Machine Vision

CSE 489

Project 1

Submitted by:

Mohamed Abd El Hamed Mostafa - 16p8173
George Medhat Halim - 16p8197
Essam Eldin Amr - 16p8198

Date 19/12/2020
Senior 2
Mechatronics

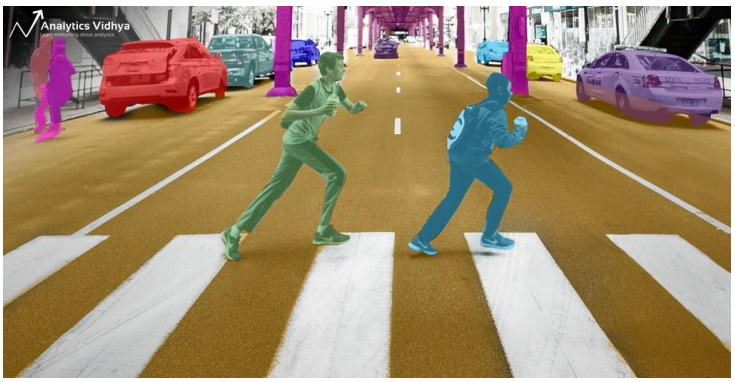
Problem definition:

The problem we are trying to solve in this project is to segment colored images using 5 different methods. Color images segmentation is to divide an image into n number of regions of similar content to separate each object in an image and color them with a different color.

This problem requires sophisticated algorithms to allow the computer to differentiate between the image objects based on the image features which in our case are the color intensities of each pixel, other image features are its histogram or the SIFT Feature extraction algorithm.

Importance of Image Segmentation:

Colored image segmentation is a very important tool that has many applications, like Object detection, Machine vision,
Medical imaging, Traffic control,
Video surveillance.

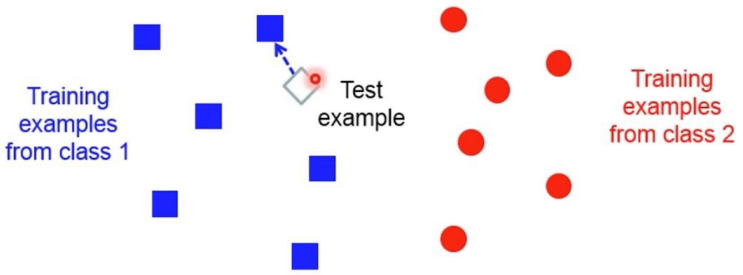


All those applications need image segmentation first to run as image segmentation outputs the exact position of all pixels of an object thus making it easier to the neural networks to learn the exact object rather than other background noise.

Methods and Algorithms:

Nearest Neighbor

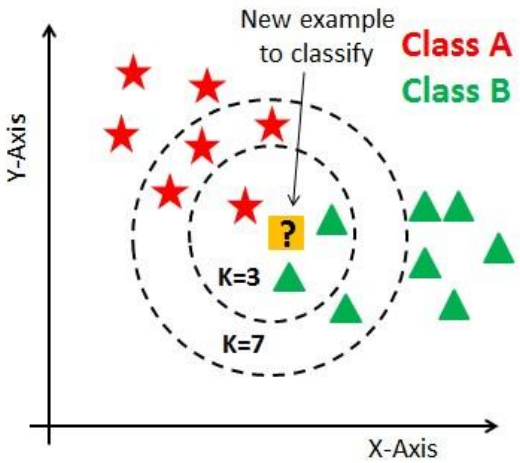
The first method implemented was the NN (nearest neighbor) segmentation this method requires a sample of the object and its background as an input and the algorithm loops through all the pixels of the image and calculates the Euclidean distance from that pixel color intensity to the mean of the both input samples and the sample with the least distance is label this pixel according to that sample category. Then we calculate the mean of each segment from the original image and update the label with the mean.



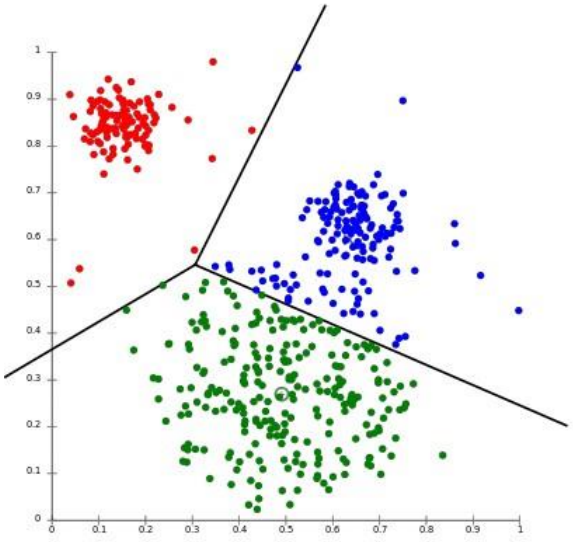
$$d(\mathbf{p},\mathbf{q}) = \sqrt{\sum_{i=1}^{n}(q_i - p_i)^2}$$

K- Nearest Neighbor

The second method is the K-NN which is exactly like the NN but K is the number of distances taken into consideration when decision which category is this pixel are going to be categorized. The most category with least distances is the correct category for this pixel.



C-Means



The third methods is the C Means which requires the generation of random color intensities for each segment we want to segment, 3 values are generated randomly for each cluster which represent the RGB colors, then the color intensities of each pixel is compared with the cluster's generated values, the least distance is classified to that region and after all the pixels are classified the mean for each cluster is calculated and this process is made again till the algorithm converges which is done by calculating the difference between the old mean and the new mean is less than epsilon which is defined at the start.

Bayes Classifier Algorithm

The fourth method is the Bayes Classifier Algorithm which calculates the posterior probability of each class for each pixel and they are compared and the pixel is classified to the class with higher probability. The probability

$$p(\mathbf{x}) = \frac{1}{(2\pi)^{d/2}|\boldsymbol{\Sigma}|^{1/2}} \exp\left[-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^t \boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\right]$$

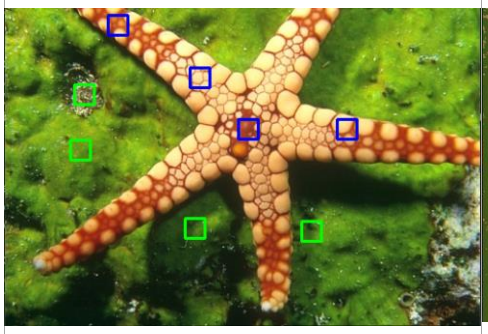
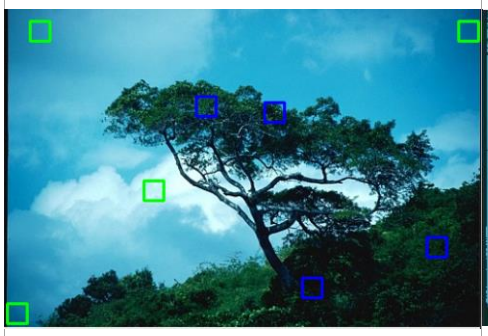
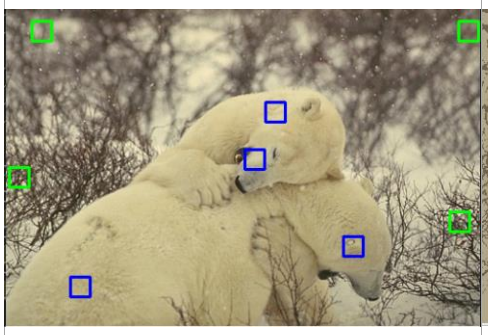
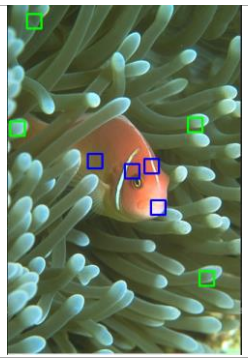
$$\boldsymbol{\mu} = \frac{1}{n}\sum_{k=1}^{n}\mathbf{x}_k \qquad \boldsymbol{\Sigma} = \frac{1}{n}\sum_{k=1}^{n}(\mathbf{x}_k-\boldsymbol{\mu})(\mathbf{x}_k-\boldsymbol{\mu})^t$$

