DIGITAL IMAGE PROCESSING
ELL715
ASSIGNMENT 3

10 September 2023

Group Members

1. Bhavik Shankla (2020MT60873)
2. Ritika Soni (2020MT10838)
3. Sai Kiran Gunnala (2020MT60889)
4. Sai Niketh Varanasi (2020MT60895)
5. Yash Pravin Shirke (2020MT60986)

Question 1

Design the Interactive Interface: Devise an intuitive user interface that allows users to upload images and interactively mark areas of their choice, they may assume some object is present.

Solution

A seperate readme text file has been submitted for other information regarding the first part.

Instructions

Give image path in console, a ROI window will open, select the desired section in the image and press "Enter" the section of image will be cropped and stored in a file named "cropped-part.jpeg".

Python Code

```

import cv2

img_path=input('Give Image Path\n')

img_raw = cv2.imread(img_path)


roi = cv2.selectROI(img_raw)


roi_cropped = img_raw[int(roi[1]):int(roi[1]+roi[3]), int(roi[0]):int(roi[0]+roi[2])]
cv2.imwrite("cropped_part.jpeg",roi_cropped)
```

Question 2

Implement Background Subtraction: Develop an algorithm that intelligently subtracts the background of the image based on user markings.

Solution

Background

In this question we have implemented 2 types of algorithms for background subtraction.

1. Based on edge detection created a foreground model

2. Graphcut algorithm

In both the methods the region of interest is given in terms of rectangle.

Method 1

The image first gets cropped to region of interest and then all the below process is performed on it.

First we create a foreground model by taking the absolute difference between original image and it's Gaussian blur, which enables us to extract edges. After extracting edges by giving a threshold we create a binary mask. Then we dilate it to fill the foreground (which in turn fills a bit of background around the edges) and then we do a morphological operation to clean up the mask. Now we do bitwise-and of mask and image to get the final output.

Method 2

The part of image which is not in region of interest gets filled with pixels as it indicates that it is background. (This is done by marking the bits in mask as 0).

As the region is given in the form of rectangle we use Graphcut algorithm with rectangle.

The working of this algorithm is based on min-cut max-flow problem. We represent every pixel as a vertex of a graph and define a similarity of 2 pixels using a function and quantify is as an edge between them.

A mask is created with 0's initially. Then, the region of interest is marked as probable background. Then we apply the graphcut algorithm and get an updated mask (mask2). Now we perform bitwise-and of this mask2 and image to get the final output.

Python Code

```
import cv2
import numpy as np
from google.colab.patches import cv2_imshow

# Loading the image
image0 = cv2.imread('fish.JPEG')

h0,w0,c = image0.shape

print('width: ',w0)
print('height: ',h0)

print('Enter the values and make sure they are in bounds: ')

x1 = int(input('Enter x-coordinate of top left: '))
y1 = int(input('Enter y-coordinate of top left: '))
x2 = int(input('Enter x-coordinate of bottom right: '))
y2 = int(input('Enter y-coordinate of bottom right: '))
# changing image according to region of interest
roi = image0[y1:y2,x1:x2]

```

```python
# Converting the image to grayscale (necessary for background subtraction)
gray_image = cv2.cvtColor(roi, cv2.COLOR_BGR2GRAY)

# created a background model using gaussian blur (can use other types also but
    gaussian blur gives better results)
background_model = cv2.GaussianBlur(gray_image, (3,3), 0)

# Extracting foreground model by taking difference between original gray-scale
    image and blurred image (this helps us to extract edges)
foreground_mask = cv2.absdiff(gray_image, background_model)
print('Foreground mask')
cv2_imshow(foreground_mask)
print()

# Applying threshold to create a binary mask
_, thresholded_mask = cv2.threshold(foreground_mask, 15, 255,
    cv2.THRESH_BINARY)
print('Thresholded mask')
cv2_imshow(thresholded_mask)
print()

# Apply morphological operations to clean up the mask to get a better output
kernel = np.ones((15, 15), np.uint8)

# Here dilation is done first because if we directly because we the part of
    foreground will have almost 0 intensity and very less white pixels
# So, even having 1 white pixel inside the edges will prove that it is part of
    foreground and we try to increase that region using dilate
dilated_mask = cv2.dilate(thresholded_mask,kernel,iterations=2)
# Now we do morphology to clean up isolated white pixels (erosion) and then
    again dilate if anything significant remains.
# Hence we used morphology here
cleaned_mask = cv2.morphologyEx(dilated_mask, cv2.MORPH_CLOSE, kernel)
print('Cleaned mask')
cv2_imshow(cleaned_mask)
print()

# Perform bitwise-and with roi to get the foreground
result_image = cv2.bitwise_and(roi, roi, mask=cleaned_mask)
print('Final output')
cv2_imshow(result_image)

# In the image we can that there is a region of background getting included in
    this is because we are dilating around the edges and hence some
# of the background gets included in it

cv2.waitKey(0)
cv2.destroyAllWindows()


# Loading the image
image1 = cv2.imread('robin_bird.JPEG')

h1,w1,c = image1.shape

print('width: ',w1)
print('height: ',h1)

print('Enter the values and make sure they are in bounds: ')

x1 = int(input('Enter x-coordinate of top left: '))
y1 = int(input('Enter y-coordinate of top left: '))
x2 = int(input('Enter x-coordinate of bottom right: '))
```

```python
y2 = int(input('Enter y-coordinate of bottom right: '))

# changing image according to region of interest
roi1 = image1[y1:y2,x1:x2]

# Convert the image to grayscale (necessary for background subtraction)
gray_image = cv2.cvtColor(roi1, cv2.COLOR_BGR2GRAY)

# created a background model using gaussian blur (can use other types also but gaussian blur gives better results)
background_model = cv2.GaussianBlur(gray_image, (3,3), 0)

# Extracting foreground model by taking difference between original gray-scale image and blurred image (this helps us to extract edges)
print('Foreground mask')
foreground_mask = cv2.absdiff(gray_image, background_model)
cv2_imshow(foreground_mask)
print()

# Applying threshold to create a binary mask
_, thresholded_mask = cv2.threshold(foreground_mask, 20, 255, cv2.THRESH_BINARY)
print('Thresholded mask')
cv2_imshow(thresholded_mask)
print()

# Here dilation is done first because if we directly because we the part of foreground will have almost 0 intensity and very less white pixels
# So, even having 1 white pixel inside the edges will prove that it is part of foreground and we try to increase that region using dilate
dilated_mask = cv2.dilate(thresholded_mask,kernel,iterations=2)
# Now we do morphology to clean up isolated white pixels (erosion) and then again dilate if anything significant remains.
# Hence we used morphology here
cleaned_mask = cv2.morphologyEx(dilated_mask, cv2.MORPH_CLOSE, kernel)
print('Cleaned mask')
cv2_imshow(cleaned_mask)
print()

# Perform bitwise-and with roi to get the foreground
result_image = cv2.bitwise_and(roi1, roi1, mask=cleaned_mask)
print('Final output')
cv2_imshow(result_image)
# In the image we can that there is a region of background getting included in this is because we are dilating around the edges and hence some
# of the background gets included in it

cv2.waitKey(0)
cv2.destroyAllWindows()

#grab cut
# In this part I have used grabcut algoritm to get better results around the edges of the fish

# Loading the image
image2 = cv2.imread('fish.JPEG')  # Replace 'input_image.jpg' with your image file path

# Created a mask to initialize the background and foreground regions
mask = np.zeros(image2.shape[:2], np.uint8)

# Define a rectangle indicating region of interest rect(x,y,w,h)
h2,w2,c = image2.shape
```

```python

print('width: ',w2)
print('height: ',h2)

print('Enter the values and make sure they are in bounds: ')

x1 = int(input('Enter x-coordinate of top left: '))
y1 = int(input('Enter y-coordinate of top left: '))
x2 = int(input('Enter the with of region of interest: '))
y2 = int(input('Enter the height of region of interest: '))
rect = (x1, y1, x2, y2)

# setting the rectangle region as probable foreground which is indicated with 3
mask[rect[1]:rect[1] + rect[3], rect[0]:rect[0] + rect[2]] = 1

# Apply GrabCut algorithm to segment the object(s)
bgdModel = np.zeros((1, 65), np.float64)
fgdModel = np.zeros((1, 65), np.float64)
cv2.grabCut(image2, mask, rect, bgdModel, fgdModel, 5, cv2.GC_INIT_WITH_RECT)

# Creating a new mask to classify probable foreground and foreground as 1 and
    probable background and background as 0
mask2 = np.where((mask == 2) | (mask == 0), 0, 1).astype('uint8')

# Perform bitwise and with image to get the foreground
result_image = cv2.bitwise_and(image2, image2, mask=mask2)

# Displaying the original image and the foreground object(s)
print('Input image')
cv2_imshow(image2)
print()
print('Output image')
cv2_imshow(result_image)
# Here we get almost null background and it finely detects the objects and
    apart from region of interest we made everything 0(i.e background)
# and displaying total image

cv2.waitKey(0)
cv2.destroyAllWindows()


#grab cut
# In this part I have used grabcut algoritm to get better results around the
    edges of the foreground

# Loading the image
image3 = cv2.imread('robin_bird.JPEG')  # Replace 'input_image.jpg' with your
    image file path

# Created a mask to initialize the background and foreground regions
mask = np.zeros(image3.shape[:2], np.uint8)

# Define a rectangle indicating region of interest rect(x,y,w,h)
h3,w3,c = image2.shape

print('width: ',w3)
print('height: ',h3)

print('Enter the values and make sure they are in bounds: ')

x1 = int(input('Enter x-coordinate of top left: '))
y1 = int(input('Enter y-coordinate of top left: '))
x2 = int(input('Enter the with of region of interest: '))
```

```python
y2 = int(input('Enter the height of region of interest: '))
rect = (x1, y1, x2, y2)

# setting the rectangle region as probable foreground which is indicated with 3
mask[rect[1]:rect[1] + rect[3], rect[0]:rect[0] + rect[2]] = 1

# Apply GrabCut algorithm to segment the object(s)
bgdModel = np.zeros((1, 65), np.float64)
fgdModel = np.zeros((1, 65), np.float64)
cv2.grabCut(image3, mask, rect, bgdModel, fgdModel, 5, cv2.GC_INIT_WITH_RECT)

# Creating a new mask to classify probable foreground and foreground as 1 and
    probable background and background as 0
mask2 = np.where((mask == 2) | (mask == 0), 0, 1).astype('uint8')

# Perform bitwise and with image to get the foreground
result_image = cv2.bitwise_and(image3, image3, mask=mask2)

# Displaying the original image and the foreground object(s)
print('Input image')
cv2_imshow(image3)
print()
print('Output image')
cv2_imshow(result_image)
# Here we get almost null background and it finely detects the objects and
    apart from region of interest we made everything 0(i.e background)
# and displaying total image

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

Conclusion

1. The 1st method works based on edges and tries to fill the part of foreground inside, due to which some of the background gets into our image.
2. The kernel size for Gaussian blur, and morphological operation is dependent on image and we need to parameterize them to get a good output.
3. The graphcut algorithm gives us fine results from this we can conclude that we get better results using graph cut than morphological operations based algorithms.

Question 3

Innovative Object Highlighting: Design a creative way to highlight the segmented objects within the image.

Solution

Problem Statement

The objective of this project is to devise innovative techniques for enhancing object visibility and visual appeal within images. We aim to creatively highlight segmented objects, making them stand out from their backgrounds.

Solutions

Thresholding

1. **Trial and Error Approach**
   - This technique begins with loading a grayscale image.
   - To experiment with different threshold values, two thresholds (25 and 50) are applied.
   - The grayscale image and the thresholded versions are displayed side by side for visual comparison.
   - This approach allows for manual adjustment of the threshold value to achieve the desired highlighting effect.
2. **Otsu Method**
   - The Otsu method is an automated thresholding technique.
   - It calculates an optimal threshold to separate foreground objects from the background.
   - Key steps include calculating the histogram, cumulative sum, and cumulative mean of pixel intensities.
   - Between-class variance is computed to find the optimal threshold.
   - The image is then binarized using this threshold, highlighting objects effectively.

Watershed Segmentation

- Watershed segmentation treats pixel intensities as topographical elevations.
- It segments objects based on their spatial characteristics,
- making it suitable for scenarios with intricate object boundaries or overlaps.
- The code applies the watershed algorithm to a given image, partitioning it into regions, each corresponding to a distinct object.
- A unique color map is assigned to each segmented region, enhancing object differentiation in complex contexts.

HSV Color Segmentation

- This technique operates on the Hue, Saturation, and Value (HSV) color space.
- It focuses on displaying the Value channel, which represents the brightness of colors.
- By displaying the Value channel, the code highlights objects based on their brightness, making them visually distinct.
- The user can adjust the figure size to customize the visualization.

Clustering-Based Segmentation Algorithms

- This approach utilizes clustering algorithms to group similar data points, particularly relevant in color images.
- The code demonstrates the K-means clustering algorithm applied to an image.
- Pixels are clustered based on their color similarity, allowing objects to be segmented by color.
- The code displays the original image and the K-means segmented image for various values of K, allowing for visual exploration of different segmentation results.

Discussion

These techniques offer a spectrum of approaches to highlight objects within images, catering to a variety of scenarios and visual objectives. They can be used individually or in combination with creative post-processing strategies to achieve desired effects.

- **Thresholding** provides flexibility for manual adjustment of thresholds and the automated Otsu method for optimal separation.
- **Watershed Segmentation** excels in complex scenarios with overlapping objects, providing a holistic perspective of the scene.
- **HSV Color Segmentation** emphasizes brightness as a criterion for highlighting objects, useful for specific applications.
- **Clustering-Based Segmentation Algorithms** leverage color similarity for object segmentation, offering a range of visual outcomes.

Python Code

