Andrew Quinn
EECS 332 - Machine Problem # 2
Dr. Ying Wu
Due 10/06/2020

Final homework checklist - cross off as done:

- ~~*Finish up code*~~
- ~~*Zip GitHub private repo for submission, put in GDrive*~~
- ~~*Review grading criteria*~~
- ~~*Zip up some processed images*~~
- ~~*Finish report*~~
- ~~*Export report to PDF*~~
- *Submit everything!*

You can see all of the algorithms in action with each SE in `gun_images.zip` and `palm_images.zip`. (I didn't include them all in the report this time around - too many of them.)

A full copy of my source code and images, including with several test images, has also been included.

Running `mp2.py` requires the third-party libraries Pillow and numpy to be installed, and it is best done by unzipping the full archive to make sure everything is in the right place.

MP 2 :: Morphological Operators

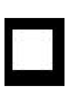
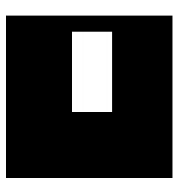
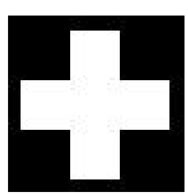
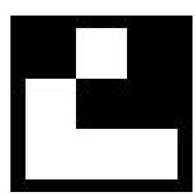
A morphological operator, in our image-based context, essentially takes in 2 images, *A* and *B*. Usually *B* is much smaller than *A*, and is called the structuring element.

In this MP, we were asked to implement 5 such morphological operators and demonstrate they worked via signal processing: erosion, dilation, opening, closing, and boundary. (Opening, closing and boundary are all relatively straightforward extensions on the first two.)

What structural elements look like

Structural elements (SEs) can be of any size, and they essentially act as inputs for per-pixel arguments.

For gun.bmp and palm.bmp, I used the following 4 structural elements to generate the 2*4*5=40 processed images, although my code does allow for larger structural elements (take a look at test.py if you want to see that yourself).

These are all found in `structure_elems/` respectively called

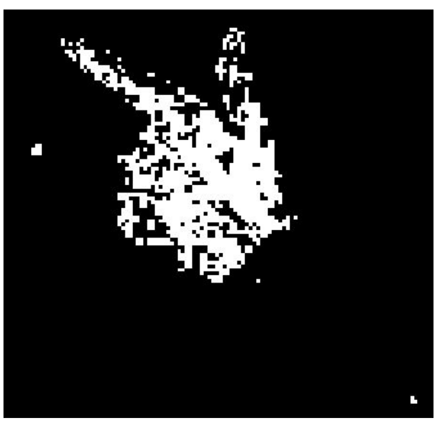
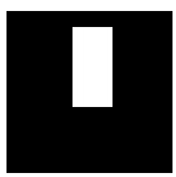
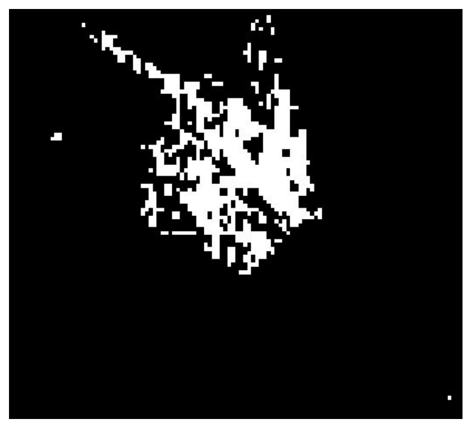
- `se_identity_1`
- `se_north_3`
- `se_cross_3`
- `se_glider_3`

`identity_1` is a 1x1 structural element; the other three are 3x3.

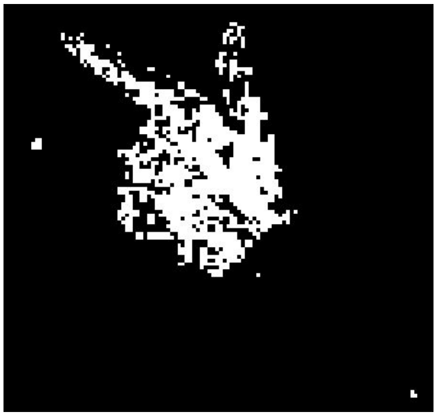
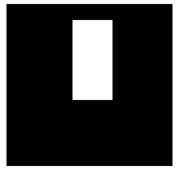
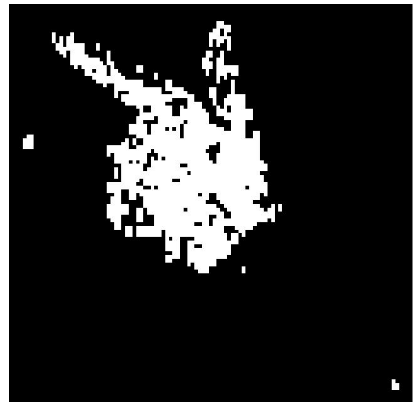
Dilation and erosion algorithm

Dilation and erosion are duals, so they are very similar in implementation. Essentially, you step through every pixel P = [x, y] in the original image A, and superimpose your centered structural element over it, to see *what values the local neighborhood of pixels around P have.* This, combined with the form of the specific structural element you are using, tells you what to do with the value of the pixel at P itself.

