Ikhlas Attarwala / 10-14-2019 / MP #2
EECS 332 – Introduction to Computer Vision

For my results, it would be easier for you to run ***main.py*** via a python IDE. When you do, you can test the basic functions in the IDE's terminal. The 4 basic operators take in 2 arguments (the image file name and the structuring element design you want to use), for example **erosion('gun.bmp', se_square3)**, where 'se_square3' is a 3x3 2D array of 1's that I created as an example. I offer various example structuring elements, but ANY 2d binary array can be used for the structuring element. The ***boundary()*** function does not require a structuring element as it defaults to using 'se_square3', but another structuring element can be offered if necessary. Please note that the file might run with less errors through Jupyter Notebook.

Morphological Transformation is an interesting concept. Simply put, it is the processing of geometrical structures, commonly in the form of digital images but may also translate to graphs and spatial structures. This homework looks at some of the most basic morphological operators. Our test data consists of black & white images, so we apply a binary morphology, which converts the image into a subset of a 2-dimensional integer grid. The basic 4 operators I discuss below probe a larger image with a simple pre-defined shape (the ***structuring element***) to transform its elements in various ways. The larger image is an input, as is the smaller shape that we call a structuring element. The structuring element can have various shapes and designs. In my code, I provide many examples of structuring elements the user can use ranging from 1x5, 3x3, or even 9x9 sizes, all with various binary designs. My code is dynamic and can use any binary 2d-array the user provides as the structuring element. For the below definitions, let A denote the binary image, and B the structuring element.

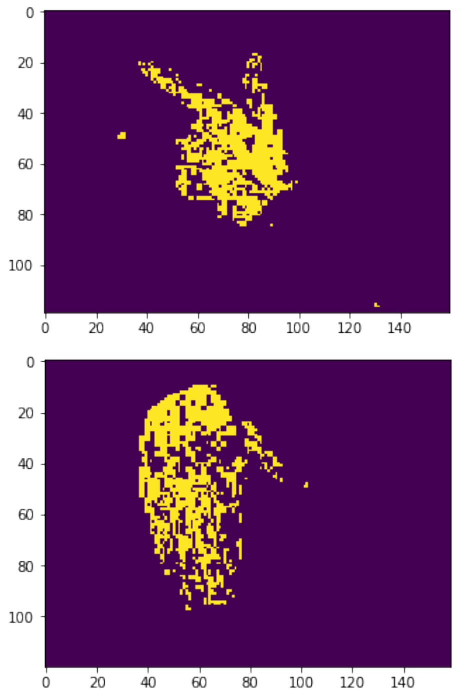
Erosion [$A\Theta B$] takes in the binary image and structuring element, and essentially shrinks the foreground pixels of the binary image via superimposing the structuring element over the image. For example, if a 3x3 structuring element that contains 1's in each pixel is run over a binary image, every 3x3 set of pixels on the binary image is evaluated and if any of this image's 3x3 sets exactly matches the structuring element (eg. a 3x3 arrangement of 1's in the binary image), it is left as is, or else the centroid is removed from the foreground.

Dilation [$A\oplus B$] works in reverse, where when every foreground pixel in the binary image is superimposed with the structuring element, all of the structuring element's foreground pixels are placed atop the selected foreground pixel of the binary image. This thickens the image's elements. For example, if a 5x5 binary image containing only one foreground pixel in the centroid is superimposed with a 1x5 horizontal structuring element of all 1's in each pixel, then the result will be a 5x5 binary image with the entire middle row containing the foreground's five 1's, while the rest of the grid contains 0's.

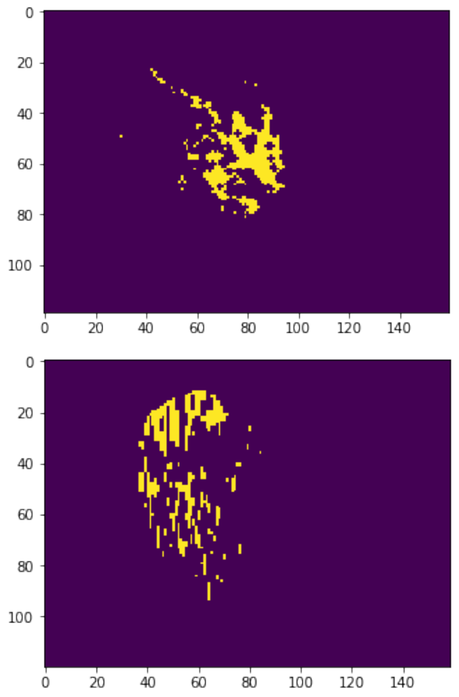
Now that we have covered Erosion and Dilation, the following are very simple. ***Opening*** [$(A\Theta B)\oplus B$] is simply a transformation of an image that *first* employs erosion across the image exactly as we defined above, and *then* dilation across the result, using the same structuring element from the erosion process. This is less destructive than erosion and mostly trims the edges of the foreground elements. ***Closing*** [$(A\oplus B)\Theta B$] is exactly the reverse of opening, where we *first* employ dilation on the image, and *then* erosion, using the same structuring element from the dilation process. This is less expansive than dilation, and has more of an effect on the inside, concave-like foreground pixels of the image.

