CSE489 Machine Vision

Project #02: Color Image Segmentation

Student Name: Menna Allah Muhammed Farouk

ID: 15P3050

Due date 16-12-2019

Due handed in 16-12-2019

- Abstract

In this project we are going to use several different techniques foe segmentation, NN, KNN, K-means and Bayes Classifier.

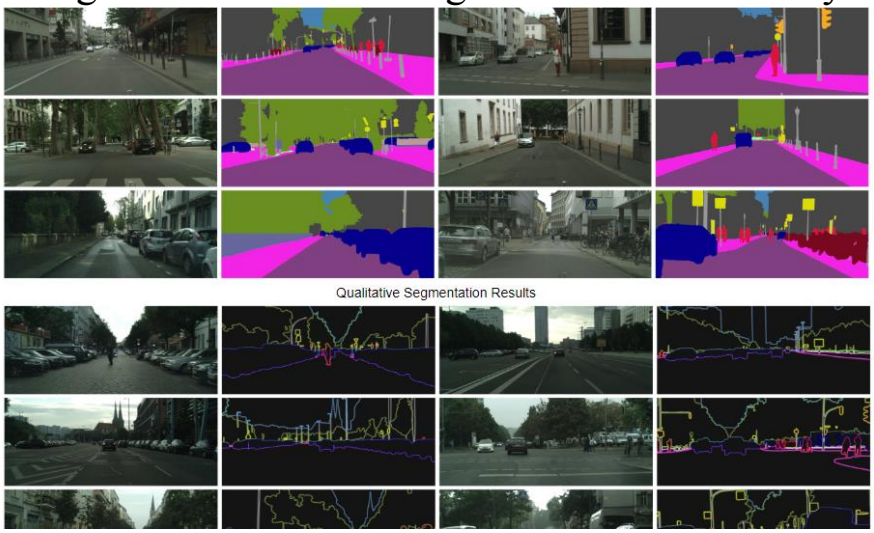
The main goal of this project to observe and compare the output of these techniques by taking samples of object and background manually then classify them by using each technique mentioned above.

How are we going to do this?

Each group is given a set of data points and classify each data point into a specific group. In theory, data points that are in the same group should have similar properties and/or features, while data points in different groups should have highly dissimilar properties and/or features.

• Problem definition and Importance

What is image segmentation? – "A process of partitioning a digital image into multiple segments. The goal of segmentation is to simplify and/or change the representation of an image into something that is more meaningful and easier to analyze."



Why is it important?

Image segmentation is a vital part of image analysis process. It differentiates between the objects we want to inspect further and the other objects or their background.

It can be used in medical applications, for example, to find cancer cells or tumors , it also can be used for detection of pedestrians for automated vehicles, faces for cameras, etc. not just that it also used to count things in images, such as, complete blood count, localization of objects in satellite images (roads, forests, crops, etc.).

What are the methods used in segmentation? – "Several general-purpose algorithms and techniques have been developed for image segmentation."
Examples of methods used in segmentation:

1- KNN 2- K-Means 3-Bayes Classifier 4- SVM

- Methods and algorithms

NN- Algorithm:

1. Take samples from object and background. Then, calculate the mean.

2- Calculate the minimum distance (using Euclidian distance) between the current pixel and the pixels from both object region and background region

3- If the pixel intensity closer to object mean intensity then it will be colored as the mean intensity of the object otherwise it will be colored as the mean intensity of the background.

KNN- Algorithm

1. Take samples from object and background. Then, calculate the mean.

2- Initialize the value of K (number of neighbors that will be taken in consideration)

3- Calculate the minimum distance (using Euclidian distance) between the current pixel and the pixels from both object region and background region. (repeating this step depends on the K value)

4-- If the pixel intensity closer to object mean intensity then it will be colored as the mean intensity of the object otherwise it will be colored as the mean intensity of the background.

K-Means Clustering:

1. Manually select the number of classes/groups to use and randomly initialize their respective center points. To figure out the number of classes to use, it's good to take a quick look at the data and try to identify any distinct groupings. For example,

two groups for an image with an apple (object) and it's background.

2. Randomly initialize the group centers.

3. Each data point is classified by computing the distance between that point and each group center, and then classifying the point to be in the group whose center is closest to it.

4. Based on these classified points, recompute the group center by taking the mean of all the vectors in the group.

5. Repeat these steps for a set number of iterations or until the group centers don't change much between iterations.

Bayes' Classifier:

1. Take samples from object and background. Then, calculate the mean.

2. Assume prior probabilities.

3. Calculate the covariance of both object region and background region.

4.Calculate likelihood (by using MLE rule).

5- If the prior of the object multiplied by the likelihood of object is bigger than the prior of the background multiplied by likelihood of the background.

MLE Rule:

$$\log(p(x)) = -(d/2)\log(2\pi) - (1/2)\log(|\Sigma|) - (1/2)(x-\mu)^t \Sigma^{-1}(x-\mu)$$

SVM Classifier:

1- Initialize training samples and labels for these samples.

