Image Processing Report

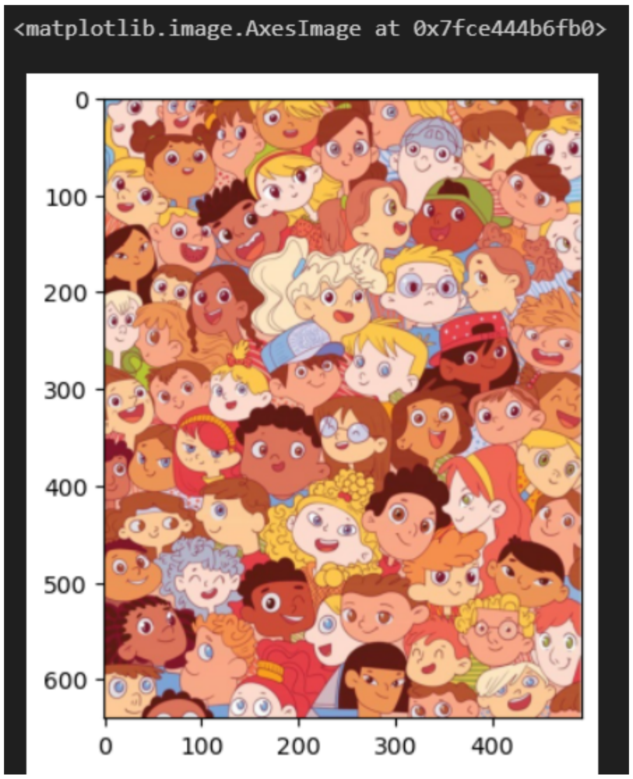
Vu Van Long, 22022501

October 12, 2024

1 Finding

1.1 General Workflow

First off, we will use mpimg.imread here to use matplotlib, and instantly convert image to numpy form



```
<matplotlib.image.AxesImage at 0x7fce444b6fb0>
```

Figure 1: Loaded image

Futhermore, We need to cut the object ourself in order to find it

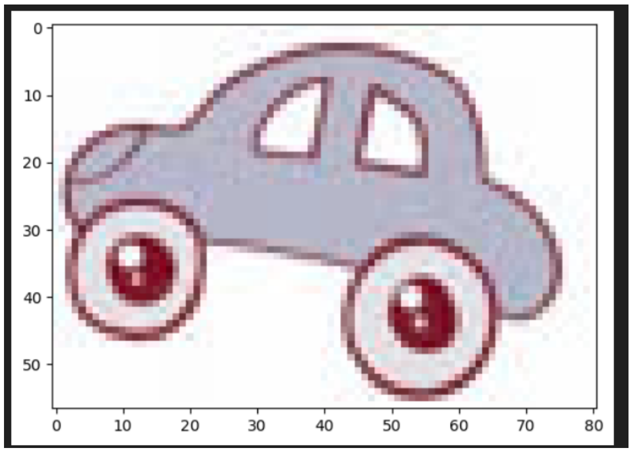
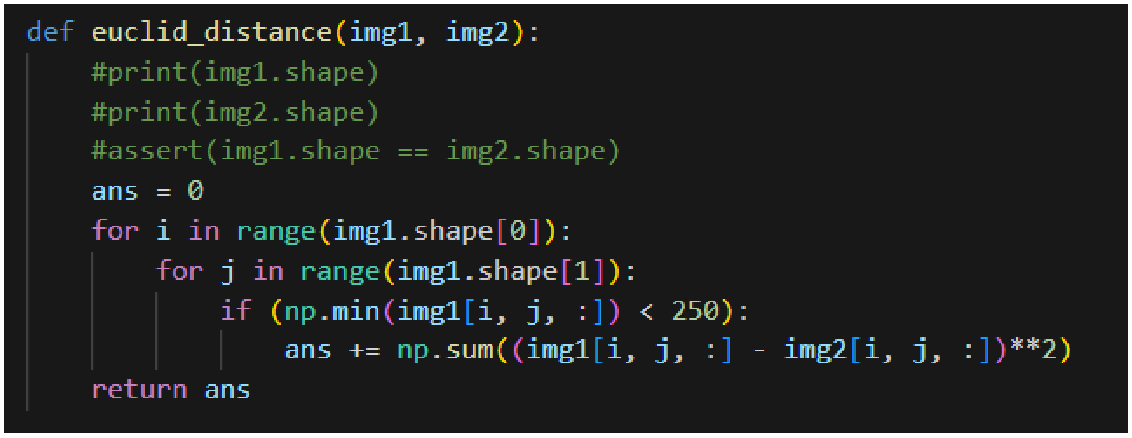


