selective_search

September 27, 2020

[139]:
```
import cv2
import os
from matplotlib import pyplot as plt
import numpy as np
import argparse
import xml.etree.ElementTree as ET
```

[140]:
```
images_path = "/Users/sagarsudhakara/Documents/CV_Course/Assignment_2/HW2_Data/
↪JPEGImages"
annotated_path = "/Users/sagarsudhakara/Documents/CV_Course/Assignment_2/
↪HW2_Data/Annotations"
```

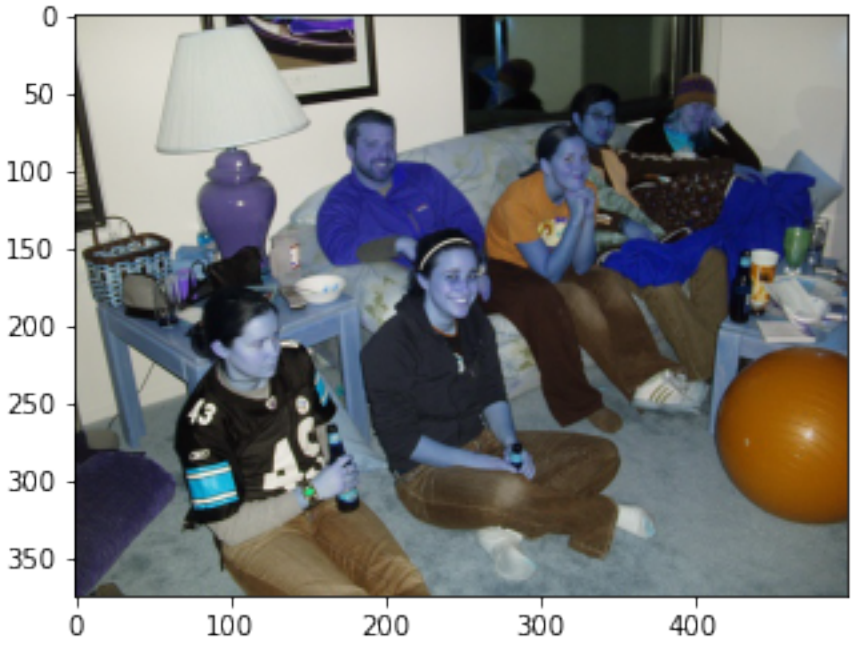
[141]:
```
def load_images_from_folder(folder):
    images = []
    for filename in os.listdir(folder):
        img = cv2.imread(os.path.join(folder, filename))
        if img is not None:
            images.append(img)
    return images

def load_annot_from_folder(folder):
    annot=[]
    for filename in os.listdir(folder):
        ann=os.path.join(folder, filename)
        annot.append(ann)
    return annot
```

[142]:
```
pic = load_images_from_folder(images_path)
annotations=load_annot_from_folder(annotated_path)

print(annotations)
```

```
['/Users/sagarsudhakara/Documents/CV_Course/Assignment_2/HW2_Data/Annotations/00
0009.xml', '/Users/sagarsudhakara/Documents/CV_Course/Assignment_2/HW2_Data/Anno
tations/000220.xml', '/Users/sagarsudhakara/Documents/CV_Course/Assignment_2/HW2
_Data/Annotations/002129.xml', '/Users/sagarsudhakara/Documents/CV_Course/Assign
ment_2/HW2_Data/Annotations/006919.xml']
```

[143]:
```
plt.imshow(pic[3])
```

[143]:
```
<matplotlib.image.AxesImage at 0x126339160>
```



[144]:
```
#cv2.destroyAllWindows()
#ss = cv2.ximgproc.segmentation.createSelectiveSearchSegmentation()
#ss.setBaseImage(img[0])

#bboxes=ss.process()
```

