Home Work 2

Name : Sai Teja Chava
Course Name : Advanced Computer Vision(CSCI 677)
USC ID : 5551847788
Instructor Name : Prof. Ram Nevatia
Due

Problem 1:

Mean Shift Segmentor:

3 Steps are involved in MeanShiftFiltering. They are as follows.

1. Look for neighboring pixels whose spatial location falls within the spatial radius and whose Euclidean distance between spectra falls within the range radius.
2. Avg those pixel positions to form a new spatial position for the current pixel, and average those pixels' spectra to form a new spectrum for the current pixel.
3. Iterates the process with the new position and spectrum until convergence conditions are met (max number of iterations or move below defined threshold).

For each pixel in the image, we end up with a new spectrum and a proposed spatial position for this. If we look at the image of new spectrum for each pixel, it will look smoother than the initial image, while preserving sharp edges. Therefore, mean- shift can be used for denoising. The image of new spatial position has no meaning by itself, but it is useful for segmentation. Hence, if two pixels fall within range and spatial radius of each other, it does not necessarily mean they will end up in the same segment. It is likely, but it depends on the distribution of the other pixels around them.

We need to use LAB Color space and use Level-1 Pyramid and try out different spatial and color window radius.

Steps involved in the code:

1. Read the input image using cv2.imread()
2. Convert to LAB space as required using cv2.cvtColor()
3. Call the pyrMeanShiftFiltering function implemented in the OpenCV to define the Level-1 pyramid with different spatial window radius and color window radius
4. Display the image to gain some intuitons.

Results

After lot of experiment on the values of spatial window radius and color window radius, the following results were drawn:

Image 1

Having spatial radius as 15 and varying color radius as 15,30,45:

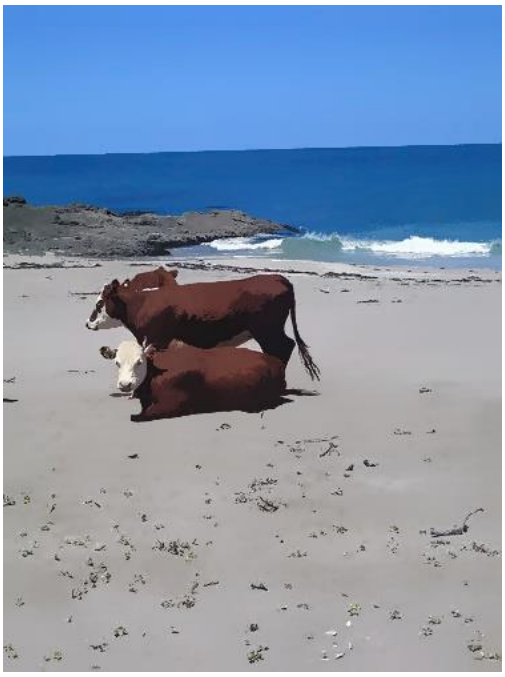


Image 1:

Having color as 15 and varying spatial radius as 30,45 :


Image 2:

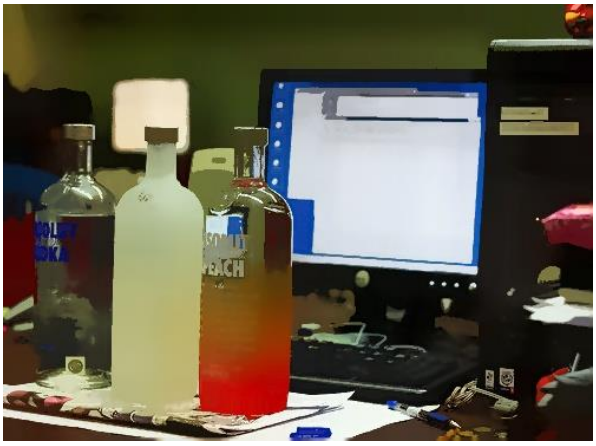
Having spatial radius as 15 and varying color radius as 15,30,45:

Image 2:

Having color as 15 and varying spatial radius as 30,45 :

Image 3:

Having spatial radius as 15 and varying color radius as 15,30,45:



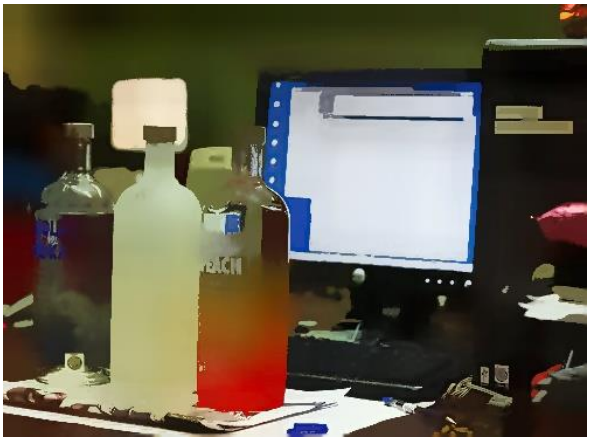


Image 3:

Having color as 15 and varying spatial radius as 30,45 :

Image 4:

Having spatial radius as 15 and varying color radius as 15,30,45:

Image 4:

Having color as 15 and varying spatial radius as 30,45 :

Analysis:

1. Having spatial radius as 15 and varying color radius caused more and more smoothness and having the spatial radius of 15 and color radius of 45 gave best results.
2. Having color radius of 15 and changing the spatial radius did not lead to any significant change is results.
3. The observation that can be made is that changing the color radius has more effect than that of changing the spatial radius.
4. One general observation that can be made about the segmentation using mean shift Segmentor is that the difference between both the radii must be significantly low and most importantly the value of the color window radius must be within a defined limit.
5. When comparing to the ground truth, lower values of both color and spatial radius are performing well than higher values. Having values of spatial radius as 15 and color radius as 45 gave better results.
6. Once you go beyond too much in color radius it becomes too blurry.

Code:

```
#from __future__ import print_function

import os
import numpy as np
import cv2

def mean_shift_segmentor(img_filename, spatial_radius, color_radius):
        src_img = cv2.imread(img_filename, cv2.IMREAD_COLOR)
        img_lab = cv2.cvtColor(src_img, cv2.COLOR_BGR2Lab)

        # Find the peak of a color-spatial distribution
        # pyrMeanShiftFiltering(src, spatialRadius, colorRadius, max_level)
        # For 640x480 color image, it works well to set spatialRadius equal to 2 and colorRadius
equal to 40
        # max_level describes how many levels of scale pyramid you want to use for
segmentation
        # A max_level of 2 or 3 works well for a 640x480 color image
        dst = cv2.pyrMeanShiftFiltering(img_lab, spatial_radius, color_radius, 1)
        dst = cv2.cvtColor(dst, cv2.COLOR_Lab2BGR)

        # filename
        dst_filename = os.path.splitext(img_filename)[0] + '_meanshift_spatial_' +
str(spatial_radius) + '_color_' + str(color_radius) + os.path.splitext(img_filename)[1]
        print('dst_filename: ' + dst_filename)
```

```
        cv2.imwrite(dst_filename, dst)

        return dst

images = ['HW2_ImageData/HW2_ImageData/Images/2007_000464.jpg',
'HW2_ImageData/HW2_ImageData/Images/2007_001288.jpg',
'HW2_ImageData/HW2_ImageData/Images/2007_002953.jpg','HW2_ImageData/HW2_ImageD
ata/Images/2007_005989.jpg']

for image in images:
        color_radius = 30
        for spatial_radius in range(15, 60, 15):
                mean_shift_segmentor(image, spatial_radius, color_radius)

        spatial_radius = 15
        for color_radius in range(15,60,15):
                mean_shift_segmentor(image, spatial_radius, color_radius)
```

Problem 2

Selective Search

Selective Search is a region proposal algorithm used in object detection. It is designed to be fast with a very high recall. It is based on computing hierarchical grouping of similar regions based on color, texture, size and shape compatibility.

Selective Search starts by over-segmenting the image based on intensity of the pixels using a graph-based segmentation method by Felzenszwalb and Huttenlocher.