```
import cv2
import numpy as np
from IPython.display import Image, display
from matplotlib import pyplot as plt

# The path to image file
image_path = '/content/panda.jpg'

# Load the image
input_image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

# Check if the image was loaded successfully
if input_image is not None:
    sample_g = input_image
    # Experimented threshold values
    sample_t = sample_g > 25
    sample_t1 = sample_g > 50

    import matplotlib.pyplot as plt

    fig, ax = plt.subplots(1, 3, figsize=(15, 5))
    im = ax[0].imshow(sample_g, cmap='gray')
    fig.colorbar(im, ax=ax[0])
    ax[1].imshow(sample_t, cmap='gray')
    ax[0].set_title('Grayscale Image', fontsize=15)
    ax[1].set_title('Threshold at 25', fontsize=15)
    ax[2].imshow(sample_t1, cmap='gray')
    ax[2].set_title('Threshold at 50', fontsize=15)
    plt.show()
else:
    print("Failed to load the image.")
```

```
import cv2

image_path = '/content/flower.jpeg'
# Load the image using OpenCV
input_image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

# Check if the image was loaded successfully
if input_image is not None:
    sample_g = input_image

    # Experimented threshold values
    sample_t = sample_g > 50
    sample_t1 = sample_g > 100

    import matplotlib.pyplot as plt

    fig, ax = plt.subplots(1, 3, figsize=(15, 5))
    im = ax[0].imshow(sample_g, cmap='gray')
    fig.colorbar(im, ax=ax[0])
    ax[1].imshow(sample_t, cmap='gray')
    ax[0].set_title('Grayscale Image', fontsize=15)
    ax[1].set_title('Threshold at 50', fontsize=15)
    ax[2].imshow(sample_t1, cmap='gray')
    ax[2].set_title('Threshold at 100', fontsize=15)
    plt.show()
else:
    print("Failed to load the image.")

import cv2

image_path = '/content/street.jpg'

# Load the image using OpenCV
input_image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

# Check if the image was loaded successfully
if input_image is not None:
    sample_g = input_image

    # Experimented threshold values
    sample_t = sample_g > 50
    sample_t1 = sample_g > 100

    import matplotlib.pyplot as plt

    fig, ax = plt.subplots(1, 3, figsize=(15, 5))
    im = ax[0].imshow(sample_g, cmap='gray')
    fig.colorbar(im, ax=ax[0])
    ax[1].imshow(sample_t, cmap='gray')
    ax[0].set_title('Grayscale Image', fontsize=15)
    ax[1].set_title('Threshold at 50', fontsize=15)
    ax[2].imshow(sample_t1, cmap='gray')
    ax[2].set_title('Threshold at 100', fontsize=15)
    plt.show()
else:
    print("Failed to load the image.")

# Load the image
image = cv2.imread("/content/rose.jpg", cv2.IMREAD_GRAYSCALE)

# Create figures with a specific size
plt.figure(1, figsize=(8, 6))
```

```
plt.imshow(image, cmap='gray')
plt.title("Original image.")

# Calculate the histogram and thresholds using OpenCV
hist = cv2.calcHist([image], [0], None, [256], [0, 256])
hist = hist.flatten()
total_pixels = image.shape[0] * image.shape[1]

# Calculate cumulative sum and cumulative mean
cumsum = np.cumsum(hist)
cummean = cumsum / total_pixels

# Calculate between-class variance
variance = (cummean * (1 - cummean))
max_variance = np.max(variance)
optimal_threshold = np.argmax(variance)

# Apply the optimal threshold to perform Otsu's thresholding
otsu = (image > optimal_threshold).astype(np.uint8) * 255

# Create a figure for the segmented image
plt.figure(2, figsize=(8, 6))
plt.imshow(otsu, cmap='gray')
plt.title("Image segmentation with Otsu thresholding.")

plt.show()


# Load the image
image = cv2.imread("/content/chick.jpg", cv2.IMREAD_GRAYSCALE)

plt.figure(1, figsize=(8, 6))
plt.imshow(image, cmap='gray')
plt.title("Original image.")

# Calculate the histogram and thresholds using OpenCV
hist = cv2.calcHist([image], [0], None, [256], [0, 256])
hist = hist.flatten()
total_pixels = image.shape[0] * image.shape[1]

# Calculate cumulative sum and cumulative mean
cumsum = np.cumsum(hist)
cummean = cumsum / total_pixels

# Calculate between-class variance
variance = (cummean * (1 - cummean))
max_variance = np.max(variance)
optimal_threshold = np.argmax(variance)

# Apply the optimal threshold to perform Otsu's thresholding
otsu = (image > optimal_threshold).astype(np.uint8) * 255

# Create a figure for the segmented image
plt.figure(2, figsize=(8, 6))
plt.imshow(otsu, cmap='gray')
plt.title("Image segmentation with Otsu thresholding.")

plt.show()


# Plot the image
def imshow(img, ax=None):
    if ax is None:
```

```
        ret, encoded = cv2.imencode(".jpg", img)
        display(Image(encoded))
    else:
        ax.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
        ax.axis('off')

#Image loading
img = cv2.imread("/content/yellow_black.jpg")

#image grayscale conversion
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
# Show image
imshow(img)

imshow(gray)


#Threshold Processing
ret, bin_img = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY_INV +
    cv2.THRESH_OTSU)
imshow(bin_img)

# Invert the binary image
inverted_bin_img = cv2.bitwise_not(bin_img)

# Display the inverted binary image
imshow(inverted_bin_img)

# noise removal
kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (3, 3))
bin_img = cv2.morphologyEx(inverted_bin_img, cv2.MORPH_OPEN, kernel,
    iterations=2)
imshow(bin_img)


# Create subplots with 1 row and 2 columns
fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(8, 8))
# sure background area
sure_bg = cv2.dilate(bin_img, kernel, iterations=3)
imshow(sure_bg, axes[0,0])
axes[0, 0].set_title('Sure Background')

# Distance transform
dist = cv2.distanceTransform(bin_img, cv2.DIST_L2, 5)
imshow(dist, axes[0,1])
axes[0, 1].set_title('Distance Transform')

#foreground area
ret, sure_fg = cv2.threshold(dist, 0.5 * dist.max(), 255, cv2.THRESH_BINARY)
sure_fg = sure_fg.astype(np.uint8)
imshow(sure_fg, axes[1,0])
axes[1, 0].set_title('Sure Foreground')

# unknown area
unknown = cv2.subtract(sure_bg, sure_fg)
imshow(unknown, axes[1,1])
axes[1, 1].set_title('Unknown')

plt.show()

# Marker labelling
# sure foreground
ret, markers = cv2.connectedComponents(sure_fg)
```

```

# Add one to all labels so that background is not 0, but 1
markers += 1
# mark the region of unknown with zero
markers[unknown == 255] = 0

fig, ax = plt.subplots(figsize=(6, 6))
ax.imshow(markers, cmap="tab20b")
ax.axis('off')
plt.show()

# watershed Algorithm
markers = cv2.watershed(img, markers)

fig, ax = plt.subplots(figsize=(5, 5))
ax.imshow(markers, cmap="tab20b")
ax.axis('off')
plt.show()


labels = np.unique(markers)

coins = []
for label in labels[2:]:

# Create a binary image in which only the area of the label is in the
    foreground
#and the rest of the image is in the background
    target = np.where(markers == label, 255, 0).astype(np.uint8)

  # Perform contour extraction on the created binary image
    contours, hierarchy = cv2.findContours(
        target, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE
    )
    coins.append(contours[0])

# Draw the outline
img = cv2.drawContours(img, coins, -1, color=(0, 23, 223), thickness=2)
imshow(img)


import cv2
import numpy as np
from skimage.color import rgb2hsv

# Load the grayscale image using OpenCV
image_path = '/content/daisy.jpg'
sample = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

# Convert the grayscale image to a fake RGB image
fake_rgb = cv2.cvtColor(sample, cv2.COLOR_GRAY2BGR)

# Now, you can convert the fake RGB image to HSV
sample_h = cv2.cvtColor(fake_rgb, cv2.COLOR_BGR2HSV)

import matplotlib.pyplot as plt

# Assuming 'sample_h' contains your HSV image
fig, ax = plt.subplots(figsize=(6, 6))  # You can adjust the width and height
    as needed
ax.imshow(sample_h[:, :, 2], cmap='hsv')  # Display only the Value channel
ax.set_title('Value', fontsize=15)
plt.show()
```

```python


# imports
import numpy as np
import cv2 as cv
import matplotlib.pyplot as plt

plt.rcParams["figure.figsize"] = (12, 50)

