing a weighted average of the pixel values under the filter. This average replaces the original pixel value, resulting in a smoother version of the image.

Applying Gaussian Filter:

To apply a Gaussian filter to your JPEG image and remove salt and pepper noise, follow these steps:

a. Choose the Filter Size: The size of the Gaussian filter (also known as the kernel size) determines the extent of blurring. A larger kernel size results in more smoothing. However, larger kernels can also cause loss of fine details. A common choice is a 3x3 or 5x5 kernel.

b. Choose the Standard Deviation (σ): The standard deviation controls the spread of the Gaussian distribution. A smaller σ leads to a narrower, sharper Gaussian curve, while a larger σ results in a broader curve and more extensive blurring.

Overview

c. Convolution Operation: For each pixel in the image, place the Gaussian filter centered on that pixel. Multiply the filter coefficients by the corresponding pixel values and sum up the results. The sum becomes the new pixel value for that location.

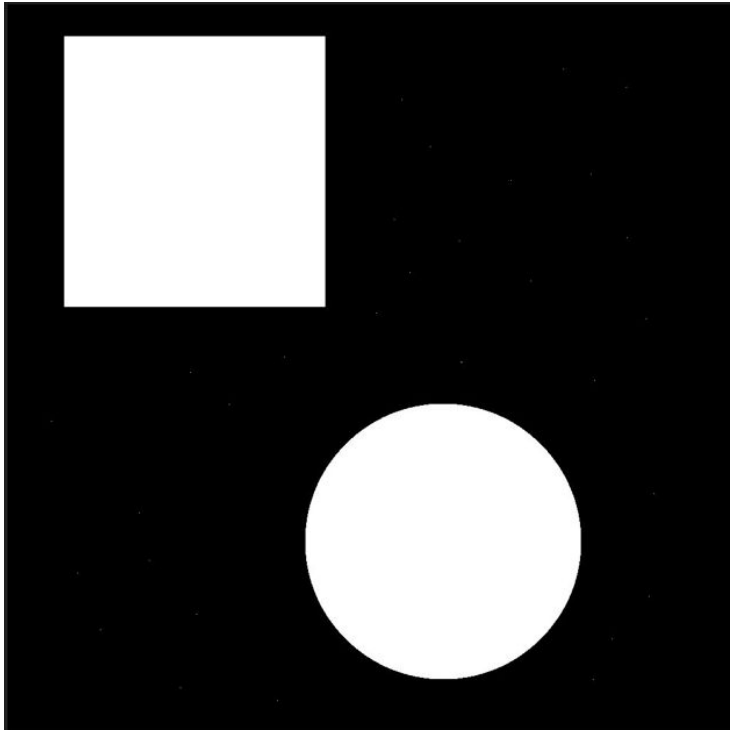
d. Repeat for All Pixels: Slide the filter over the entire image, pixel by pixel, applying the convolution operation at each location.

Result:

After applying the Gaussian filter, the salt and pepper noise will be reduced, and the image will appear smoother. However, some fine details might also be slightly blurred due to the filtering process.