[145]:
```
def get_intersection_area(box1, box2):
    """
    Calculates the intersection area of two bounding boxes where (x1,y1)␣
 ↪indicates the top left corner and (x2,y2)
    indicates the bottom right corner
    :param box1: List of coordinates(x1,y1,x2,y2) of box1
    :param box2: List of coordinates(x1,y1,x2,y2) of box2
    :return: float: area of intersection of the two boxes
    """
    x1 = max(box1[0], box2[0])
    x2 = min(box1[2], box2[2])
    y1 = max(box1[1], box2[1])
    y2 = min(box1[3], box2[3])
    # Check for the condition if there is no overlap between the bounding boxes␣
 ↪(either height or width
```

```
    # of intersection box are negative)
    if (x2 - x1 < 0) or (y2 - y1 < 0):
        return 0.0
    else:
        return (x2 - x1 + 1) * (y2 - y1 + 1)
```

[146]:
```
def calculate_iou(proposal_boxes, gt_boxes):
    """
    Returns the bounding boxes that have Intersection over Union (IOU) > 0.5␣
↪with the ground truth boxes
    :param proposal_boxes: List of proposed bounding boxes(x1,y1,x2,y2) where␣
↪(x1,y1) indicates the top left corner
    and (x2,y2) indicates the bottom right corner of the proposed bounding box
    :param gt_boxes: List of ground truth boxes(x1,y1,x2,y2) where (x1,y1)␣
↪indicates the top left corner and (x2,y2)
    indicates the bottom right corner of the ground truth box
    :return iou_qualified_boxes: List of all proposed bounding boxes that have␣
↪IOU > 0.5 with any of the ground
    truth boxes
    :return final_boxes: List of the best proposed bounding box with each of␣
↪the ground truth box (if available)
    """
    iou_qualified_boxes = []
    final_boxes = []
    for gt_box in gt_boxes:
        best_box_iou = 0
        best_box = 0
        area_gt_box = (gt_box[2] - gt_box[0]) * (gt_box[3] - gt_box[1])
        for prop_box in proposal_boxes:
            area_prop_box = (prop_box[2] - prop_box[0] + 1) * (prop_box[3] -␣
↪prop_box[1] + 1)
            intersection_area = get_intersection_area(prop_box, gt_box)
            union_area = area_prop_box + area_gt_box - intersection_area
            iou = float(intersection_area) / float(union_area)
            if iou > 0.5:
                iou_qualified_boxes.append(prop_box)
                if iou > best_box_iou:
                    best_box_iou = iou
                    best_box = prop_box
        if best_box_iou != 0:
            final_boxes.append(best_box)
    return iou_qualified_boxes, final_boxes
```

[147]:
```
print(get_intersection_area([1,3,3,1], [2,2,4,0]))
```

```
0.0
```

[148]:
```
def get_groundtruth_boxes(annoted_img_path):
    """
    Parses the xml file of the annotated image to obtain the ground truth boxes
    :param annoted_img_path: String: File path of the annotated image␣
↪containing the ground truth
    :return gt_boxes: List of ground truth boxes(x1,y1,x2,y2) where (x1,y1)␣
↪indicates the top left corner and (x2,y2)
    indicates the bottom right corner of the ground truth box
    """
    gt_boxes = []
    tree = ET.parse(annoted_img_path)
    root = tree.getroot()
    for items in root.findall('object/bndbox'):
        xmin = items.find('xmin')
        ymin = items.find('ymin')
        xmax = items.find('xmax')
        ymax = items.find('ymax')
        gt_boxes.append([int(xmin.text), int(ymin.text), int(xmax.text),␣
↪int(ymax.text)])
    return gt_boxes
```