# load image
img = cv.imread('/content/scenery.jfif')
Z = img.reshape((-1, 3))
# convert to np.float32
Z = np.float32(Z)

# define stopping criteria, number of clusters(K) and apply kmeans()
# TERM_CRITERIA_EPS : stop when the epsilon value is reached
# TERM_CRITERIA_MAX_ITER:  stop when Max iteration is reached
criteria = (cv.TERM_CRITERIA_EPS + cv.TERM_CRITERIA_MAX_ITER, 10, 1.0)

fig, ax = plt.subplots(10, 2, sharey=True)
for i in range(10):
    K = i + 3
    # apply K-means algorithm
    ret, label, center = cv.kmeans(Z, K, None, criteria, 10,
        cv.KMEANS_RANDOM_CENTERS)
    # Now convert back into uint8, and make the original image
    center = np.uint8(center)
    res = center[label.flatten()]
    res2 = res.reshape(img.shape)
    # plot the original image and K-means image
    ax[i, 1].imshow(res2)
    ax[i, 1].set_title('K = %s Image' % K)
    ax[i, 0].imshow(cv.cvtColor(img, cv.COLOR_BGR2RGB))
    ax[i, 0].set_title('Original Image')

plt.show()
```

Question 4

Real-time Object Detection: Integrate real-time object detection using pre-trained models or custom algorithms. As users interact with the image, the system should instantly identify objects based on their markings.

Solution

We have used a pre-trained model called YOLO v3 for object detection. The model used in the code was trained on COCO data set and it can detect 80 objects. The configuration file and weight files of the model are required for running the program which are provided through the file paths in the code.

For each of the 80 objects a different color has been assigned which will be used to show the detected object in the final image. With the provided weights and configuration we create a network and then prepare the input image to run through that Deep neural network.

In the YOLO v3 architecture there are multiple output layers giving out the predictions. get_output_layers() function provides the names of the output layers. We run through all the output layers and detections to neglect the detections with confidence¡=0.5 and draw a rectangle over the object. We then use non-max suppression for handling multiple detections and remove rectangles which are highly overlapping.

Python Code

```
import cv2
import argparse
import numpy as np
from matplotlib import pyplot as plt

#following are the paths to files for input, YOLO configuration file, pre-trained YOLO weights and a text file containing class on which YOLO had been pre-trained
#the model being used here trained on COCO dataset
#model can detect 80 objects and those objects names are in the text file below
input_image_path='C:\\Users\\HP\\Desktop\\7th Semester\\ELL715\\Assignment 3\\Question4\\MalteseDog.JPEG'
config_path='C:\\Users\\HP\\Desktop\\7th Semester\\ELL715\\Assignment 3\\Question4\\yolov3.cfg'
weights_path='C:\\Users\\HP\\Desktop\\7th Semester\\ELL715\\Assignment 3\\Question4\\yolov3.weights'
classes_path='C:\\Users\\HP\\Desktop\\7th Semester\\ELL715\\Assignment 3\\Question4\\yolov3.txt'

#reading the input image from the path given above
image = cv2.imread(input_image_path)

plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
plt.show()

#obtaining the width and height of the input image
Width = image.shape[1]
Height = image.shape[0]
scale = 0.00392


#obtaining the classes and adding them to a list
model_classes = None
with open(classes_path, 'r') as f:
    model_classes = [line.strip() for line in f.readlines()]

#generating different colors for all the classes
COLORS = np.random.uniform(0, 255, size=(len(model_classes), 3))

#reading the configuration and weights files to create a network
```

```python
network = cv2.dnn.readNet(weights_path, config_path)

#preparing the input image to run through the dnn
blob = cv2.dnn.blobFromImage(image, scale, (416,416), (0,0,0), True,
    crop=False)

network.setInput(blob)

#proides the names of the multiple output layers of the network
def get_output_layers(net):

    layer_names = network.getLayerNames()

    output_layers = [layer_names[i - 1] for i in net.getUnconnectedOutLayers()]

    return output_layers


#drawing rectangles over the detected object region
def draw_bounding_box(img, class_id, confidence, x, y, x_plus_w, y_plus_h):

    label = str(model_classes[class_id])

    color = COLORS[class_id]

    cv2.rectangle(img, (x,y), (x_plus_w,y_plus_h), color, 2)

    cv2.putText(img, label, (x-10,y-10), cv2.FONT_HERSHEY_SIMPLEX, 0.5, color,
        2)

#feed forward through the network happens from this line
outs = network.forward(get_output_layers(network))

class_ids = []
confidences = []
boxes = []
conf_threshold = 0.5
nms_threshold = 0.4

# for each detetion from each output layer get the confidence, class id,
    bounding box params and ignore weak detections (confidence < 0.5)
for out in outs:
    for detection in out:
        scores = detection[5:]
        class_id = np.argmax(scores)
        confidence = scores[class_id]
        if confidence > 0.5:
            center_x = int(detection[0] * Width)
            center_y = int(detection[1] * Height)
            w = int(detection[2] * Width)
            h = int(detection[3] * Height)
            x = center_x - w / 2
            y = center_y - h / 2
            class_ids.append(class_id)
            confidences.append(float(confidence))
            boxes.append([x, y, w, h])

#using non-max suppression to remove boxes which are high overlapping
indices = cv2.dnn.NMSBoxes(boxes, confidences, conf_threshold, nms_threshold)

for i in indices:
    box = boxes[i]
    x = box[0]
```

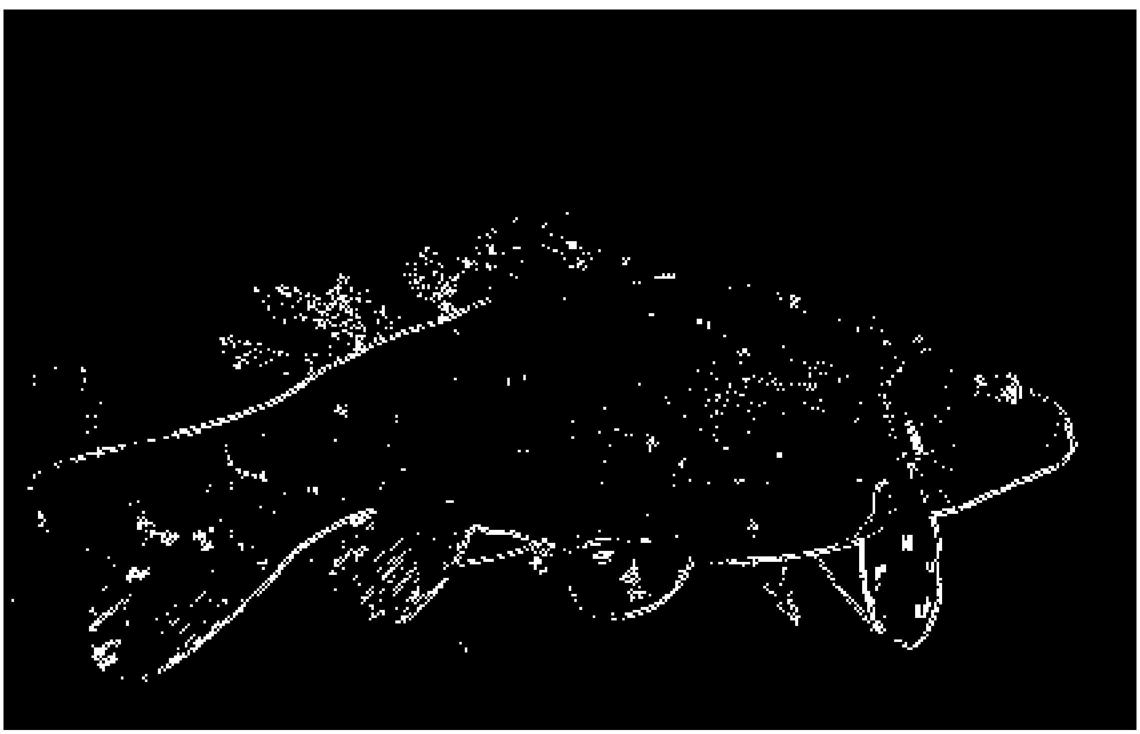
```python
    y = box[1]
    w = box[2]
    h = box[3]

    draw_bounding_box(image, class_ids[i], confidences[i], round(x), round(y),
        round(x+w), round(y+h))


#displaying the output image
plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
plt.show()
```

```python
import cv2
import numpy as np
from google.colab.patches import cv2_imshow

# Loading the image
image0 = cv2.imread('fish.JPEG')

h0,w0,c = image0.shape

print('width: ',w0)
print('height: ',h0)

print('Enter the values and make sure they are in bounds: ')

x1 = int(input('Enter x-coordinate of top left: '))
y1 = int(input('Enter y-coordinate of top left: '))
x2 = int(input('Enter x-coordinate of bottom right: '))
y2 = int(input('Enter y-coordinate of bottom right: '))
# changing image according to region of interest
roi = image0[y1:y2,x1:x2]

# Converting the image to grayscale (necessary for background
subtraction)
gray_image = cv2.cvtColor(roi, cv2.COLOR_BGR2GRAY)

# created a background model using gaussian blur (can use other types
also but gaussian blur gives better results)
background_model = cv2.GaussianBlur(gray_image, (3,3), 0)

# Extracting foreground model by taking difference between original
gray-scale image and blurred image (this helps us to extract edges)
foreground_mask = cv2.absdiff(gray_image, background_model)
print('Foreground mask')
cv2_imshow(foreground_mask)
print()

# Applying threshold to create a binary mask
_, thresholded_mask = cv2.threshold(foreground_mask, 15, 255,
cv2.THRESH_BINARY)
print('Thresholded mask')
cv2_imshow(thresholded_mask)
print()

# Apply morphological operations to clean up the mask to get a better
output
kernel = np.ones((15, 15), np.uint8)

# Here dilation is done first because if we directly because we the
part of foreground will have almost 0 intensity and very less white
pixels
```

```python
# So, even having 1 white pixel inside the edges will prove that it is
part of foreground and we try to increase that region using dilate
dilated_mask = cv2.dilate(thresholded_mask,kernel,iterations=2)
# Now we do morphology to clean up isolated white pixels (erosion) and
then again dilate if anything significant remains.
# Hence we used morphology here
cleaned_mask = cv2.morphologyEx(dilated_mask, cv2.MORPH_CLOSE, kernel)
print('Cleaned mask')
cv2_imshow(cleaned_mask)
print()

# Perform bitwise-and with roi to get the foreground
result_image = cv2.bitwise_and(roi, roi, mask=cleaned_mask)
print('Final output')
cv2_imshow(result_image)

# In the image we can that there is a region of background getting
included in this is because we are dilating around the edges and hence
some
# of the background gets included in it

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

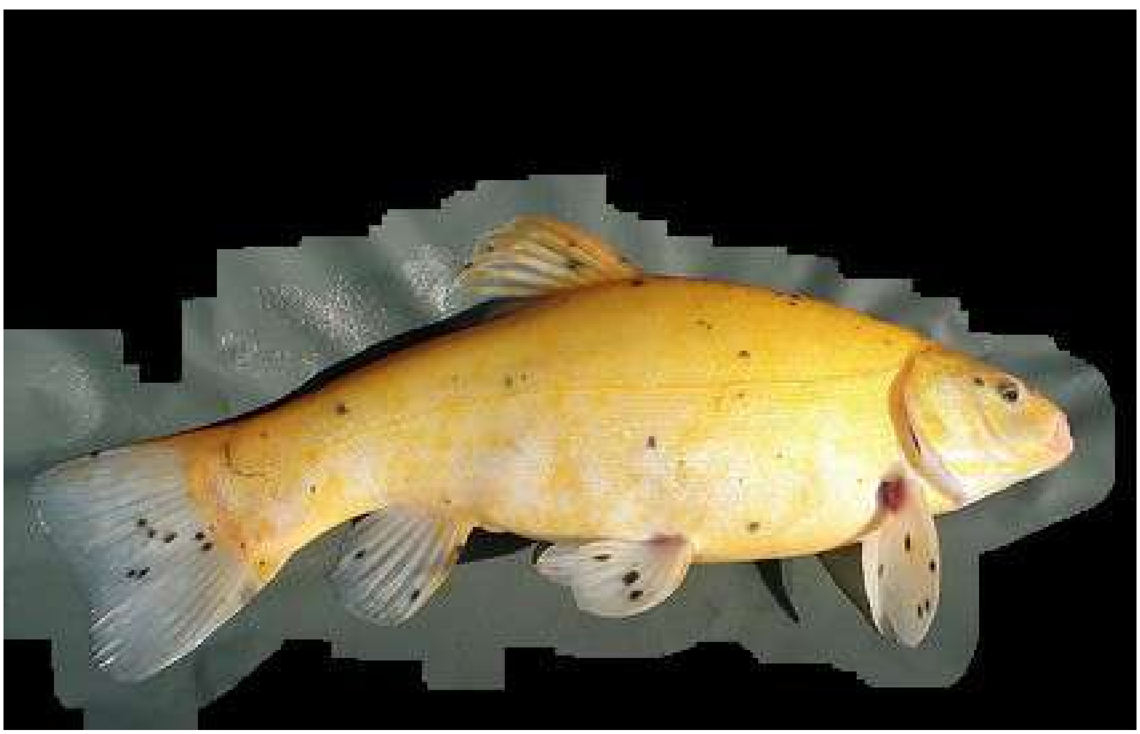
```
width:  500
height:  375
Enter the values and make sure they are in bounds:
Enter x-coordinate of top left: 25
Enter y-coordinate of top left: 50
Enter x-coordinate of bottom right: 450
Enter y-coordinate of bottom right: 320
Foreground mask
```

```
Thresholded mask
```



```
Cleaned mask
```

```
Final output
```