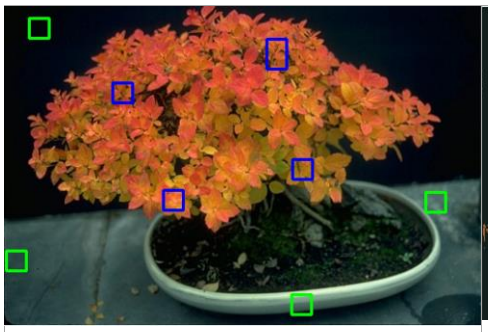
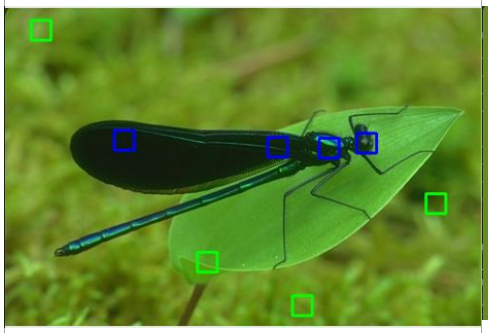
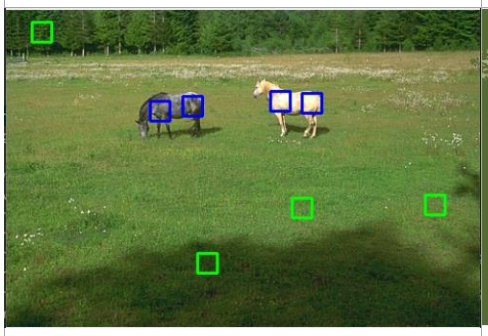
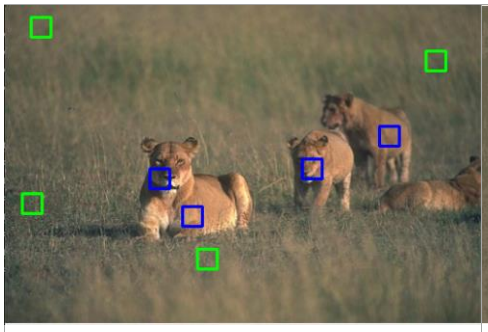
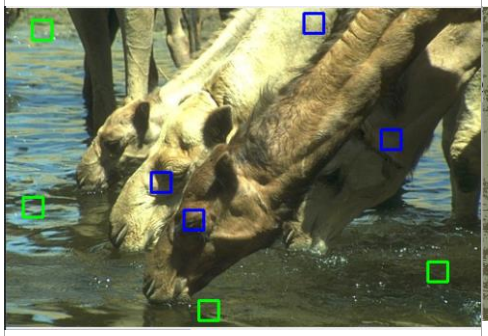
But we use the log of the posterior function to decrease the effect of the exponential as it becomes very large and the computers data types can't contain all of it.

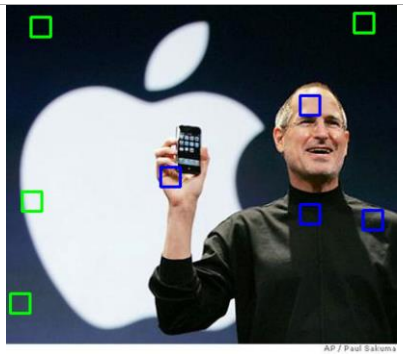
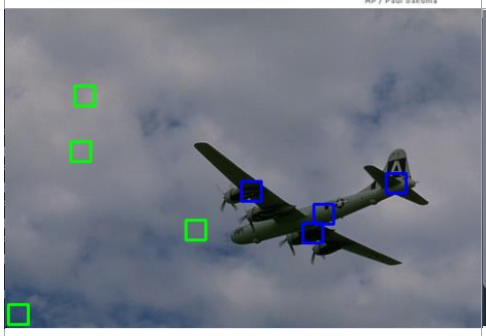
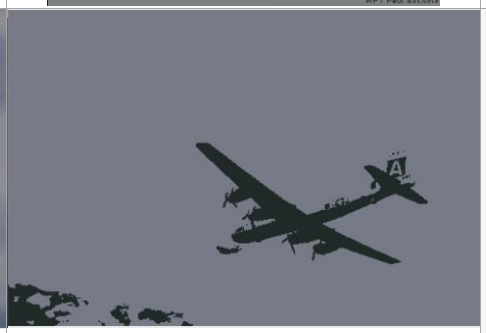
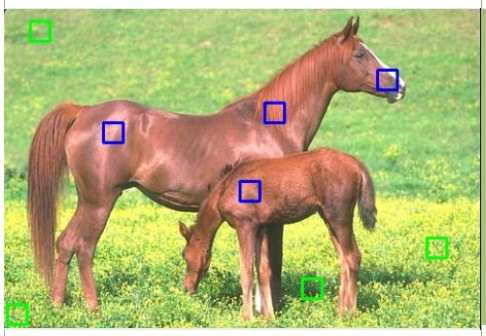
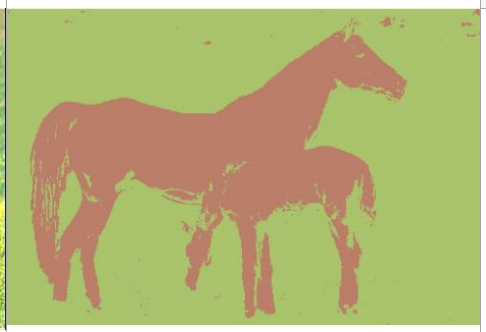
Linear Support Vector Machine SVM

The fifth and last algorithm is the linear support vector machine SVM which produce a line two divide the two dataset into two classes and the input image is checked whether it falls on the left or right of the line and belong to which. 50,000 iterations was used for all the images bellow.

Experimental Results and Discussion Nearest Neighbor

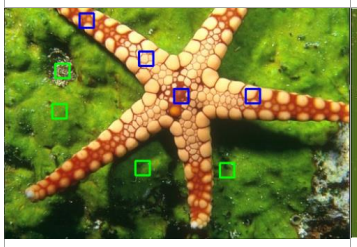
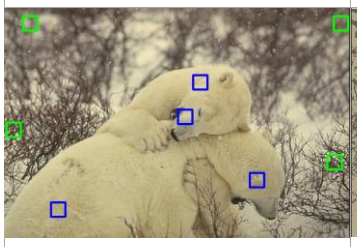
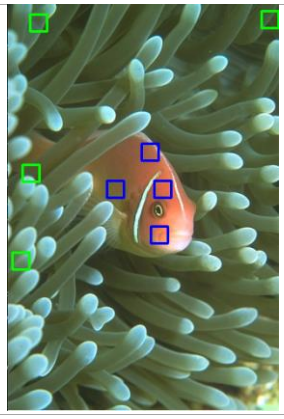
Original Image	Segmented Image	Comment
		The results are quite good as the background is mostly homogeneous but the starfish has 2 colors but it was segmented fully correctly.
		The segmentation of this image is very good due to how homogeneous both the background and the object are.
		This image is quite hard because the color variation between the object and the background is very low thus the segmentation is quite bad.
		On this image only the tips of the coral reefs are identified as background due to the similarity of the back body of the fish with the lower part of the coral reef.