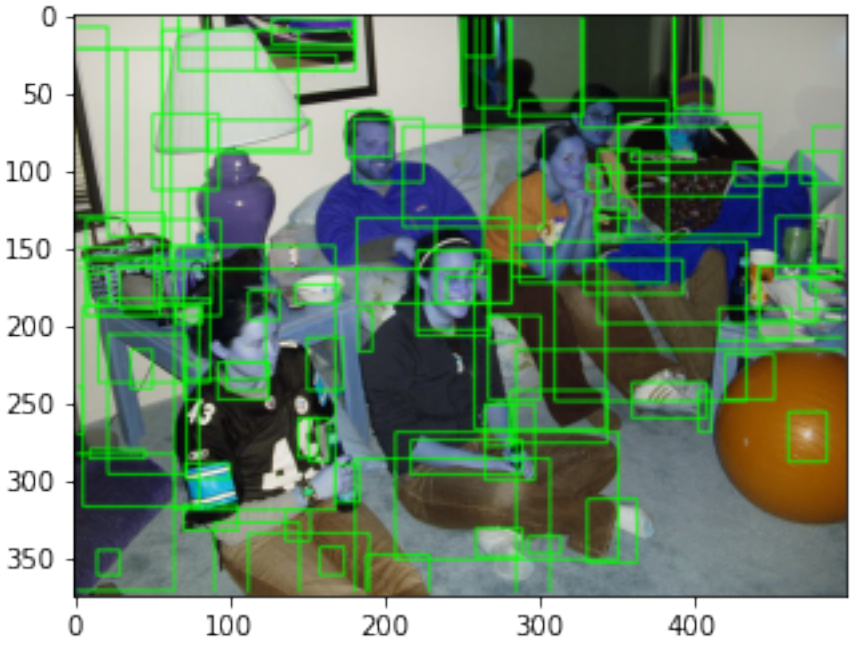
[149]:
```
if __name__ == "__main__":
    #parser = argparse.ArgumentParser()
    #parser.add_argument("input_image_path", default="./HW2_Data/JPEGImages/
↪000480.jpg", type=str, help="Enter the image path")
    #parser.add_argument("annotated_image_path", default="./HW2_Data/
↪Annotations/000480.xml", type=str, help="Enter the annotated image path")
    #parser.add_argument("strategy", default="color", type=str, help="Enter the␣
↪strategy - color for color strategy, all for all strategies")
    #args = parser.parse_args()

    #img_path = args.input_image_path
    #annotated_img_path = args.annotated_image_path
    img=pic[3]
    annotated_img_path=annotations[3]
    #img = cv2.imread(img_path)   it is pic[0]
    ss = cv2.ximgproc.segmentation.createSelectiveSearchSegmentation()

    # Convert image from BGR (default color in OpenCV) to RGB
    rgb_im = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    ss.addImage(rgb_im)
    gs = cv2.ximgproc.segmentation.createGraphSegmentation()
    # gs.setK(150)
    # gs.setSigma(0.8)
    ss.addGraphSegmentation(gs)
    ss.clearStrategies()
```