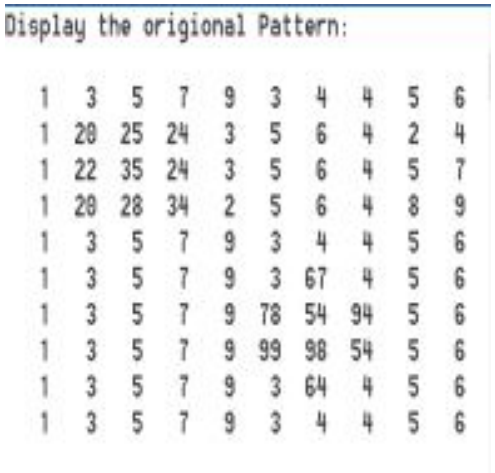
Design

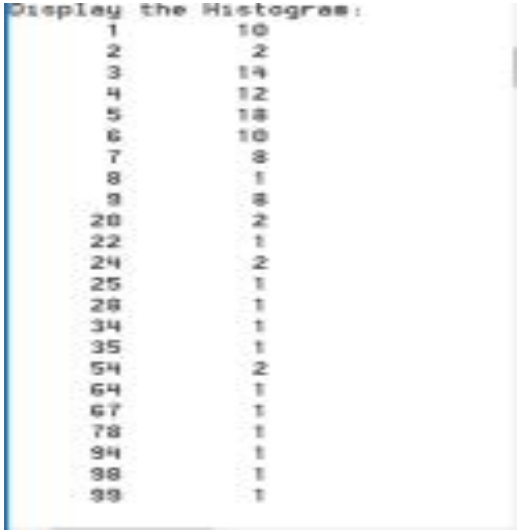
Step 1: read the digital image from a file, and store into an array for future operations.



Design

Histogram: in statistics, a Histogram is a graphical representation showing a visual impression of the distribution of data. A histogram consists of tabular frequencies. (Anon, 2011) Method: Design a program to compute the histogram of a 2D array. Use array elements as an index to another array element. The frequency counts of the population of elements of each array value. F(g)=Ng. (Ng—The number of pixels in the image)





Design

Threshold: Thresholding is a method to convert a Gray Scale Image into a Binary Image, so that objects of interest are separated from the background.(Djahromi,)

Otsu's method: is an image segmentation technique used to automatically determine an optimal threshold value for converting a grayscale image into a binary image. The goal of Otsu's method is to find a threshold that minimizes the variance within the two classes of pixels in the image: the background and the foreground.

In computer vision and image processing, Otsu's method is used to automatically perform histogram shape-based image thresholding, or, the reduction of a gray-level image to a binary image. (Anon, 2013)

Design

Convert a gray-level image into a binary image

```
Display Binary Image

0 0 0 0 0 0 0 0 0 0
0 1 1 1 0 0 0 0 0 0
0 1 1 1 0 0 0 0 0 0
0 1 1 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 1 0 0 0
0 0 0 0 0 1 1 1 0 0
0 0 0 0 0 1 1 1 0 0
0 0 0 0 0 0 1 0 0 0
0 0 0 0 0 0 0 0 0 0
```

Design

Connectivity Analysis: Consider each pixel with number 0 as the background, and number 1 as the foreground. Labels pattern with series numbers(Salem, 2013) (Anon, 2013)

From top to down, left to right, assign labels to each pixel by checking down, right down-left, and down-right 4 points; if they are on foreground(1), mark it with label, else mark it as background(0). Increase label when reach each left border of different object.

From down to top, right to left, check if each object with different label, and choice the largest one as the object label.

Result: each object has its own label. And any pixel with label 0 is on the background.