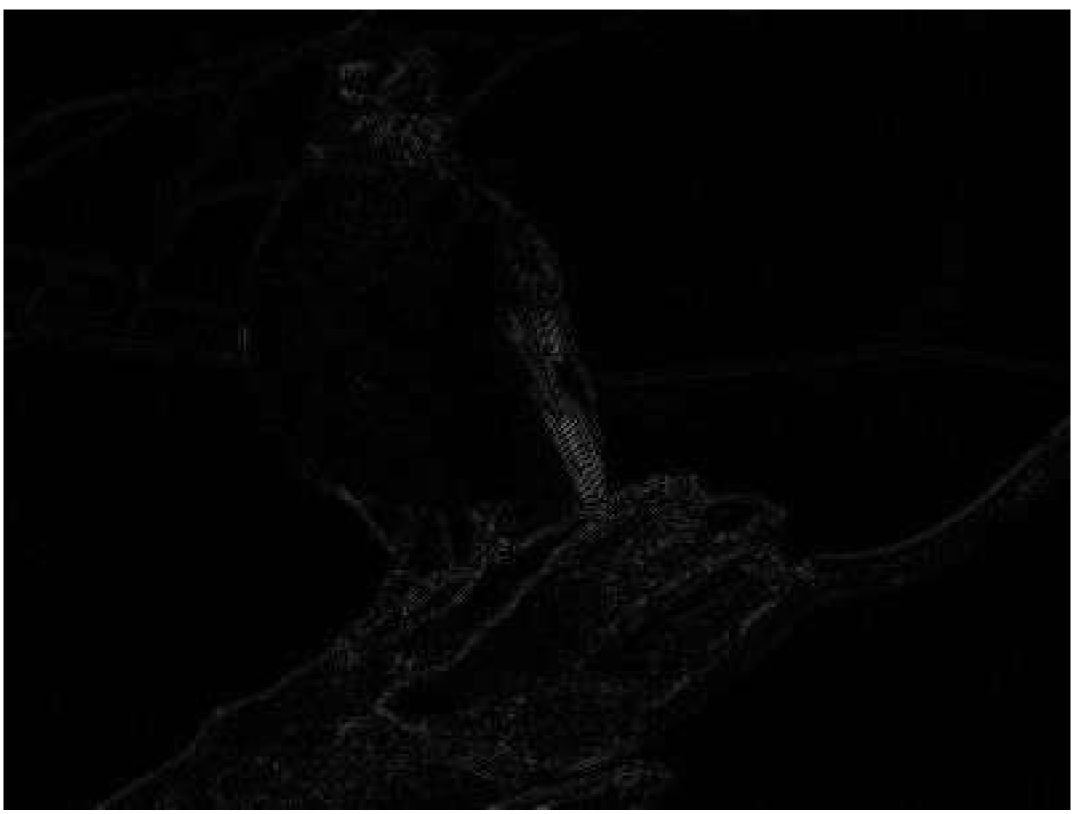


```
# Loading the image
image1 = cv2.imread('robin_bird.JPEG')
```

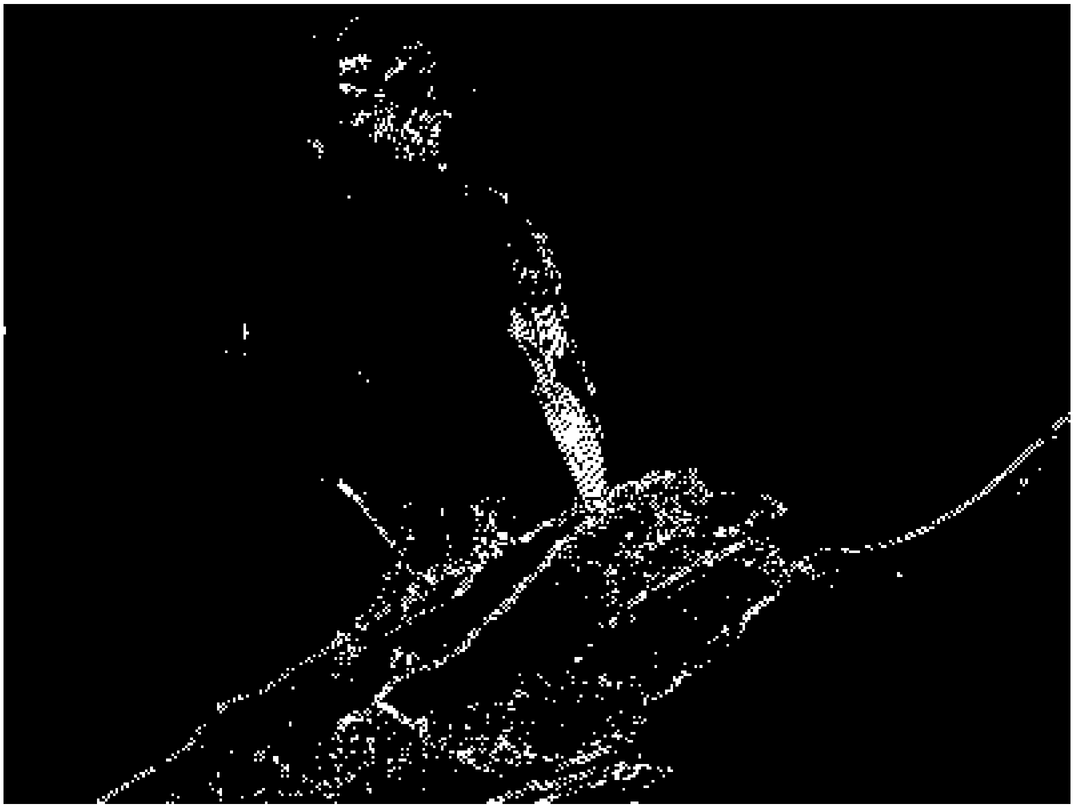
```
h1,w1,c = image1.shape

print('width: ',w1)
print('height: ',h1)

print('Enter the values and make sure they are in bounds: ')

x1 = int(input('Enter x-coordinate of top left: '))
y1 = int(input('Enter y-coordinate of top left: '))
x2 = int(input('Enter x-coordinate of bottom right: '))
y2 = int(input('Enter y-coordinate of bottom right: '))

# changing image according to region of interest
roi1 = image1[y1:y2,x1:x2]

# Convert the image to grayscale (necessary for background
subtraction)
gray_image = cv2.cvtColor(roi1, cv2.COLOR_BGR2GRAY)

# created a background model using gaussian blur (can use other types
also but gaussian blur gives better results)
background_model = cv2.GaussianBlur(gray_image, (3,3), 0)

# Extracting foreground model by taking difference between original
gray-scale image and blurred image (this helps us to extract edges)
print('Foreground mask')
foreground_mask = cv2.absdiff(gray_image, background_model)
cv2_imshow(foreground_mask)
print()

# Applying threshold to create a binary mask
_, thresholded_mask = cv2.threshold(foreground_mask, 20, 255,
cv2.THRESH_BINARY)
print('Thresholded mask')
cv2_imshow(thresholded_mask)
print()

# Here dilation is done first because if we directly because we the
part of foreground will have almost 0 intensity and very less white
pixels
# So, even having 1 white pixel inside the edges will prove that it is
part of foreground and we try to increase that region using dilate
dilated_mask = cv2.dilate(thresholded_mask,kernel,iterations=2)
# Now we do morphology to clean up isolated white pixels (erosion) and
then again dilate if anything significant remains.
# Hence we used morphology here
cleaned_mask = cv2.morphologyEx(dilated_mask, cv2.MORPH_CLOSE, kernel)
print('Cleaned mask')
cv2_imshow(cleaned_mask)
```

```
print()

# Perform bitwise-and with roi to get the foreground
result_image = cv2.bitwise_and(roi1, roi1, mask=cleaned_mask)
print('Final output')
cv2_imshow(result_image)
# In the image we can that there is a region of background getting
included in this is because we are dilating around the edges and hence
some
# of the background gets included in it

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

```
width:  500
height:  375
Enter the values and make sure they are in bounds:
Enter x-coordinate of top left: 50
Enter y-coordinate of top left: 50
Enter x-coordinate of bottom right: 450
Enter y-coordinate of bottom right: 350
Foreground mask
```



```
Thresholded mask
```



```
Cleaned mask
```

```
Final output
```

```
#grab cut
# In this part I have used grabcut algoritm to get better results
around the edges of the fish

# Loading the image
image2 = cv2.imread('fish.JPEG')  # Replace 'input_image.jpg' with
your image file path

# Created a mask to initialize the background and foreground regions
mask = np.zeros(image2.shape[:2], np.uint8)

# Define a rectangle indicating region of interest rect(x,y,w,h)
h2,w2,c = image2.shape

print('width: ',w2)
print('height: ',h2)

print('Enter the values and make sure they are in bounds: ')

x1 = int(input('Enter x-coordinate of top left: '))
y1 = int(input('Enter y-coordinate of top left: '))
x2 = int(input('Enter the with of region of interest: '))
y2 = int(input('Enter the height of region of interest: '))
rect = (x1, y1, x2, y2)

# setting the rectangle region as probable foreground which is
```

```
indicated with 3
mask[rect[1]:rect[1] + rect[3], rect[0]:rect[0] + rect[2]] = 1

# Apply GrabCut algorithm to segment the object(s)
bgdModel = np.zeros((1, 65), np.float64)
fgdModel = np.zeros((1, 65), np.float64)
cv2.grabCut(image2, mask, rect, bgdModel, fgdModel, 5,
cv2.GC_INIT_WITH_RECT)

# Creating a new mask to classify probable foreground and foreground
as 1 and probable background and background as 0
mask2 = np.where((mask == 2) | (mask == 0), 0, 1).astype('uint8')

# Perform bitwise and with image to get the foreground
result_image = cv2.bitwise_and(image2, image2, mask=mask2)

# Displaying the original image and the foreground object(s)
print('Input image')
cv2_imshow(image2)
print()
print('Output image')
cv2_imshow(result_image)
# Here we get almost null background and it finely detects the objects
and apart from region of interest we made everything 0(i.e background)
# and displaying total image

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

```
width:  500
height:  375
Enter the values and make sure they are in bounds:
Enter x-coordinate of top left: 50
Enter y-coordinate of top left: 50
Enter the with of region of interest: 400
Enter the height of region of interest: 270
Input image
```