Because these are binary images, there are only two options - True (white) or False (black). With **erosion, pixel P is white iff every pixel in the local neighborhood covered by the white pixels of the structural element are also white.** For example, eroding an image with `se_north_3` will result in a new image where only pixels that were *both* white themselves *and* had a white pixel directly above them in the old image are still white:

erode( , ) = 

On the other hand, with **dilation, pixel P is white if *any* pixel in the local neighborhood covered by the white pixels of the structural element are also white**. Dilating an image with `se_north_3` will result in a new image where pixels that either were white themselves, *or* had a white pixel directly above them, are now white:

dilate( ,  **)** = 

`identity_1` with either of these operations returns the original image - hence its name.

Erosion and dilation are mathematical duals; in this context,

    invert(erode(A, B)) = dilate(invert(A), B-hat)

where `invert()` turns black pixels white and vice versa, and `B-hat` is just the original structural element rotated 180 degrees about its center.

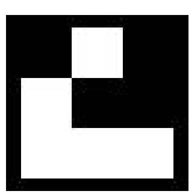
Opening, closing and boundary

Opening and closing are dead-simple application orders of dilation and erosion. My Python code for them are as close to the mathematical formalism as I'm likely to get:

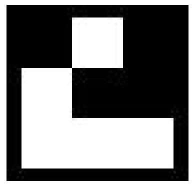
```
def opening(img, se):
      return dilate(erode(img, se), se)

def closing(img, se):
      return erode(dilate(img, se), se)
```

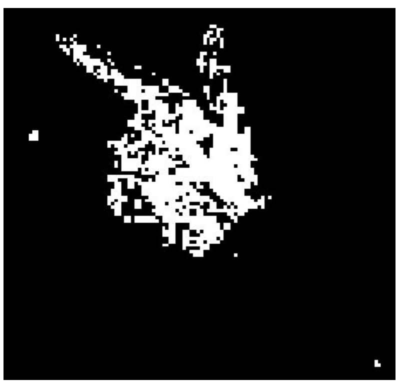
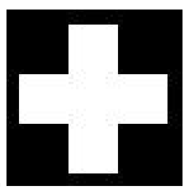
Because opening first erodes and then dilates an image, it's good for smoothing out a picture while still removing small bits of noise from the foreground.

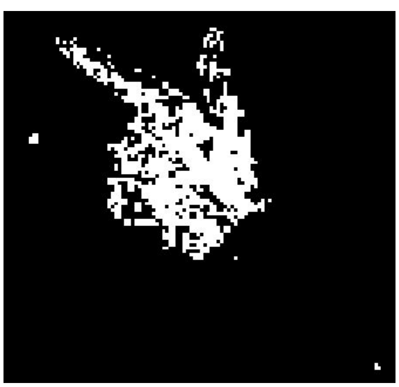
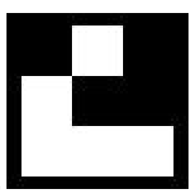
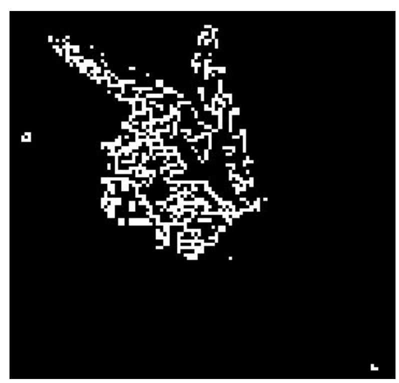
opening( , ) = 

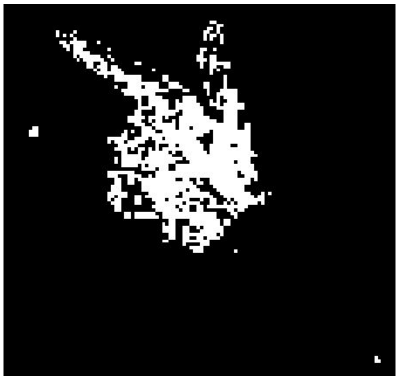
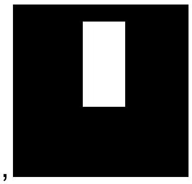
Closing, meanwhile, dilates an image first, then erodes it, which has the opposite effect of removing small *holes* from the *background* of the image.

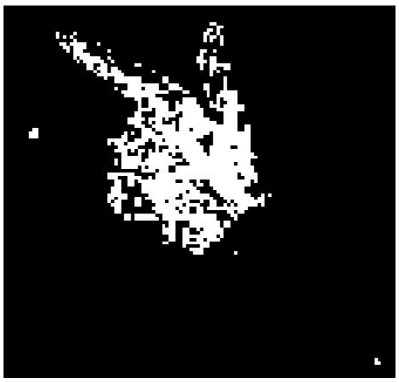
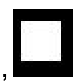
closing(, ) = 

Boundary, meanwhile, is a set difference based on erosion. Let C be the eroded picture of A, using some structural element B. Then boundary(A, B) has **white pixels for every pixel that is white in A but *not* in C**, and **black pixels everywhere else**. This has the effect of leaving only the edge pixels of contiguous regions on:

boundary(, ) = 

boundary(, ) = 

boundary(, ) = 

boundary(, ) = 

You can combine all of these for whatever effect you're looking for.

Again, **you can see all of the algorithms in action with each SE in `gun_images.zip` and `palm_images.zip`.** I only included what I thought was needed to illustrate the concepts here.