Design

Connectivity Analysis:

```
Image pattern
0 0 0 0 0 0 0 0 0 0
0 1 1 1 1 1 1 1 1 0
0 1 1 1 1 1 1 1 1 0
0 1 1 0 0 0 0 0 0 0
0 1 1 0 1 1 0 1 1 0
0 1 1 0 1 1 0 1 1 0
0 1 1 0 0 0 0 1 1 0
0 1 1 0 1 1 1 1 1 0
0 1 1 0 1 1 1 1 1 0
0 0 0 0 0 0 0 0 0 0
```

```
Image pattern after the first step
0 0 0 0 0 0 0 0 0 0
0 1 1 1 1 1 1 1 1 0
0 1 1 1 1 1 1 1 1 0
0 1 1 0 0 0 0 0 0 0
0 1 1 0 2 2 0 3 3 0
0 1 1 0 2 2 0 3 3 0
0 1 1 0 0 0 0 3 3 0
0 1 1 0 2 2 3 3 3 0
0 1 1 0 2 2 2 3 3 0
0 0 0 0 0 0 0 0 0 0
```

```
Image pattern after the second step
0 0 0 0 0 0 0 0 0 0
0 1 1 1 1 1 1 1 1 0
0 1 1 1 1 1 1 1 1 0
0 1 1 0 0 0 0 0 0 0
0 1 1 0 2 2 0 3 3 0
0 1 1 0 2 2 0 3 3 0
0 1 1 0 0 0 0 3 3 0
0 1 1 0 3 3 3 3 3 0
0 1 1 0 3 3 3 3 3 0
0 0 0 0 0 0 0 0 0 0
```

```
Image pattern
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 1 1 1 1 1 1 0 0
0 0 1 1 1 1 1 1 0 0
0 0 1 0 0 0 1 1 0 0
0 0 1 0 0 0 1 1 0 0
0 0 1 1 1 1 1 1 0 0
0 0 1 1 1 1 1 1 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
```

```
Image pattern after the first step
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 1 1 1 1 1 1 0 0
0 0 1 1 1 1 1 1 0 0
0 0 1 0 0 0 1 1 0 0
0 0 1 0 0 0 1 1 0 0
0 0 1 1 1 1 1 1 0 0
0 0 1 1 1 1 1 1 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
```

```
Image pattern after the second step
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 1 1 1 1 1 1 0 0
0 0 1 1 1 1 1 1 0 0
0 0 1 0 0 0 1 1 0 0
0 0 1 0 0 0 1 1 0 0
0 0 1 1 1 1 1 1 0 0
0 0 1 1 1 1 1 1 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
```

```
Image pattern
0 0 0 0 0 0 0 0 0 0
0 1 1 1 0 0 0 0 0 0
0 1 1 1 0 0 0 0 0 0
0 1 1 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 1 0 0 0
0 0 0 0 0 0 1 1 1 0
0 0 0 0 0 0 1 1 1 0
0 0 0 0 0 0 0 0 0 0
```

```
Image pattern after the first step
0 0 0 0 0 0 0 0 0 0
0 1 1 1 0 0 0 0 0 0
0 1 1 1 0 0 0 0 0 0
0 1 1 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 2 0 0 0
0 0 0 0 0 0 2 2 2 0
0 0 0 0 0 0 2 2 2 0
0 0 0 0 0 0 0 2 0 0
0 0 0 0 0 0 0 0 0 0
```

```
Image pattern after the second step
0 0 0 0 0 0 0 0 0 0
0 1 1 1 0 0 0 0 0 0
0 1 1 1 0 0 0 0 0 0
0 1 1 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 2 0 0 0
0 0 0 0 0 0 2 2 2 0
0 0 0 0 0 0 2 2 2 0
0 0 0 0 0 0 0 2 0 0
0 0 0 0 0 0 0 0 0 0
```

Design

Compute attributes and recognize objects:

This step detect what object we have in the image.

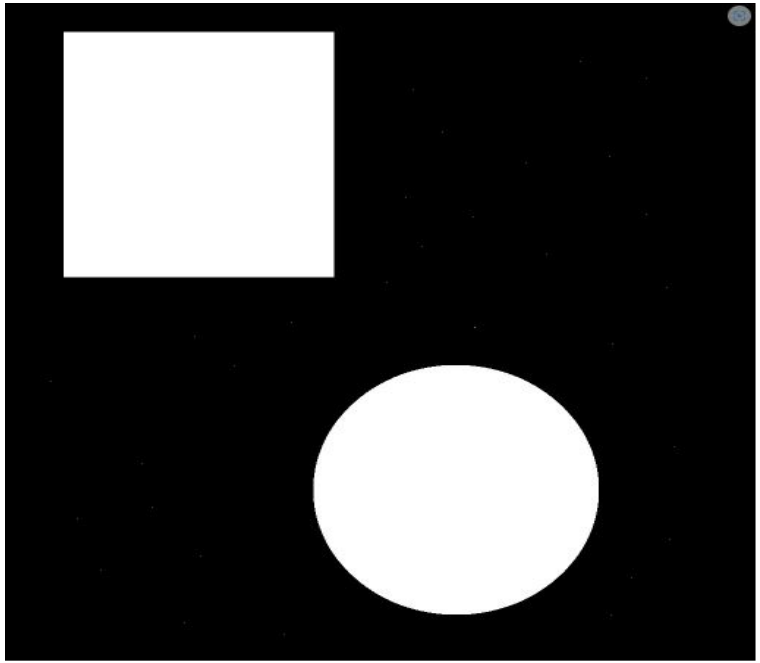
Remove any salt and pepper noises present in the image