Output image

```
#grab cut
# In this part I have used grabcut algoritm to get better results
around the edges of the foreground

# Loading the image
image3 = cv2.imread('robin_bird.JPEG')  # Replace 'input_image.jpg'
with your image file path

# Created a mask to initialize the background and foreground regions
mask = np.zeros(image3.shape[:2], np.uint8)

# Define a rectangle indicating region of interest rect(x,y,w,h)
h3,w3,c = image2.shape

print('width: ',w3)
print('height: ',h3)

print('Enter the values and make sure they are in bounds: ')

x1 = int(input('Enter x-coordinate of top left: '))
y1 = int(input('Enter y-coordinate of top left: '))
x2 = int(input('Enter the with of region of interest: '))
```

```
y2 = int(input('Enter the height of region of interest: '))
rect = (x1, y1, x2, y2)

# setting the rectangle region as probable foreground which is
indicated with 3
mask[rect[1]:rect[1] + rect[3], rect[0]:rect[0] + rect[2]] = 1

# Apply GrabCut algorithm to segment the object(s)
bgdModel = np.zeros((1, 65), np.float64)
fgdModel = np.zeros((1, 65), np.float64)
cv2.grabCut(image3, mask, rect, bgdModel, fgdModel, 5,
cv2.GC_INIT_WITH_RECT)

# Creating a new mask to classify probable foreground and foreground
as 1 and probable background and background as 0
mask2 = np.where((mask == 2) | (mask == 0), 0, 1).astype('uint8')

# Perform bitwise and with image to get the foreground
result_image = cv2.bitwise_and(image3, image3, mask=mask2)

# Displaying the original image and the foreground object(s)
print('Input image')
cv2_imshow(image3)
print()
print('Output image')
cv2_imshow(result_image)
# Here we get almost null background and it finely detects the objects
and apart from region of interest we made everything 0(i.e background)
# and displaying total image

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

```
width:  500
height:  375
Enter the values and make sure they are in bounds:
Enter x-coordinate of top left: 50
Enter y-coordinate of top left: 50
Enter the with of region of interest: 400
Enter the height of region of interest: 300
Input image
```

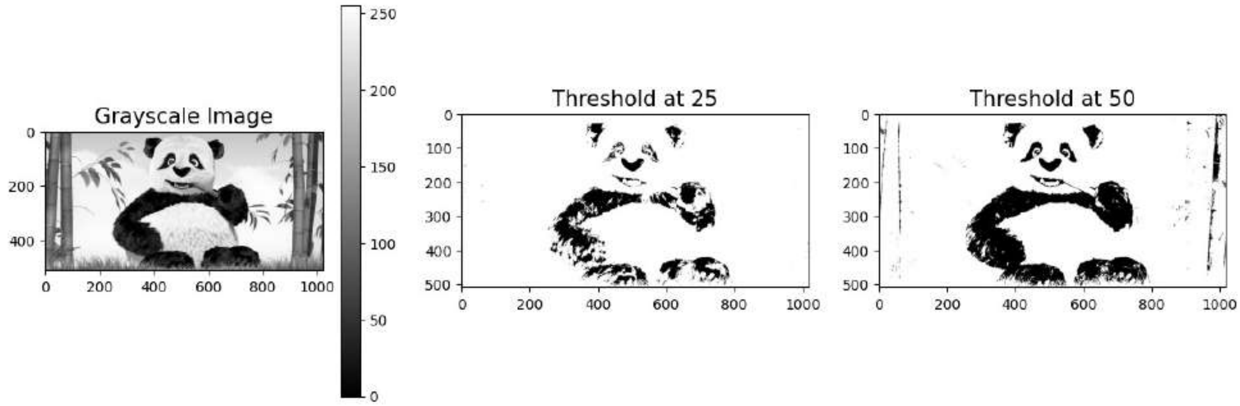
```
Output image
```

1. Thresholding
- Trial and Error
- Otsu Method
1. Watershed Segmentation
2. HSV Color Segmentation
3. Clustering-Based Segmentation Algorithms

---

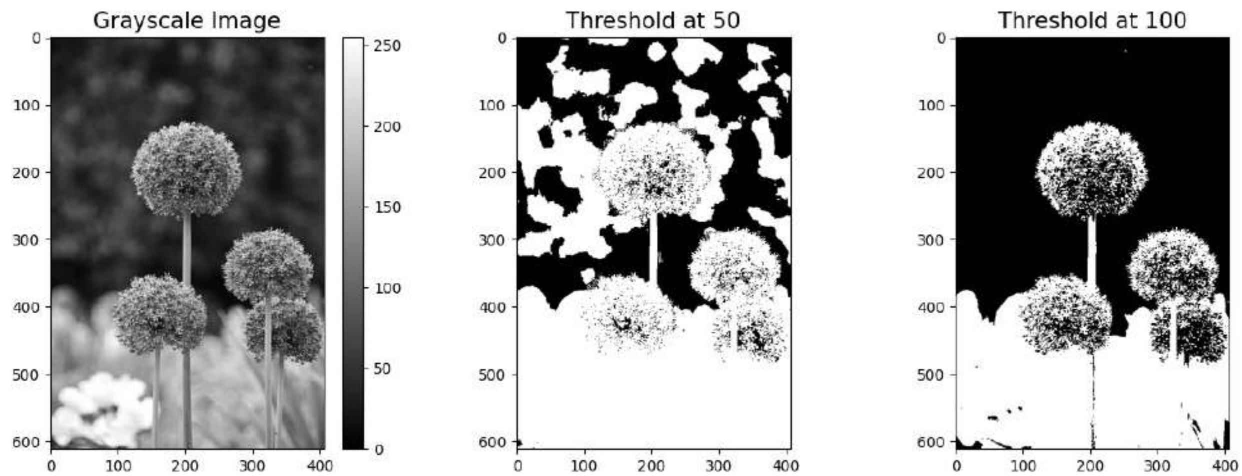
1. Thresholding - Trial and Error

```
import cv2
import numpy as np
from IPython.display import Image, display
from matplotlib import pyplot as plt

# The path to image file
image_path = '/content/panda.jpg'

# Load the image
input_image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

# Check if the image was loaded successfully
if input_image is not None:
    sample_g = input_image
    # Experimented threshold values
    sample_t = sample_g > 25
    sample_t1 = sample_g > 50

    import matplotlib.pyplot as plt

    fig, ax = plt.subplots(1, 3, figsize=(15, 5))
    im = ax[0].imshow(sample_g, cmap='gray')
    fig.colorbar(im, ax=ax[0])
    ax[1].imshow(sample_t, cmap='gray')
    ax[0].set_title('Grayscale Image', fontsize=15)
    ax[1].set_title('Threshold at 25', fontsize=15)
    ax[2].imshow(sample_t1, cmap='gray')
    ax[2].set_title('Threshold at 50', fontsize=15)
    plt.show()
else:
    print("Failed to load the image.")
```

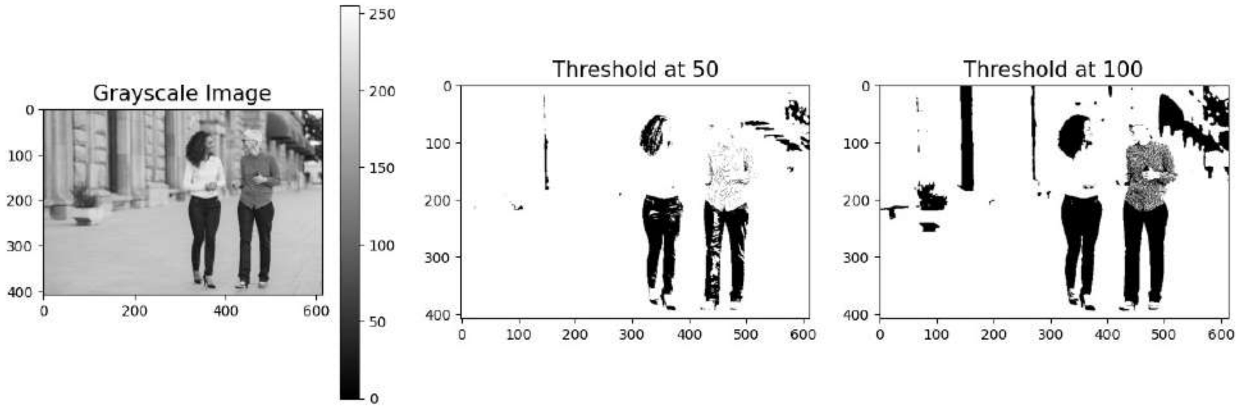


1. Thresholding - Otsu's Method

```
import cv2

image_path = '/content/flower.jpeg'
# Load the image using OpenCV
input_image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

# Check if the image was loaded successfully
if input_image is not None:
    sample_g = input_image

    # Experimented threshold values
    sample_t = sample_g > 50
    sample_t1 = sample_g > 100

    import matplotlib.pyplot as plt

    fig, ax = plt.subplots(1, 3, figsize=(15, 5))
    im = ax[0].imshow(sample_g, cmap='gray')
    fig.colorbar(im, ax=ax[0])
    ax[1].imshow(sample_t, cmap='gray')
    ax[0].set_title('Grayscale Image', fontsize=15)
    ax[1].set_title('Threshold at 50', fontsize=15)
    ax[2].imshow(sample_t1, cmap='gray')
    ax[2].set_title('Threshold at 100', fontsize=15)
    plt.show()
else:
    print("Failed to load the image.")
```



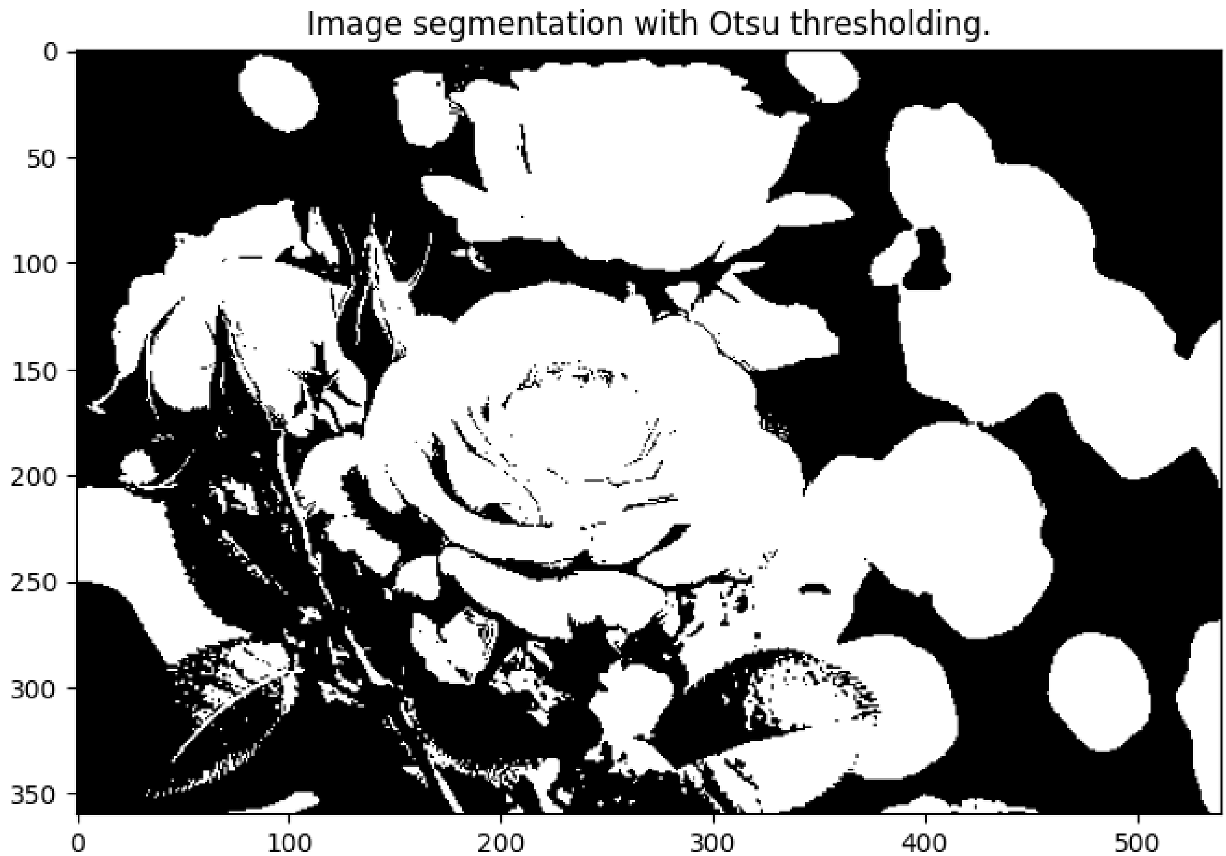

