Digital Image Processing: Homework #4

Due on June, 28th, 2020 at 11:59pm

Yuyao Zhang

Tianyuan Wu
63305667

Note: The implementation of image inpainting is in "hw4.py", and the implementation of bonus problem is in "detect_inpainting.py"

Problem 1

Solution

1) Description of the process of calculating priority of boundary points
We denote the priority of the a boundary point P as $\Psi(P)$, which can be calculated by:

$$\Psi(P) = D(P) \cdot C(P)$$

where $D(P)$ is the priority in data term (highest strong edges from outside the hole hit the hole at right angles), and $C(P)$ describes the confidence term (high confidence if pixel is surrounded by known pixels).
In order to calculate $C(P)$, for a specific pixel $P(r, c)$, we just need to count the percentage of known pixels in the $n \times n$ grid which center is P (i.e. count the percentage of known pixels in the square area which has the upper-left corner as $(r - \frac{n}{2}, c - \frac{n}{2})$). In more formal terms,

$$C(P) = \frac{\#\text{ of known pixels in the } n \times n \text{ cell surrounds } P}{n \cdot n}$$

Then we calculate $D(P)$ by the following formula:

$$D(P) = ||\nabla I(P)||(\nabla^2 I(P) \cdot \vec{n})$$

Where $\nabla I(P)$ denotes the gradient of the image at pixel P, $\nabla^2 I(P)$ denotes the Laplacian of the image at pixel P, and $\vec{n}$ denotes the normal vector to the hole. By following equations, we can finally calculate the priority $\Psi(P)$.

2) Implementation of image inpainting algorithm.

I. Create mask of of the girl. First, we select some points in MATLAB, connect them to create a polygon, which is the mask shown in Fig.(1) and (2), the red points in Fig.2. denotes points I selected in MATLAB.

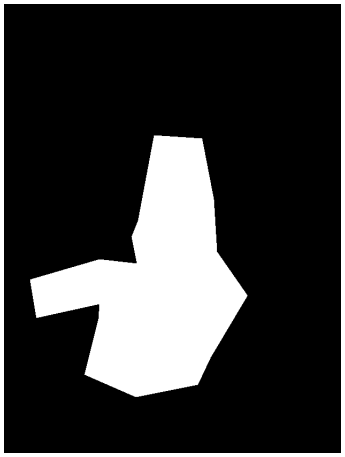


Figure 1: The mask create by selected points

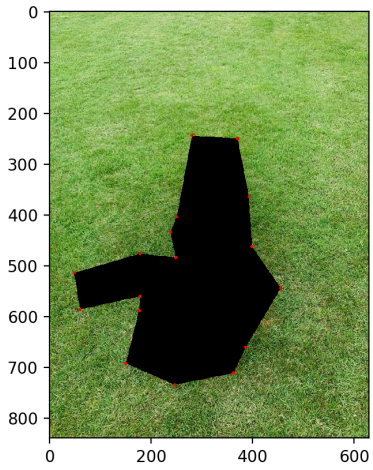


Figure 2: Use the mask to cover the girl

II. By equations shown in problem (1), we calculate the priority of boundary points, lighter points means higher priority, the result is shown in Fig.3.

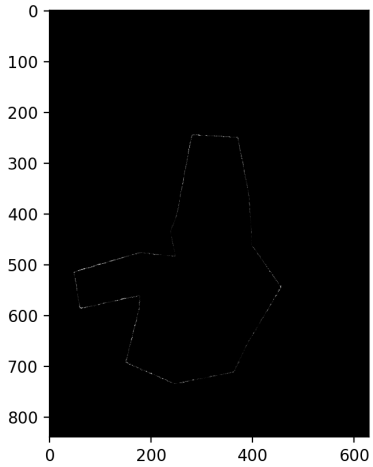
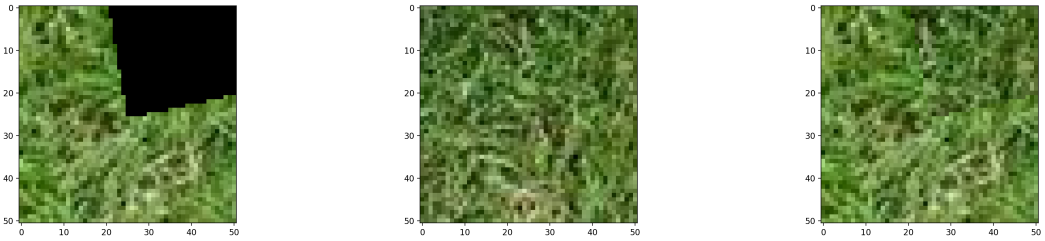


Figure 3: Use the mask to cover the girl

III. We search the whole image, and find the best matching patch, then replace the hole with pixels in the found patch. Fig.4(a) denotes the patch need to be inpainted, Fig.4(b) shows the best matching patch, and Fig.4(c) shows the result of replacement.