Perfect segmentation is not the goal. We just want to predict many region proposals such that some of them should have very high overlap with actual objects.

Selective search uses over segments from Felzenszwalb and Huttenlocher's method as an initial input and performs the following steps

1. Add all bounding boxes corresponding to segmented parts to the list of regional proposals
2. Group adjacent segments based on similarity
3. Go to step 1

At each iteration, larger segments are formed and added to the list of region proposals. Hence we create region proposals from smaller segments to larger segments in a bottom-up approach. This is what we mean by computing "hierarchical" segmentations using Felzenszwalb and Huttenlocher's over segments.

There are several similarities measures that can be used. They are as follows:

1. Color
2. Texture
3. Size
4. Shape

You can also use combination of all.

Steps involved in the code are as follows:

1. Reading the image.

2. Create Selective Search Segmentation object.
3. Set input image on which we will run segmentation.
4. Choose among fast but low recall Selective Search method or high recall but slow Selective Search method. I choose slow and quality method.
5. Run selective search segmentation on input image
6. Limit the number of region proposals to show to 100.
7. Calculate the IoU and display the proposals that have IoU of >0.5
8. Display the output

Intersection over union:

Intersection over Union is an evaluation metric used to measure the accuracy of an object detector on a particular dataset. We often see this evaluation metric used in object detection challenges such as the popular PASCAL VOC challenge.

Intersection over Union is simply an evaluation metric. Any algorithm that provides predicted bounding boxes as output can be evaluated using IoU.
More formally, in order to apply Intersection over Union to evaluate an (arbitrary) object detector we need:

1. The ground-truth bounding boxes (i.e., the hand labeled bounding boxes from the testing set that specify where in the image our object is).
2. The predicted bounding boxes from our model.

As long as we have these two sets of bounding boxes we can apply Intersection over Union.

Results(Using Color only Similarity):

Image 1:

Displaying first 100 region proposals

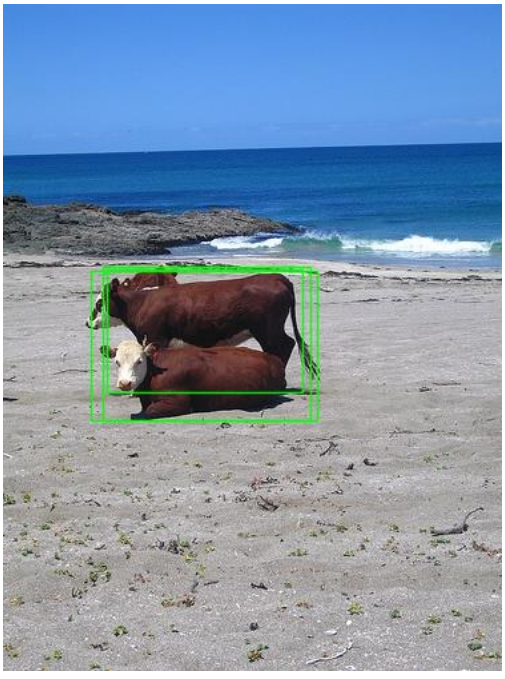


Image 2:

Displaying first 100 region proposals

Displaying the boxes that have an IoU > 0.5 among 100 region proposals shown above.

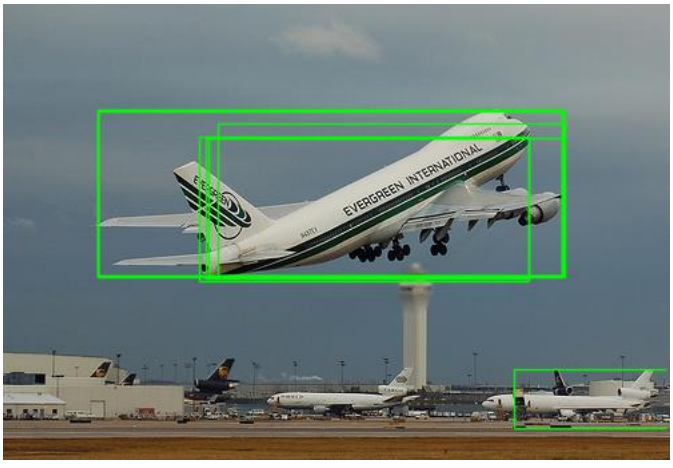


Image 3:

Displaying first 100 region proposals

Displaying the boxes that have an IoU > 0.5 among 100 region proposals shown above.