Implementation

Pattern recognition process:

The goal is to do JPG image convolution with Gaussian Filter to remove salt and pepper noises.

Original Image

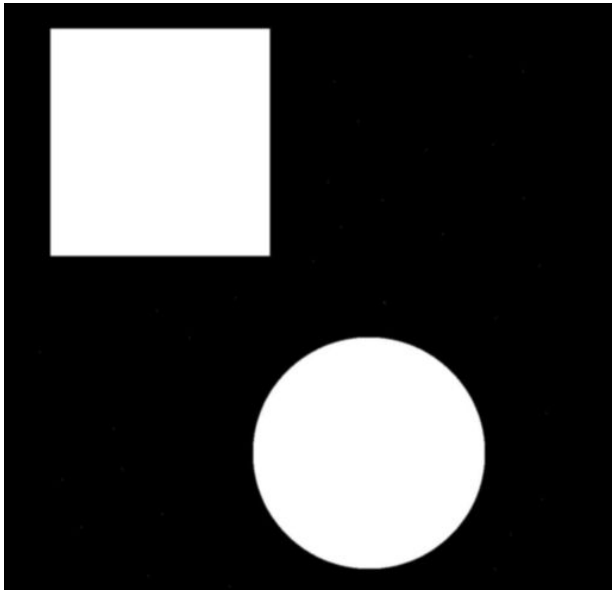


Implementation

1. Noise reduction by blurring:

We use Guassian filter to blur the noise.

```
# Apply Gaussian blur for noise reduction
blurred_img = cv2.GaussianBlur(img, (7, 7), 0)

# Display the OpenCV version
print("OpenCV Version:", cv2.__version__)

# Display the blurred image:
from google.colab.patches import cv2_imshow
cv2_imshow(blurred_img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

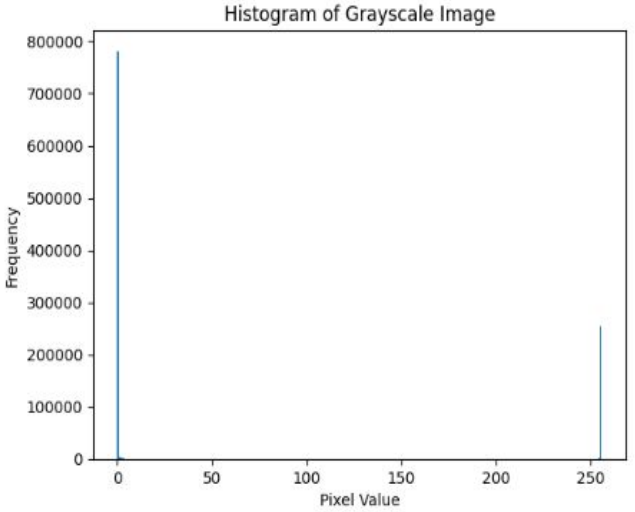


Implementation

2. Histogram

Histogram is used to determine the threshold value that will be used for the next step