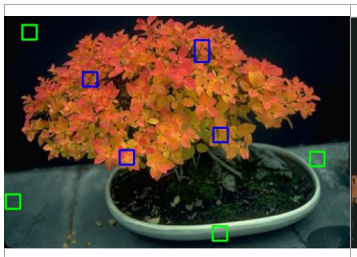
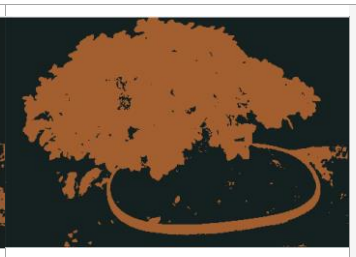
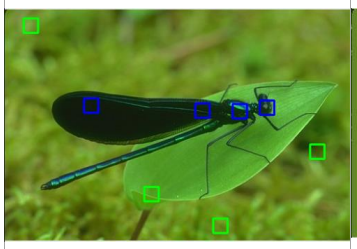
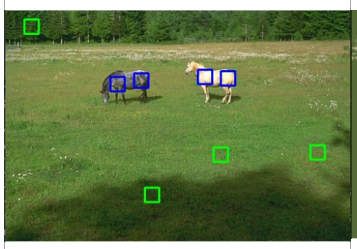
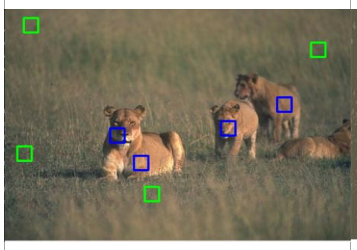
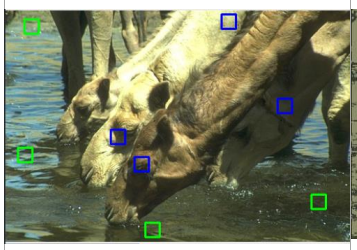
		Most of the object is classified correctly but some of the background is also classified as object
		The segmentation of this image is nearly perfect all the background is segmented correctly and only small parts of the insect is segmented wrongly
		This image is somewhat hard due to the variation in the colors of the object horses but the segmentation is fairly acceptable
		The segmentation of this image is not unusable at all due to the similarity in color for both the object and the background
		This image is quite hard because the objects each has a different color and some of them have the same color as the background water.

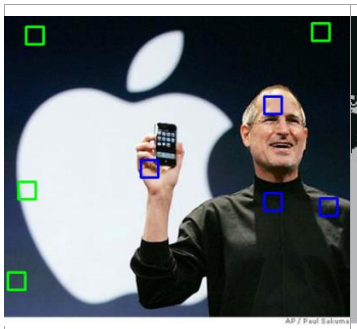
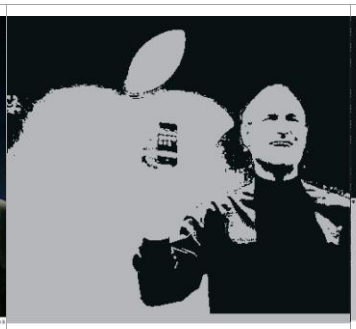
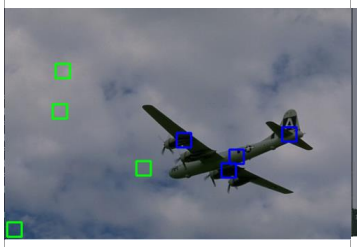
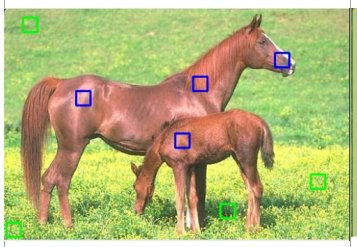
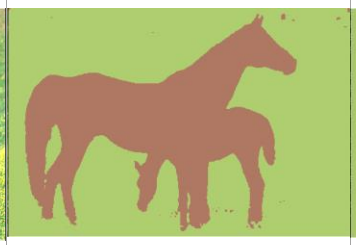
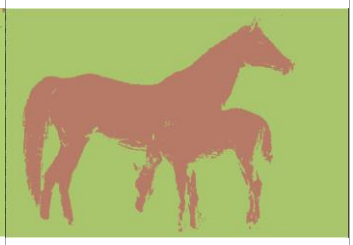
		The segmentation of this image is not the best because the the closeness of the intensities of the object and the background.
		The top of the plane was segmented incorrectly and the corner of the image.
		This image was quite easy as both objects have similar colors and the background is homogenous.

4 samples was taken for the objects and other 4 was taken for the background, the mean was calculated for each one and the Euclidean distance is measured from each pixel to the mean of the samples and the closest to the pixel is the category of that pixel and at the end of the image the cluster's mean color is calculated and replace the variation in each class.

K-Nearest Neighbor

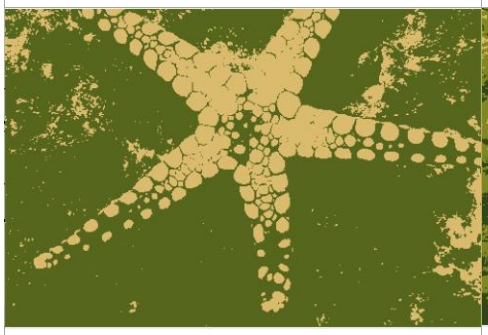
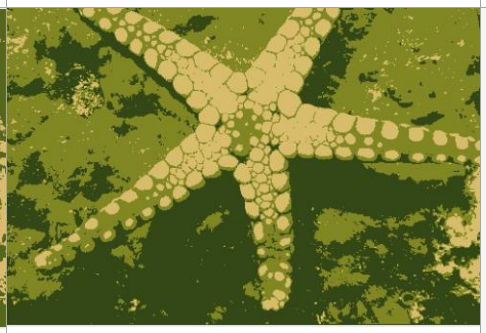
Original Image	Segmented Image K = 3	Segmented Image K = 5	Comment
			The result of K=3 is better than the NN but not by much While the K=5 is worse than K=3
			Both the NN and KNN are perfect for this image. And both Ks are very similar
			In the KNN algorithm the segmentation is also not good. And both Ks are very similar
			The KNN is marginally better than the NN but its not perfect as all the tips of the coral reefs are segmented as the object and no difference between k=3 and k=5

			The KNN is actually worse than the NN for this image, probably due to the use of more than 1 sample to classify each pixel. But the K=5 is better.
			The object it self was segmented better in the KNN but some of the background is segmented incorrectly now.The K=5 is slightly worse
			No visible difference was noted between the KNN and the NN algorithms for this image but the K=5 is slightly worse.
			The KNN is slightly better but no where near what was suppose to be segmented.
			This image is quite hard and no improvement was noted. The k=5 is worse

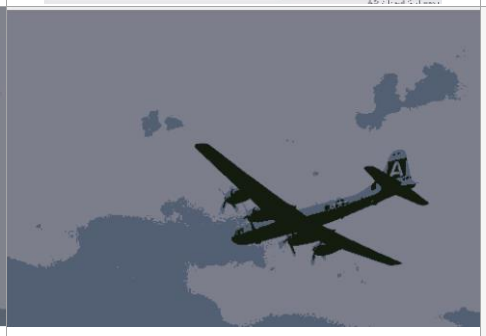
			The segmentation in this image is logical but not correct. The K=5 is slightly better.
			No difference was noted on this image.
			An improvement was noted and its nearly segmented 99% correctly but the k=5 is slightly worse about 95%

The KNN is the same as the NN but the only difference is that each pixel is segmented based on K number of closest samples. the K value for those images was 3 and 5. Some images was better with the k = 5 while others are worse so it’s better to calculate both and compare for each specific image.

C Means

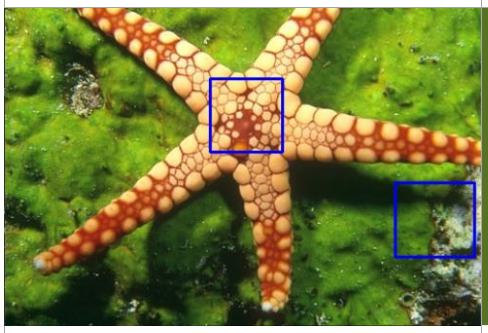
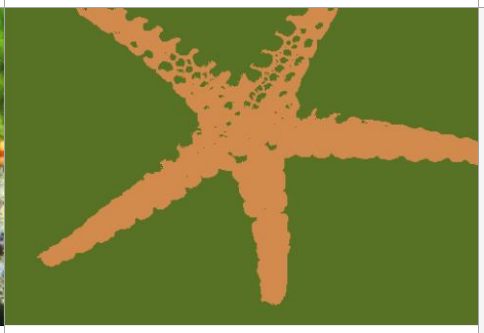
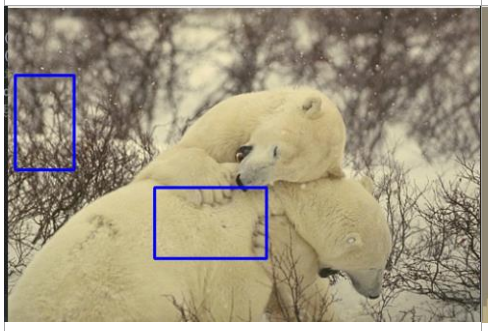
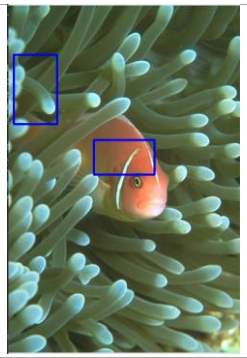
2 clusters	3 clusters	Comment
		The results are worse than both the KNN and the NN
		The results are very good in the C Means algorithm
		In the 2 clusters there is no visible difference between it and the KNN and the NN while the 3 clusters is not that better but it has much more details
		The 2 cluster segmentation is cleaner than KNN and NN but not accurate while the 3 clusters is not that better as the object was not separated correctly.