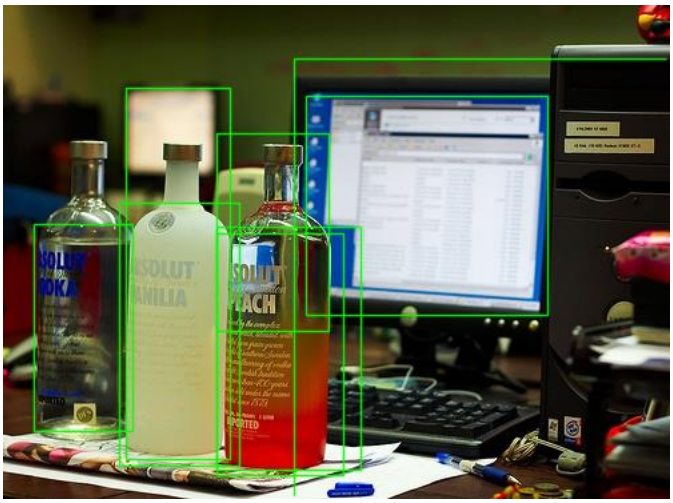
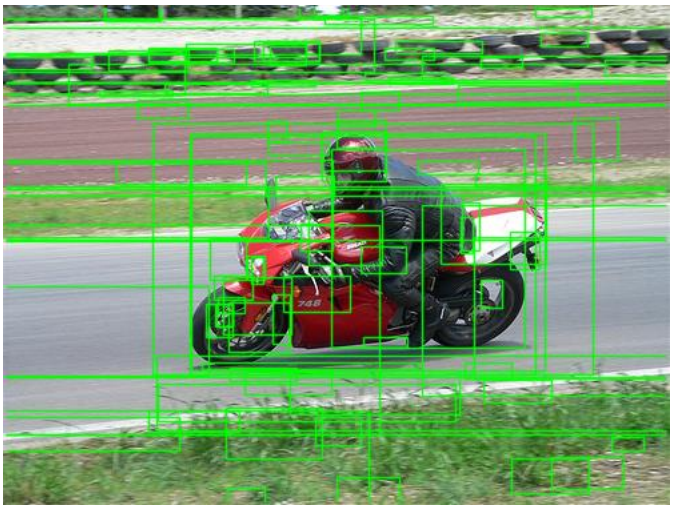
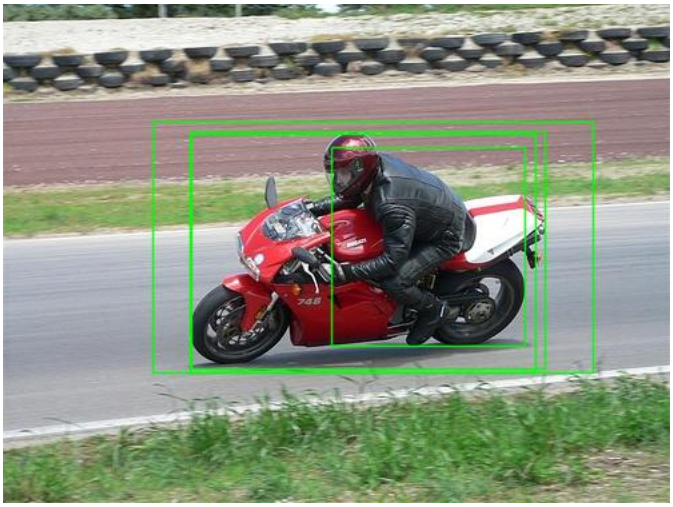


Image 4:

Displaying first 100 region proposals



Displaying the boxes that have an IoU > 0.5 among 100 region proposals shown above.



