Angel Nieto 2211798052

Homework Assignment #2

1. Selective search
   1. Source code

homework.py # class to read a filename, parse xml associated with that name an run selective search and edge boxes

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)

args = parser.parse\_args()

utils.parse\_xmlfile(args.filename)

ss.run\_selective\_search(args.filename)

es.run\_edge\_boxes(args.filename)

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

import xml.etree.ElementTree as ET

import cv2

gt\_boxes = []

NUM\_PROPOSALS = 100

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)

# 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 compute\_recall(test\_box):

for box in gt\_boxes:

if intersection\_over\_union(test\_box, box) > 0.5:

return True

return False

selective\_search.py # run ss algorithm for each strategy existent in cv2.ximgproc.segmentation

import cv2

import utils

def run\_selective\_search(image\_file):

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\_color, strategy\_texture, strategy\_size, strategy\_fill, strategy\_combined]

for strategy in strategies:

ss.clearStrategies()

ss.addStrategy(strategy)

ss.switchToSelectiveSearchFast()

# ss.switchToSelectiveSearchQuality()

bboxes = ss.process()

image\_output = img.copy()

# draw ground truth

utils.draw\_groundtruth(image\_output)

# iterate over all the region proposals

for i, rect in enumerate(bboxes):

# draw rectangle for region proposal till numShowRects

if (i < utils.NUM\_PROPOSALS):

x, y, w, h = rect

if utils.compute\_recall((x , y, x + w, y + h)):

cv2.rectangle(image\_output, (x, y), (x + w, y + h), (0, 255, 0), 1, cv2.LINE\_AA)

else:

break

cv2.imshow("Selective search output " + str(strategy), image\_output)

cv2.waitKey(0)

cv2.destroyAllWindows()

1. Edge boxes

2.1 Source code

import cv2

import numpy as np

import utils

NUM\_PROPOSALS = 100

def run\_edge\_boxes(image\_file):

img = cv2.imread('./JPEGImages/' + image\_file + '.jpg')

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()

boxes, scores = edge\_boxes.getBoundingBoxes(edges, orientation)

# draw ground truth

utils.draw\_groundtruth(img)

if len(boxes) > 0:

boxes\_scores = zip(boxes, scores)

for i, b\_s in enumerate(boxes\_scores):

if (i < utils.NUM\_PROPOSALS):

box = b\_s[0]

x, y, w, h = box

if utils.compute\_recall((x , y, x + w, y + h)):

cv2.rectangle(img, (x, y), (x + w, y + h), (0, 255, 0), 1, cv2.LINE\_AA)

score = b\_s[1][0]

print("score={:f}".format(score))

else:

break

cv2.imshow("Edgeboxes output", img)

cv2.waitKey(0)

cv2.destroyAllWindows()

2.2 Results

1. Conclusion

Seems like in all circumstances edge\_boxes algorithms performs better thatn selective\_search with any kind of strategy not only obtaining more proposals than selective search but also higher scores in the proposals that satisfy IoU > 0.5