Figure 2: Cutted object

alright i forgot to save my code so all the tried-and-error experiments is gone =(((. But in short, cross-correlation doesnt work very well and require cubersome preparation. From now on i will only demonstrate my best result.

The cross-correlation gove terrible result, now i will try euclid distance. If some pixel is pure white (or semi-pure-white), we will just ignore it.



```python
def euclid_distance(img1, img2):
    #print(img1.shape)
    #print(img2.shape)
    #assert(img1.shape == img2.shape)
    ans = 0
    for i in range(img1.shape[0]):
        for j in range(img1.shape[1]):
            if (np.min(img1[i, j, :]) < 250):
                ans += np.sum((img1[i, j, :] - img2[i, j, :])**2)
    return ans
```

We may need some rescale, let use cv2. But to what scale ? Should we do it manually ? Or is there any auto method ?
If we just naively take the euclid score among each scaled image smaller image will always give smaller euclid distance, dead wrong.
OK so let try this, if the image is 2 times smaller so euclid distance will be x2.
It run a litte slow, so let add stride = 5, 6, 7, etc... Detailed implementation in jupyter notebook.

This turn out to run better then cross-correlation with gray-scale (and mean-subtract) normalize as it can utilize the color infomation.

1.2 Final Result

For 5 selected object in the code, it successfully detect 4 of them, the blue one, who knows (why his head look so alike to an ice cream). The light-green one on the left of the image is a little bit hard to see, but it is correct.