```
# Plot the histogram
plt.hist(img.ravel(), 256, [0, 256])
plt.xlabel('Pixel Value')
plt.ylabel('Frequency')
plt.title('Histogram of Grayscale Image')
plt.show()
```



Implementation

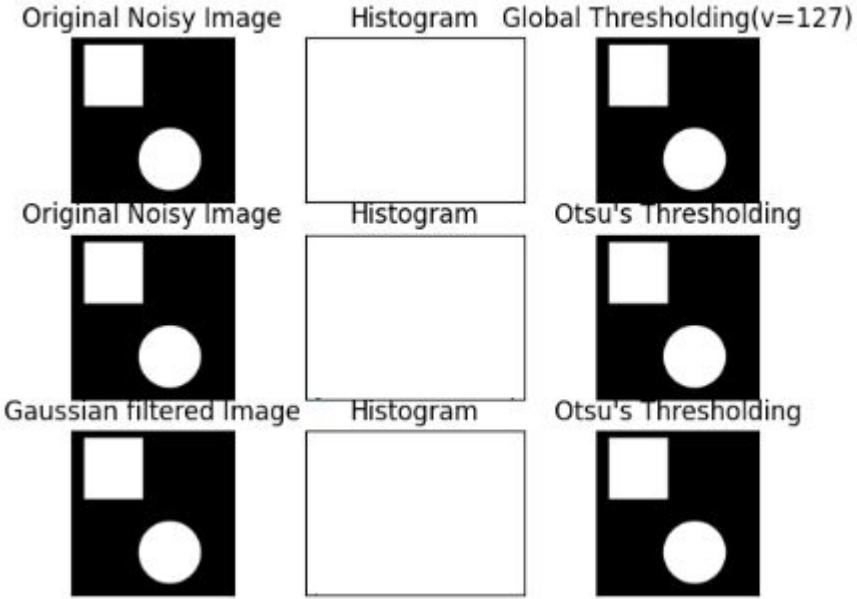
3. Thresholding

We use Otsu's method for Thresholding

```
# Global thresholding
ret1,th1 = cv2.threshold(img,127,255,cv2.THRESH_BINARY)
# Otsu's thresholding
ret2,th2 = cv2.threshold(img,0,255,cv2.THRESH_BINARY+cv2.THRESH_OTSU)
# Otsu's thresholding after Gaussian filtering
blur = cv2.GaussianBlur(img,(5,5),0)
ret3,th3 = cv2.threshold(blur,0,255,cv2.THRESH_BINARY+cv2.THRESH_OTSU)
# plot all the images and their histograms
images = [img, 0, th1,
 img, 0, th2,
 blur, 0, th3]
titles = ['Original Noisy Image','Histogram','Global Thresholding(v=127)',
 'Original Noisy Image','Histogram',"Otsu's Thresholding",
 'Gaussian filtered Image','Histogram',"Otsu's Thresholding"]
for i in range(3):
 plt.subplot(3,3,i*3+1),plt.imshow(images[i*3],'gray')
 plt.title(titles[i*3]), plt.xticks([]), plt.yticks([])
 plt.subplot(3,3,i*3+2),plt.hist(images[i*3].ravel(),256)
 plt.title(titles[i*3+1]), plt.xticks([]), plt.yticks([])
 plt.subplot(3,3,i*3+3),plt.imshow(images[i*3+2],'gray')
 plt.title(titles[i*3+2]), plt.xticks([]), plt.yticks([])
plt.show()
```

Implementation

3. Thresholding



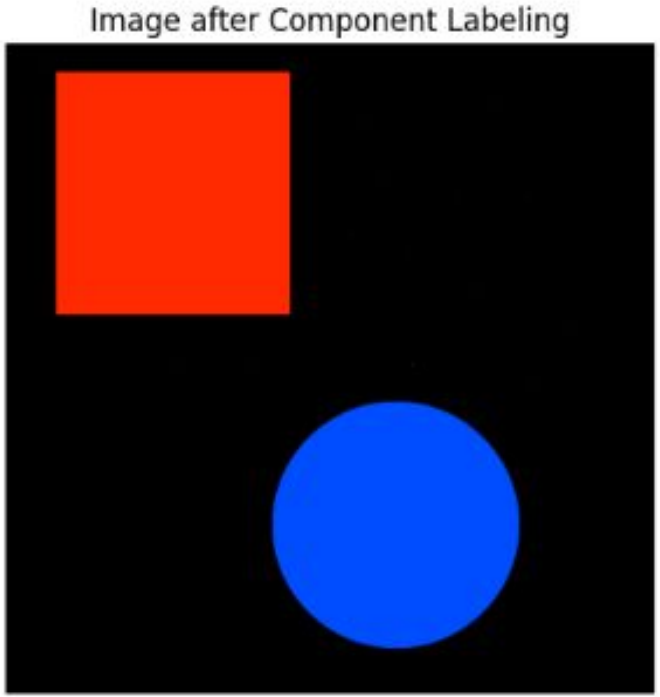
Implementation

4. Connectivity Analysis: This step distinguishes separate objects in an image by using either a 4-pixel or 8-pixel connectivity. 4-pixel connects pixels with the same value along the edges. 8-pixel does the same with the addition of edges and corners.

