EE148 HW2

Jinrui Hou

https://github.com/kinreehou/caltech-ee148-spring2020-hw02

Deliverable #1

Done

Deliverable #2

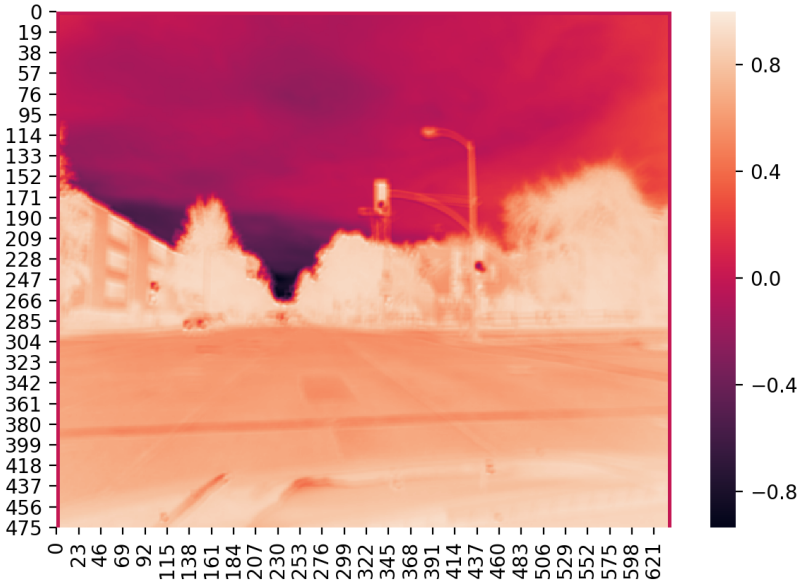
1.

- 1 First pick a template. I just cropped a red light image manually and normalized it.
- 2 Then find all candidate areas (the same size as the template) in a photo and perform 3d convolution (RGB channels altogether). In this step, a heatmap of the original photo is generated.
- 3 Find the maximum value pixel in the heatmap and explore all its connected areas above the threshold by recursion. The pixel with the maximum value is stored as one of the 'centers'.
- 4 Repeat last step to get all 'centers' above the threshold.
- 5 Return all the bounding boxes that center at 'centers' in step3. Bounding boxes are of the same size as the template.

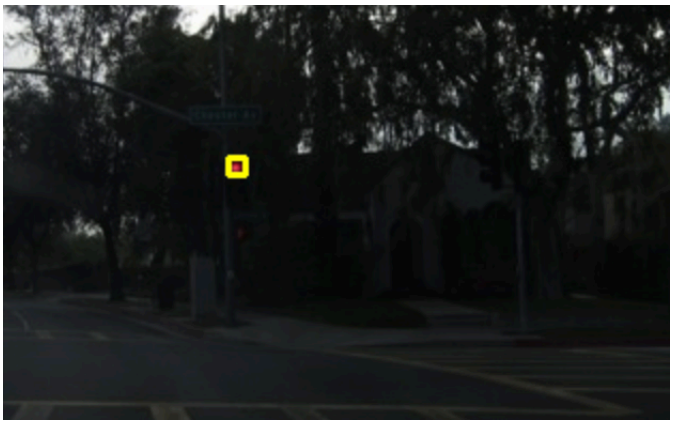
2.

Template:

Heatmap:



3.

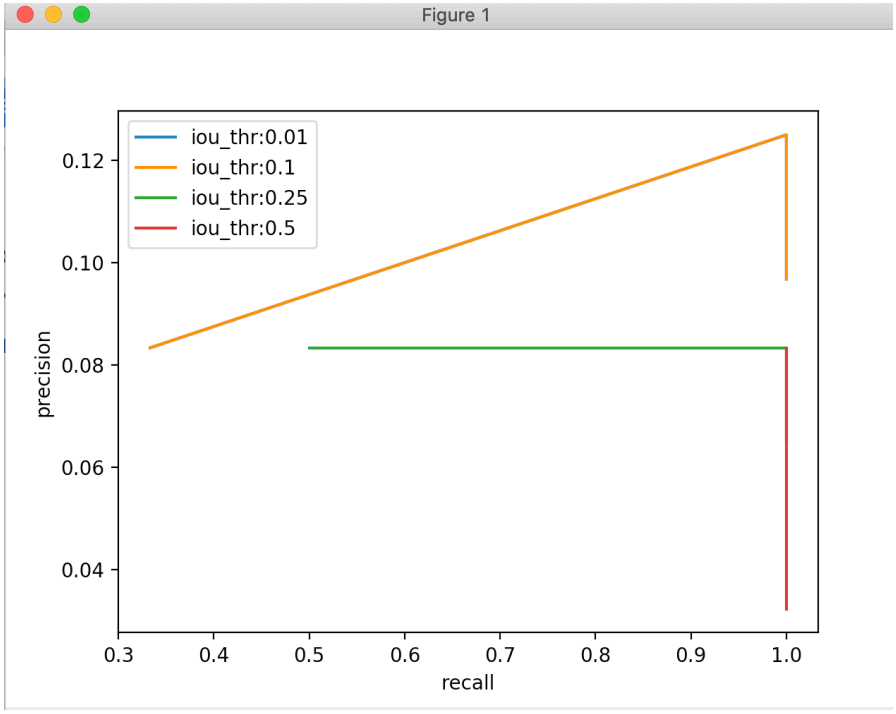


These images are easy for the algorithms because the red light show a sharp contrast to the background. Besides, the size and color of the red light is similar to the template.