```python
import cv2

image_path = '/content/street.jpg'

# Load the image using OpenCV
input_image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

# Check if the image was loaded successfully
if input_image is not None:
    sample_g = input_image

    # Experimented threshold values
    sample_t = sample_g > 50
    sample_t1 = sample_g > 100

    import matplotlib.pyplot as plt

    fig, ax = plt.subplots(1, 3, figsize=(15, 5))
    im = ax[0].imshow(sample_g, cmap='gray')
    fig.colorbar(im, ax=ax[0])
    ax[1].imshow(sample_t, cmap='gray')
    ax[0].set_title('Grayscale Image', fontsize=15)
    ax[1].set_title('Threshold at 50', fontsize=15)
    ax[2].imshow(sample_t1, cmap='gray')
    ax[2].set_title('Threshold at 100', fontsize=15)
    plt.show()
else:
    print("Failed to load the image.")
```



1. Thresholding Segmentation - Otsu Method

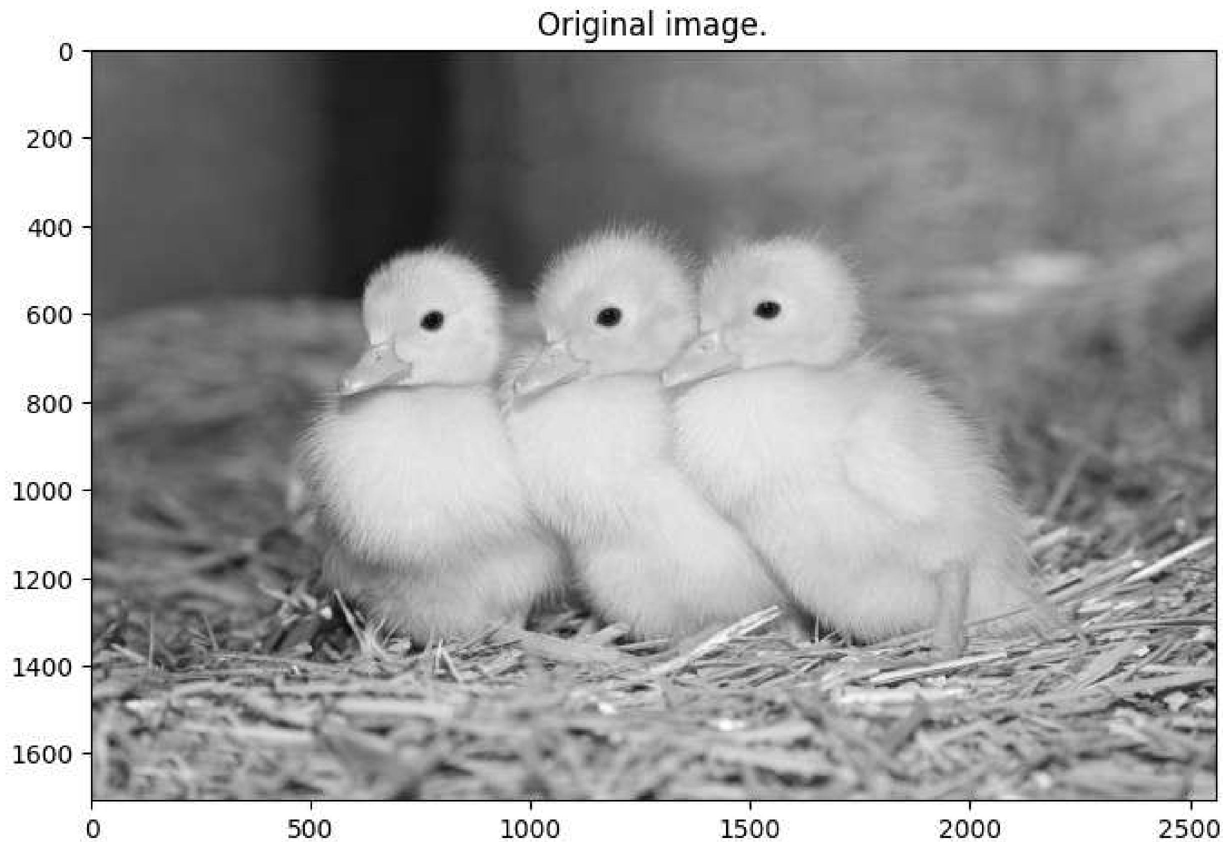
```
# Load the image
image = cv2.imread("/content/rose.jpg", cv2.IMREAD_GRAYSCALE)

# Create figures with a specific size
plt.figure(1, figsize=(8, 6))
plt.imshow(image, cmap='gray')
plt.title("Original image.")

# Calculate the histogram and thresholds using OpenCV
hist = cv2.calcHist([image], [0], None, [256], [0, 256])
hist = hist.flatten()
total_pixels = image.shape[0] * image.shape[1]

# Calculate cumulative sum and cumulative mean
cumsum = np.cumsum(hist)
cummean = cumsum / total_pixels

# Calculate between-class variance
variance = (cummean * (1 - cummean))
max_variance = np.max(variance)
optimal_threshold = np.argmax(variance)

# Apply the optimal threshold to perform Otsu's thresholding
otsu = (image > optimal_threshold).astype(np.uint8) * 255

# Create a figure for the segmented image
plt.figure(2, figsize=(8, 6))
plt.imshow(otsu, cmap='gray')
plt.title("Image segmentation with Otsu thresholding.")

plt.show()
```

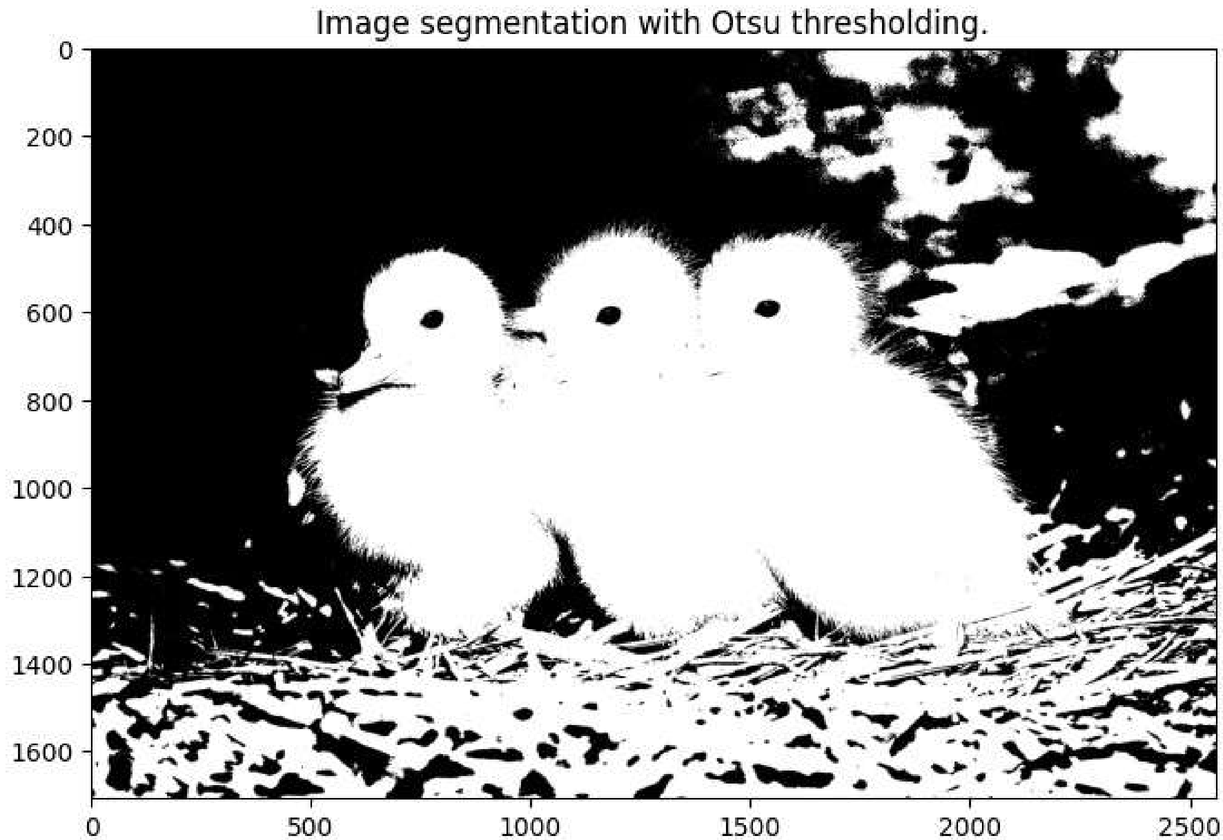
Original image.




```python
# Load the image
image = cv2.imread("/content/chick.jpg", cv2.IMREAD_GRAYSCALE)

plt.figure(1, figsize=(8, 6))
plt.imshow(image, cmap='gray')
plt.title("Original image.")

# Calculate the histogram and thresholds using OpenCV
hist = cv2.calcHist([image], [0], None, [256], [0, 256])
hist = hist.flatten()
total_pixels = image.shape[0] * image.shape[1]

# Calculate cumulative sum and cumulative mean
cumsum = np.cumsum(hist)
cummean = cumsum / total_pixels

# Calculate between-class variance
variance = (cummean * (1 - cummean))
max_variance = np.max(variance)
optimal_threshold = np.argmax(variance)

# Apply the optimal threshold to perform Otsu's thresholding
otsu = (image > optimal_threshold).astype(np.uint8) * 255
```

```python
# Create a figure for the segmented image
plt.figure(2, figsize=(8, 6))
plt.imshow(otsu, cmap='gray')
plt.title("Image segmentation with Otsu thresholding.")

plt.show()
```





1. Watershed Segmentation

```
# Plot the image
def imshow(img, ax=None):
    if ax is None:
        ret, encoded = cv2.imencode(".jpg", img)
        display(Image(encoded))
    else:
        ax.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
        ax.axis('off')

#Image loading
img = cv2.imread("/content/yellow_black.jpg")

#image grayscale conversion
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
# Show image
imshow(img)
```