```
def connected_component_label(path):
    # Getting the input image
    img = cv2.imread(next(iter(uploaded)), cv2.IMREAD_GRAYSCALE)
    # Converting those pixels with values 1-127 to 0 and others to 1
    img = cv2.threshold(img, 127, 255, cv2.THRESH_BINARY)[1]
    # Applying cv2.connectedComponents()
    num_labels, labels = cv2.connectedComponents(img)
    # Map component labels to hue val, 0-179 is the hue range in OpenCV
    label_hue = np.uint8(179*labels/np.max(labels))
    blank_ch = 255*np.ones_like(label_hue)
    labeled_img = cv2.merge([label_hue, blank_ch, blank_ch])
    # Converting cvt to BGR
    labeled_img = cv2.cvtColor(labeled_img, cv2.COLOR_HSV2BGR)
    # set bg label to black
    labeled_img[label_hue==0] = 0
    #Showing Image after Component Labeling
    plt.imshow(cv2.cvtColor(labeled_img, cv2.COLOR_BGR2RGB))
    plt.axis('off')
    plt.title("Image after Component Labeling")
    plt.show()
connected_component_label(img)
```

Implementation

4. Connectivity Analysis:



Implementation

5. Pattern recognition:

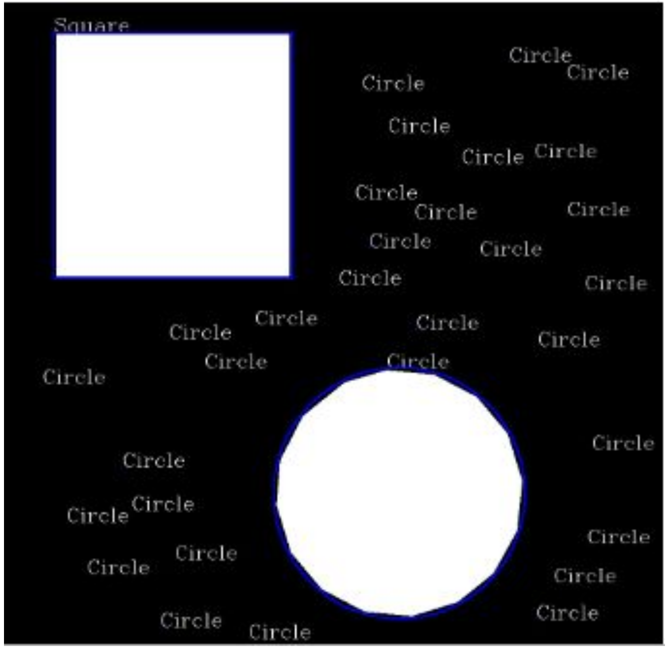
This step is to recognize what kind of objects are in the picture