Results (Using all Similarity):

Image 1:

Displaying first 100 region proposals

Displaying the boxes that have an IoU > 0.5 among 100 region proposals shown above.

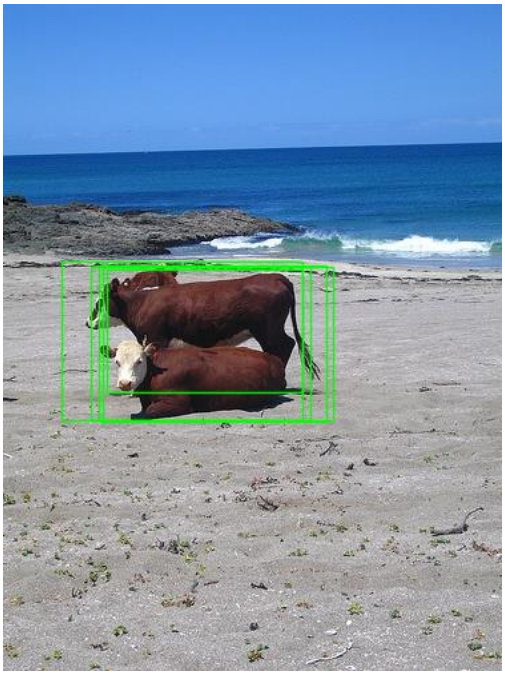


Image 2:

Displaying first 100 region proposals

Displaying the boxes that have an IoU > 0.5 among 100 region proposals shown above.

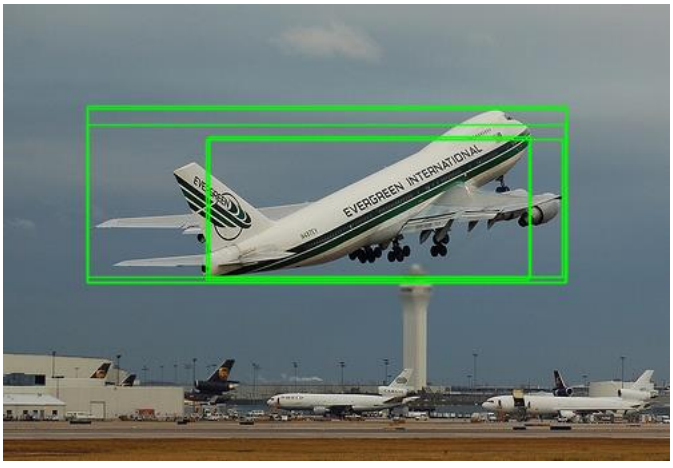
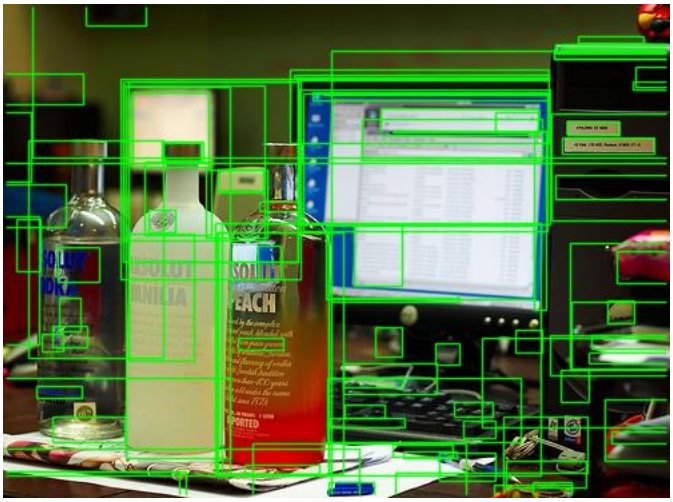


Image 3:

Displaying first 100 region proposals



Displaying the boxes that have an IoU > 0.5 among 100 region proposals shown above.