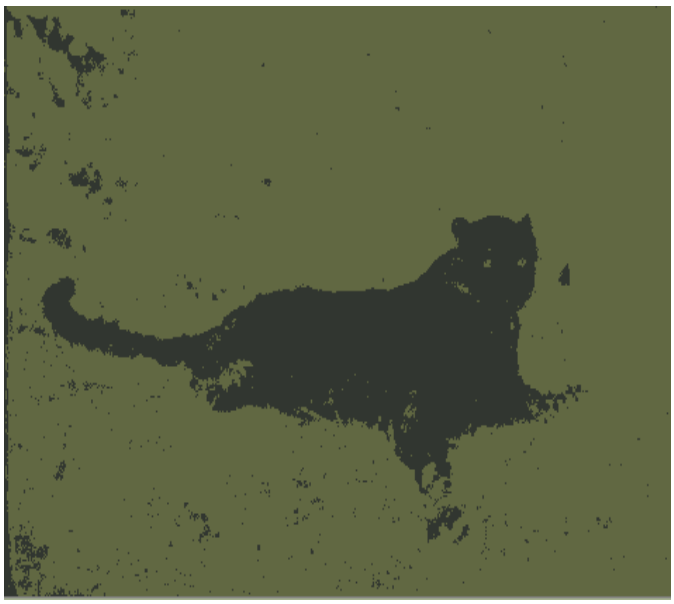
2- Resize the samples to be a 2-D array.

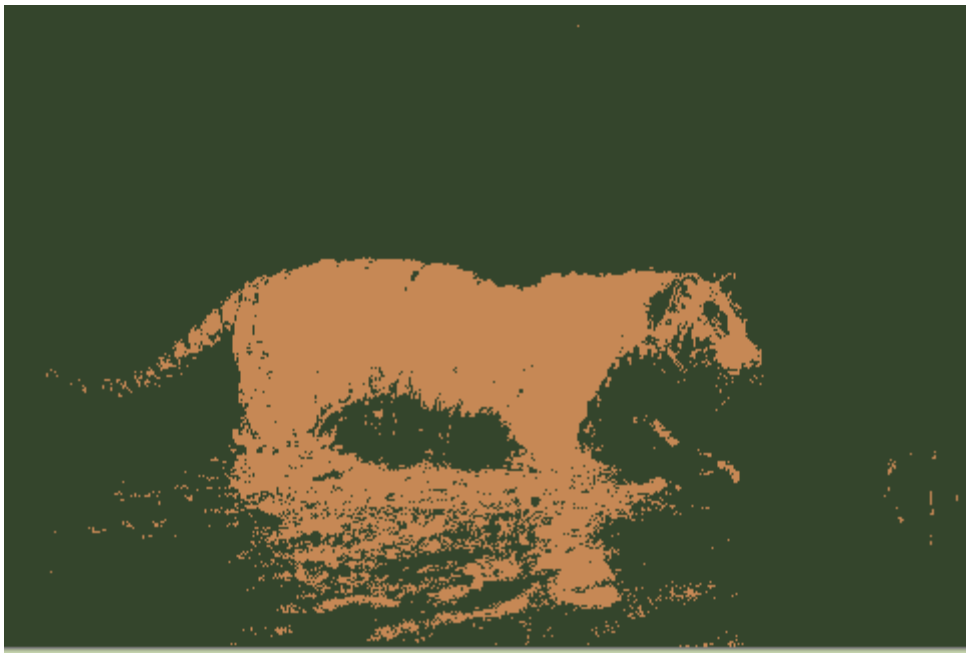
3-Train the classifier by using the training samples (object, background).

4-Predict the values of the rest of the pixels in the image.

- Results:

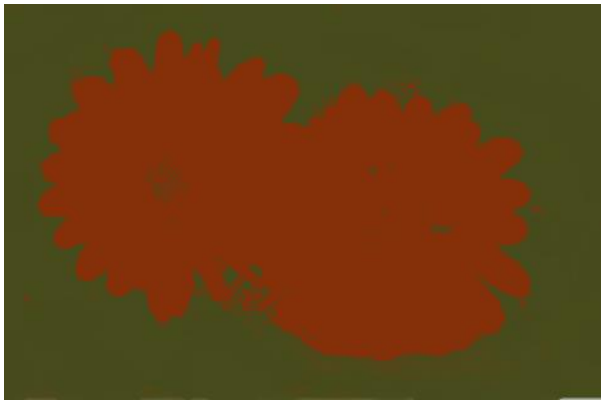
→NN Classifier

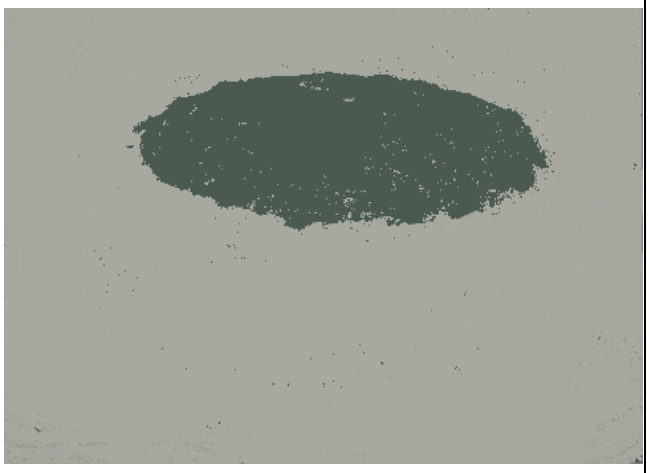


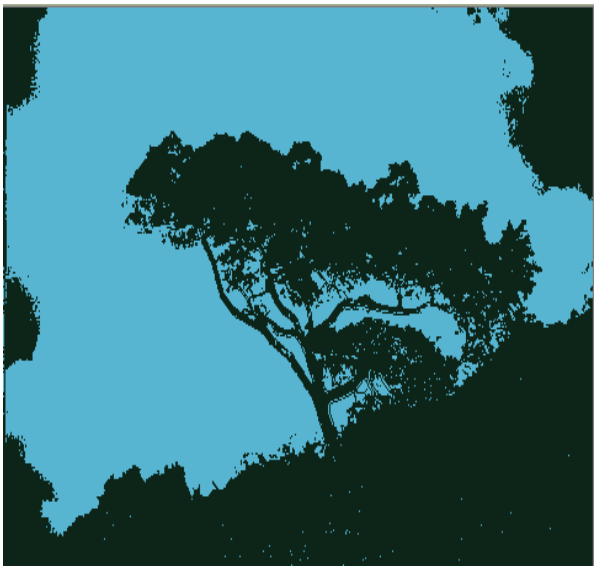


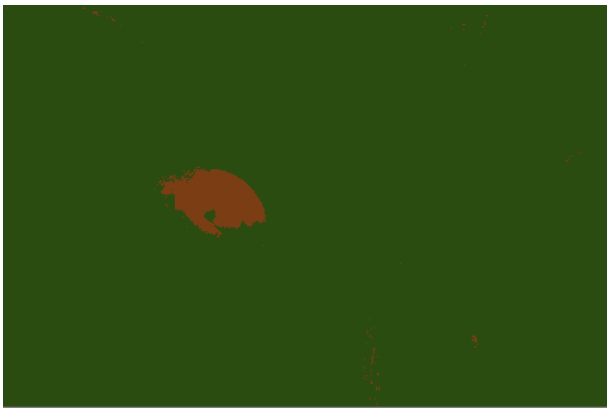
- Results:

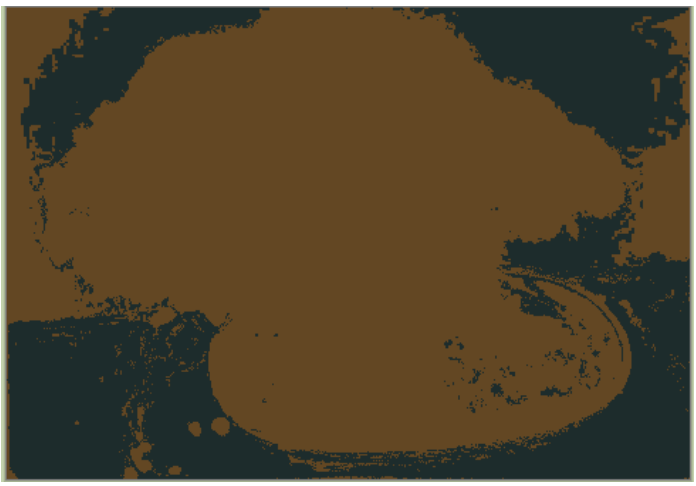
→KNN Classifier





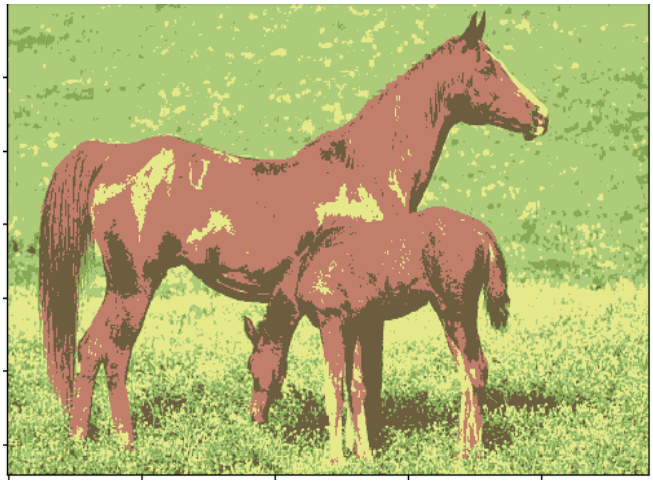


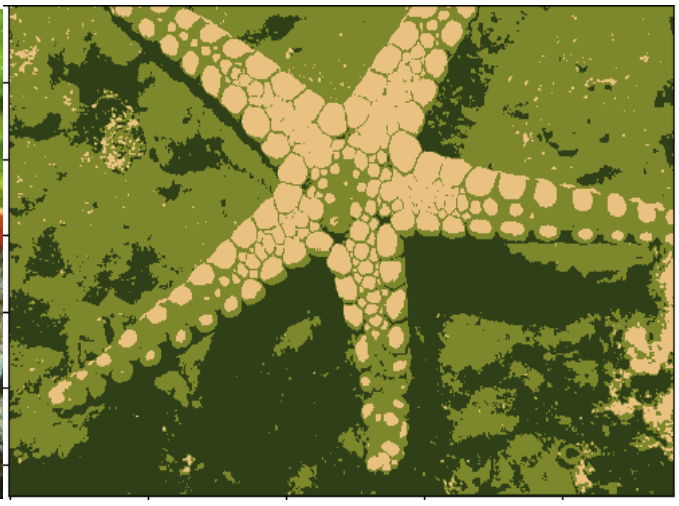


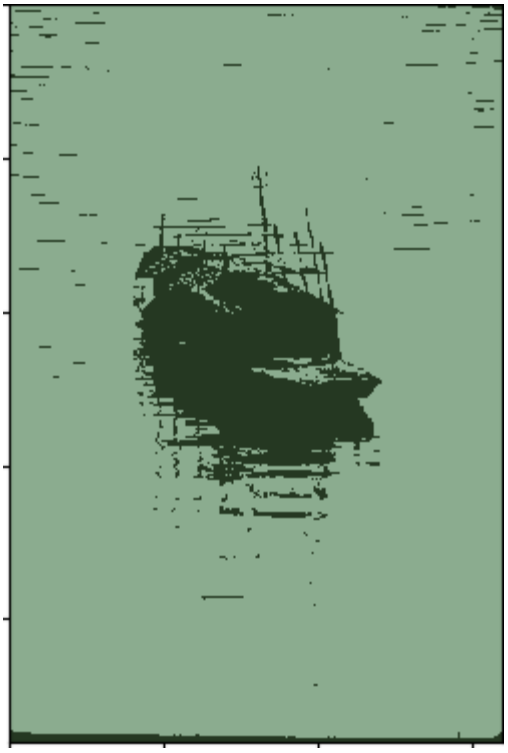


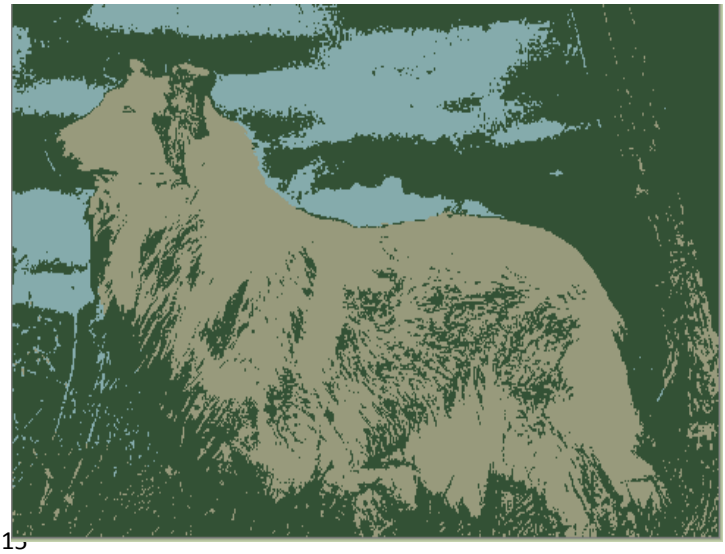
- Results:

→K-mean Clustering



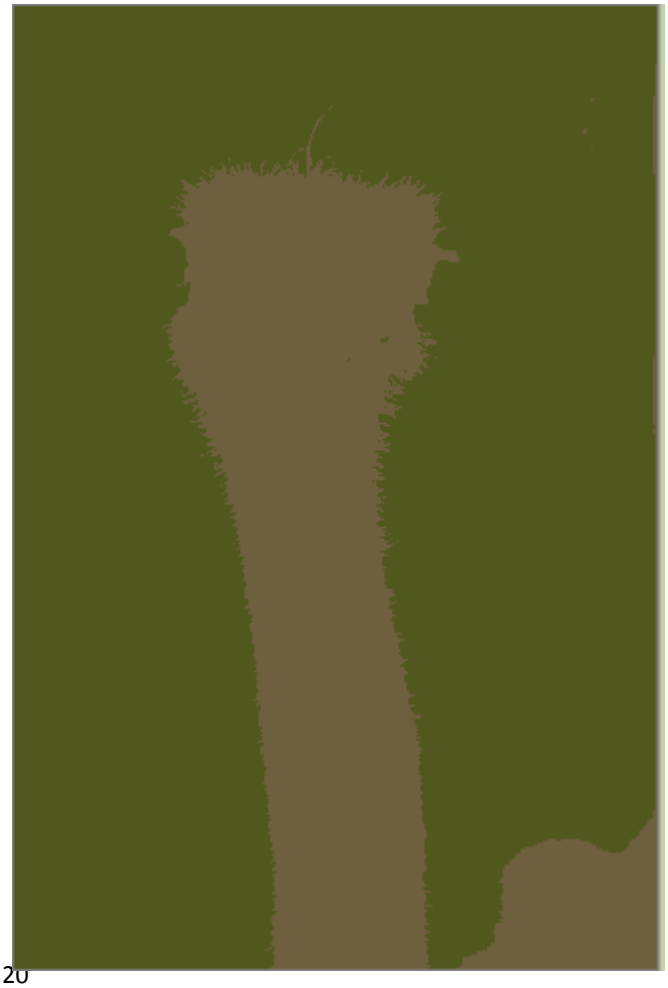






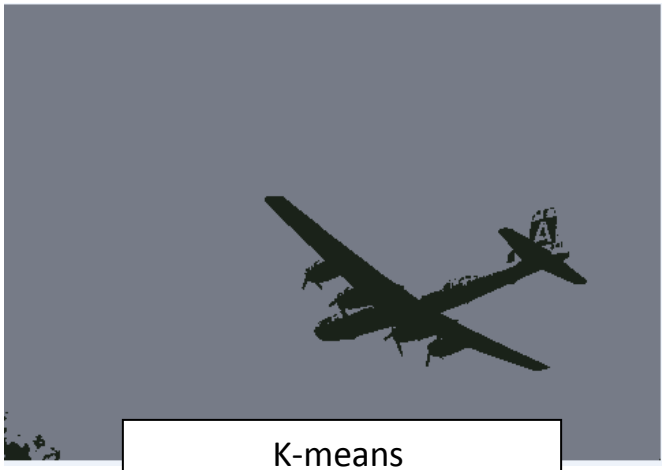
- Results:

→ Bayes Classifier



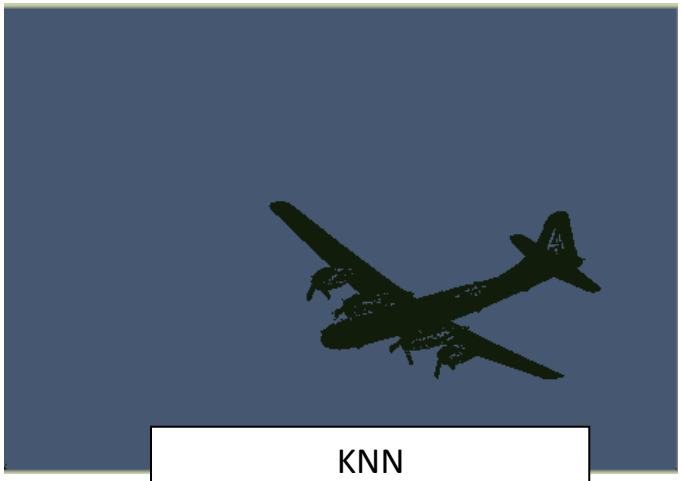
- Results:
  SVM Classifier

Discussion of results:

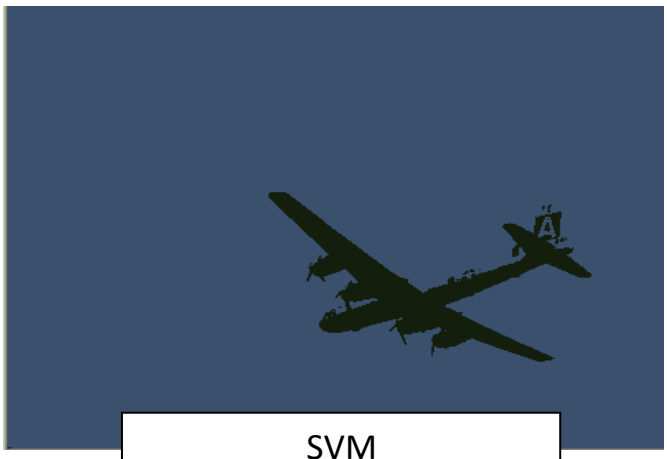


K-means

Bayes Classifier



KNN



SVM

NN

Comment on the results:

It's obvious that the best result was bayes classifier the shape of the airplane is completely detected and there were no noise in the result image, while the second best results were NN and KNN most of the shape of the airplane was detected, while the third results were K-means and SVM.

SVM works really well with easy detected objects images so do NN and KNN.

Bayes classifier works with easy and medium images perfectly but there is sort of noise in hard ones.