```
from google.colab.patches import cv2_imshow

font = cv2.FONT_HERSHEY_COMPLEX
#img = cv2.imread('Screenshot1.jpg', cv2.IMREAD_GRAYSCALE)
img = cv2.imread(next(iter(uploaded)), cv2.IMREAD_GRAYSCALE)
_, threshold = cv2.threshold(img, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)
contours, _ = cv2.findContours(threshold, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)

for cnt in contours:
    approx = cv2.approxPolyDP(cnt, 0.01 * cv2.arcLength(cnt, True), True)
    cv2.drawContours(img, [approx], 0, (0), 5)
    x = approx.ravel()[0]
    y = approx.ravel()[1]
    if len(approx) == 4:
        cv2.putText(img, "Square", (x, y), font, 1, (255))
    else:
        cv2.putText(img, "Circle", (x, y), font, 1, (255))

img = cv2.cvtColor(img, cv2.COLOR_GRAY2BGR)
cv2.drawContours(img, contours, -1, (255, 0, 0), 2)

cv2_imshow(img)
```

Implementation

5. Pattern recognition (Original Image):

This step is to recognize what kind of objects are in the picture



Implementation

6. Pattern recognition: Remove Salt and Pepper Noise (Filtered Image):

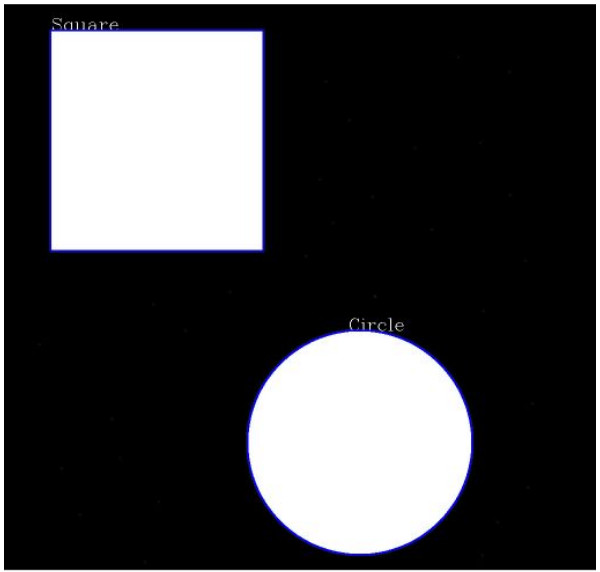
This step is to recognize what kind of objects are in the picture after the image is blurred using Gaussian filter to remove the salt and pepper noises

```
from google.colab.patches import cv2_imshow

font = cv2.FONT_HERSHEY_COMPLEX
img = cv2.imread(next(iter(uploaded)), cv2.IMREAD_GRAYSCALE)
# Apply Gaussian blur for noise reduction
filtered_imag = cv2.GaussianBlur(img, (7, 7), 0)

_, threshold = cv2.threshold(filtered_imag, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)
contours, _ = cv2.findContours(threshold, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)

for cnt in contours:
    approx = cv2.approxPolyDP(cnt, 0.01 * cv2.arcLength(cnt, True), True)
    cv2.drawContours(img, [approx], 0, (0), 5)
    x = approx.ravel()[0]
    y = approx.ravel()[1]
    if len(approx) == 4:
        cv2.putText(filtered_imag, "Square", (x, y), font, 1, (255))
    else:
        cv2.putText(filtered_imag, "Circle", (x, y), font, 1, (255))

filtered_imag = cv2.cvtColor(filtered_imag, cv2.COLOR_GRAY2BGR)
cv2.drawContours(filtered_imag, contours, -1, (255, 0, 0), 2)

cv2_imshow(filtered_imag)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Implementation

6. Pattern recognition: Remove Salt and Pepper Noise (Filtered Image):

This step is to recognize what kind of objects are in the picture after the image is blurred using Gaussian filter to remove the salt and pepper noises