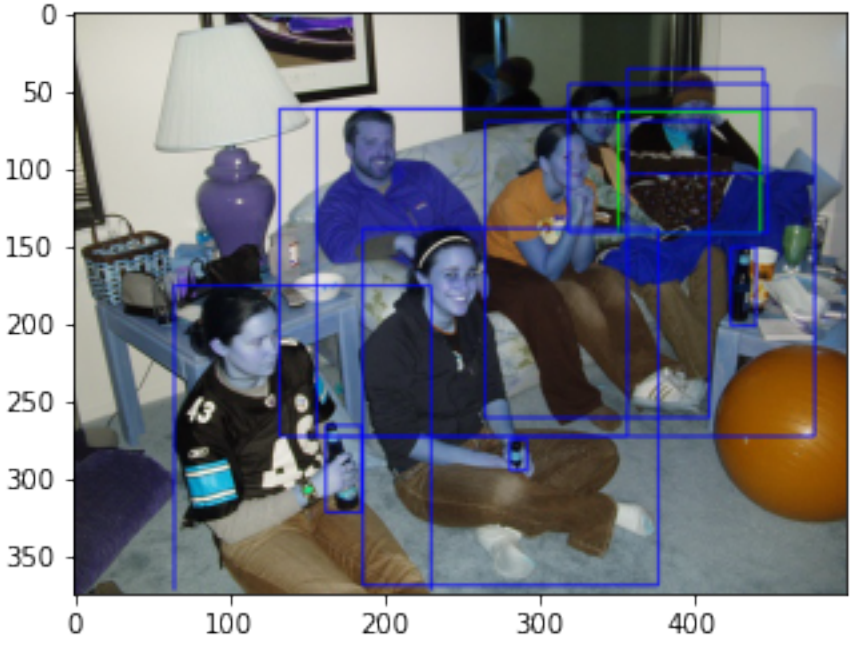
```
    # Creating strategy using color similarity
    #if args.strategy == "color":
    #strategy_color = cv2.ximgproc.segmentation.
↪createSelectiveSearchSegmentationStrategyColor()
    #ss.addStrategy(strategy_color)

    # Creating strategy using texture similarity
    #elif args.strategy == "texture":
    #strategy_texture = cv2.ximgproc.segmentation.
↪createSelectiveSearchSegmentationStrategyTexture()
    #ss.addStrategy(strategy_texture)
    #ss.clearStrategies()

    # Creating strategy using all similarities (size,color,fill,texture)
    #elif args.strategy == "all":
    strategy_color = cv2.ximgproc.segmentation.
↪createSelectiveSearchSegmentationStrategyColor()
    strategy_fill = cv2.ximgproc.segmentation.
↪createSelectiveSearchSegmentationStrategyFill()
    strategy_size = cv2.ximgproc.segmentation.
↪createSelectiveSearchSegmentationStrategySize()
    strategy_texture = cv2.ximgproc.segmentation.
↪createSelectiveSearchSegmentationStrategyTexture()
    strategy_multiple = cv2.ximgproc.segmentation.
↪createSelectiveSearchSegmentationStrategyMultiple(strategy_color,␣
↪strategy_fill, strategy_size, strategy_texture)
    ss.addStrategy(strategy_multiple)

    get_boxes = ss.process()
    print("Total proposals = ", len(get_boxes))
    proposal_box_limit = 100

    # Convert (x,y,w,h) parameters for the top 100 proposal boxes returned from␣
↪ss.process() command into
    # (x, y, x+w, y+h) parameters to be consistent with the xml tags of the␣
↪ground truth boxes where
    # (x,y) indicates the top left corner and (x+w,y+h) indicates the bottom␣
↪right corner of bounding box
    boxes = [[box[0], box[1], box[0] + box[2], box[1] + box[3]] for box in␣
↪get_boxes[0:proposal_box_limit]]

    output_img_proposal_top100 = img.copy()
    output_img_iou_qualified = img.copy()
    output_img_final = img.copy()

    # Fetch all ground truth boxes from the annotated image file
```