Appendix

1- NN- Algorithm

```python
import numpy as np
import cv2
import matplotlib.pyplot as plt


# Pick the image used for NN:

path = r'C:\Users\New-Amr\PyChram Projects\3096.jpg'
img = cv2.imread(path, 1)
cv2.imshow('Original image', img)
h, w, s = img.shape
segmented_img = np.zeros_like(img)
counter = 0
# Tke samples from the original image

(ROI_bg_x, ROI_bg_y, ROI_bg_w, ROI_bg_h) = cv2.selectROI(img, showCrosshair=0)
(ROI_obj_x, ROI_obj_y, ROI_obj_w, ROI_obj_h) = cv2.selectROI(img, showCrosshair=0)
# Calculate mean values for object and background

mean_bg = np.sum(img[ROI_bg_y:ROI_bg_y + ROI_bg_h, ROI_bg_x:ROI_bg_x + ROI_bg_w], axis=(0, 1), keepdims=True) / (
        ROI_bg_h * ROI_bg_w)
mean_obj = np.sum(img[ROI_obj_y:ROI_obj_y + ROI_obj_h, ROI_obj_x:ROI_obj_x + ROI_obj_w], axis=(0, 1),
                  keepdims=True) / (ROI_obj_h * ROI_obj_w)
for m in range(h):
    for n in range(w):
        # Calculate the distances for both object and background

        distance_obj = np.linalg.norm(img[m, n] - img[ROI_obj_y:ROI_obj_y + ROI_obj_h, ROI_obj_x:ROI_obj_x + ROI_obj_w], axis=2)
        distance_bg = np.linalg.norm(img[m, n] - img[ROI_bg_y:ROI_bg_y + ROI_bg_h, ROI_bg_x:ROI_bg_x + ROI_bg_w], axis=2)
```

```python
        # Compare between the 2 distances of object and background and assign values to new image

        if np.min(distance_obj) < np.min(distance_bg):
            segmented_img[m, n] = mean_obj
        else:
            segmented_img[m, n] = mean_bg
        print(counter)
        counter = counter + 1
print(mean_bg)
print(mean_obj)
cv2.imshow('Segmented Image', segmented_img)
cv2.waitKey(0)
```

2- KNN- Algorithm:

```python
import numpy as np
import cv2
# Select the image used for KNN:
path = r'C:\Users\New-Amr\PyChram Projects\35058.jpg'
img = cv2.imread(path, 1)
cv2.imshow('image', img)
h, w, s = img.shape
segmented_img = np.zeros_like(img)
k = 4
obj_vote = 0
bg_vote = 0

# Take samples from original image for both object and background

(ROI_bg_x, ROI_bg_y, ROI_bg_w, ROI_bg_h) = cv2.selectROI(img, showCrosshair=0)
(ROI_obj_x, ROI_obj_y, ROI_obj_w, ROI_obj_h) = cv2.selectROI(img, showCrosshair=0)

# Calculate the mean values for both object and background

mean_bg = np.sum(img[ROI_bg_y:ROI_bg_y + ROI_bg_h, ROI_bg_x:ROI_bg_x + ROI_bg_w], axis=(0, 1), keepdims=True) / (
          ROI_bg_h * ROI_bg_w)
mean_obj = np.sum(img[ROI_obj_y:ROI_obj_y + ROI_obj_h, ROI_obj_x:ROI_obj_x + ROI_obj_w], axis=(0, 1), keepdims=True) / (
          ROI_obj_h * ROI_obj_w)

for m in range(h):
    for n in range(w):
        diff_obj = img[m, n] - img[ROI_obj_y:ROI_obj_y + ROI_obj_h, ROI_obj_x:ROI_obj_x + ROI_obj_w]
        distance_obj = np.linalg.norm(diff_obj, axis=2)
        diff_bg = img[m, n] - img[ROI_bg_y:ROI_bg_y + ROI_bg_h, ROI_bg_x:ROI_bg_x + ROI_bg_w]
        distance_bg = np.linalg.norm(diff_bg, axis=2)
        distance = [np.min(distance_obj), np.min(distance_bg)]
```

```python
        # Based on the value of K the loop is repeated

        for i in range(k):
            # Calculate the distance
            if min(distance) == np.min(distance_obj):
                index = np.where(distance_obj == np.min(distance_obj))
                distance_obj[index[0], index[1]] = 1000
                obj_vote = obj_vote + 1
            elif min(distance) == np.min(distance_bg):
                index = np.where(distance_bg == np.min(distance_bg))
                distance_bg[index[0], index[1]] = 1000
                bg_vote = bg_vote + 1
# Compare between votes:
        vote = [obj_vote, bg_vote]
        if max(vote) == obj_vote:
            segmented_img[m, n] = mean_obj
        elif max(vote) == bg_vote:
            segmented_img[m, n] = mean_bg
        else:
            segmented_img[m, n] = segmented_img[m, n - 1]
        # Replace the min distance with larger one so it won't be considered again
        min_distance_bg = 1000
        min_distance_obj = 1000
        # Clear votes every time the pixel is added to either object or background
        obj_vote = 0
        bg_vote = 0

cv2.imshow('Segmented Image', segmented_img)
cv2.waitKey(0)
```

