Angel Nieto García 2211798052

Homework Assignment #2

1. Selective search
   1.1 Source code

homework.py # class to read a filename, parse xml associated with that name and run selective search and edge boxes, parameters passed in console are –filename (image to analyze) and –numproposals (max number of proposals to display)

```
import selective_search as ss
import edge_boxes as es
import utils as utils
import cv2
import argparse

if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument('--filename', default=None)
    parser.add_argument('--numproposals', default=None)
    args = parser.parse_args()

    utils.parse_xmlfile(args.filename)
    print(ss.run_selective_search(args.filename, int(args.numproposals), True))
    print(es.run_edge_boxes(args.filename, int(args.numproposals), True))
```

utils.py # which has helper methods to parse and store ground_truths, to draw ground truths in an image, get IoU, to compare to boxes and see if their IoU > 0.5 and to compute recall

```
import xml.etree.ElementTree as ET
import cv2

gt_boxes = []

def parse_xmlfile(xmlfile):
    root = ET.parse('./Annotations/' + xmlfile + '.xml')
    for bndbox in root.findall('object/bndbox'):
        xmin = int(bndbox.find('xmin').text)
        ymin = int(bndbox.find('ymin').text)
        xmax = int(bndbox.find('xmax').text)
        ymax = int(bndbox.find('ymax').text)
        gt_boxes.append((xmin, ymin, xmax, ymax))

    return gt_boxes

def draw_groundtruth(image_output):
    for ground_truth in gt_boxes:
        cv2.rectangle(image_output, (ground_truth[0], ground_truth[1]), (ground_truth[2],
ground_truth[3]), (0, 0, 255), 1, cv2.LINE_AA)
```

```
def intersection_over_union(box_a, box_b):
    # determine the (x, y)-coordinates of the intersection rectangle
    x_a = max(box_a[0], box_b[0])
    y_a = max(box_a[1], box_b[1])
    x_b = min(box_a[2], box_b[2])
    y_b = min(box_a[3], box_b[3])

    # compute the area of intersection rectangle
    intersection_area = max(0, x_b - x_a + 1) * max(0, y_b - y_a + 1)
    if intersection_area == 0:
        return 0

    # compute the area of both the prediction and ground-truth
    # rectangles
    box_a_area = (box_a[2] - box_a[0] + 1) * (box_a[3] - box_a[1] + 1)
    box_b_area = (box_b[2] - box_b[0] + 1) * (box_b[3] - box_b[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 = intersection_area / float(box_a_area + box_b_area - intersection_area)

    return iou

def compare_gt(test_box):
    for box in gt_boxes:
        if intersection_over_union(test_box, box) > 0.5:
            return True
    return False

def compute_recall(proposal_boxes):
    true_positives = 0
    for gt_box in gt_boxes:
        for proposal_box in proposal_boxes:
            x, y, w, h = proposal_box # proposal box to gt_coordinates
            if intersection_over_union(gt_box, (x , y, x + w, y + h)) > 0.5:
                true_positives += 1
                break
    return (true_positives / len(gt_boxes), len(proposal_boxes))
```

selective_search.py # run ss algorithm for each strategy existent in cv2.ximgproc.segmentation in a loop to speed up image analysis, last parameter (draw_only_positives) allows to display ground_truth and matching proposal