```
imshow(gray)
```

```
#Threshold Processing
ret, bin_img = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY_INV +
cv2.THRESH_OTSU)
imshow(bin_img)
```

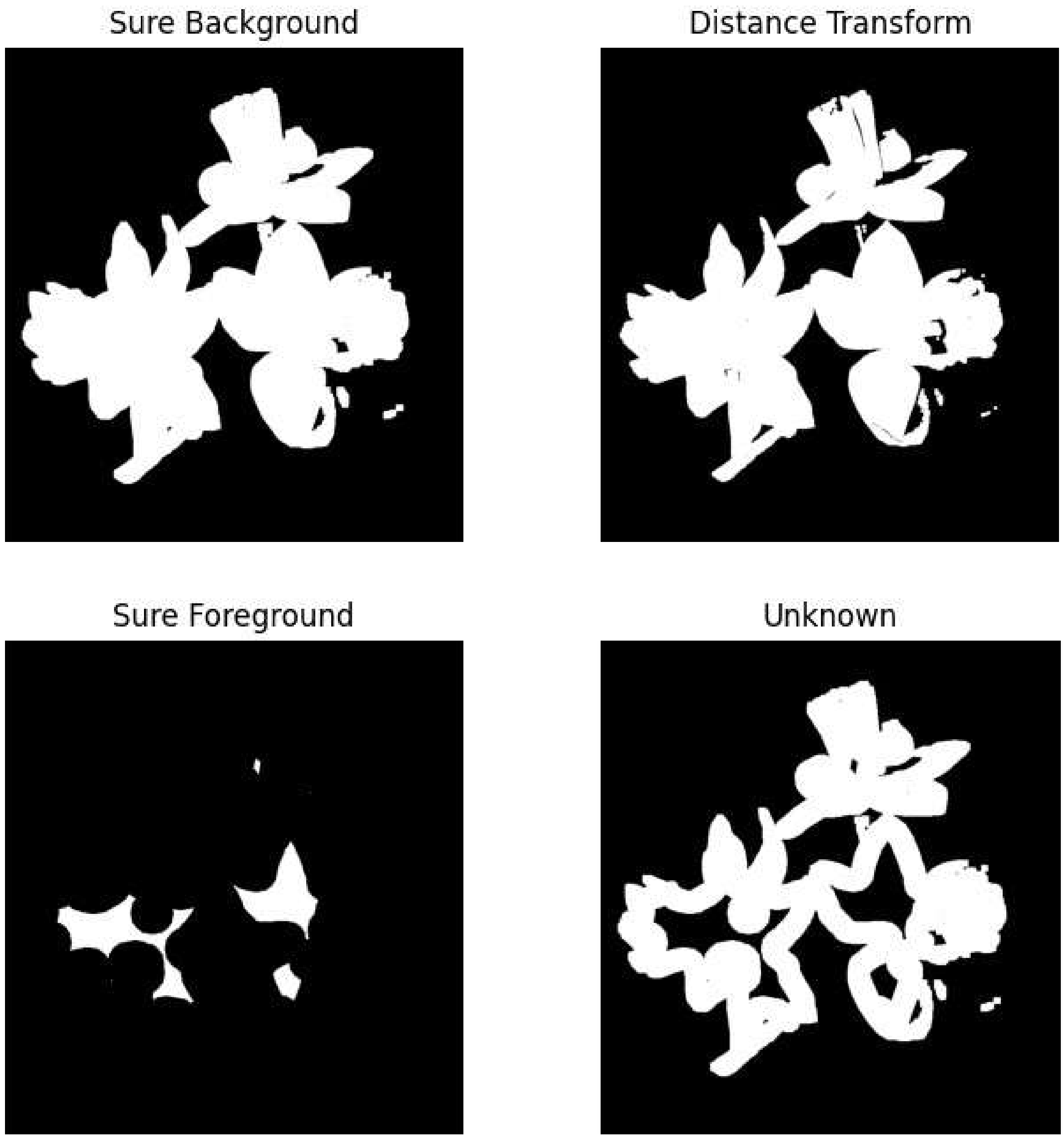
```python
# Invert the binary image
inverted_bin_img = cv2.bitwise_not(bin_img)

# Display the inverted binary image
imshow(inverted_bin_img)
```

```
# noise removal
kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (3, 3))
bin_img = cv2.morphologyEx(inverted_bin_img, cv2.MORPH_OPEN, kernel,
iterations=2)
imshow(bin_img)
```

```
# Create subplots with 1 row and 2 columns
fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(8, 8))
# sure background area
sure_bg = cv2.dilate(bin_img, kernel, iterations=3)
imshow(sure_bg, axes[0,0])
axes[0, 0].set_title('Sure Background')

# Distance transform
dist = cv2.distanceTransform(bin_img, cv2.DIST_L2, 5)
```

```
imshow(dist, axes[0,1])
axes[0, 1].set_title('Distance Transform')

#foreground area
ret, sure_fg = cv2.threshold(dist, 0.5 * dist.max(), 255,
cv2.THRESH_BINARY)
sure_fg = sure_fg.astype(np.uint8)
imshow(sure_fg, axes[1,0])
axes[1, 0].set_title('Sure Foreground')

# unknown area
unknown = cv2.subtract(sure_bg, sure_fg)
imshow(unknown, axes[1,1])
axes[1, 1].set_title('Unknown')

plt.show()
```

```
WARNING:matplotlib.image:Clipping input data to the valid range for
imshow with RGB data ([0..1] for floats or [0..255] for integers).
```

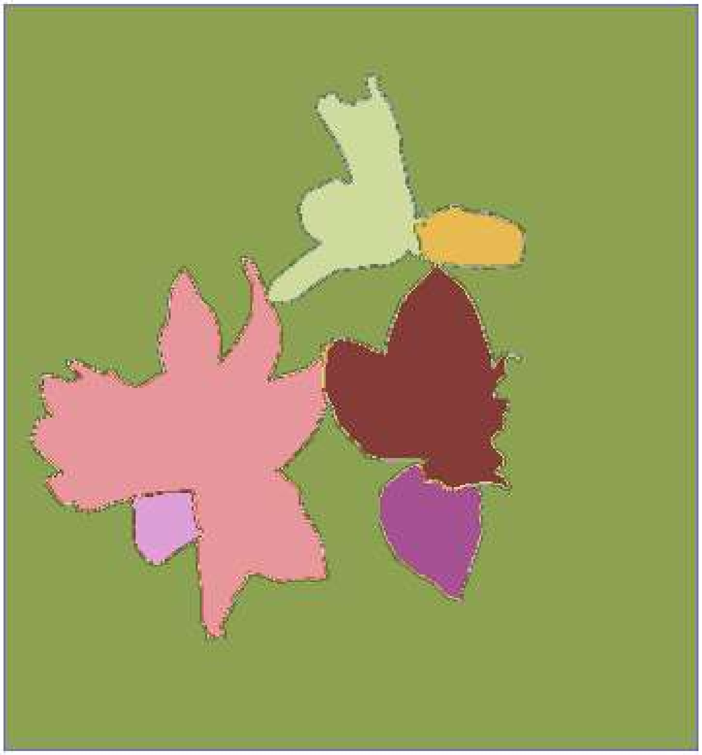



```
# Marker labelling
# sure foreground
ret, markers = cv2.connectedComponents(sure_fg)

# Add one to all labels so that background is not 0, but 1
markers += 1
# mark the region of unknown with zero
markers[unknown == 255] = 0

fig, ax = plt.subplots(figsize=(6, 6))
ax.imshow(markers, cmap="tab20b")
```

```
ax.axis('off')
plt.show()
```

```
# watershed Algorithm
markers = cv2.watershed(img, markers)

fig, ax = plt.subplots(figsize=(5, 5))
ax.imshow(markers, cmap="tab20b")
ax.axis('off')
plt.show()


labels = np.unique(markers)

coins = []
for label in labels[2:]:

# Create a binary image in which only the area of the label is in the
foreground
#and the rest of the image is in the background
    target = np.where(markers == label, 255, 0).astype(np.uint8)
```

```
# Perform contour extraction on the created binary image
    contours, hierarchy = cv2.findContours(
        target, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE
    )
    coins.append(contours[0])
```

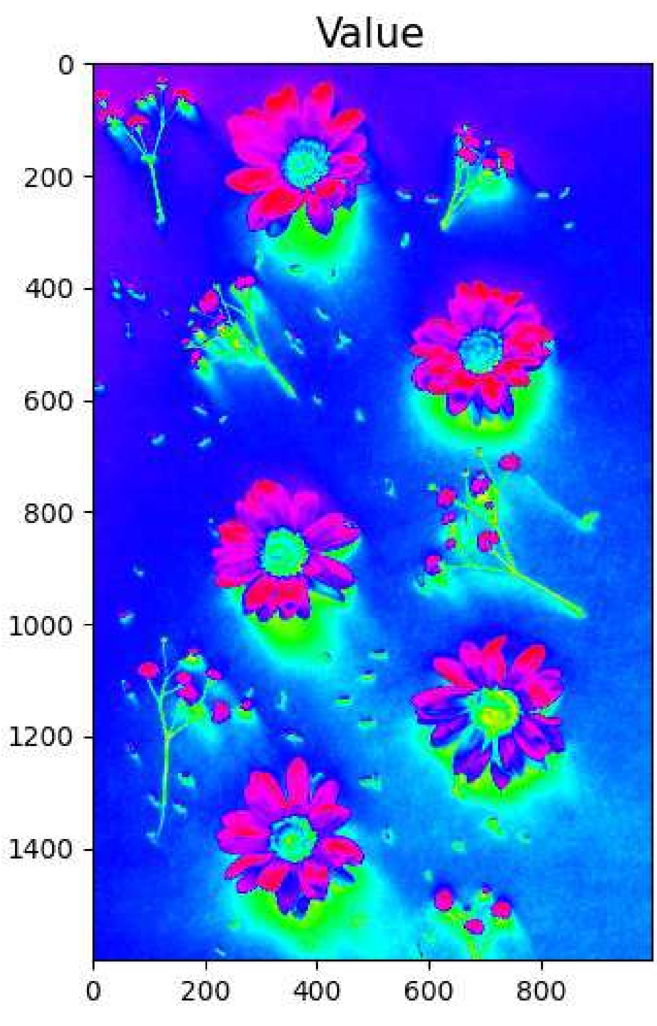


```
# Draw the outline
img = cv2.drawContours(img, coins, -1, color=(0, 23, 223),
thickness=2)
imshow(img)
```

1. HSV Color Segmentation

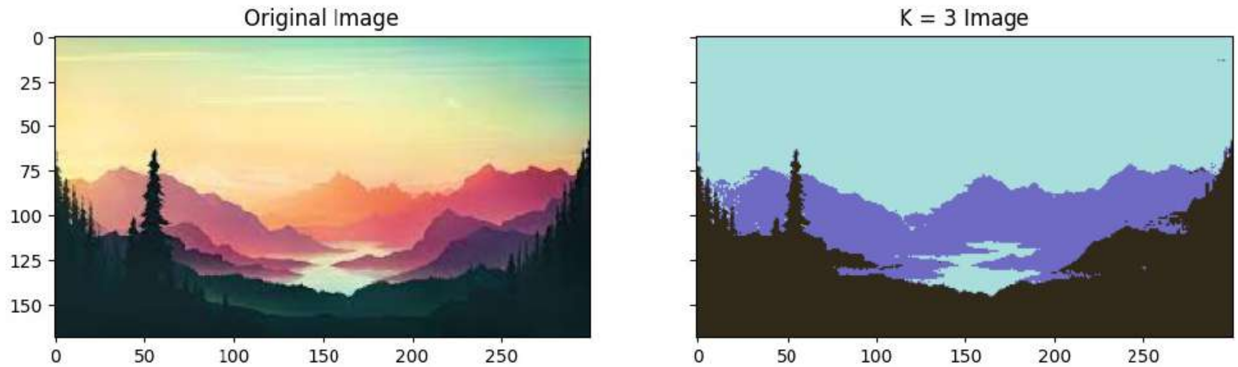
```
import cv2
import numpy as np
from skimage.color import rgb2hsv