4.

These images are hard for the algorithms because there are no red lights or red lights are not in sharp contrast to the background. In these cases, it's not easy to find obvious peaks in the heatmap.

5.



The result is not satisfying. It seems that in most cases, the recall rate is 1. By lowering the IOU threshold, I got some cases where the recall rate is less than 1.

There's a trend that when IOU threshold is lower, the precision rate is higher. The IOU threshold filters out predicted bounding boxes where the confidence score is low.

Deliverable #3

1. Matched filter algorithm

```
import os
import numpy as np
import json
from PIL import Image
import sys
import seaborn as sns
import matplotlib.pyplot as plt
import cv2
sys.setrecursionlimit(20000)

def compute_convolution(I, T, stride=None):
    '''
    This function takes an image <I> and a template <T> (both numpy arrays)
    and returns a heatmap where each grid represents the output produced by
    convolution at each location. You can add optional parameters (e.g.
stride,
    window_size, padding) to create additional functionality.
    '''
    (img_height,img_width,channeli) = np.shape(I)
    (t_height,t_width, channelt) = np.shape(T)
```

```
    '''
    BEGIN YOUR CODE
    '''

    #bad performance if no negative values
    i_norm = I[:,:,:]/255.0*2-1
    t_norm = T[:,:,:]/255.0*2-1

    #print("normalization done")

    heatmap = [[0]*(img_width) for _ in range(img_height)]

    for i in range(t_height//2, img_height-t_height//2):
        for j in range(t_width//2, img_width-t_width//2):
            test_area = i_norm[i-t_height//2:i+t_height//2+1, j-
t_width//2:j+t_width//2+1,:]
            #print(test_area)
            heatmap[i][j]=np.sum(test_area*t_norm)

    heatmap = heatmap/np.amax(heatmap)
    '''
    END YOUR CODE
    '''
    #print("max point",np.amax(heatmap))
    #print("max point posi",np.where(heatmap==np.amax(heatmap)))
    #print("gt",np.sum(t_norm*t_norm))

    return heatmap


def predict_boxes(heatmap,T,threshold=0.5): #can avoid overlapped false alarms
    '''
    This function takes heatmap and returns the bounding boxes and associated
    confidence scores.
    '''

    '''
    BEGIN YOUR CODE
    '''
    threshold = threshold*np.amax(heatmap)

    temp_h=int(T.shape[0]//2)
    temp_w=int(T.shape[1]//2)
    #print(temp_h)
    origin_map = np.copy(heatmap)

    def explore(i,j,cnt):
        if heatmap[i][j]<threshold or cnt>10000:
```

```
            return [[],[]]

        heatmap[i][j]=0
        coords = [[i],[j]]
        if i>=1:
            res1 = explore(i-1,j,cnt+1)
            coords[0]+=res1[0]
            coords[1]+=res1[1]
        if i<len(heatmap)-1:
            res2 = explore(i+1,j,cnt+1)
            coords[0]+=res2[0]
            coords[1]+=res2[1]
        if j>=1:
            res3 = explore(i,j-1,cnt+1)
            coords[0]+=res3[0]
            coords[1]+=res3[1]
        if j<len(heatmap[0])-1:
            res4 = explore(i,j+1,cnt+1)
            coords[0]+=res4[0]
            coords[1]+=res4[1]
        return coords

    connected_area = []
    for i in range(len(heatmap)):
        for j in range(len(heatmap[0])):
            if heatmap[i][j]>=threshold:
                coords = explore(i,j,0)
                tl_row = min(coords[0])
                tl_col = min(coords[1])
                br_row = max(coords[0])
                br_col = max(coords[1])
                connected_area.append([tl_row,tl_col,br_row,br_col])
    #print(connected_area)

    boxes_set = set()
    for tl_row,tl_col,br_row,br_col in connected_area:
        max_conv = np.amax(origin_map[tl_row:br_row+1, tl_col:br_col+1])
        #print(origin_map[tl_row:br_row+1, tl_col:br_col+1])
        #print(max_conv)
        center_posi = np.where(origin_map==max_conv)
        #print(center_posi)

        center_r, center_c = -1, -1
        for r,c in zip(center_posi[0], center_posi[1]):
            if tl_row<=r<=br_row and tl_col<=r<=br_col:
                center_r, center_c=r, c
        #print(center_r,center_c)
        if center_r<0:
            continue
```

```
            tl_row = max(center_r-temp_h,0)
            tl_col = max(center_c-temp_w,0)
            br_row = min(center_r+temp_h,len(heatmap))
            br_col = min(center_c+temp_w,len(heatmap[0]))
            score = origin_map[center_r][center_c]      ##### take the convolution
result of the center as score

            #change the type to 'int', numpy.float64 cannot be serialized by JSON
            tl_row = int(tl_row)
            tl_col = int(tl_col)
            br_row = int(br_row)
            br_col = int(br_col)
            score = float(score)
            print('score type', type(score))

            boxes_set.add((tl_row,tl_col,br_row,br_col,score))

    boxes = [list(x) for x in boxes_set]

    return boxes



def detect_red_light_mf(I):
    '''
    This function takes a numpy array <I> and returns a list <output>.
    The length of <output> is the number of bounding boxes predicted for <I>.
    Each entry of <output> is a list <[row_TL,col_TL,row_BR,col_BR,score]>.
    The first four entries are four integers specifying a bounding box
    (the row and column index of the top left corner and the row and column
    index of the bottom right corner).
    <score> is a confidence score ranging from 0 to 1.

    Note that PIL loads images in RGB order, so:
    I[:,:,0] is the red channel
    I[:,:,1] is the green channel
    I[:,:,2] is the blue channel
    '''

    '''
    BEGIN YOUR CODE
    ------------------------------------------------------------
    '''
    template_path = 'template2.jpg'
    T = Image.open(template_path)
    T = np.asarray(T)
    T = T[:,:,0:3]
    #T = T[0:15,0:15,:]
```

```
    #print("calculate Template done", T)
    heatmap = compute_convolution(I, T)
    #print("conv done")


    ax = sns.heatmap(heatmap)
    plt.show()


    output = predict_boxes(heatmap,T)
    '''
    ------------------------------------------------------------
    END YOUR CODE
    '''

    for i in range(len(output)):
        assert len(output[i]) == 5
        assert (output[i][4] >= 0.0) and (output[i][4] <= 1.0)
        print(output)
    return output

# Note that you are not allowed to use test data for training.
# set the path to the downloaded data:
data_path = '../data/RedLights2011_Medium'

# load splits:
split_path = '../data/hw02_splits'
file_names_train = np.load(os.path.join(split_path,'file_names_train.npy'))
file_names_test = np.load(os.path.join(split_path,'file_names_test.npy'))

# set a path for saving predictions:
preds_path = '../data/hw02_preds'
os.makedirs(preds_path, exist_ok=True) # create directory if needed

# Set this parameter to True when you're done with algorithm development:
done_tweaking = False

'''
Make predictions on the training set.
'''


preds_train = {}
for i in range(len(file_names_train)):
#for i in range(9,20):
    # read image using PIL:
    print(file_names_train[i],"Begin")
    I = Image.open(os.path.join(data_path,file_names_train[i]))
```

```
    # convert to numpy array:
    I = np.asarray(I)

    preds_train[file_names_train[i]] = detect_red_light_mf(I)
    #print(file_names_train[i], preds_train[file_names_train[i]])
    print(file_names_train[i], "Done")



    ########## visualize to check
    img = cv2.imread(os.path.join(data_path,file_names_train[i]))
    cv2.imshow("original",img)
    for y1,x1,y2,x2,score in preds_train[file_names_train[i]]:
        cv2.rectangle(img, (x1,y1), (x2,y2), (0,255,255),2)


    cv2.imshow("detected",img)
    cv2.waitKey(0)
    cv2.destroyAllWindows()



print(preds_train)
# save preds (overwrites any previous predictions!)
with open(os.path.join(preds_path,'preds_train.json'),'w') as f:
    json.dump(preds_train,f)

if done_tweaking:
    '''
    Make predictions on the test set.
    '''
    preds_test = {}
    for i in range(0,len(file_names_test)):

        # read image using PIL:
        I = Image.open(os.path.join(data_path,file_names_test[i]))

        # convert to numpy array:
        I = np.asarray(I)


        preds_test[file_names_test[i]] = detect_red_light_mf(I)


    # save preds (overwrites any previous predictions!)
    with open(os.path.join(preds_path,'preds_test.json'),'w') as f:
        json.dump(preds_test,f)
```

2. Code to generate PR curves

```
iou_thrs = [0.01, 0.1, 0.25, 0.5]
for i,iou_thr in enumerate(iou_thrs):
    confidence_thrs = [0.05, 0.1, 0.3, 0.5, 0.7, 0.95, 0.9999]
    tp_train = np.zeros(len(confidence_thrs))
    fp_train = np.zeros(len(confidence_thrs))
    fn_train = np.zeros(len(confidence_thrs))
    for i, conf_thr in enumerate(confidence_thrs):
        tp_train[i], fp_train[i], fn_train[i] = compute_counts(preds_train,
gts_train, iou_thr=iou_thr, conf_thr=conf_thr)

    # Plot training set PR curves

    if done_tweaking:
        print('Code for plotting test set PR curves.')
        precision = tp_train/(tp_train + fp_train)
        recall = tp_train/(tp_train + fn_train)
        #print(precision, recall)
        plt.plot(recall, precision)

plt.legend(['iou_thr:0.01', 'iou_thr:0.1', 'iou_thr:0.25', 'iou_thr:0.5'])
plt.xlabel('recall')
plt.ylabel('precision')
plt.show()
```

Deliverable #4

See attached files