Enhancement Ideas

- Parameter Tuning: We can experiment with different parameter values in the code to see how they affect the results. For example, we can adjust the parameters for the Gaussian blur, the thresholding methods, and the contour detection to find the values that work best for our specific images.
- Adaptive Thresholding: Instead of using global thresholding, we can consider using adaptive thresholding techniques like cv2.adaptiveThreshold. Adaptive thresholding adapts the threshold value for each pixel based on its local neighborhood, which can lead to better results in images with varying lighting conditions.
- Smoothing Contours: After drawing the contours, we can apply morphological operations like dilation or erosion to smooth out the contours and potentially fill small gaps or holes in the shapes.
- Shape Recognition: We can extend the shape recognition part of our code to identify more complex shapes, not just squares and circles. We can detect triangles, pentagons, hexagons, etc., by adjusting the number of vertices detected in the approximated polygon.

Enhancement Ideas

- Color Visualization: If we have access to the original colored image, we can overlay the contours and shape labels on the colored image instead of the grayscale one for better visualization.
- Interactive User Interface: If we want to process multiple images, we can create an interactive user interface using libraries like matplotlib or tkinter. This would allow us to load images, apply processing steps, and visualize the results in a more user-friendly manner.
- Error Handling: We can Implement error handling to gracefully handle cases where the image file is not found or there are issues with image reading or processing.
- Performance Optimization: If we are processing a large number of images, we can explore techniques for parallel processing or optimizing the code for faster execution.
- Machine Learning: Depending on our application, we might consider using machine learning techniques to classify and recognize shapes more accurately.

Conclusion

we've explored image processing techniques using OpenCV in Python to perform various tasks on an input image. We began by loading an image in grayscale and then employed thresholding methods, such as global thresholding and Otsu's thresholding, to segment the image into distinct regions. The thresholded image was then used to identify shapes by detecting contours and approximating their vertices.

To enhance the quality of our image processing, we introduced noise reduction through Gaussian blurring. The Gaussian blur operation helped to smooth out the image and eliminate noise, making subsequent processing steps more accurate and reliable. This addition showcased the significance of preprocessing in image analysis to achieve improved results.

References

1. Anonymous. 2011. Image Processing-Laboratory 3: The histogram of image intensity levels. Technical University of Cluh-Napoca, Computer Science Department. http://ftp.utcluj.ro/pub/users/nedevschi/IP/IP_Labs_2011/ipl_03e.pdf

2. A,. Djahromi, Department of Electrical Engineering University of Texas at Arlington Binary Image Processing (local copt) :

3. Otsu's method. Feb 2013. http://en.wikipedia.org/wiki/Otsu's_method

4. O,. Salem. Feb 2013. Connected Component Labeling Algorithm. http://www.codeproject.com/Articles/336915/Connected-Component-Labeling-Algorithm

5. Anonymous. March 2013. Connected component labeling. http://en.wikipedia.org/wiki/Connected-component_labeling

References

6. Otsu, Nobuyuki. A Threshold Selection Method from Gray-Level Histograms. January, 1979 http://en.wikipedia.org/wiki/Otsu's_method

7. Professor Henry Change: Exercises (sfbu.edu)

Histogram
https://opencv-pythontutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_histograms/py_histogram_begins/py _histogram_begins.html

Threshold
https://opencv-pythontutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_thresholding/py_thresholding.html
https://docs.opencv.org/master/d7/d1b/group__imgproc__misc.html#ggaa9e58d2860d4afa658ef7 0a9b1115576a147222a96556ebc1d948b372bcd7ac59

References

Connectivity analysis https://iq.opengenus.org/connected-component-labeling/
https://github.com/yashml/OpenGenus_Articles_Code/tree/master/Connected%20Component%20 Labeling

Pattern recognition
https://pysource.com/2018/09/25/simple-shape-detection-opencv-with-python-3/