# Load the grayscale image using OpenCV
image_path = '/content/daisy.jpg'
sample = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
```

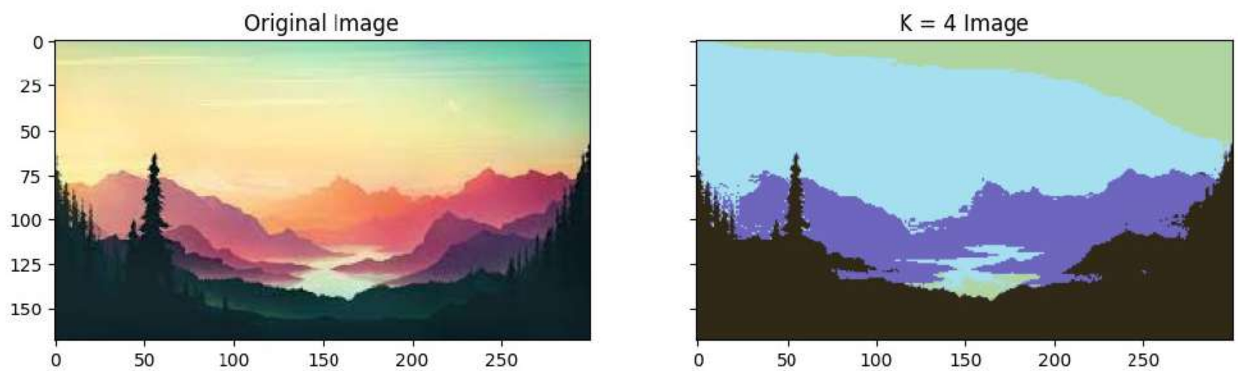
```python
# Convert the grayscale image to a fake RGB image
fake_rgb = cv2.cvtColor(sample, cv2.COLOR_GRAY2BGR)

# Now, you can convert the fake RGB image to HSV
sample_h = cv2.cvtColor(fake_rgb, cv2.COLOR_BGR2HSV)

import matplotlib.pyplot as plt

# Assuming 'sample_h' contains your HSV image
fig, ax = plt.subplots(figsize=(6, 6))  # You can adjust the width and height as needed
ax.imshow(sample_h[:, :, 2], cmap='hsv')  # Display only the Value channel
ax.set_title('Value', fontsize=15)
plt.show()
```

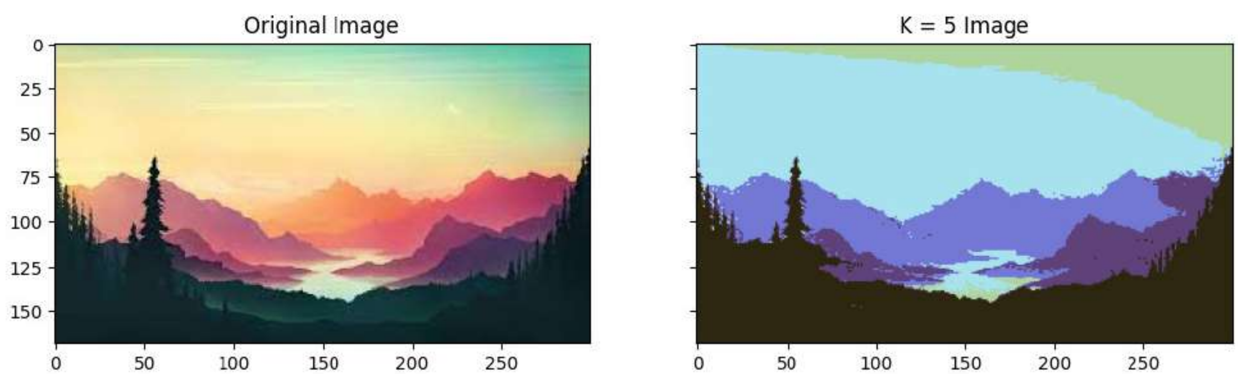


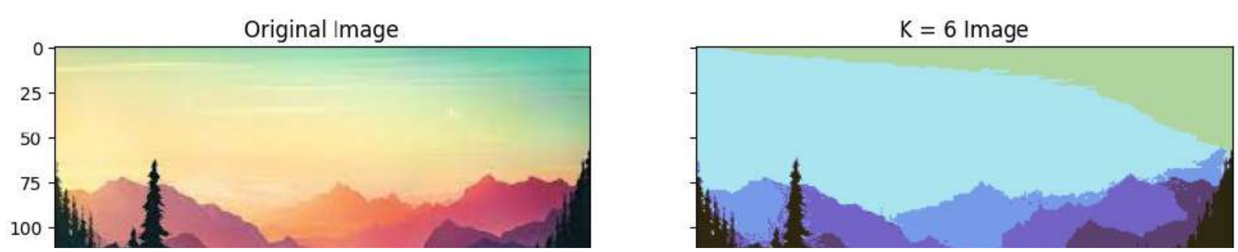
1. Clustering-Based Segmentation Algorithm - K-Means Clustering

```python
# imports
import numpy as np
import cv2 as cv
import matplotlib.pyplot as plt

plt.rcParams["figure.figsize"] = (12, 50)

# load image
img = cv.imread('/content/scenery.jfif')
Z = img.reshape((-1, 3))
# convert to np.float32
Z = np.float32(Z)

# define stopping criteria, number of clusters(K) and apply kmeans()
# TERM_CRITERIA_EPS : stop when the epsilon value is reached
# TERM_CRITERIA_MAX_ITER:  stop when Max iteration is reached
criteria = (cv.TERM_CRITERIA_EPS + cv.TERM_CRITERIA_MAX_ITER, 10, 1.0)

fig, ax = plt.subplots(10, 2, sharey=True)
for i in range(10):
    K = i + 3
    # apply K-means algorithm
    ret, label, center = cv.kmeans(Z, K, None, criteria, 10,
cv.KMEANS_RANDOM_CENTERS)
    # Now convert back into uint8, and make the original image
    center = np.uint8(center)
    res = center[label.flatten()]
    res2 = res.reshape(img.shape)
    # plot the original image and K-means image
    ax[i, 1].imshow(res2)
    ax[i, 1].set_title('K = %s Image' % K)
    ax[i, 0].imshow(cv.cvtColor(img, cv.COLOR_BGR2RGB))
    ax[i, 0].set_title('Original Image')

plt.show()
```

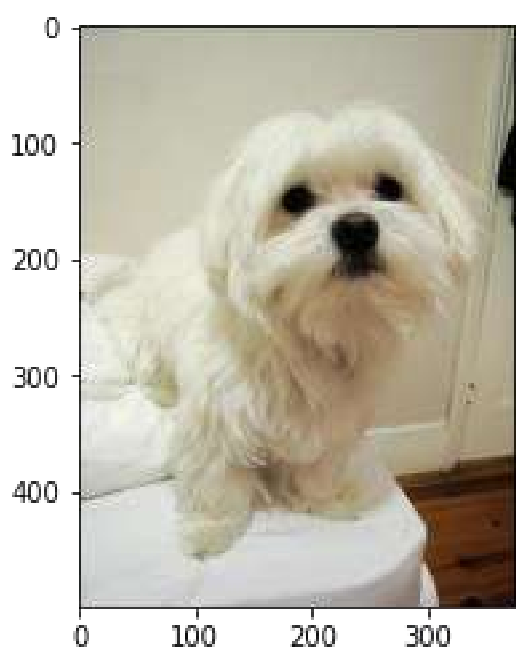

Original Image
K = 3 Image


Original Image
K = 4 Image


Original Image
K = 5 Image


Original Image
K = 6 Image

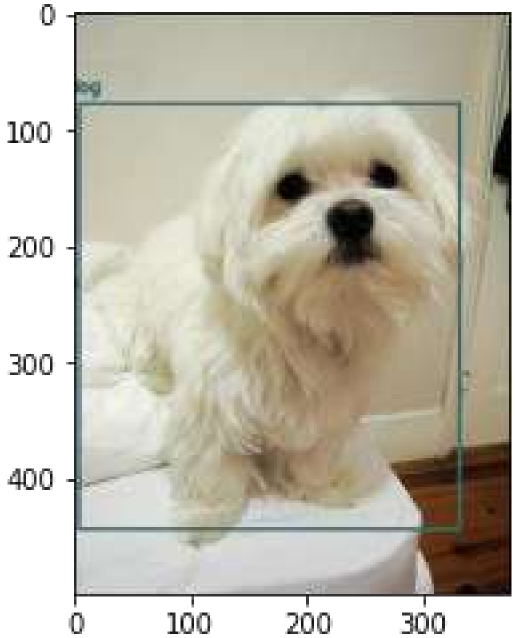
```
import cv2
import argparse
import numpy as np
from matplotlib import pyplot as plt

#following are the paths to files for input, YOLO configuration file,
pre-trained YOLO weights and a text file containing class on which
YOLO had been pre-trained
#the model being used here trained on COCO dataset
#model can detect 80 objects and those objects names are in the text
file below
input_image_path='C:\\Users\\HP\\Desktop\\7th Semester\\ELL715\\
Assignment 3\\Question4\\MalteseDog.JPEG'
config_path='C:\\Users\\HP\\Desktop\\7th Semester\\ELL715\\Assignment
3\\Question4\\yolov3.cfg'
weights_path='C:\\Users\\HP\\Desktop\\7th Semester\\ELL715\\Assignment
3\\Question4\\yolov3.weights'
classes_path='C:\\Users\\HP\\Desktop\\7th Semester\\ELL715\\Assignment
3\\Question4\\yolov3.txt'

#reading the input image from the path given above
image = cv2.imread(input_image_path)

plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
plt.show()
```




```
#obtaining the width and height of the input image
Width = image.shape[1]
```

```
Height = image.shape[0]
scale = 0.00392

#obtaining the classes and adding them to a list
model_classes = None
with open(classes_path, 'r') as f:
    model_classes = [line.strip() for line in f.readlines()]

#generating different colors for all the classes
COLORS = np.random.uniform(0, 255, size=(len(model_classes), 3))

#reading the configuration and weights files to create a network
network = cv2.dnn.readNet(weights_path, config_path)

#preparing the input image to run through the dnn
blob = cv2.dnn.blobFromImage(image, scale, (416,416), (0,0,0), True,
crop=False)

network.setInput(blob)

#proides the names of the multiple output layers of the network
def get_output_layers(net):

    layer_names = network.getLayerNames()

    output_layers = [layer_names[i - 1] for i in
net.getUnconnectedOutLayers()]

    return output_layers

#drawing rectangles over the detected object region
def draw_bounding_box(img, class_id, confidence, x, y, x_plus_w,
y_plus_h):

    label = str(model_classes[class_id])

    color = COLORS[class_id]

    cv2.rectangle(img, (x,y), (x_plus_w,y_plus_h), color, 2)

    cv2.putText(img, label, (x-10,y-10), cv2.FONT_HERSHEY_SIMPLEX,
0.5, color, 2)

#feed forward through the network happens from this line
outs = network.forward(get_output_layers(network))

class_ids = []
confidences = []
boxes = []
conf_threshold = 0.5
nms_threshold = 0.4
```

```python
# for each detetion from each output layer get the confidence, class
id, bounding box params and ignore weak detections (confidence < 0.5)
for out in outs:
    for detection in out:
        scores = detection[5:]
        class_id = np.argmax(scores)
        confidence = scores[class_id]
        if confidence > 0.5:
            center_x = int(detection[0] * Width)
            center_y = int(detection[1] * Height)
            w = int(detection[2] * Width)
            h = int(detection[3] * Height)
            x = center_x - w / 2
            y = center_y - h / 2
            class_ids.append(class_id)
            confidences.append(float(confidence))
            boxes.append([x, y, w, h])

#using non-max suppression to remove boxes which are high overlapping
indices = cv2.dnn.NMSBoxes(boxes, confidences, conf_threshold,
nms_threshold)

for i in indices:
    box = boxes[i]
    x = box[0]
    y = box[1]
    w = box[2]
    h = box[3]

    draw_bounding_box(image, class_ids[i], confidences[i], round(x),
round(y), round(x+w), round(y+h))

#displaying the output image
plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
plt.show()
```


0
100
200
300
400
0
100
200
300