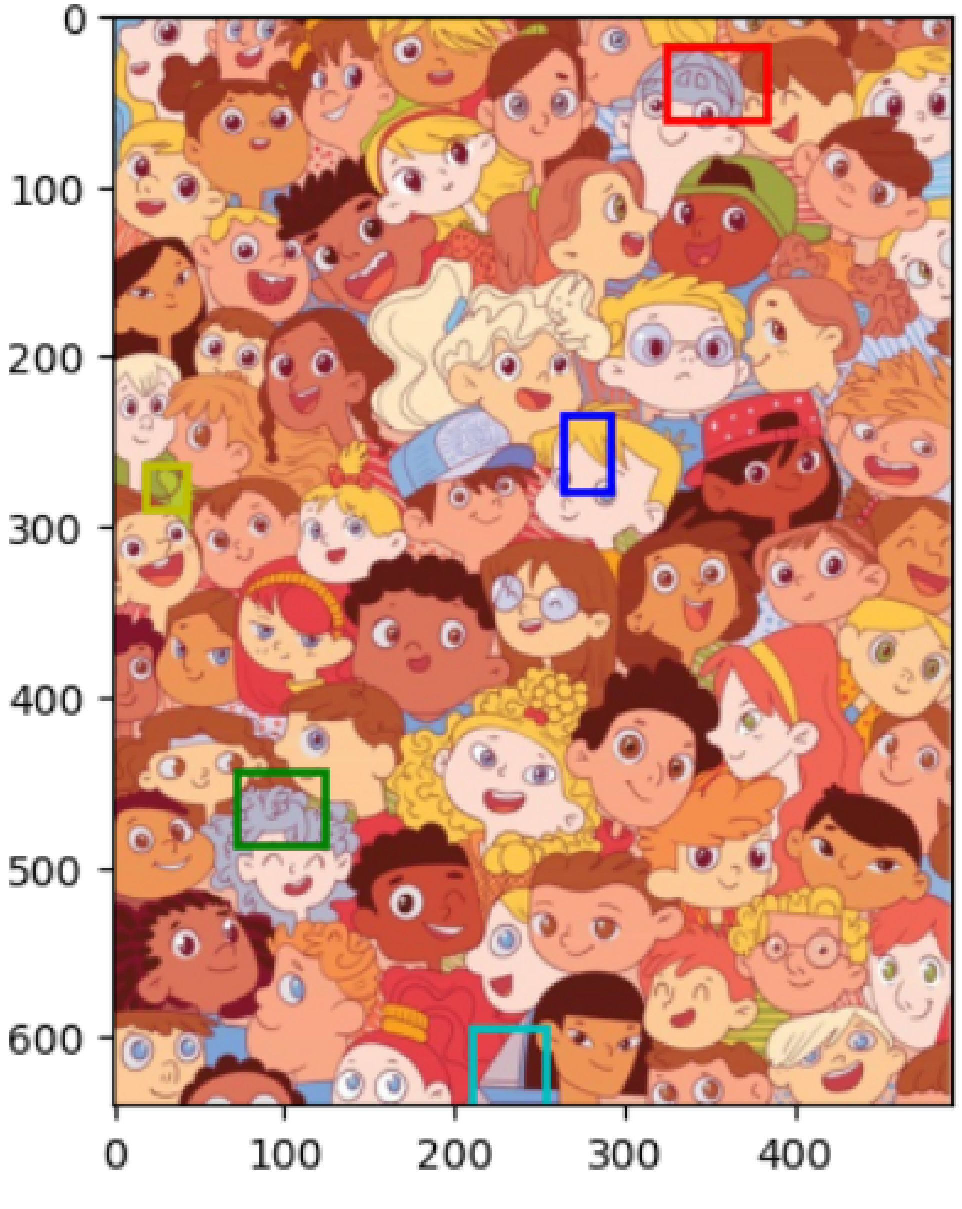


Figure 3: Final Result

2 Counting

2.1 General Workflow

This problem is somewhat less tricky than previous problem, especially for rabbit and mouse as they are distinguishable from its environment.

Let's load the image using mpimg.