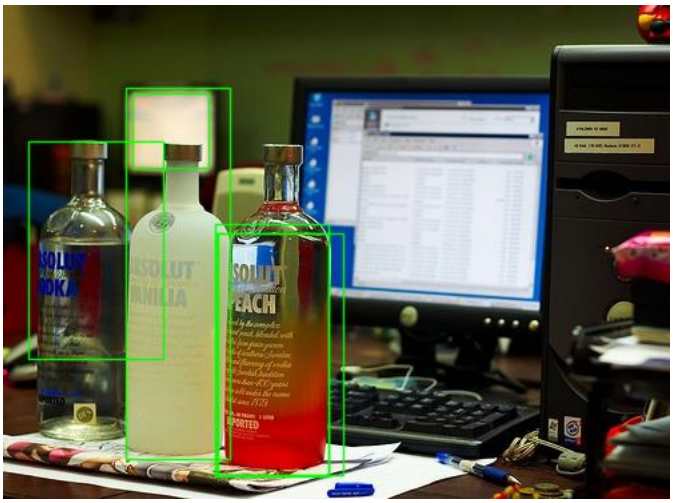
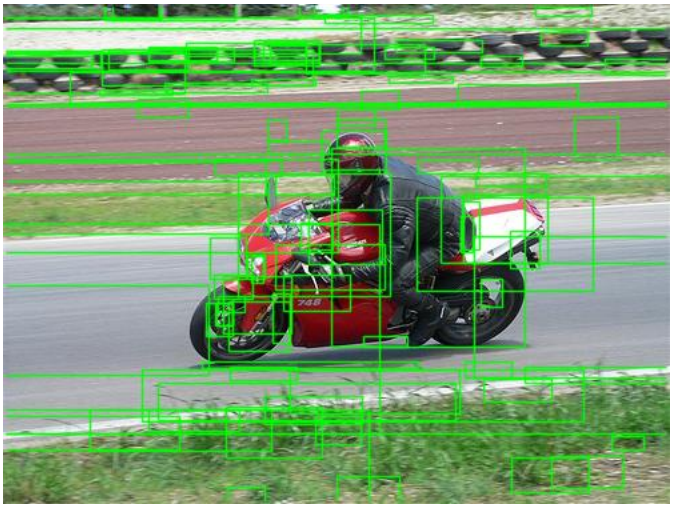


Image 4:

Displaying first 100 region proposals



Displaying the boxes that have an IoU > 0.5 among 100 region proposals shown above.

Analysis:

1. There are two different methods called Selective Search Quality and Selective Search Fast.
2. Selective search fast as the name suggests produces results much faster than selective search quality.
3. Using “color” produced less no of region proposals in all images when compared to using “all” region proposals.
4. Generating more region proposals means more chance of object being detected but more of the computation.
5. Again, it depends whether to go for only “color” or “all” as it depends on images you have and object properties that vary.
6. In case of cows, there were more no of regions that had IoU>0.5 when using “all” when compared to using “color” which means more chance of them being detected.
7. In case of bike image, there were few regions among the region proposals generated that passed the criteria of IoU > 0.5 when using all similarity when compared to using only color similarity.
8. From the results I can say that all similarity performs i.e does well than using only color similarity.