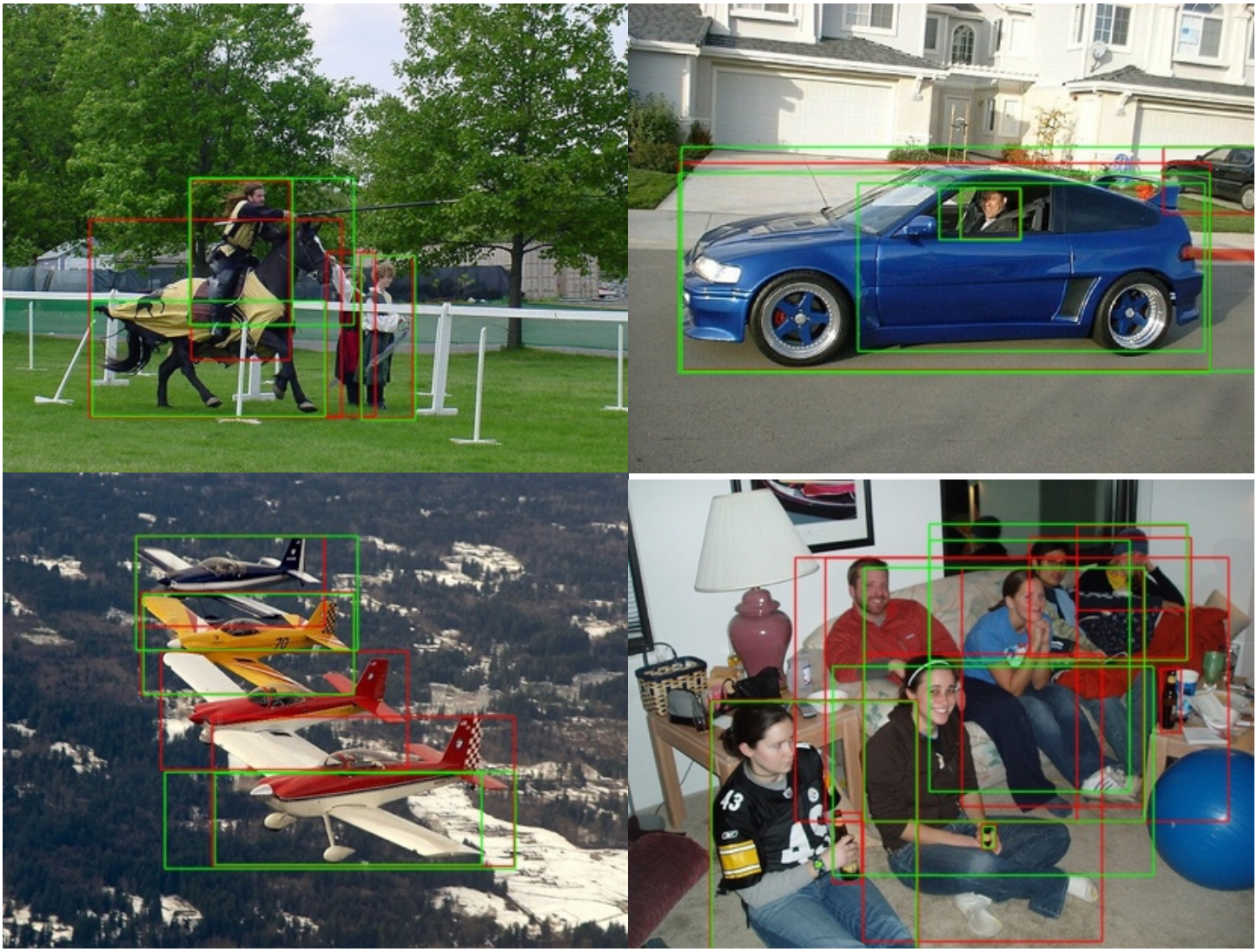
```
import cv2
import utils

def run_selective_search(image_file, num_proposals, draw_only_positives):
    img = cv2.imread('./JPEGImages/' + image_file + '.jpg')

    # create Selective Search Segmentation Object using default parameters
```

```
    ss = cv2.ximgproc.segmentation.createSelectiveSearchSegmentation()

    ss.setBaseImage(img)

    strategy_color = cv2.ximgproc.segmentation.createSelectiveSearchSegmentationStrategyColor()
    strategy_texture = cv2.ximgproc.segmentation.createSelectiveSearchSegmentationStrategyTexture()
    strategy_size = cv2.ximgproc.segmentation.createSelectiveSearchSegmentationStrategySize()
    strategy_fill = cv2.ximgproc.segmentation.createSelectiveSearchSegmentationStrategyFill()
    strategy_combined =
cv2.ximgproc.segmentation.createSelectiveSearchSegmentationStrategyMultiple(strategy_color,
strategy_texture, strategy_size, strategy_fill)

    strategies = [strategy_texture, strategy_color, strategy_size, strategy_fill,
strategy_combined]

    recall_list = []

    for strategy in strategies:

        ss.addStrategy(strategy)

        ss.switchToSelectiveSearchFast()
        # ss.switchToSelectiveSearchQuality()

        prososal_boxes = ss.process()

        image_output = img.copy()

        # iterate over all the region proposals
        for i, rect in enumerate(prososal_boxes):
            # draw rectangle for region proposal till numShowRects
            if (i < num_proposals):
                x, y, w, h = rect
                if draw_only_positives and not utils.compare_gt((x , y, x + w, y + h)):
                    continue
                cv2.rectangle(image_output, (x, y), (x + w, y + h), (0, 255, 0), 1, cv2.LINE_AA)
            else:
                break

        if draw_only_positives:
            utils.draw_groundtruth(image_output)

        recall_list.append(utils.compute_recall(prososal_boxes))
        # cv2.imshow("Selective search output " + str(strategy), image_output)
        cv2.imwrite("./ss_output_" + "_" + image_file + "_" + str(strategy) + ".jpg", image_output)
        # cv2.waitKey(0)
        ss.clearStrategies()
        # cv2.destroyAllWindows()

    return recall_list
```

1.2 Images result

1.2.1 Selective search strategy color 100 first proposals

First 100 matching proposals with IoU > 0.5



1.2.2 Selective search strategy texture 100 first proposals

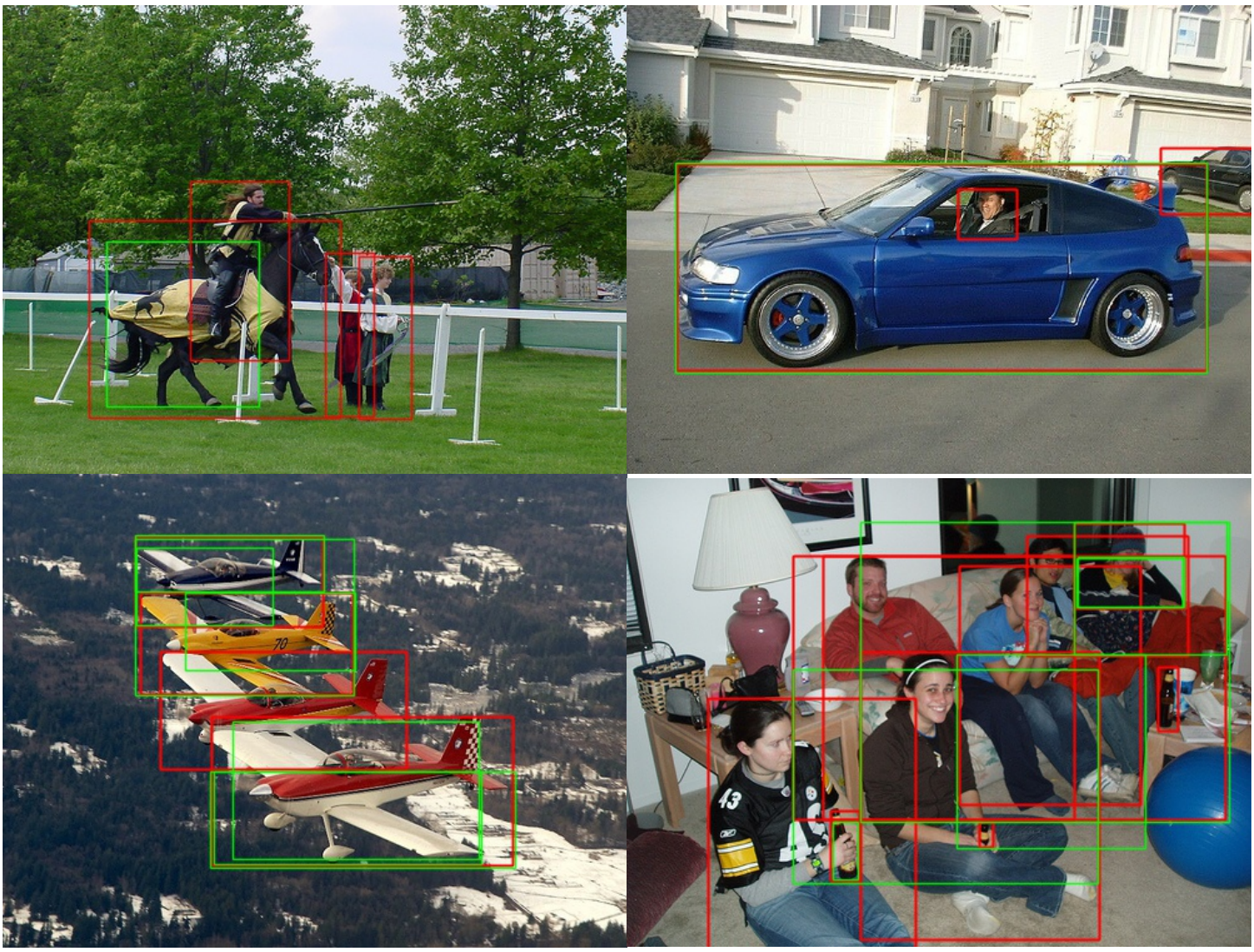
First 100 matching proposals with IoU > 0.5

1.2.3 Selective search strategy size 100 first proposals

First 100 matching proposals with IoU > 0.5

1.2.4 Selective search strategy fill 100 first proposals

First 100 matching proposals with IoU > 0.5



1.2.5 Selective search combining all previous strategies 100 first proposals

First 100 matching proposals with IoU > 0.5

2 Edge boxes

2.1 Source code

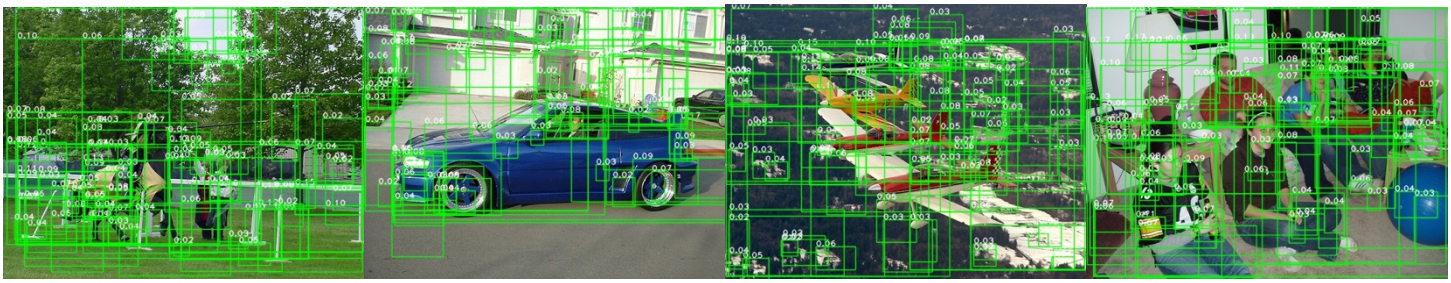
selective_search.py # run edgeboxes algorithm for each strategy existent in cv2.ximgproc.segmentation in a loop to speed up image analysis, last parameter (draw_only_positives) allows to display ground_truth and matching proposal

```
import cv2
import numpy as np
import utils

def run_edge_boxes(image_file, num_proposals, draw_only_positives):
    img = cv2.imread('./JPEGImages/' + image_file + '.jpg')

    parameters_combinations = [(0.25, 0.85), (0.45, 0.45), (0.65, 0.75), (0.85, 0.35), (0.85,
0.85)]
    recall_list = []

    for parameter_tuple in parameters_combinations:
        edge_detection = cv2.ximgproc.createStructuredEdgeDetection('model.yml.gz')

        rgb_im = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        edges = edge_detection.detectEdges(np.float32(rgb_im / 255.0))
        orientation = edge_detection.computeOrientation(edges)

        edges = edge_detection.edgesNms(edges, orientation)
        edge_boxes = cv2.ximgproc.createEdgeBoxes(alpha = parameter_tuple[0], beta =
parameter_tuple[1])
        proposal_boxes, scores = edge_boxes.getBoundingBoxes(edges, orientation)

        image_output = img.copy()

        if len(proposal_boxes) > 0:
            boxes_scores = zip(proposal_boxes, scores)

            for i, b_s in enumerate(boxes_scores):
                if (i < num_proposals):
                    box = b_s[0]
                    x, y, w, h = box
                    if draw_only_positives and not utils.compare_gt((x , y, x + w, y + h)):
                        continue
                    cv2.rectangle(image_output, (x, y), (x + w, y + h), (0, 255, 0), 1,
cv2.LINE_AA)
                    score = b_s[1][0]
                    cv2.putText(image_output, "{:.2f}".format(score), (x, y),
cv2.FONT_HERSHEY_PLAIN, 0.8, (255, 255, 255), 1, cv2.LINE_AA)
                    # print("score={:f}".format(score))
                else:
                    break
```

```
        if draw_only_positives:
            utils.draw_groundtruth(image_output)

        recall_list.append(utils.compute_recall(proposal_boxes))
        # cv2.imshow("Edgeboxes output " + str(parameter_tuple), image_output)
        cv2.imwrite("./eb_output_" + image_file + "_" + str(parameter_tuple) + ".jpg",
image_output)
        # cv2.waitKey(0)
        # cv2.destroyAllWindows()

    return recall_list
```

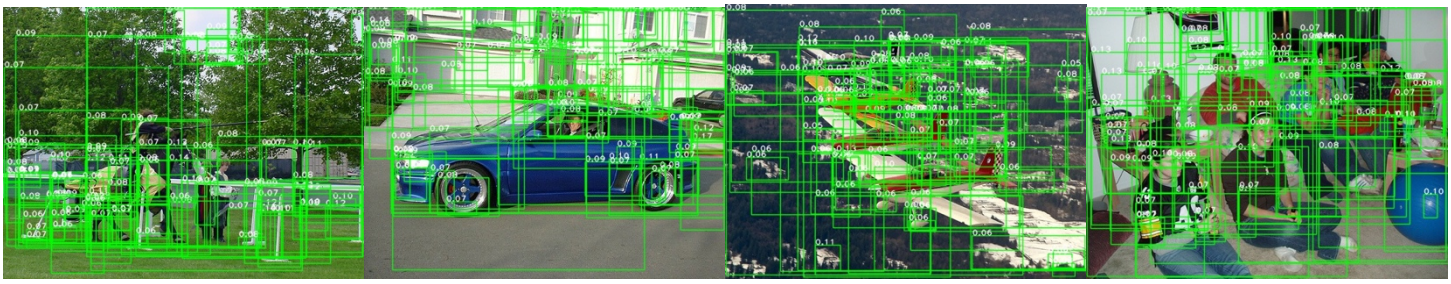
2.2 Images result (score numbers appear on top of proposal)

2.2.1 Edgeboxes with $\alpha = 0.25$ and $\beta = 0.85$



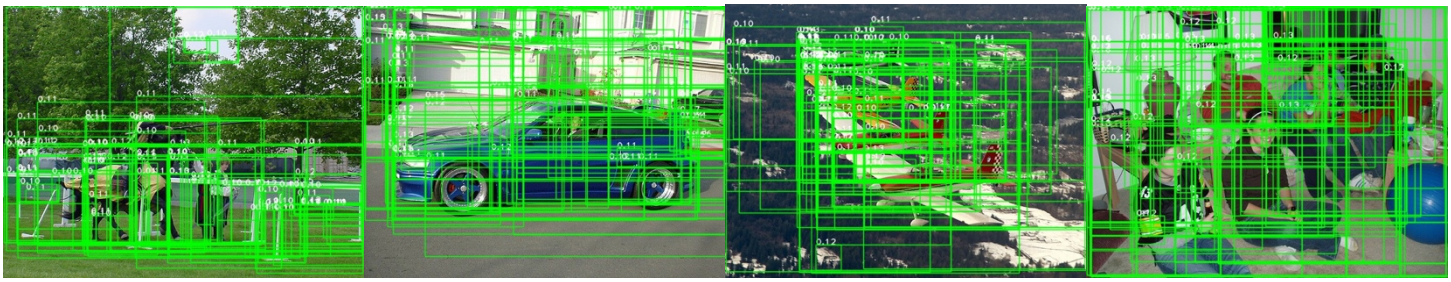
100 Highest score matching proposals with IoU > 0.5

2.2.2 Edgeboxes with $\alpha = 0.45$ and $\beta = 0.45$* (program seems to hang indefinitely when α or $\beta >= 0.9$ so 0.45 will be the middle point

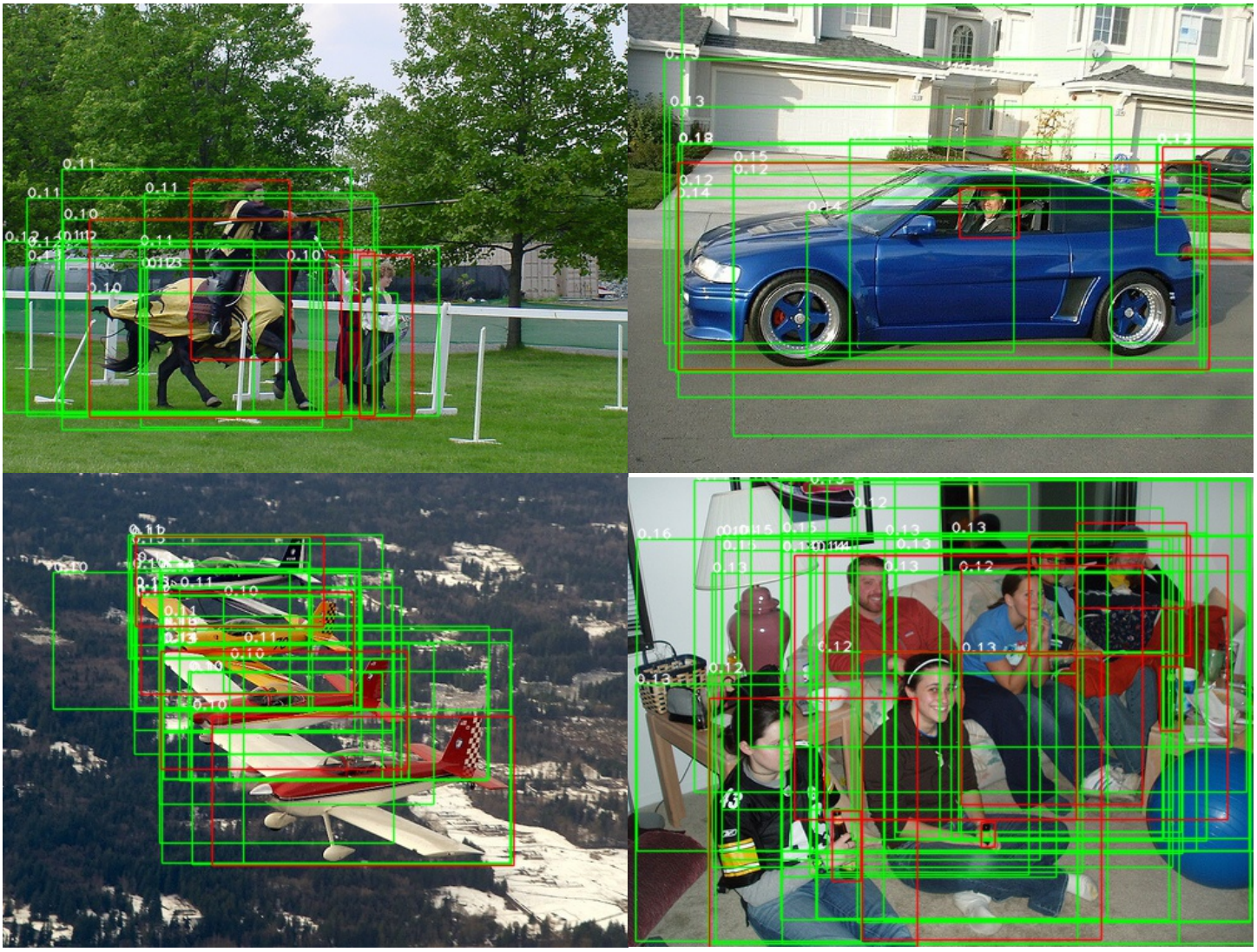


100 Highest score matching proposals with IoU > 0.5

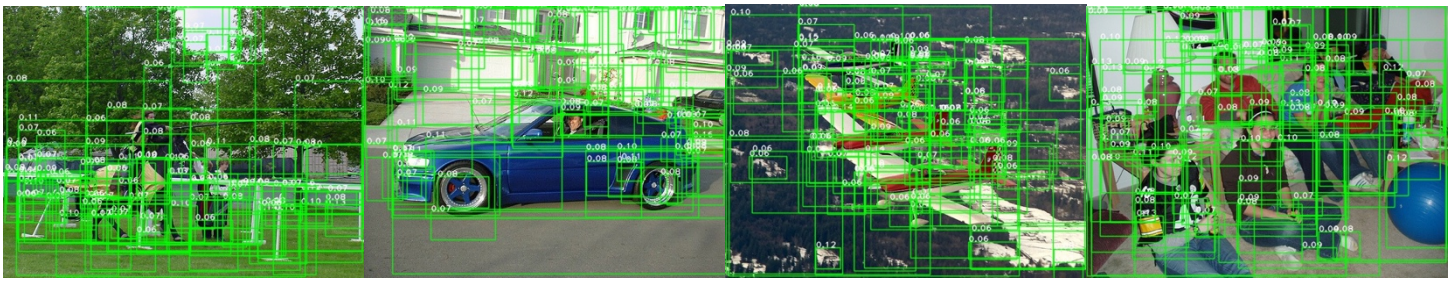
2.2.3 Edgeboxes with $\alpha = 0.65$ and $\beta = 0.75$ (standard paramters in OpenCV)



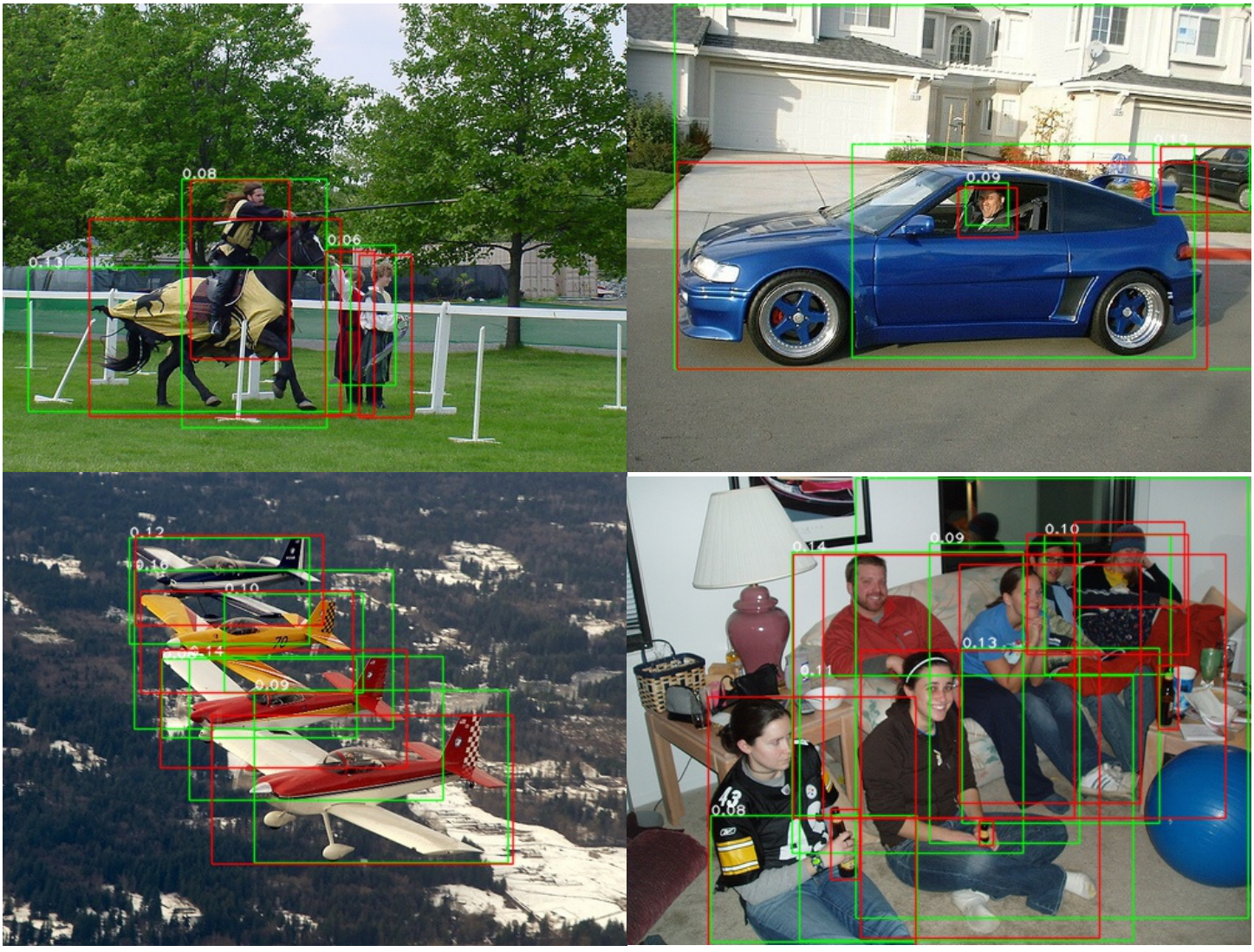
100 Highest score matching proposals with IoU > 0.5



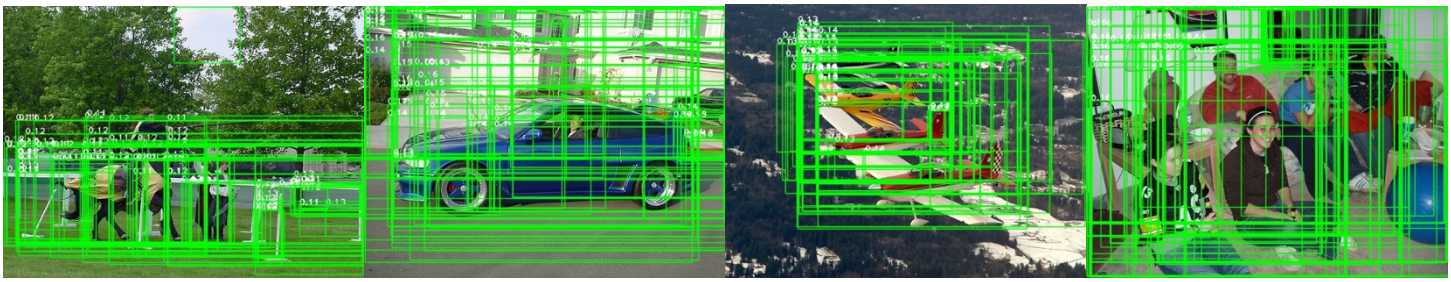
2.2.4 Edgeboxes with $\alpha = 0.85$ and $\beta = 0.35$



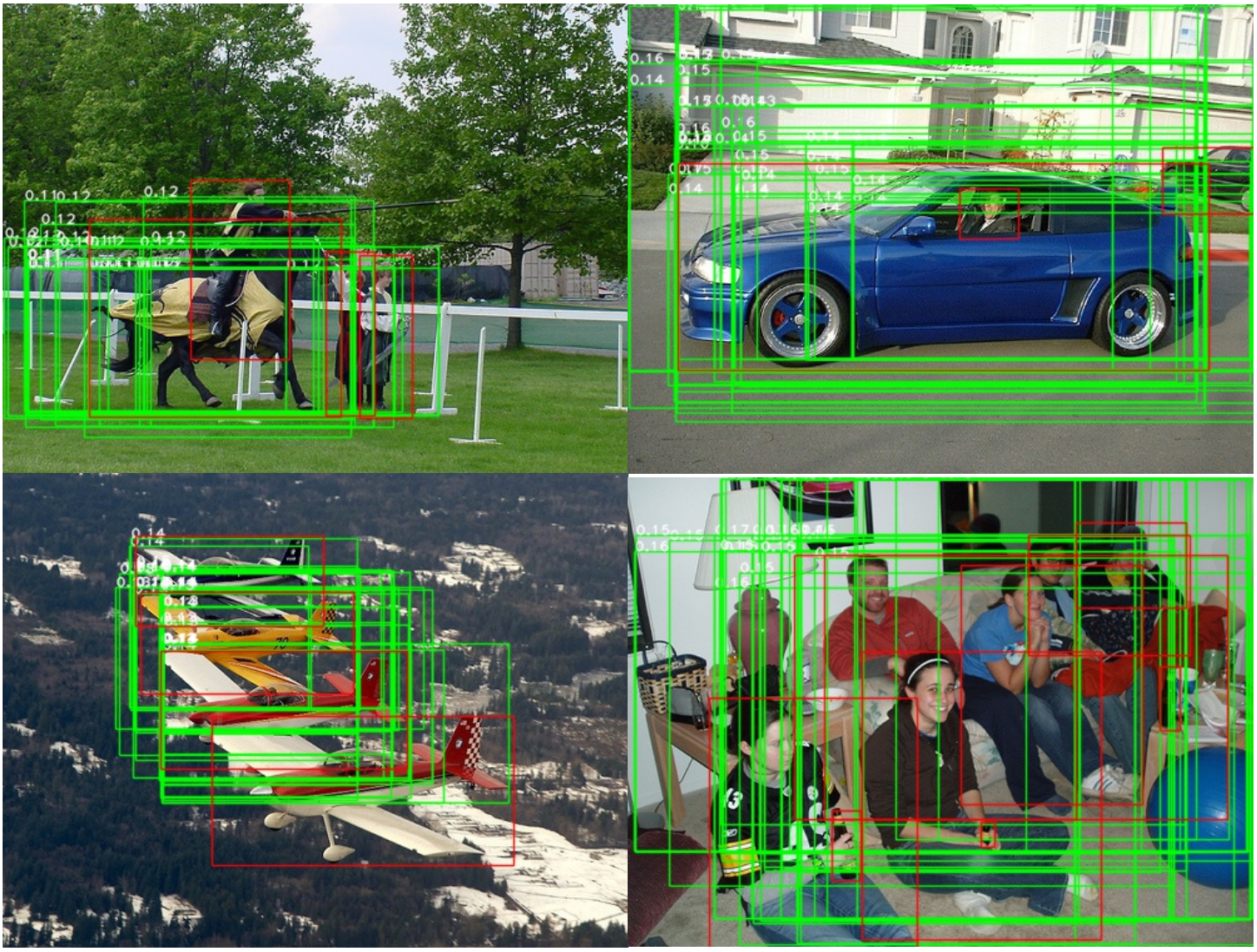
100 Highest score matching proposals with IoU > 0.5



2.2.5 Edgeboxes with $\alpha = 0.85$ and $\beta = 0.85$



100 Highest score matching proposals with IoU > 0.5



3. IoU (Intersection over Union) definition

Intersection over union or IoU is an evaluation of metric used to measure the accuracy of an object detector on a particular dataset.[1]

It is calculated using the following formula: $\text{IoU} = \frac{Area\ of\ Overlap}{Area\ of\ Union}$

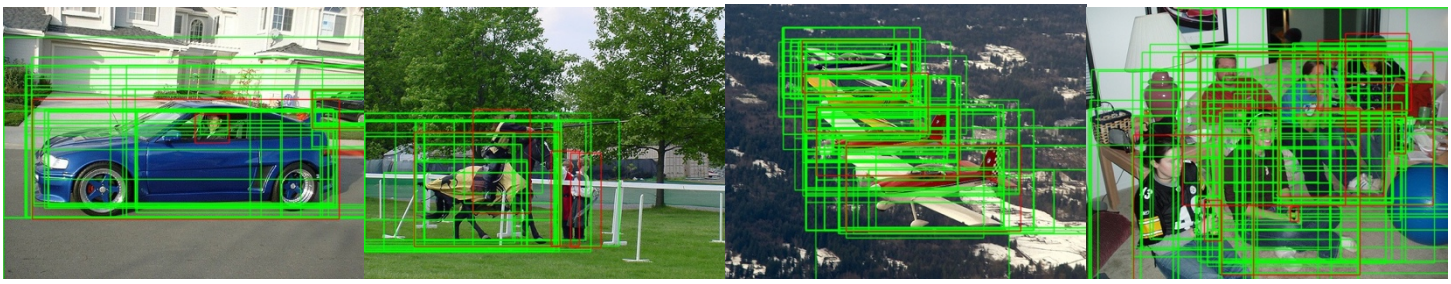
3.1 Recall table (over all proposals)

Algorithm + strategy/parameters	Recall value			
	Knight image	Car image	Airplanes image	Gathering image
SS + Color strategy	0.5	1.0	1.0	0.9
SS + Texture strategy	0.5	1.0	1.0	0.9
SS + Size strategy	0.5	1.0	1.0	0.9
SS + Fill strategy	0.5	1.0	1.0	0.9
SS + Combined strategy	0.5	1.0	1.0	0.9
Edgeboxes $\alpha = 0.25, \beta = 0.85$	0.5	1.0	0.5	0.4
Edgeboxes $\alpha = 0.45, \beta = 0.45$	0.75	1.0	1.0	0.7
Edgeboxes $\alpha = 0.65, \beta = 0.75$	1.0	1.0	1.0	0.9
Edgeboxes $\alpha = 0.85, \beta = 0.35$	0.75	1.0	1.0	0.7
Edgeboxes $\alpha = 0.85, \beta = 0.85$	1.0	1.0	1.0	0.9

Recall was computed taking with the following formula: True Positives / (True Positives + False Negatives)

From recall table we can see that Selective search algorithm return similar recall values where the strategy applied has little or no relevance to the final recall value, even if we combined all strategies into a single strategy. On the other hand, in edge boxes algorithm is very important to select the appropriate values since the step size (α) and the threshold (β) take care of the density of the sampling and directly affect the number of proposals edge boxes algorithm generates.

3.2 Display all proposals with property IoU >= 0.5



Selective search + color strategy all proposals with IoU >= 0.5 in green rectangles along ground truth in red rectangles

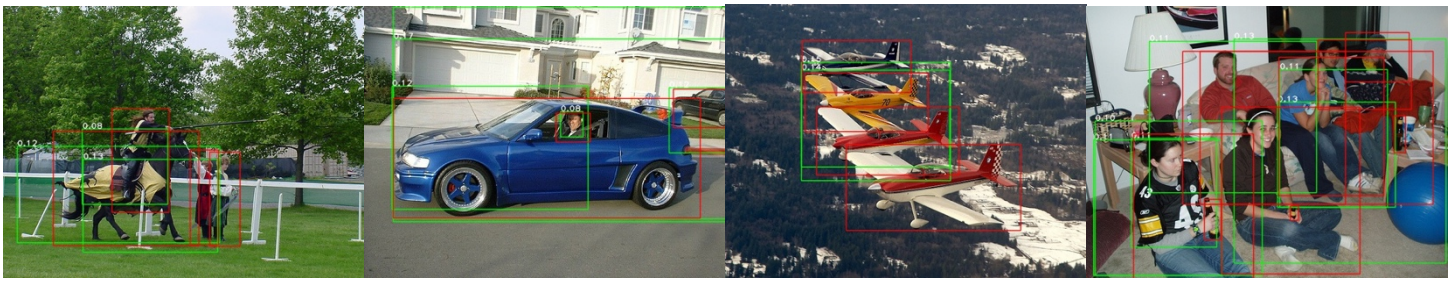
[1] Rosebrock, Adrian, "Intersection over Union (IoU) for object detection" November 7, 2016. Accessed September 25, 2020, https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/

Selective search + fill strategy all proposals with IoU >= 0.5 in green rectangles along ground truth in red rectangles

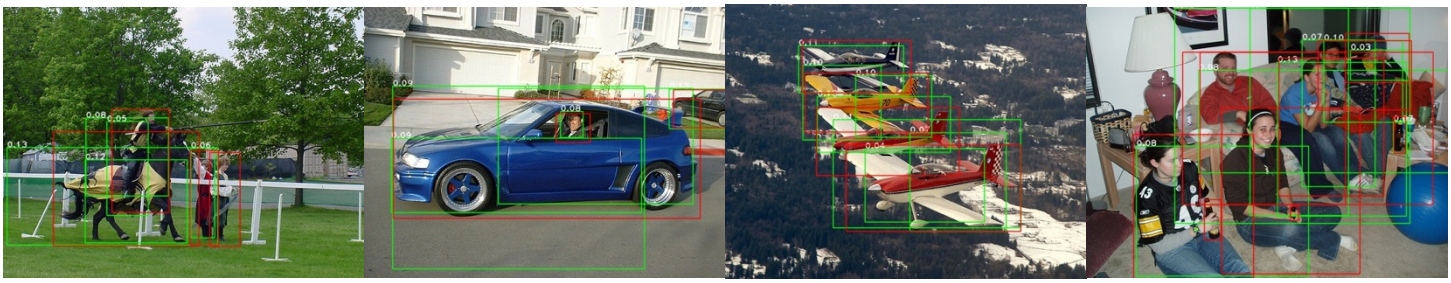
Selective search + size strategy all proposals with IoU >= 0.5 in green rectangles along ground truth in red rectangles

Selective search + texture strategy all proposals with IoU >= 0.5 in green rectangles along ground truth in red rectangles

Selective search + combined strategy all proposals with IoU >= 0.5 in green rectangles along ground truth in red rectangles

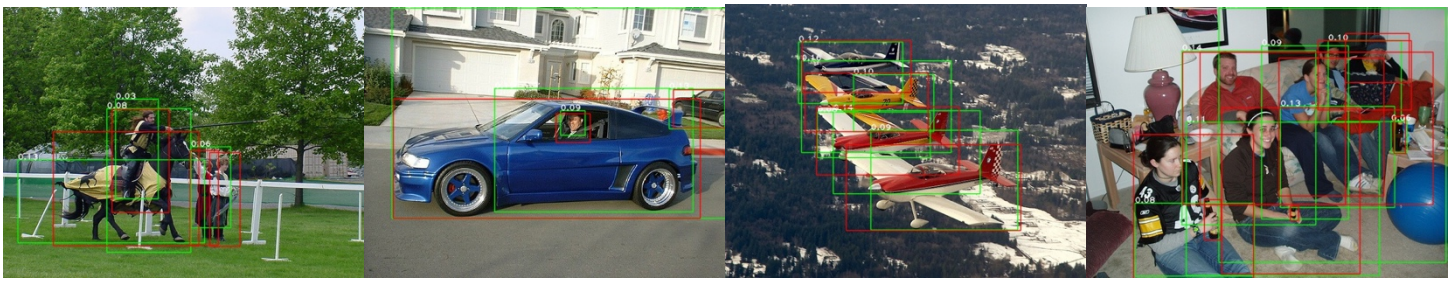


Edgeboxes with $\alpha = 0.25$, $\beta = 0.85$ all proposals with IoU >= 0.5 in green rectangles along ground truth in red rectangles

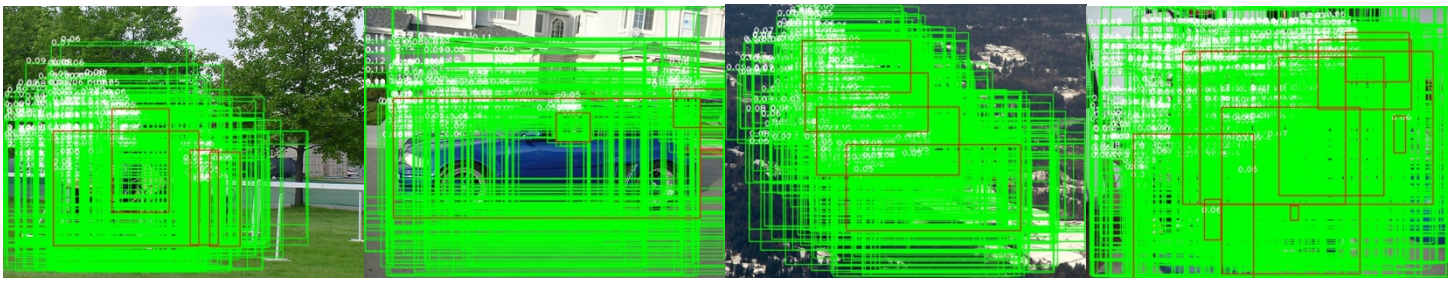


Edgeboxes with $\alpha = 0.45$, $\beta = 0.45$ all proposals with IoU >= 0.5 in green rectangles along ground truth in red rectangles

Edgeboxes with $\alpha = 0.65$, $\beta = 0.75$ all proposals with IoU >= 0.5 in green rectangles along ground truth in red rectangles



Edgeboxes with $\alpha = 0.85$, $\beta = 0.35$ all proposals with IoU >= 0.5 in green rectangles along ground truth in red rectangles



Edgeboxes with $\alpha = 0.85$, $\beta = 0.85$ all proposals with IoU >= 0.5 in green rectangles along ground truth in red rectangles

4. Conclusions

Is hard to tell just with recall value which algorithm performs better. As we can appreciate in 3.1 recall table, edge boxes algorithm performs equal or better than selective search in most scenarios under the condition that correct parameters for edge boxes are chosen. Additionally, it is worth mentioning that Edge boxes with standard parameters α and β generate more than twice the number of proposals than selective search, and when $\alpha = 0.85$, $\beta = 0.85$ it generates 10,000 proposals this is because when alpha is directly proportional to the density of the sampling and β being the threshold for Edgeboxes algorithm it would compute more precise proposals if β value is close to the IoU value we require for estimations. More proposals in edge boxes algorithm gives it a clear advantage in the recall value. It will be worth to calculate precision and maybe increase the IoU to see how the performance changes in both algorithms.