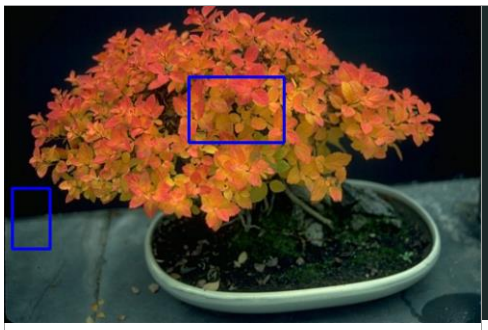
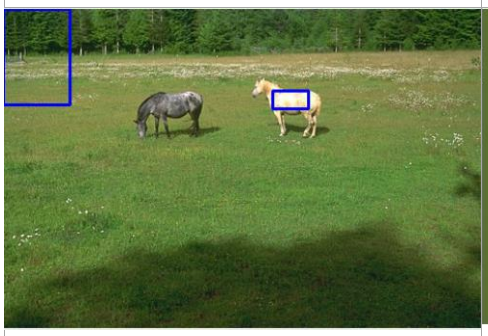
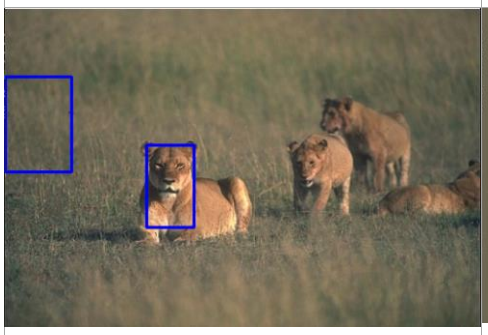
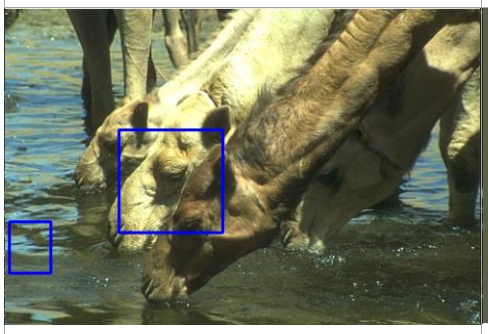
		In the two clusters the output is better than the NN and the 3 clusters are the best for this image because it separated the leafs and the pot but some of the table is segmented as pot
		In the 2 clusters the output is worse than the KNN and the NN due to the randomness of the initialization while the 3 clusters is slightly better for the object.
		The output in the 2 clusters are worst than the KNN while for the 3 clusters only the white horse is segmented correctly.
		In the two clusters the output is better than the KNN and the NN while the 3 clusters are not that much better in isolating the object.
		This image is quite hard and no improvement was noted in the 2 clusters while the 3 clusters is better for the lighter skinned camels

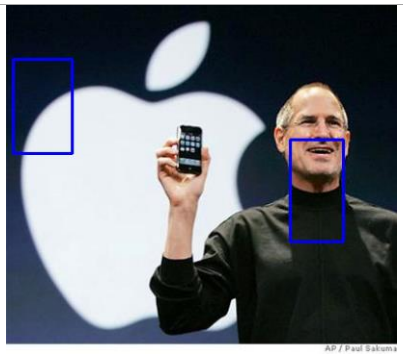
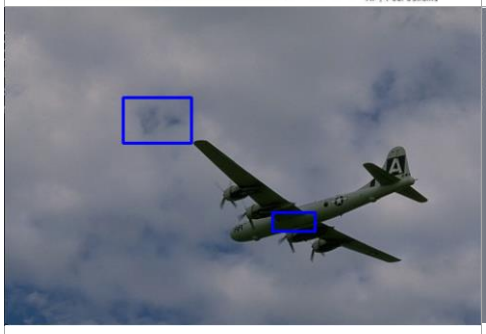
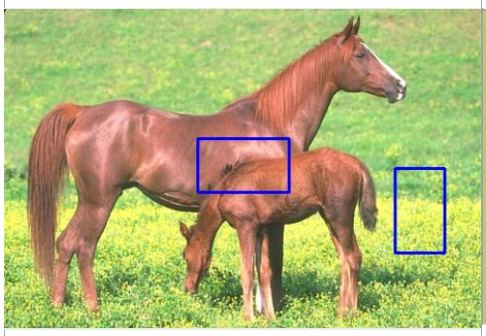
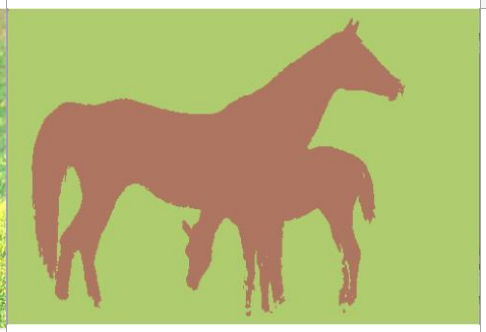
		The Segmentation in the 2 clusters is better than the KNN and NN but its not isolating the object because the similarity in color while in the 3 clusters is better
		The 2 clusters is worse while the 3 clusters is much better than the 2 clusters
		The 2 clusters output is worse than the KNN and NN due to the randomness while the 3 clusters is not that much better.

The C Mean uses C number of clusters which generates at the start random color intensities for each cluster then it computes the Euclidean distance from each pixel on the image to the C clusters and the pixel with the least distance to a cluster then this pixel is labeled to that cluster, then the mean for each cluster is calculated then we iterate again on all pixels with the new mean till the mean of the difference between current iteration minus one and the mean of the current iteration is less than the predefined constant Epsilon, the epsilon used for those images was 0.1. As we can see some images was segmented better than the KNN and NN while other were segmented worse. Due to the randomness of the initialization of the cluster intensities at the beginning each time an image is segmented it produces a slightly different segmentation and sometimes especially in the 3 cluster mode one of the cluster becomes zero and will not show up in the final image.

Bayes Algorithm

Original Image	Segmented Image	Comment
		The Bayes Algorithm shows the best results for this image
		The segmentation of this image is 100% correct
		This image is quite hard but the Bayes algorithm segmented it to some extend perfectly
		This image was also segmented about 98% correctly