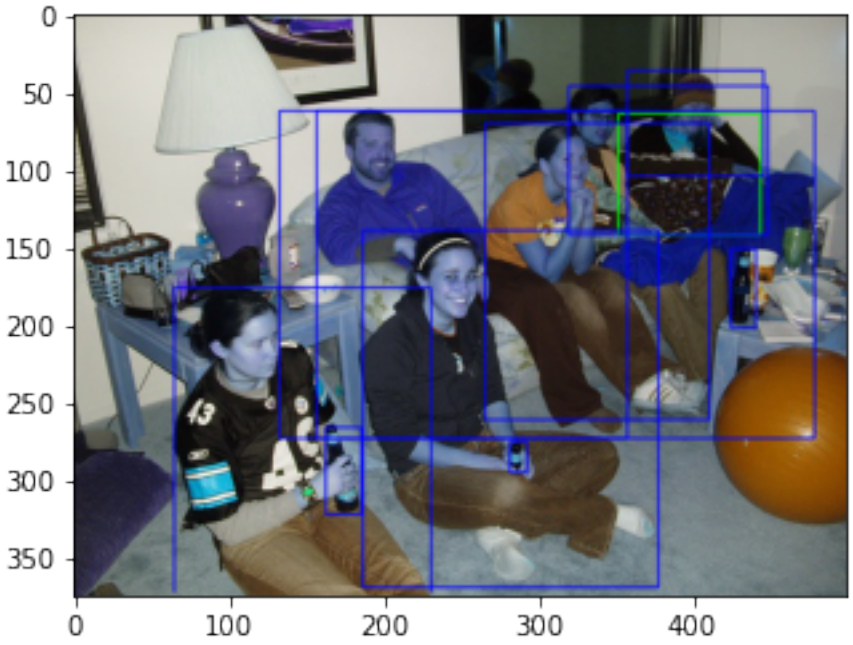
```
    gt_boxes = get_groundtruth_boxes(annotated_img_path)
    print("Number of Ground Truth Boxes = ", len(gt_boxes))

    # Draw bounding boxes for top 100 proposals
    for i in range(0, len(boxes)):
        top_x, top_y, bottom_x, bottom_y = boxes[i]
        cv2.rectangle(output_img_proposal_top100, (top_x, top_y), (bottom_x,␣
↪bottom_y), (0, 255, 0), 1, cv2.LINE_AA)
        #cv2.imshow("Output_Top_100_Proposals", output_img_proposal_top100)
        plt.figure(1)
        plt.imshow( output_img_proposal_top100)
    cv2.imwrite("./SelectiveSearch_results/Output_Top_100_Proposals4b.png",␣
↪output_img_proposal_top100)
    #cv2.waitKey()
    #cv2.destroyAllWindows()


    # Fetch all proposed bounding boxes that have IOU > 0.5 with any of the␣
↪ground truth boxes and also the bounding box
    # that has the maximum/best overlap for each ground truth box
    iou_qualified_boxes, final_boxes = calculate_iou(boxes, gt_boxes)
    print("Number of Qualified Boxes with IOU > 0.5 = ",␣
↪len(iou_qualified_boxes))
    print("Qualified Boxes = ", iou_qualified_boxes)

    # Draw bounding boxes for iou_qualified_boxes
    for i in range(0, len(iou_qualified_boxes)):
        top_x, top_y, bottom_x, bottom_y = iou_qualified_boxes[i]
        cv2.rectangle(output_img_iou_qualified, (top_x, top_y), (bottom_x,␣
↪bottom_y), (0, 255, 0), 1, cv2.LINE_AA)
    for i in range(0, len(gt_boxes)):
        top_x, top_y, bottom_x, bottom_y = gt_boxes[i]
        cv2.rectangle(output_img_iou_qualified, (top_x, top_y), (bottom_x,␣
↪bottom_y), (0, 0, 255), 1, cv2.LINE_AA)
    #cv2.imshow("Output_IOU_Qualified_Proposals", output_img_iou_qualified)
    plt.figure(2)
    plt.imshow( output_img_iou_qualified)
    cv2.imwrite("./SelectiveSearch_results/Output_IOU_Qualified_Proposals4b.
↪png", output_img_iou_qualified)
    #cv2.waitKey()
    #cv2.destroyAllWindows()

    print("Number of final boxes = ", len(final_boxes))
    print("Final boxes = ", final_boxes)
```

```
    # Recall is calculated as the fraction of ground truth boxes that overlap␣
↪with at least one proposal box with
    # Intersection over Union (IoU) > 0.5
    recall = len(final_boxes) / len(gt_boxes)
    print("Recall = ", recall)

    # Draw bounding boxes for final_boxes
    for i in range(0, len(final_boxes)):
        top_x, top_y, bottom_x, bottom_y = final_boxes[i]
        cv2.rectangle(output_img_final, (top_x, top_y), (bottom_x, bottom_y),␣
↪(0, 255, 0), 1, cv2.LINE_AA)
    for i in range(0, len(gt_boxes)):
        top_x, top_y, bottom_x, bottom_y = gt_boxes[i]
        cv2.rectangle(output_img_final, (top_x, top_y), (bottom_x, bottom_y),␣
↪(0, 0, 255), 1, cv2.LINE_AA)
    #cv2.imshow("Output_Final_Boxes", output_img_final)
    plt.figure(3)
    plt.imshow( output_img_final)
    cv2.imwrite("./SelectiveSearch_results/output_img_final4b.png",␣
↪output_img_final)
    #cv2.waitKey()
    #cv2.destroyAllWindows()
```

```
Total proposals =  302
Number of Ground Truth Boxes =  10
Number of Qualified Boxes with IOU > 0.5 =  1
Qualified Boxes =  [[352, 64, 444, 142]]
Number of final boxes =  1
Final boxes =  [[352, 64, 444, 142]]
Recall =  0.1
```


0
50
100
150
200
250
300
350
0
100
200
300
400


0
50
100
150
200
250
300
350
0
100
200
300
400


0
50
100
150
200
250
300
350
0
100
200
300
400

[ ]: