TASK 1

Image Crop association

Approach:

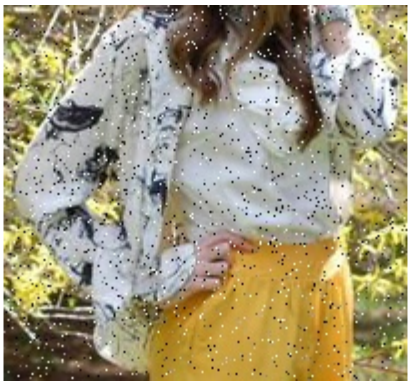
- Given a set of images and a set of crops we find the association between them as follows.
- For a single image (say *img*), we iterate through all the crops present and match it against the image.
    - We use the function ***match template*** to return the coordinates of the bounding box in the image that match the most with the given crop.
    - We define the region within the bounding box as *compare_img*.
    - Now we check how well the *compare_img* is related to the *crop*.
    - We perform a pixel wise operation. We find the difference between the pixel value(calculated separately for the individual R, G, B components) of *compare_img and crop*.
    - If the value exceeds a threshold of 15(assumption) we count the pixel as one that deviates.
    - If there are minimum number of values that deviate (say <1500), then we conclude that the *crop* is found in the *img*.

- Initially we create an empty list (say *my_list*) . For every *crop* that is found in *img,* we append the *crop* filename and the bounding box coordinates to the list.
- After iterating through all the crops for a single image, we write *my_list* against the *img* filename to our json file (*output.json)*
- We repeat the same steps for every image in the given set.
- If any given image has no crop association, we include that as well.

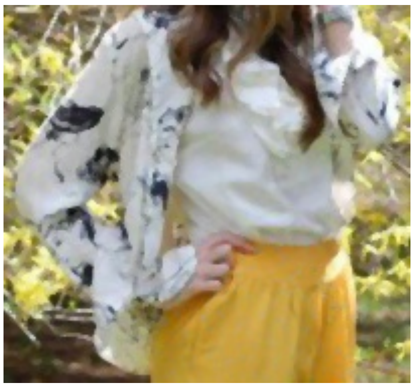
Merits:

- The script works well against crops that are affected with salt and pepper noise.
- To rectify the *crop,* we apply median filtering (*medianBlur)* to the crop and then compare it against the *img*.

(a)



(b)



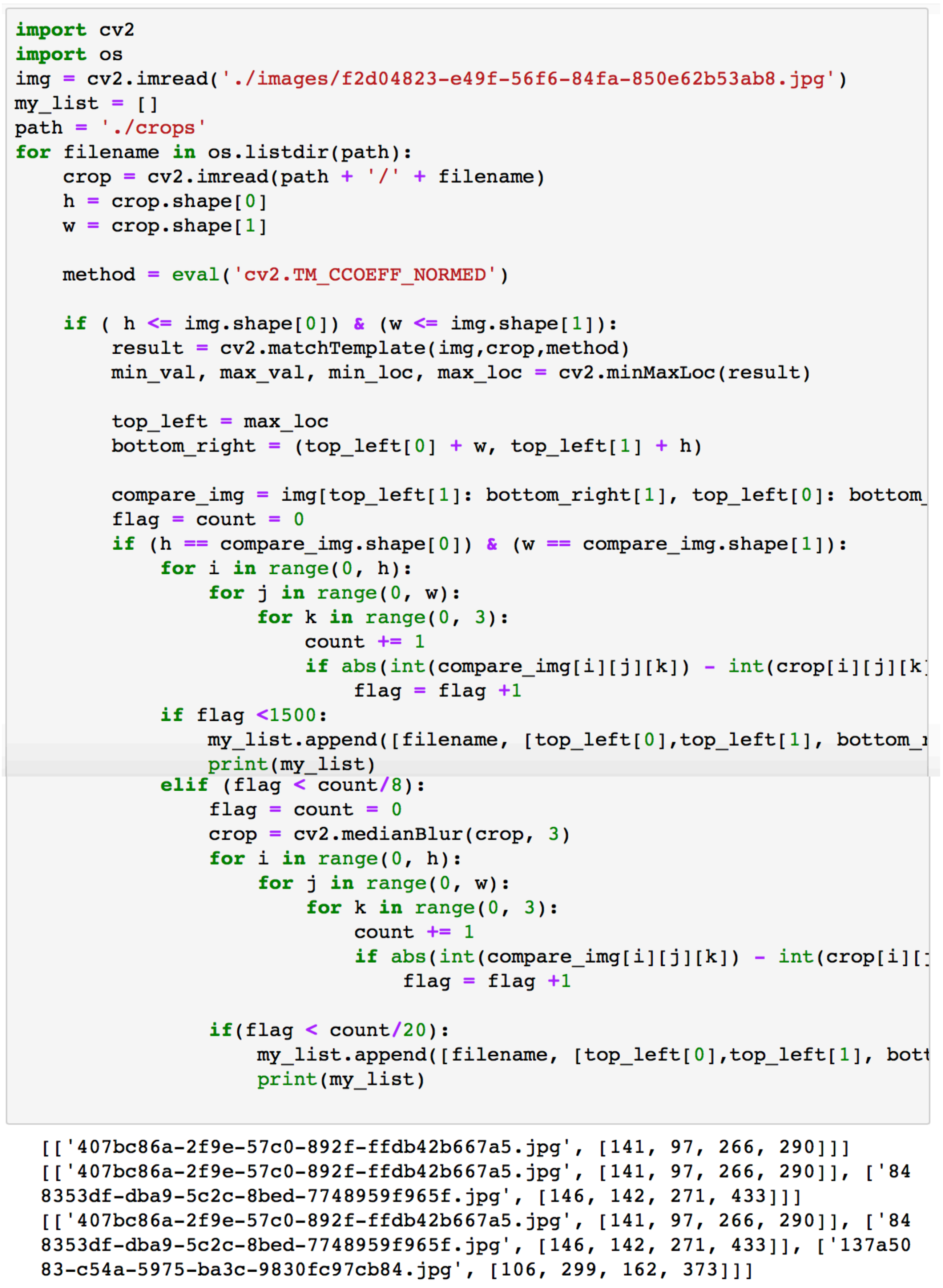
- Consider the following *crop* that is affected with salt/ pepper noise (a).
- Median blur is applied to remove the noise(b).
- After removal of noise, the *crop* is matched against the *img* and its association is returned.

Demerits:

- The script doesn't work against *crops* that are sheared/ rotated by an angle.
- To rectify this, one approach that would work would be to rotate the *img* by an angle successively and for every rotated *img,* apply template matching to it with the *crop*.

Unit Test:

- The following code block runs against an *img* 'f2d04823-e49f-56f6-84fa-850e62b53ab8.jpg' and returns its association found with 3 *crops:*
    - 07bc86a-2f9e-57c0-892f-ffdb42b667a5.jpg
    - 137a5083-c54a-5975-ba3c-9830fc97cb84.jpg
    - 848353df-dba9-5c2c-8bed-7748959f965f.jpg



```
import cv2
import os
img = cv2.imread('./images/f2d04823-e49f-56f6-84fa-850e62b53ab8.jpg')
my_list = []
path = './crops'
for filename in os.listdir(path):
    crop = cv2.imread(path + '/' + filename)
    h = crop.shape[0]
    w = crop.shape[1]

    method = eval('cv2.TM_CCOEFF_NORMED')

    if ( h <= img.shape[0]) & (w <= img.shape[1]):
        result = cv2.matchTemplate(img,crop,method)
        min_val, max_val, min_loc, max_loc = cv2.minMaxLoc(result)

        top_left = max_loc
        bottom_right = (top_left[0] + w, top_left[1] + h)

        compare_img = img[top_left[1]: bottom_right[1], top_left[0]: bottom_
        flag = count = 0
        if (h == compare_img.shape[0]) & (w == compare_img.shape[1]):
            for i in range(0, h):
                for j in range(0, w):
                    for k in range(0, 3):
                        count += 1
                        if abs(int(compare_img[i][j][k]) - int(crop[i][j][k
                            flag = flag +1
            if flag <1500:
                my_list.append([filename, [top_left[0],top_left[1], bottom_r
                print(my_list)
            elif (flag < count/8):
                flag = count = 0
                crop = cv2.medianBlur(crop, 3)
                for i in range(0, h):
                    for j in range(0, w):
                        for k in range(0, 3):
                            count += 1
                            if abs(int(compare_img[i][j][k]) - int(crop[i][
                                flag = flag +1

                if(flag < count/20):
                    my_list.append([filename, [top_left[0],top_left[1], bott
                    print(my_list)
```

```
[['407bc86a-2f9e-57c0-892f-ffdb42b667a5.jpg', [141, 97, 266, 290]]]
[['407bc86a-2f9e-57c0-892f-ffdb42b667a5.jpg', [141, 97, 266, 290]], ['84
8353df-dba9-5c2c-8bed-7748959f965f.jpg', [146, 142, 271, 433]]]
[['407bc86a-2f9e-57c0-892f-ffdb42b667a5.jpg', [141, 97, 266, 290]], ['84
8353df-dba9-5c2c-8bed-7748959f965f.jpg', [146, 142, 271, 433]], ['137a50
83-c54a-5975-ba3c-9830fc97cb84.jpg', [106, 299, 162, 373]]]
```

Evaluation Method:

- To find precision and recall, we do:

  $$\text{Precision} = \frac{TP}{TP + FP}$$

  $$\text{Precision} = \frac{TP}{TP + FN}$$

  TP = True positive (when a *crop* that is found in *img* exists in ground truth)
  FP = False positive (when a *crop* that is found in *img* does not exist in ground truth)
  FN = False Negative (when a *crop* not found in *img* exists in ground truth)

- On evaluation, the model yields the following results:
  - Precision rate: `0.819277108434`
  - Recall rate: `0.701030927835`