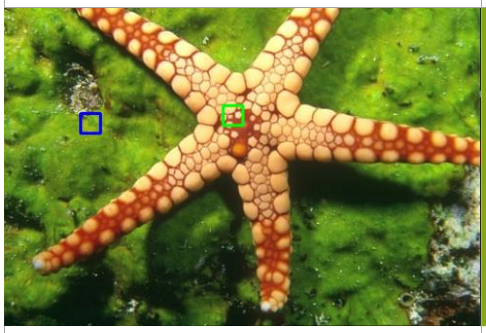
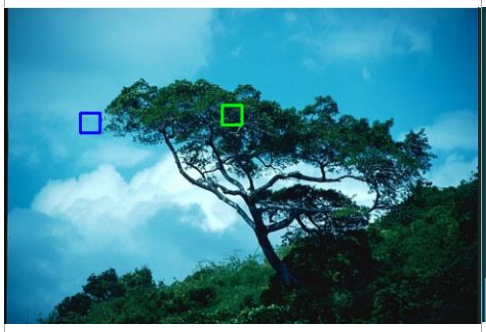
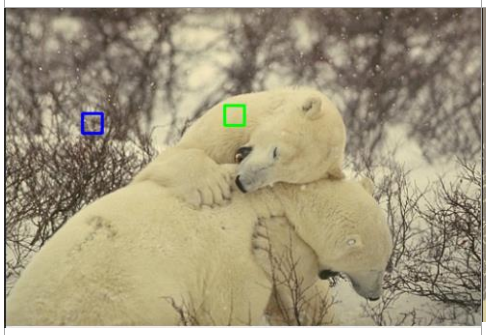
		This Bayes algorithm segmented this image the best among all other methods
		In the 2 clusters the output is worse than the KNN and the NN due to the randomness of the initialization while the 3 clusters is slightly better for the object.
		Given the sampled input the output is good but the other horse was note segmented as shown.
		This Bayes algorithm segmented this image the best among all other methods
		This image is quite hard and no improvement was noted in the 2 clusters while the 3 clusters is better for the lighter skinned camels

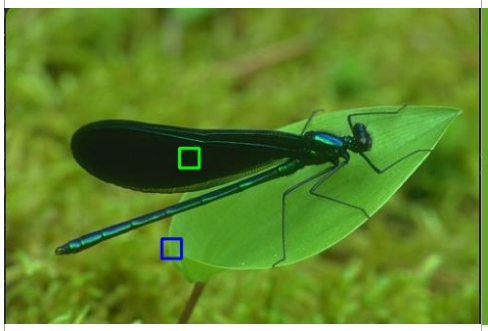
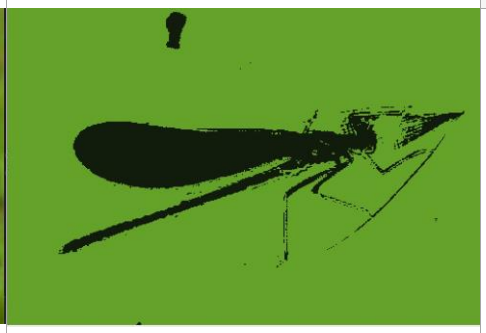
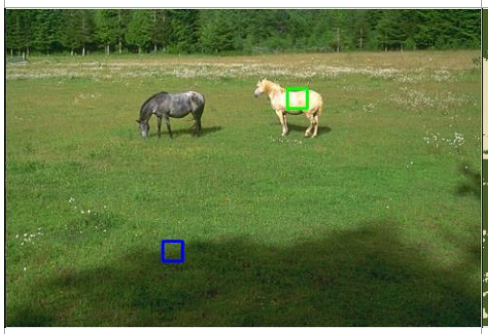
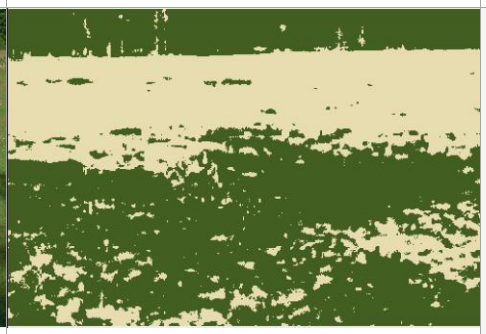
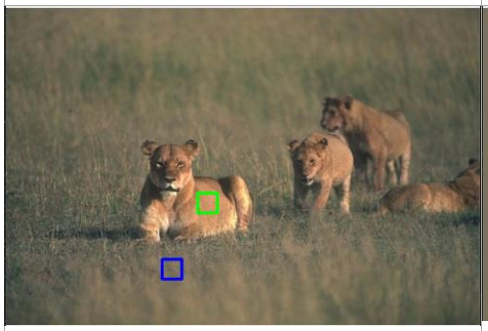
		While the segmentation is not 100 percent but its the best among all other methods.
		About 100% segmentation
		About 100% segmentation

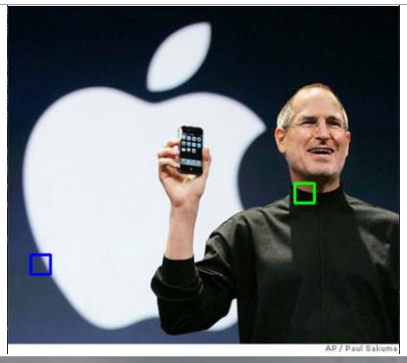
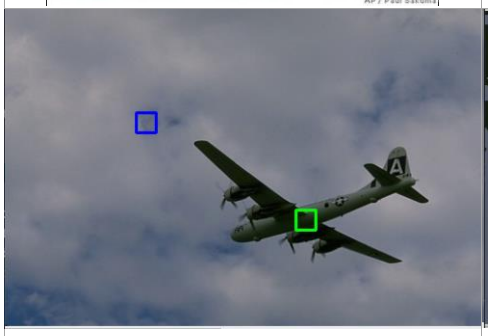
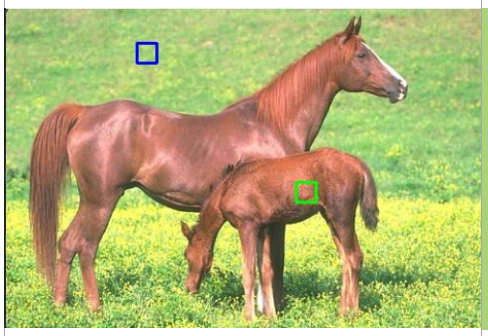
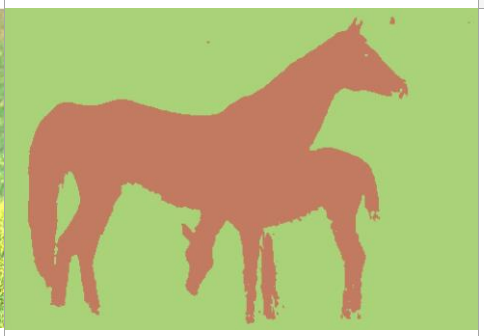
The Bayes Algorithm works by having a sampled input form the background and the object and it calculates the mean and covariance of each sample and calculates the posterior probability function and with the Bayes rule the category with higher probability to a given pixel is the selected category for that pixel, the P1 was defined by

0.7 for the object and 0.3 for the background.

SVM Algorithm

Original Image	Segmented Image	Comment
		The Segmentation is about 95%
		The background is segmented perfectly while the object is 99% good
		The segmentation is very similar to the Bayes but its slightly worse here.
		This image was also segmented about 98% correctly like the Bayes

		This image was segmented the best with the SVM method
		The object was segmented correctly but the background had small part which was segmented wrongly.
		This image is hard due to the variation in the objects colors so it wasn't segmented in a good way.
		This Bayes algorithm and the SVM segmented this image the best among all other methods
		The SVM segmented this image the best

		While the segmentation is not 100 percent but its the best among all other methods.
		This image is slightly worse than Bayes
		About 99% segmentation

The Linear SVM or support vector machine algorithm is a supervised learning model which divides the dataset into two categories with a line. Using the SKLearn library. As we can see some hard images which weren't segmented correctly in any other method was segmented in the SVM method like the image of the camels and the lions while other easy images was slightly worse. 50,000 iterations was used for all the images.

Conclusion

All of the methods show some potential but the ones that stand out are the Bayes and the SVM they show great segmentation in most of the test images. but the Bayes show better segmentation due to having a sample from the image itself while the SVM doesn't have that luxury whilst providing great results too. There is no drastic difference between the NN and KNN methods. The C means shows great potential, its randomness provide good segmentation most of the time but sometimes it can't segment the image and a restart is needed to provide fresh initial cluster intensities. The next step would be grouping the segmented areas and discarding the very small ones based on their areas.

Appendix NN

Code:

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

img = cv2.imread('TestingImages/applelogos_be022.jpg')
cv2.imshow('Old img', img)
print(img.shape)
F1Y, F1Y2, F1X, F1X2 = 92,112,300,320
F2Y, F2Y2, F2X, F2X2 = 165,185,158,178
F3Y, F3Y2, F3X, F3X2 = 209,229,364,384
F4Y, F4Y2, F4X, F4X2 = 203,223,300,320

B1Y, B1Y2, B1X, B1X2 = 12,32,26,46
B2Y, B2Y2, B2X, B2X2 = 8,28,355,375
B3Y, B3Y2, B3X, B3X2 = 190,210,17,37
B4Y, B4Y2, B4X, B4X2 = 294,314,5,25


Frontsample1 = img[F1Y:F1Y2,F1X:F1X2]
old_img = cv2.rectangle(img, (F1X, F1Y), (F1X2, F1Y2), (255, 0, 0), 2)
Frontsample2 = img[F2Y:F2Y2,F2X:F2X2]
old_img = cv2.rectangle(img, (F2X, F2Y), (F2X2, F2Y2), (255, 0, 0), 2)
Frontsample3 = img[F3Y:F3Y2,F3X:F3X2]
old_img = cv2.rectangle(img, (F3X, F3Y), (F3X2, F3Y2), (255, 0, 0), 2)
Frontsample4 = img[F4Y:F4Y2,F4X:F4X2]
old_img = cv2.rectangle(img, (F4X, F4Y), (F4X2, F4Y2), (255, 0, 0), 2)
```

```python

Backsample1 = img[B1Y:B1Y2,B1X:B1X2]
old_img = cv2.rectangle(img, (B1X, B1Y), (B1X2, B1Y2), (0, 255, 0), 2)
Backsample2 = img[B2Y:B2Y2,B2X:B2X2]
old_img = cv2.rectangle(img, (B2X, B2Y), (B2X2, B2Y2), (0, 255, 0), 2)
Backsample3 = img[B3Y:B3Y2,B3X:B3X2]
old_img = cv2.rectangle(img, (B3X, B3Y), (B3X2, B3Y2), (0, 255, 0), 2)
Backsample4 = img[B4Y:B4Y2,B4X:B4X2]
old_img = cv2.rectangle(img, (B4X, B4Y), (B4X2, B4Y2), (0, 255, 0), 2)
cv2.imshow('Old img', old_img)
```

```python
Frontsample1 = np.reshape(Frontsample1, (Frontsample1.shape[0] * Frontsample1.shape[1], 3))
Frontsample1 = np.average(Frontsample1,axis=0)
Frontsample2 = np.reshape(Frontsample2, (Frontsample2.shape[0] * Frontsample2.shape[1], 3))
Frontsample2 = np.average(Frontsample2,axis=0)
Frontsample3 = np.reshape(Frontsample3, (Frontsample3.shape[0] * Frontsample3.shape[1], 3))
Frontsample3 = np.average(Frontsample3,axis=0)
Frontsample4 = np.reshape(Frontsample4, (Frontsample4.shape[0] * Frontsample4.shape[1], 3))
Frontsample4 = np.average(Frontsample4,axis=0)
```

```
average_samples = np.zeros((2, 4, 3))

average_samples[0,0,:] =Frontsample1
average_samples[0,1,:] =Frontsample2
average_samples[0,2,:] =Frontsample3
average_samples[0,3,:] =Frontsample4

Backsample1 = np.reshape(Backsample1, (Backsample1.shape[0] * Backsample1.shape[1], 3))
Backsample1 = np.average(Backsample1,axis=0)

Backsample2 = np.reshape(Backsample2, (Backsample2.shape[0] * Backsample2.shape[1], 3))
Backsample2 = np.average(Backsample2,axis=0)

Backsample3 = np.reshape(Backsample3, (Backsample3.shape[0] * Backsample3.shape[1], 3))
Backsample3 = np.average(Backsample3,axis=0)

Backsample4 = np.reshape(Backsample4, (Backsample4.shape[0] * Backsample4.shape[1], 3))
Backsample4 = np.average(Backsample4,axis=0)

average_samples[1,0,:] =Backsample1
average_samples[1,1,:] =Backsample2
average_samples[1,2,:] =Backsample3
average_samples[1,3,:] =Backsample4
```

```
img = cv2.imread('TestingImages/applelogos_be022.jpg')

Flattned_avg_samp= average_samples.reshape(average_samples.shape[0] * average_samples.shape[1], average_samples.shape[2])
EuclideanDistance = np.zeros((img.shape[0], img.shape[1], Flattned_avg_samp.shape[0]))

for p in range(Flattned_avg_samp.shape[0]):
    for i in range(img.shape[0]):
        for j in range(img.shape[1]):
            EuclideanDistance[i, j, p] = math.sqrt((img[i, j,0] - Flattned_avg_samp[p, 0]) ** 2 +
                                                   (img[i, j,1] - Flattned_avg_samp[p, 1]) ** 2 +
                                                   (img[i, j,2] - Flattned_avg_samp[p, 2]) ** 2)

IndexMinDistance = np.argmin(EuclideanDistance, axis=2)
IndexMinDistance = np.where(IndexMinDistance < 4, 0, IndexMinDistance)
IndexMinDistance = np.where(IndexMinDistance >= 4, 1, IndexMinDistance)
```

```
average_region1 = []
average_region2 = []
newU_1 = []
newU_2 = []
img_test = np.reshape(img,(img.shape[0]*img.shape[1],3))
IndexMinDistance = np.reshape(IndexMinDistance,(img.shape[0]*img.shape[1]))
print(IndexMinDistance.shape)
for i in range(IndexMinDistance.shape[0]):
    if IndexMinDistance[i] > 0:
        average_region1.append(img_test[i, :])
    if IndexMinDistance[i] < 1:
        average_region2.append(img_test[i, :])

#if len(average_region1) > 0:
mean1 = np.true_divide(np.sum(average_region1, axis=0), len(average_region1))
#if (len(average_region2) > 0):
mean2 = np.true_divide(np.sum(average_region2, axis=0), len(average_region2))
IndexMinDistance = np.reshape(IndexMinDistance, (img.shape[0], img.shape[1]))
for i in range(img.shape[0]):
    for j in range(img.shape[1]):
        if IndexMinDistance[i, j] == 0:
            img[i, j] = mean2.astype(int)
        else:
            img[i, j] = mean1.astype(int)

cv2.imshow('new img', img)
while cv2.waitKey(1) != ord('q'):
    z = 0
cv2.destroyAllWindows()
```

KNN Code:

```python
import numpy as np
from scipy import stats
import math
import cv2
K = 5

img = cv2.imread('TestingImages/113044.jpg')
cv2.imshow('Old img', img)

F1Y, F1Y2, F1X, F1X2 = 177,197,240,260
F2Y, F2Y2, F2X, F2X2 = 378,398,59,79
F3Y, F3Y2, F3X, F3X2 = 86,106,286,306
F4Y, F4Y2, F4X, F4X2 = 116,136,101,121

B1Y, B1Y2, B1X, B1X2 = 9,29,20,40
B2Y, B2Y2, B2X, B2X2 = 289,309,308,328
B3Y, B3Y2, B3X, B3X2 = 300,320,3,23
B4Y, B4Y2, B4X, B4X2 = 235,255,431,451

Frontsample1 = img[F1Y:F1Y2,F1X:F1X2]
old_img = cv2.rectangle(img, (F1X, F1Y), (F1X2, F1Y2), (255, 0, 0), 2)
Frontsample2 = img[F2Y:F2Y2,F2X:F2X2]
old_img = cv2.rectangle(img, (F2X, F2Y), (F2X2, F2Y2), (255, 0, 0), 2)
Frontsample3 = img[F3Y:F3Y2,F3X:F3X2]
old_img = cv2.rectangle(img, (F3X, F3Y), (F3X2, F3Y2), (255, 0, 0), 2)
Frontsample4 = img[F4Y:F4Y2,F4X:F4X2]
old_img = cv2.rectangle(img, (F4X, F4Y), (F4X2, F4Y2), (255, 0, 0), 2)

Backsample1 = img[B1Y:B1Y2,B1X:B1X2]
old_img = cv2.rectangle(img, (B1X, B1Y), (B1X2, B1Y2), (0, 255, 0), 2)
Backsample2 = img[B2Y:B2Y2,B2X:B2X2]
old_img = cv2.rectangle(img, (B2X, B2Y), (B2X2, B2Y2), (0, 255, 0), 2)
Backsample3 = img[B3Y:B3Y2,B3X:B3X2]
old_img = cv2.rectangle(img,  (B3X, B3Y),(B3X2, B3Y2), (0, 255, 0), 2)
Backsample4 = img[B4Y:B4Y2,B4X:B4X2]
old_img = cv2.rectangle(img, (B4X, B4Y), (B4X2, B4Y2), (0, 255, 0), 2)
cv2.imshow('Old img', old_img)

average_samples = np.zeros((2, 4, 3))
Frontsample1 = np.reshape(Frontsample1, (Frontsample1.shape[0] * Frontsample1.shape[1], 3))
Frontsample1 = np.average(Frontsample1, axis=0)

Frontsample2 = np.reshape(Frontsample2, (Frontsample2.shape[0] * Frontsample2.shape[1], 3))
Frontsample2 = np.average(Frontsample2, axis=0)

Frontsample3 = np.reshape(Frontsample3, (Frontsample3.shape[0] * Frontsample3.shape[1], 3))
Frontsample3 = np.average(Frontsample3, axis=0)

Frontsample4 = np.reshape(Frontsample4, (Frontsample4.shape[0] * Frontsample4.shape[1], 3))
Frontsample4 = np.average(Frontsample4, axis=0)

average_samples[0, 0, :] = Frontsample1
average_samples[0, 1, :] = Frontsample2
average_samples[0, 2, :] = Frontsample3
average_samples[0, 3, :] = Frontsample4

Backsample1 = np.reshape(Backsample1, (Backsample1.shape[0] * Backsample1.shape[1], 3))
Backsample1 = np.average(Backsample1, axis=0)
```

```
Backsample2 = np.reshape(Backsample2, (Backsample2.shape[0] * Backsample2.shape[1], 3))
Backsample2 = np.average(Backsample2, axis=0)

Backsample3 = np.reshape(Backsample3, (Backsample3.shape[0] * Backsample3.shape[1], 3))
Backsample3 = np.average(Backsample3, axis=0)

Backsample4 = np.reshape(Backsample4, (Backsample4.shape[0] * Backsample4.shape[1], 3))
Backsample4 = np.average(Backsample4, axis=0)

average_samples[1, 0, :] = Backsample1
average_samples[1, 1, :] = Backsample2
average_samples[1, 2, :] = Backsample3
average_samples[1, 3, :] = Backsample4

img = cv2.imread('TestingImages/113044.jpg')

Flattned_avg_samp = average_samples.reshape(average_samples.shape[0] * average_samples.shape[1],
                                            average_samples.shape[2])
EuclideanDistance = np.zeros((img.shape[0], img.shape[1], Flattned_avg_samp.shape[0]))

for p in range(Flattned_avg_samp.shape[0]):
    for i in range(img.shape[0]):
        for j in range(img.shape[1]):
            EuclideanDistance[i, j, p] = math.sqrt((img[i, j, 0] - Flattned_avg_samp[p, 0]) ** 2 +
                                                   (img[i, j, 1] - Flattned_avg_samp[p, 1]) ** 2 +
                                                   (img[i, j, 2] - Flattned_avg_samp[p, 2]) ** 2)


```

```
IndexMinDistance = EuclideanDistance.argsort(axis=-1)[:, :, :K]
IndexMinDistance = np.where(IndexMinDistance < 4, 0, IndexMinDistance)
IndexMinDistance = np.where(IndexMinDistance >= 4, 1, IndexMinDistance)
predicted = stats.mode(IndexMinDistance, axis=2)

IndexMinDistance = predicted[0]

average_region1 = []
average_region2 = []
newU_1 = []
newU_2 = []
img_test = np.reshape(img,(img.shape[0]*img.shape[1],3))
IndexMinDistance = np.reshape(IndexMinDistance,(img.shape[0]*img.shape[1]))

for i in range(IndexMinDistance.shape[0]):
    if IndexMinDistance[i] > 0:
        average_region1.append(img_test[i, :])
    if IndexMinDistance[i] < 1:
        average_region2.append(img_test[i, :])
#if len(average_region1) > 0:
mean1 = np.true_divide(np.sum(average_region1, axis=0), len(average_region1))
#if (len(average_region2) > 0):
mean2 = np.true_divide(np.sum(average_region2, axis=0), len(average_region2))
IndexMinDistance = np.reshape(IndexMinDistance, (img.shape[0], img.shape[1]))
for i in range(img.shape[0]):
    for j in range(img.shape[1]):
        if IndexMinDistance[i, j] == 0:
            img[i, j] = mean2.astype(int)
        else:
            img[i, j] = mean1.astype(int)
```

```
cv2.imshow('new img', img)
while cv2.waitKey(1) != ord('q'):
    z = 0
cv2.destroyAllWindows()

```

C Means:

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

def Cmean(img,size):

    a = img.shape[0]
    b= img.shape[1]
    average_region1= []
    average_region2 = []
    average_region3 = []
    print(a)
    print(b)
    label = np.zeros([a,b,3])
    U= np.zeros([size,3])
    for i in range(size):
        U[i,:] = [random.randint(0,255),random.randint(0,255),random.randint(0,255)]

    U = np.array(U)
    x=50
```

```python
    if size == 2:
        while x>0:
            x=x-1
            print(x)
            for i in range(a):
                for j in range(b):
                    dis1 = math.sqrt((U[0,0]-img[i,j,0])**2 + (U[0,1]-img[i,j,1])**2+(U[0,2]-img[i,j,2])**2)
                    dis2 = math.sqrt((U[1,0]-img[i,j,0])**2 + (U[1,1]-img[i,j,1])**2+ (U[1,2]-img[i,j,2])**2)
                    if dis1 <= dis2:
                        label[i,j] =1
                        average_region1.append(img[i,j,:])
                    else:
                        label[i,j]=0
                        average_region2.append(img[i, j, :])
            if len(average_region1)>0:
                newU_1 = np.true_divide(np.sum(average_region1,axis=0),len(average_region1))
            else:
                newU_1 = U[0,:]
            if(len(average_region2)>0):
                newU_2 = np.true_divide(np.sum(average_region2,axis=0),len(average_region2))
            else:
                newU_2 = U[1,:]

            if (abs(np.sum(np.subtract(U[0],newU_1)))<0.5 and abs(np.sum(np.subtract(U[1],newU_2)))<0.5):
                x=0
            print(abs(np.sum(np.subtract(U[0],newU_1))))
            print(abs(np.sum(np.subtract(U[1],newU_2))))
            U[0]=newU_1
            U[1]=newU_2
```

```python
        New_img = np.where(label>0,newU_1.astype(int),newU_2.astype(int))
        cv2.imshow("label",label)
        New_img = New_img .astype(np.uint8)
        cv2.imshow("img", New_img)
        cv2.moveWindow('img', 0, 0)
        while cv2.waitKey(1) != ord('q'):
            z = 0

        cv2.destroyAllWindows()
```

```
    elif size == 3:
        while x > 0:
            x = x - 1
            print(x)
            for i in range(a):
                for j in range(b):
                    dis1 = math.sqrt(
                        (U[0, 0] - img[i, j, 0]) ** 2 + (U[0, 1] - img[i, j, 1]) ** 2 + (U[0, 2] - img[i, j, 2]) ** 2)
                    dis2 = math.sqrt(
                        (U[1, 0] - img[i, j, 0]) ** 2 + (U[1, 1] - img[i, j, 1]) ** 2 + (U[1, 2] - img[i, j, 2]) ** 2)
                    dis3 = math.sqrt(
                        (U[2, 0] - img[i, j, 0]) ** 2 + (U[2, 1] - img[i, j, 1]) ** 2 + (U[2, 2] - img[i, j, 2]) ** 2)
                    if dis1 <= dis2 and dis1<= dis3:
                        label[i, j] = 1
                        average_region1.append(img[i, j, :])
                        # average_region1.extend(img[i,j,:])
                        # average_region1 = np.append(average_region1,img[i,j,:])

                    elif dis2<=dis1 and dis2<=dis3:
                        label[i, j] = 0
                        average_region2.append(img[i, j, :])
                        # average_region2.extend(img[i, j, :])
                    #   average_region2 = np.append(average_region2, img[i,j,:])
                    else:
                        label[i, j] = 2
                        average_region3.append(img[i, j, :])
            print(label.shape)
```

```
            if len(average_region1) > 0:
                newU_1 = np.true_divide(np.sum(average_region1, axis=0), len(average_region1))
            else:
                newU_1 = U[0, :]
            if (len(average_region2) > 0):
                newU_2 = np.true_divide(np.sum(average_region2, axis=0), len(average_region2))
            else:
                newU_2 = U[1, :]
            if (len(average_region3) > 0):
                newU_3 = np.true_divide(np.sum(average_region3, axis=0), len(average_region3))
            else:
                newU_3 = U[2, :]

            if (abs(np.sum(np.subtract(U[0], newU_1))) < 0.1 and abs(np.sum(np.subtract(U[1], newU_2))) < 0.1 and abs(np.sum(np.subtract(U[2
                x = 0
            print(abs(np.sum(np.subtract(U[0], newU_1))))
            print(abs(np.sum(np.subtract(U[1], newU_2))))
            print(abs(np.sum(np.subtract(U[2], newU_3))))
            U[0] = newU_1
            U[1] = newU_2
            U[2] = newU_3
        # U= np.array(U)
        # print(U)
        # print(newU_1.astype(int))
        # print(newU_2.astype(int))
        print(label.shape)
        New_img = np.where(label < 1, newU_2.astype(int), (np.where(label > 1, newU_3.astype(int), newU_1.astype(int))))
        cv2.imshow("label", label)
        New_img = New_img.astype(np.uint8)
        cv2.imshow("img", New_img)
```

```
            cv2.imshow("img", New_img)
            cv2.moveWindow('img', 0, 0)
            while cv2.waitKey(1) != ord('q'):
                z = 0

            cv2.destroyAllWindows()




img  = cv2.imread("TestingImages/113044.jpg")
#cv2.imshow("raw",img)
Cmean(img,3)
```