Code:

```
import os
import numpy as np
import cv2


def bb_intersection_over_union(boxA, boxB):
        # determine the (x, y)-coordinates of the intersection rectangle
        xA = max(boxA[0], boxB[0])
        yA = max(boxA[1], boxB[1])
        xB = min(boxA[2], boxB[2])
        yB = min(boxA[3], boxB[3])

        # compute the area of intersection rectangle
        interArea = max(0, xB - xA + 1) * max(0, yB - yA + 1)

        # compute the area of both the prediction and ground-truth
        # rectangles
```

```
        boxAArea = (boxA[2] - boxA[0] + 1) * (boxA[3] - boxA[1] + 1)
        boxBArea = (boxB[2] - boxB[0] + 1) * (boxB[3] - boxB[1] + 1)

        # compute the intersection over union by taking the intersection
        # area and dividing it by the sum of prediction + ground-truth
        # areas - the interesection area
        iou = interArea / float(boxAArea + boxBArea - interArea)

        # return the intersection over union value
        return iou

def SelectiveSearch(image,method,strategy,image_num):

        img = cv2.imread(image)
        cv2.imshow('Window',img)
        cv2.waitKey()

        #Strategies
        stra_color    =
cv2.ximgproc.segmentation.createSelectiveSearchSegmentationStrategyColor()
        stra_texture  =
cv2.ximgproc.segmentation.createSelectiveSearchSegmentationStrategyTexture()
        stra_size     =
cv2.ximgproc.segmentation.createSelectiveSearchSegmentationStrategySize()
        stra_fill     =
cv2.ximgproc.segmentation.createSelectiveSearchSegmentationStrategyFill()
        ss          = cv2.ximgproc.segmentation.createSelectiveSearchSegmentation()

        #Feeding an image
        ss.setBaseImage(img)

        if method=='q':
                ss.switchToSelectiveSearchQuality()
        elif method=='f':
                ss.switchToSelectiveSearchFast()


        #Selecting the Strategy:
        if strategy=="color":
                ss.clearStrategies()
                ss.addStrategy(stra_color)

        elif strategy == "all":
                stra_multi    =
cv2.ximgproc.segmentation.createSelectiveSearchSegmentationStrategyMultiple(stra_texture,str
a_color,stra_size,stra_fill)
```

```
                ss.clearStrategies()
                ss.addStrategy(stra_multi)

        # Get the bounding boxes
        bboxes = ss.process()
        print(len(bboxes))

        numShowRects = 100
        while True:
        # create a copy of original image
                imOut = img.copy()
                imIoU = img.copy()
                # itereate over all the region proposals
                for i, box in enumerate(bboxes):
                        # draw rectangle for region proposal till numShowRects
                        if (i < numShowRects):
                                x, y, w, h = box
                                box1  = [x, y, x+w, y+h]
                                cv2.rectangle(imOut, (x, y), (x+w, y+h), (0, 255, 0), 1,
cv2.LINE_AA)
                                if image_num == 1:
                                        for k in gt1:
                                                iou =
bb_intersection_over_union(k,box1)
                                                if iou>0.5:
                                                        cv2.rectangle(imIoU, (x, y),
(x+w, y+h), (0, 255, 0), 1, cv2.LINE_AA)
                                elif image_num == 2:
                        for k in gt2:
                            iou = bb_intersection_over_union(k,box1)
                            if iou>0.5:
                                cv2.rectangle(imIoU, (x, y), (x+w, y+h), (0, 255, 0), 1,
cv2.LINE_AA)
                                elif image_num == 3:
                        for k in gt3:
                            iou = bb_intersection_over_union(k,box1)
                            if iou>0.5:
                                cv2.rectangle(imIoU, (x, y), (x+w, y+h), (0, 255, 0), 1,
cv2.LINE_AA)
                                elif image_num == 4:
                        for k in gt4:
                            iou = bb_intersection_over_union(k,box1)
                            if iou>0.5:
                                cv2.rectangle(imIoU, (x, y), (x+w, y+h), (0, 255, 0), 1,
cv2.LINE_AA)
                            else:
```

```
                    break


        # show output
                cv2.imshow("Output", imOut)

                # Show IoU
                cv2.imshow("IoU",imIoU)

                # record key press
                k = cv2.waitKey(0) & 0xFF


        # q is pressed
                if k == 113:
                break



images = ['HW2_ImageData/HW2_ImageData/Images/2007_000464.jpg',
'HW2_ImageData/HW2_ImageData/Images/2007_001288.jpg',
'HW2_ImageData/HW2_ImageData/Images/2007_002953.jpg','HW2_ImageData/HW2_ImageD
ata/Images/2007_005989.jpg']

gt1 = [[71,252,216,314],[58,202,241,295]]
gt2 = [[68,76,426,209],[357,272,497,317],[196,272,316,314]]
gt3 = [[25,102,93,325],[94,104,166,337],[165,103,247,347],[216,52,416,304],[89,61,162,141]]
gt4 = [[140,130,408,273],[213,96,355,260]]

i = 1
for image in images:
        SelectiveSearch(image,'q',"color",i)
        i+=1
```