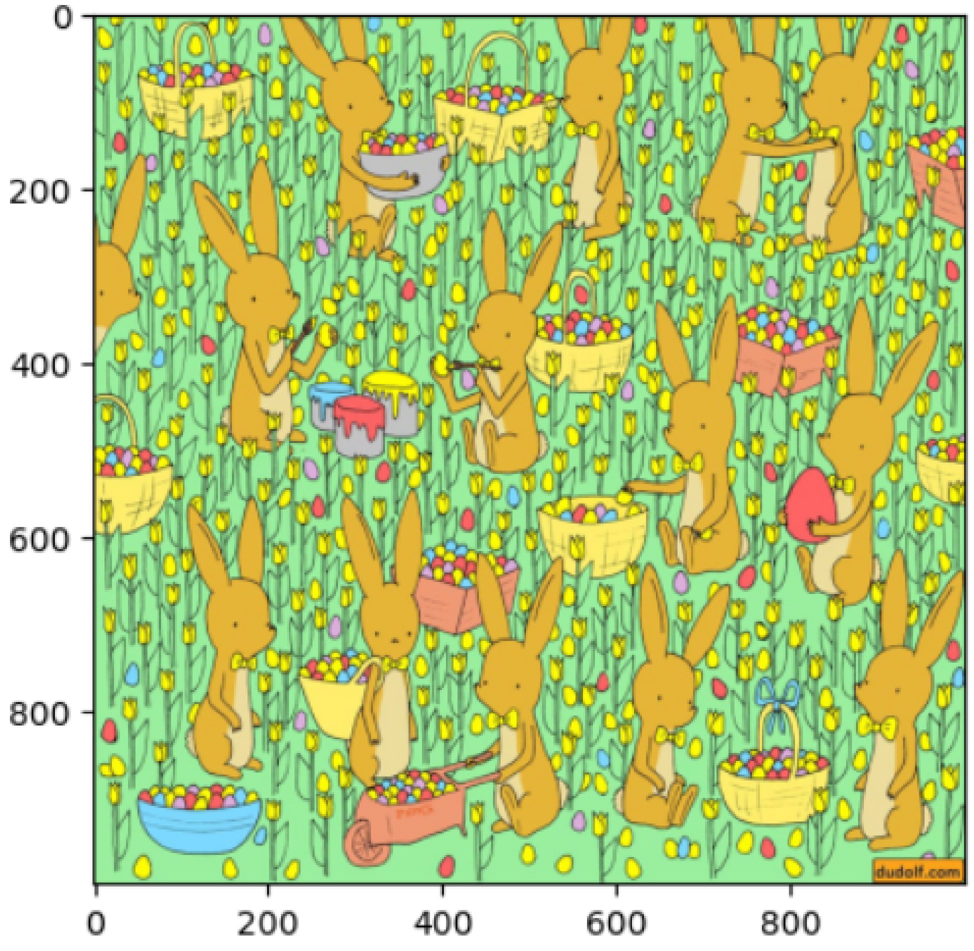
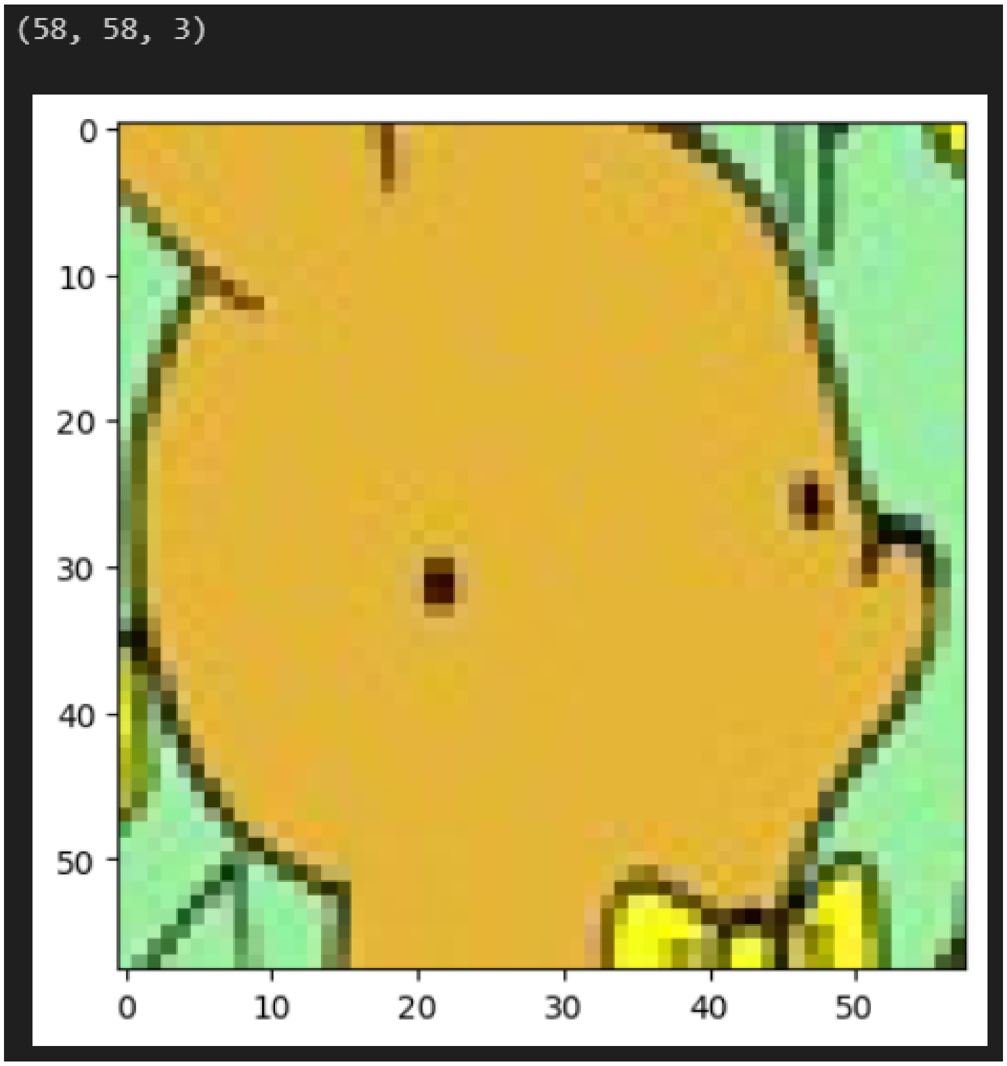