3- K-Means Clustering

```python
import matplotlib.pyplot as plt
from PIL import Image, ImageStat
import numpy as np


def converged(centroids, old_centroids):
    if len(old_centroids) == 0:
        return False

    if len(centroids) <= 5:
        a = 1
    elif len(centroids) <= 10:
        a = 2
    else:
        a = 4

    for i in range(0, len(centroids)):
        cent = centroids[i]
        old_cent = old_centroids[i]

        if ((int(old_cent[0]) - a) <= cent[0] <= (int(old_cent[0]) + a)) and ((int(old_cent[1]) - a) <= cent[1] <= (int(old_cent[1]) + a)) and ((int(old_cent[2]) - a) <= cent[2] <= (int(old_cent[2]) + a
            continue
        else:
            return False

    return True


def Min_distance(pixel, centroids):
    minDist = 9999
    minIndex = 0
```

```python
    for i in range(0, len(centroids)):
        d = np.sqrt(int((centroids[i][0] - pixel[0])) ** 2 + int((centroids[i][1] - pixel[1])) ** 2 + int(
            (centroids[i][2] - pixel[2])) ** 2)
        if d < minDist:
            minDist = d
            minIndex = i

    return minIndex


def Assign_Pixels(centroids):
    clusters = {}

    for x in range(0, img_width):
        for y in range(0, img_height):
            p = px[x, y]
            minIndex = Min_distance(px[x, y], centroids)

            try:
                clusters[minIndex].append(p)
            except KeyError:
                clusters[minIndex] = [p]

    return clusters


def Adjust_Centroids(centroids, clusters):
    new_centroids = []
    keys = sorted(clusters.keys())
    # print(keys)

    for k in keys:
        n = np.mean(clusters[k], axis=0)
```

```
        new = (int(n[0]), int(n[1]), int(n[2]))
        print(str(k) + ": " + str(new))
        new_centroids.append(new)

    return new_centroids



def K_means(K):
    centroids = []
    old_centroids = []
    rgb_range = ImageStat.Stat(im).extrema
    i = 1

    # Initializes someK number of centroids for the clustering
    for j in range(0, K):
        cent = px[np.random.randint(0, img_width), np.random.randint(0, img_height)]
        centroids.append(cent)

    print("Centroids Initialized. Starting Assignments")
    print("==========================================")

    while not converged(centroids, old_centroids) and i <= 20:
        print("Iteration #" + str(i))
        i += 1

        old_centroids = centroids  # Make the current centroids into the old centroids
        clusters = Assign_Pixels(centroids)  # Assign each pixel in the image to their respective centroids
        centroids = Adjust_Centroids(old_centroids,  clusters)  # Adjust the centroids to the center of their assigned pixels
    return centroids
```

```
def result_image(result):
    img = Image.new('RGB', (img_width, img_height), "white")
    p = img.load()

    for x in range(img.size[0]):
        for y in range(img.size[1]):
            RGB_value = result[Min_distance(px[x, y], result)]
            p[x, y] = RGB_value
    plt.imshow(img)
    plt.show()
    img.show()


k_input = int(input("Enter K value: "))

im = Image.open('16077.jpg')
img_width, img_height = im.size
px = im.load()

result = K_means(k_input)
result_image(result)
```

4- Bayes Classifier

```python
import numpy as np
import cv2
import math

# Pick the image used for Bayes Classifier:
path = r'C:\Users\New-Amr\PyChram Projects\41033.jpg'
img = cv2.imread(path, 1)
cv2.imshow('Original Image', img)
h, w, s = img.shape

# Get THREE samples for the image --> one for object, and two for the background to ensure the result

(ROI_bg1_x, ROI_bg1_y, ROI_bg1_w, ROI_bg1_h) = cv2.selectROI(img, showCrosshair=0)
(ROI_obj_x, ROI_obj_y, ROI_obj_w, ROI_obj_h) = cv2.selectROI(img, showCrosshair=0)

mean_bg1 = np.sum(img[ROI_bg1_y:ROI_bg1_y + ROI_bg1_h, ROI_bg1_x:ROI_bg1_x + ROI_bg1_w], axis=(0, 1),
                  keepdims=True) / (ROI_bg1_h * ROI_bg1_w)
mean_obj = np.sum(img[ROI_obj_y:ROI_obj_y + ROI_obj_h, ROI_obj_x:ROI_obj_x + ROI_obj_w], axis=(0, 1),
                  keepdims=True) / (ROI_obj_h * ROI_obj_w)

# Calculate the covariance of objet

P_obj_matrix = np.array(img[ROI_obj_y:ROI_obj_y + ROI_obj_h, ROI_obj_x:ROI_obj_x + ROI_obj_w])
sum_obj = 0
k = P_obj_matrix[0, 0, :]

for x in range(P_obj_matrix.shape[0]):
    for y in range(P_obj_matrix.shape[1]):
        a1 = np.reshape(P_obj_matrix[x, y, :], (3, 1))
        a2 = np.reshape(mean_obj, (3, 1))
        a3 = np.reshape(P_obj_matrix[x, y, :], (1, 3))
        a4 = np.reshape(mean_obj, (1, 3))
```

```
        b1 = (a1 - a2) * (a3 - a4)
        sum_obj = b1 + sum_obj

obj_cov = sum_obj / (P_obj_matrix.shape[0] * P_obj_matrix.shape[1])
print('obj', obj_cov)

##############
# Calculate covariance of background 1

P_bg1_matrix = np.array(img[ROI_bg1_y:ROI_bg1_y + ROI_bg1_h, ROI_bg1_x:ROI_bg1_x + ROI_bg1_w])
sum_bg1 = 0
for x in range(P_bg1_matrix.shape[0]):
    for y in range(P_bg1_matrix.shape[1]):
        a1 = np.reshape(P_bg1_matrix[x, y, :], (3, 1))
        a2 = np.reshape(mean_bg1, (3, 1))
        a3 = np.reshape(P_bg1_matrix[x, y, :], (1, 3))
        a4 = np.reshape(mean_bg1, (1, 3))

        b1 = (a1 - a2) * (a3 - a4)
        sum_bg1 = b1 + sum_bg1

bg1_cov = sum_bg1 / (P_bg1_matrix.shape[0] * P_bg1_matrix.shape[1])
print('bg1', bg1_cov)
##################

##############
# Calculate likehood

counter = 0
segmented_img = np.zeros_like(img)
P_O = np.zeros_like(img)
P_bg1 = np.zeros_like(img)
```

```
for l in range(img.shape[0]):
    for t in range(img.shape[1]):
        a1 = np.reshape(img[l, t, :], (3, 1))
        a2 = np.reshape(img[l, t, :], (1, 3))
        b1 = np.reshape(mean_obj, (3, 1))
        b2 = np.reshape(mean_obj, (1, 3))
        c = a1 - b1
        c1 = a2 - b2
        c111 = np.linalg.inv(obj_cov)
        c11 = np.dot(c1, c111)
        c1111 = np.dot(c11, c)
        print(counter)
        ################
        a11 = np.reshape(img[l, t, :], (3, 1))
        a22 = np.reshape(img[l, t, :], (1, 3))
        b11 = np.reshape(mean_bg1, (3, 1))
        b22 = np.reshape(mean_bg1, (1, 3))
        d = a11 - b11
        d1 = a22 - b22
        d111 = np.linalg.inv(bg1_cov)
        d11 = np.dot(d1, d111)
        d1111 = np.dot(d11, d)

        ##############################
        counter = counter + 1
        P_O = -3 / 2 * math.log(2 * math.pi) - 0.5 * math.log(abs(np.linalg.det(obj_cov))) - 0.5 * c1111
        P_bg1 = -3 / 2 * math.log(2 * math.pi) - 0.5 * math.log(abs(np.linalg.det(bg1_cov))) - 0.5 * d1111
        if 0.6 * P_O > 0.4 * P_bg1:
            segmented_img[l, t] = mean_obj
        else:
            segmented_img[l, t] = mean_bg1
cv2.imshow('seg', segmented_img)
cv2.waitKey(0)
```