Another transformation is the ***boundary*** [$\beta(A)=A-(A\Theta B)$] of an image. Mathematically put, to acquire the boundary of an image, we simply remove the result of an erosion process of an image from the image itself prior to erosion, ie. we subtract the 2d-array of the eroded result from the 2d-array of the original image. For this assignment, we also had to reduce a lot of *noise* around the images, so this required multiple transformations of various structuring element designs in specific orders to acquire a more exact looking boundary of the image we really want to acquire. Some of my results are on the following page:

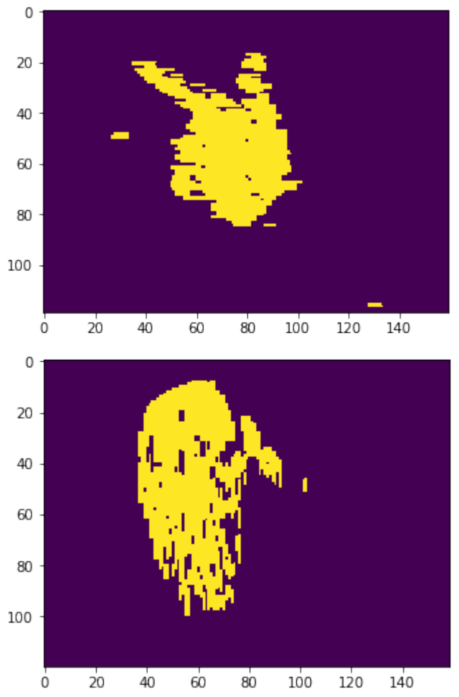
Original:



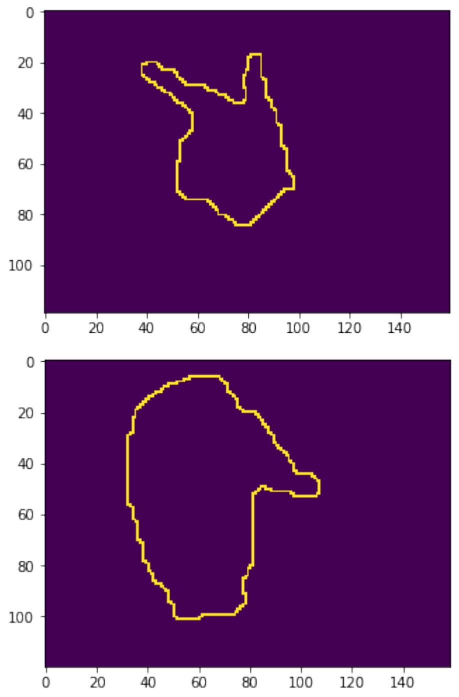
Example ***Erosion***:



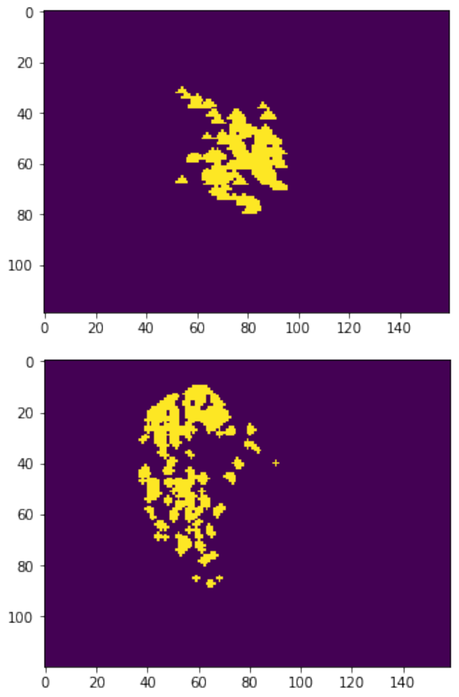
Example ***Dilation***:



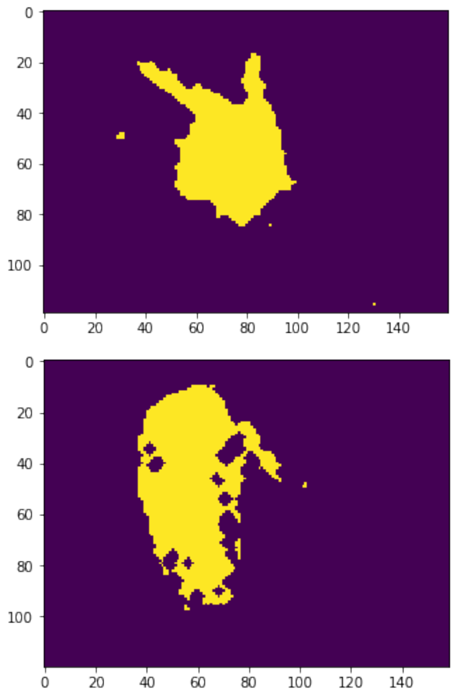
Example ***Boundary*** (used multiple transformations to reduce noise!!):



Example ***Opening***
(First erosion, then dilation):



Example ***Closing***:
(First dilation, then erosion)


0
20
40
60
80
100
0 20 40 60 80 100 120 140