(a) The hole to be inpainted (b) Best matching patch (c) Result

Figure 4: Results of inpainting

IV. After the inpainting process, the whole image is shown in Fig.5, the red circle notes the inpainted area.

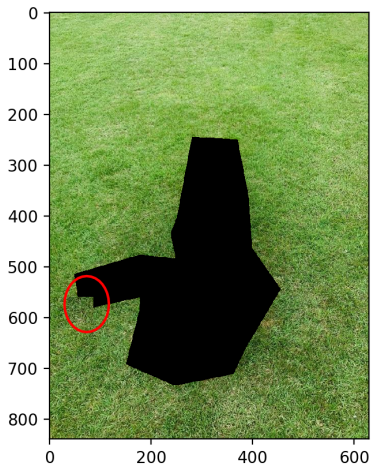


Figure 5: Image after inpainting (1 iteration)

V. After many iterations, the image is finally inpainted, and the result is shown in Fig.6 (patch_size = 51), and Fig.7. (patch_size = 11).

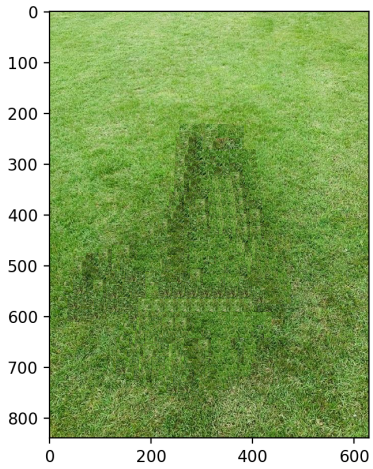


Figure 6: Image after inpainting

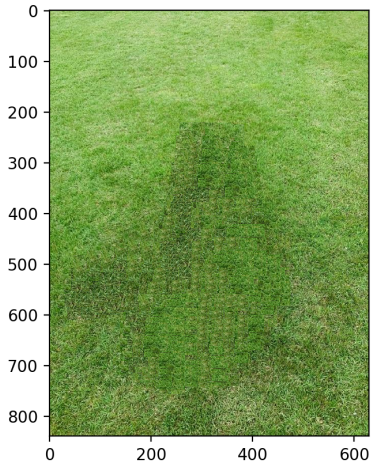


Figure 7: Image after inpainting

And for the other image "family.jpg", we also create the mask at first. The mask is shown in Fig.7 Then do

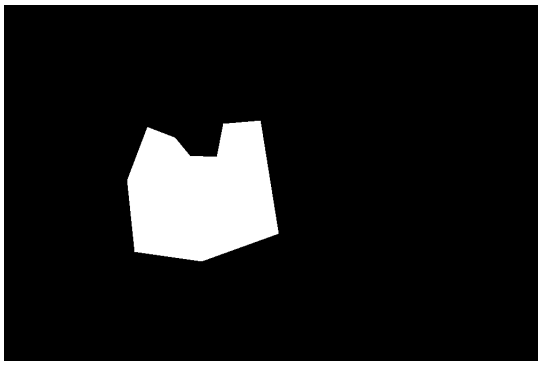


Figure 8: Mask of "family.jpg"

the patch-based image inpainting, the result is shown in Fig.8

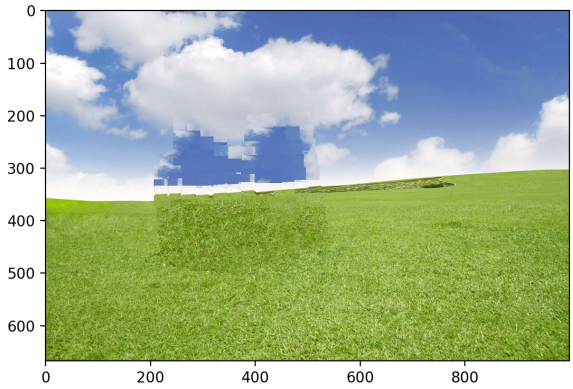


Figure 9: inpainting result of "family.jpg"

Problem 2

Solution to Bonus Problem

We can consider this problem on the view of information theory. That is, the inpainting process cannot bring extra information to the original image. It's just an algorithm to find the best matching patch, and cover the hole with the patch. Consequently, if we use a "inverse process" of the inpainting algorithm, we can observe if there are many same patches occured. We can use a small "kernel" (size $= k \times k$) to traverse the whole image, where k is smaller than the patch size. Then we can record all $k \times k$ patches in the image. Then, do the image hashing of all patches (here I use pHash). If the image is inpainted, then there must be a lot of same $k \times k$ squares (i.e. their hash value is the same). But if an image is not inpainted by this way, the probability of existing many patches with the same hash is very low.
In more formal terms:

1. Input an image I

2. Choose a patch size k, where k should less or equal that the patch size when inpainting (e.g. the input image is inpainted by patch size $= 51 \times 51$, but when detecting, use the patch size $k = 8$).
3. Enumerate all $k \times k$ patches in the input image.
4. Calculate the hash value of all patches.
5. Count the number of patches with the same hash, and sort the occur frequency in descending order.
6. If there are many patches with the same hash, then the image is inpainted with high probability.

Here are the top 10 number of patches with the same hash in inpainted image and original image and inpainted image (I enumerated all 8×8 patches in 2 images):

```
Inpainted Image: [7303, 3031, 2211, 1108, 929, 904, 883, 734, 613, 568]
Original Image:  [5926, 2637, 1956, 1021, 861, 858, 849, 680, 588, 549]
```

It's obviously that the inpainted image has much more same patches, so we can detect whether an image is inpainted by the patch-based algorithm by this way.