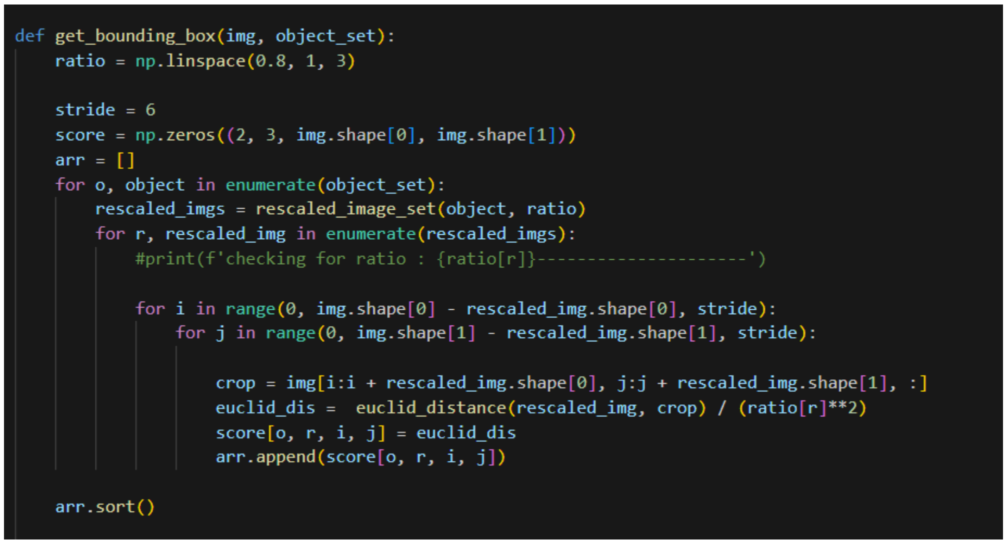
Figure 4: Loaded image

OK for this image, we can clearly see that the rabbit head is pretty unique. Let's take their head. I took 2 type of their head, one looking to the right, one to the left (or we can just flip the image).



```
(58, 58, 3)
```