Bayes Algorithm:

```python
import cv2
import numpy as np
pi = 3.14
def Bayes(img,BX,BY,BX2,BY2,FX,FY,FX2,FY2):


    pFront = np.zeros([img.shape[0]*img.shape[1]])

    pBack = np.zeros([img.shape[0]*img.shape[1]])
    img_test = np.reshape(img,(img.shape[0]*img.shape[1],3))
    average_region1 = []
    average_region2 = []

    newU_1= []
    newU_2= []


    BackSample = img[BY:BY2,BX:BX2,:]
    FrontSample = img[FY:FY2,FX:FX2,:]
    BackSample = np.reshape(BackSample,(BackSample.shape[0]*BackSample.shape[1],3))
    FrontSample = np.reshape(FrontSample,(FrontSample.shape[0]*FrontSample.shape[1],3))
    BackMean = np.true_divide(np.sum(BackSample, axis=0), len(BackSample))
    DifferenceBack = np.subtract(BackSample,BackMean)
    BackCovariance = np.true_divide(np.transpose(DifferenceBack).dot(DifferenceBack),len(BackSample))
    FrontMean = np.true_divide(np.sum(FrontSample, axis=0), len(FrontSample))
    DifferenceFront = np.subtract(FrontSample,FrontMean)
    FrontCovariance = np.true_divide(np.transpose(DifferenceFront).dot(DifferenceFront),len(FrontSample))

    for i in range(pFront.shape[0]):
        pBack[i] = -1.19727 - (0.5) * np.log(np.linalg.det(BackCovariance)) - 0.5 * (np.transpose(np.subtract(img_test[i,:],BackMean))).\
            dot(np.linalg.inv(BackCovariance)).dot(np.transpose(np.subtract(img_test[i,:],BackMean)))
        pFront[i]= -1.19727 - (0.5)*np.log(np.linalg.det(FrontCovariance)) - 0.5 *(np.transpose(np.subtract(img_test[i,:],FrontMean))).\
            dot(np.linalg.inv(FrontCovariance)).dot(np.transpose(np.subtract(img_test[i,:],FrontMean)))
    new_img = np.where(0.3*pBack > 0.7*pFront,0,255)
    for i in range(pFront.shape[0]):
        if new_img[i] > 100:
            average_region1.append(img_test[i,:])
        if new_img[i] < 100:
            average_region2.append(img_test[i,:])
    if len(average_region1) > 0:
        newU_1 = np.true_divide(np.sum(average_region1, axis=0), len(average_region1))
    if (len(average_region2) > 0):
        newU_2 = np.true_divide(np.sum(average_region2, axis=0), len(average_region2))

    New_imgColor = np.zeros((img.shape[0]* img.shape[1],3))
    #New_imgColor = np.where(new_img < 1, newU_2.astype(int),  newU_1.astype(int))
    for v in range(pFront.shape[0]):
        if new_img[v] >1:
            New_imgColor[v] = newU_1.astype(int)
        else:
            New_imgColor[v] = newU_2.astype(int)

    new_img = np.reshape(new_img, (img.shape[0],img.shape[1]))
    new_img = new_img.astype(np.uint8)
    New_imgColor = np.reshape(New_imgColor, (img.shape[0], img.shape[1],3))
    New_imgColor = New_imgColor.astype(np.uint8)

    #cv2.imshow("new_img",new_img)
    cv2.imshow("new_imgColor", New_imgColor)

img = cv2.imread("TestingImages/271008.jpg")
BX =63
BY = 61
BX2 = 108
BY2 = 155
FX = 160
FY = 73
FX2 = 214
FY2 = 263
old_img = cv2.rectangle(img, (BX, BY), (BX2, BY2), (255,0,0), 2)
old_img = cv2.rectangle(old_img, (FX, FY), (FX2, FY2), (255, 0, 0), 2)
img = cv2.imread("TestingImages/271008.jpg")
cv2.imshow("old_img",old_img)
Bayes(img,BX,BY,BX2,BY2,FX,FY,FX2,FY2)

while cv2.waitKey(1) != ord('q'):
    z = 0

cv2.destroyAllWindows()
```

Linear SVM:

```python
import numpy as np
from sklearn.svm import LinearSVC
import cv2

img = cv2.imread("TestingImages/113044.jpg")
X = img.reshape(img.shape[0] * img.shape[1], img.shape[2])

BX1 =132
BY1 =34
BX2 =152
BY2 =54

FX1 =293
FY1 =174
FX2 =313
FY2 =194

old_img = cv2.rectangle(img, (FX1, FY1), (FX2, FY2), (0, 255, 0), 2)
old_img = cv2.rectangle(img, (BX1, BY1), (BX2, BY2), (255, 0, 0), 2)
cv2.imshow('old_img', old_img)
img = cv2.imread("TestingImages/113044.jpg")

X = img.reshape(img.shape[0] * img.shape[1], img.shape[2])


Label = 2* np.ones(shape=(img.shape[0], img.shape[1]), dtype=np.int)
Label[BY1:BY2, BX1:BX2] = 0
Label[FY1:FY2, FX1:FX2] = 1

BackImg = img[BY1:BY2, BX1:BX2]
FrontImg = img[FY1:FY2, FX1:FX2]
```

```python
BackImg = np.reshape(BackImg,[BackImg.shape[0]*BackImg.shape[1],3])
FrontImg = np.reshape(FrontImg,[FrontImg.shape[0]*FrontImg.shape[1],3])
MeanBack = np.average(BackImg,axis=0).astype(int)
MeanFront = np.average(FrontImg,axis=0).astype(int)
mean = np.vstack((MeanBack, MeanFront))

y = Label.ravel()
train = np.flatnonzero(Label < 2)
test = np.flatnonzero(Label == 2)

clf = LinearSVC(max_iter=50000)
clf.fit(X[train], y[train])
y[test] = clf.predict(X[test])
Label = y.reshape(img.shape[0], img.shape[1])

cv2.imshow('img2', mean[Label].astype(np.uint8))
while cv2.waitKey(1) != ord('q'):
    z = 0

cv2.destroyAllWindows()
```