5- SVM Classifier

```
import numpy as np
import cv2
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import classification_report, confusion_matrix

# Pick the image used for SVM:
path = r'C:\Users\New-Amr\PyChram Projects\271008.jpg'
img = cv2.imread(path, 1)
cv2.imshow('Original Image', img)
h, w, s = img.shape

# Get THREE samples for the image --> one for object, and one for the background to ensure the result

(ROI_bg1_x, ROI_bg1_y, ROI_bg1_w, ROI_bg1_h) = cv2.selectROI(img, showCrosshair=0)
(ROI_obj_x, ROI_obj_y, ROI_obj_w, ROI_obj_h) = cv2.selectROI(img, showCrosshair=0)

mean_bg1 = np.sum(img[ROI_bg1_y:ROI_bg1_y + ROI_bg1_h, ROI_bg1_x:ROI_bg1_x + ROI_bg1_w], axis=(0, 1),
                  keepdims=True) / (ROI_bg1_h * ROI_bg1_w)
mean_obj = np.sum(img[ROI_obj_y:ROI_obj_y + ROI_obj_h, ROI_obj_x:ROI_obj_x + ROI_obj_w], axis=(0, 1),
                  keepdims=True) / (ROI_obj_h * ROI_obj_w)

# Create two arrays for object and background

bg = np.array(img[ROI_bg1_y:ROI_bg1_y + ROI_bg1_h, ROI_bg1_x:ROI_bg1_x + ROI_bg1_w])
obj = np.array(img[ROI_obj_y:ROI_obj_y + ROI_obj_h, ROI_obj_x:ROI_obj_x + ROI_obj_w])

# flatten both the object and background

bg1 = bg.reshape(-1, bg.shape[-1])
obj1 = obj.reshape(-1, bg.shape[-1])
```

```
# Create  labels array for training data

y = np.array(np.zeros((obj1.shape[0] + bg1.shape[0], 1)))

# assign values to object and background

for i in range(obj1.shape[0]):
    y[i, :] = 1

# Add the 2 arrays of object and background to X

X = np.concatenate([bg1, obj1])

# Training my data

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20)

# Choose type of classifier

svclassifier = SVC(kernel='linear', C=1.0)
svclassifier.fit(X, y.ravel())
counter = 0
new_img = np.zeros_like(img)

# Assign mean intensity to pixel to classify them:

for r in range(img.shape[0]):
    for c in range(img.shape[1]):
        y_pred = svclassifier.predict([img[r, c, :]])
        counter = counter + 1
        print(counter)
        print('yx', y_pred)
        if y_pred == 0:
            new_img[r, c, :] = mean_obj
        else:
            new_img[r, c, :] = mean_bg1

cv2.imshow('Segmented Image', new_img)
cv2.waitKey(0)
```