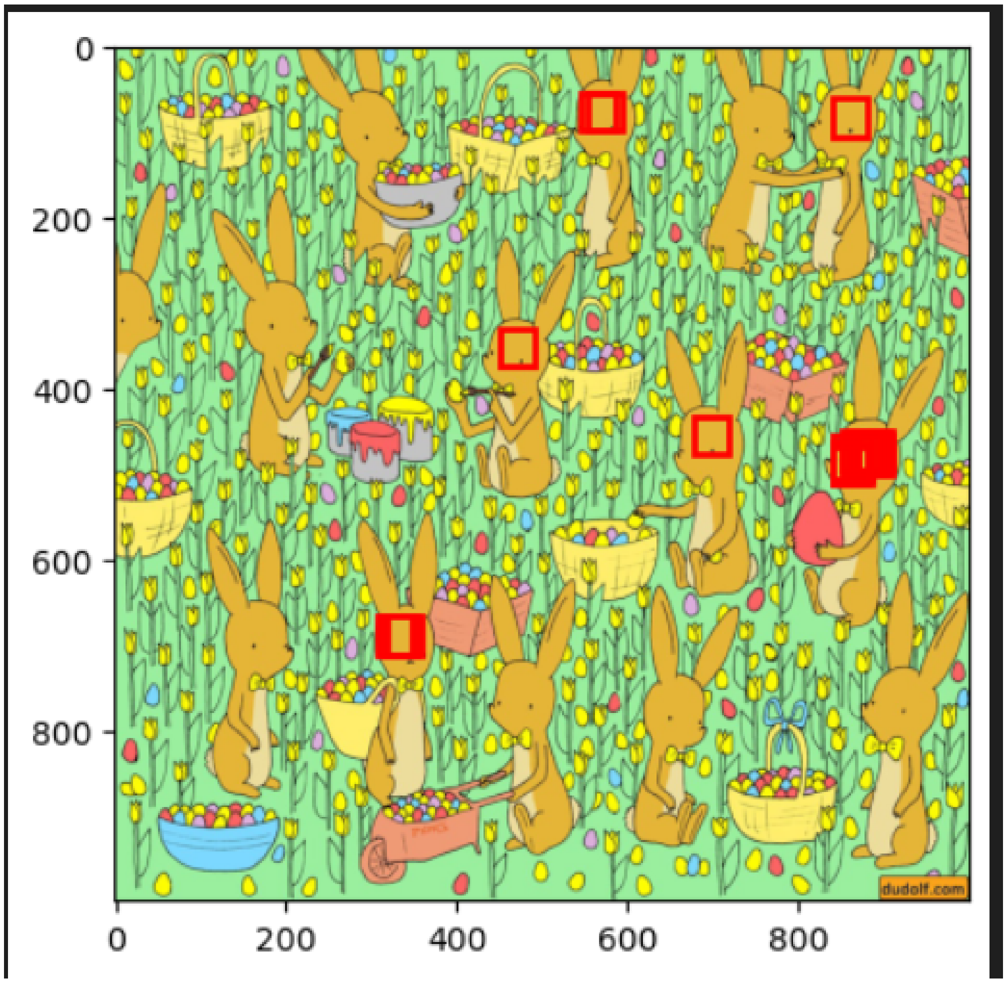
This time the rabbit head look pretty much the same (in size), let adjust our scaling a little bit and we need to take top k position (because there are many rabbits, implementation detail in code).



```python
def get_bounding_box(img, object_set):
    ratio = np.linspace(0.8, 1, 3)

    stride = 6
    score = np.zeros((2, 3, img.shape[0], img.shape[1]))
    arr = []
    for o, object in enumerate(object_set):
        rescaled_imgs = rescaled_image_set(object, ratio)
        for r, rescaled_img in enumerate(rescaled_imgs):
            #print(f'checking for ratio : {ratio[r]}---------------------')

            for i in range(0, img.shape[0] - rescaled_img.shape[0], stride):
                for j in range(0, img.shape[1] - rescaled_img.shape[1], stride):

                    crop = img[i:i + rescaled_img.shape[0], j:j + rescaled_img.shape[1], :]
                    euclid_dis =  euclid_distance(rescaled_img, crop) / (ratio[r]**2)
                    score[o, r, i, j] = euclid_dis
                    arr.append(score[o, r, i, j])

    arr.sort()
```

2.2 Final Result

Not looking so good =((((. IN conclusion, these matching method require A LOT prior knowledge.


0
200
400
600
800
0
200
400
600
800
dudolf.com