






PROJECT: IMAGE MORPHOLOGICAL OPERATIONS

GROUP 6

Team Objective We aim to provide the user with an interface, where he can upload binary or grayscale image and choose various among morphological operators to be applied on, to obtain the desired resulting image.

Implementation

Erosion
In all the results “I” correspond to iterations and N correspond to “Kernel Size”
The results of erosion operation on binary image are as follows:

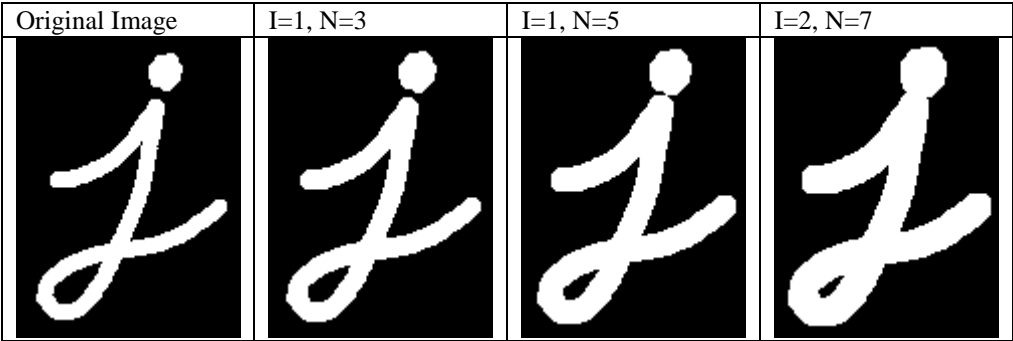
Original Image	I=1, N=3	I=1, N=5	I=2, N=3	I=2, N=5
				

The results for binary image are as follows:

Original Image	I=1, N=3	I=2, N=5
		

Dilation

Binary



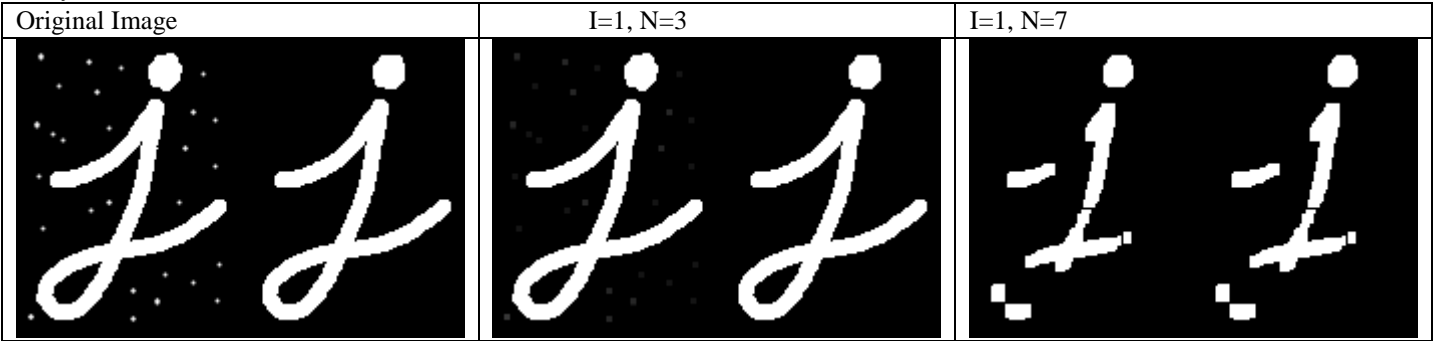
Gary Scale



Open

Open is implemented by calling erosion function followed by dilation

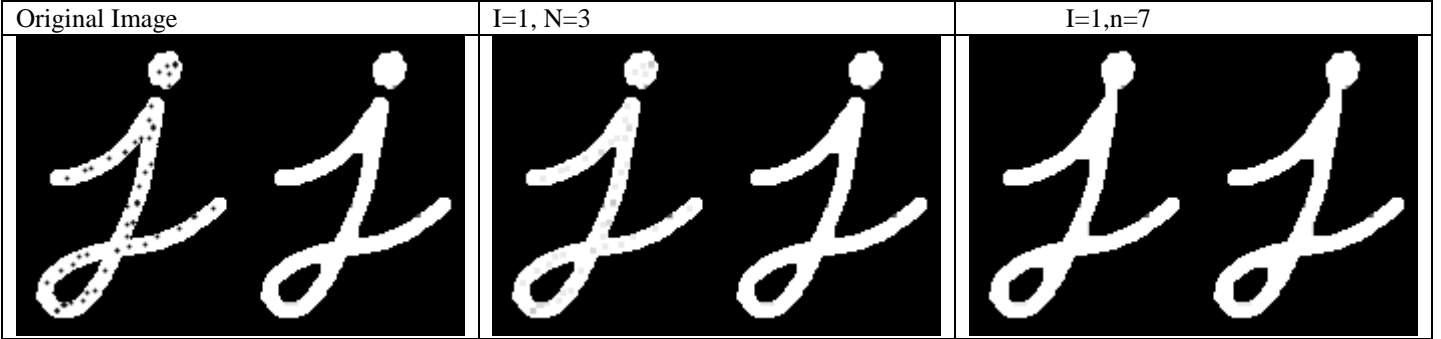
Binary



Gray Scale

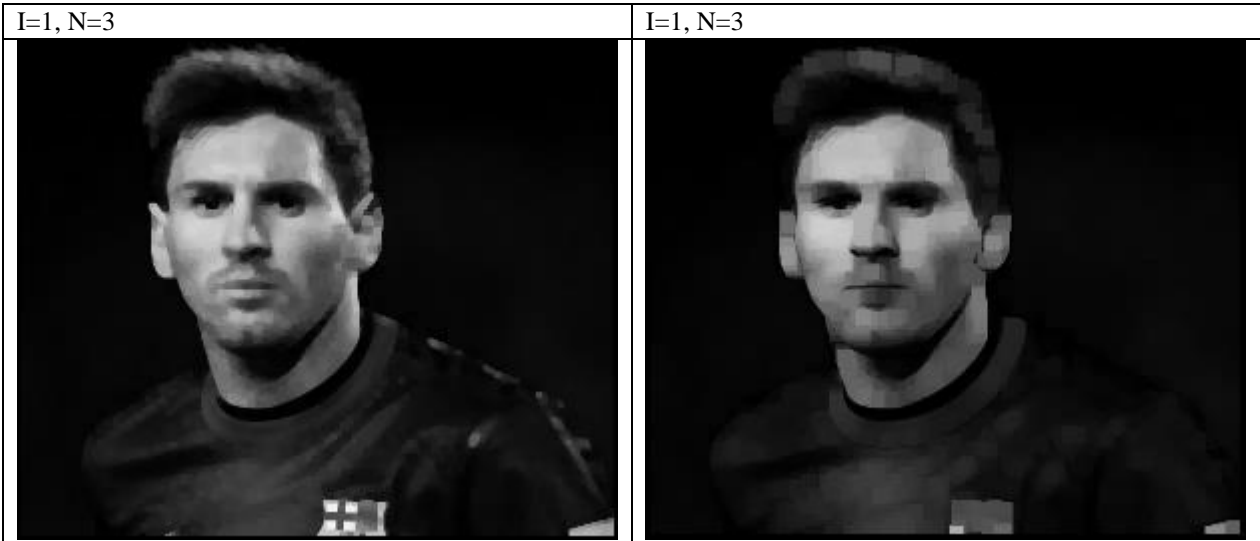
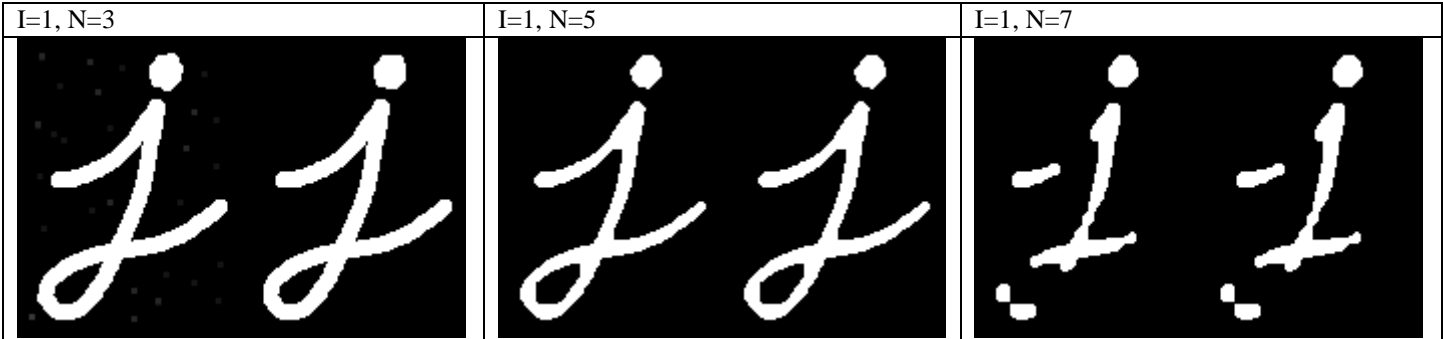


Close Close is implemented by calling dilation function followed by Erosion.

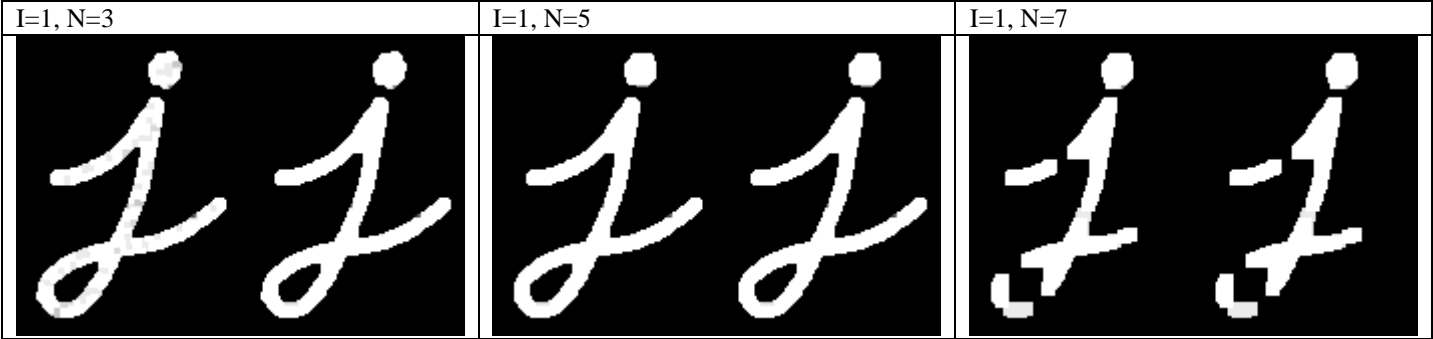




Open-Close Input considered is same as open.
 Binary

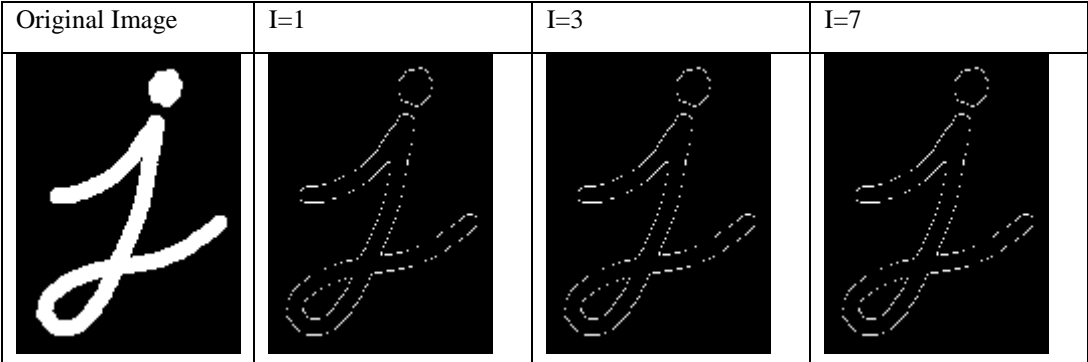


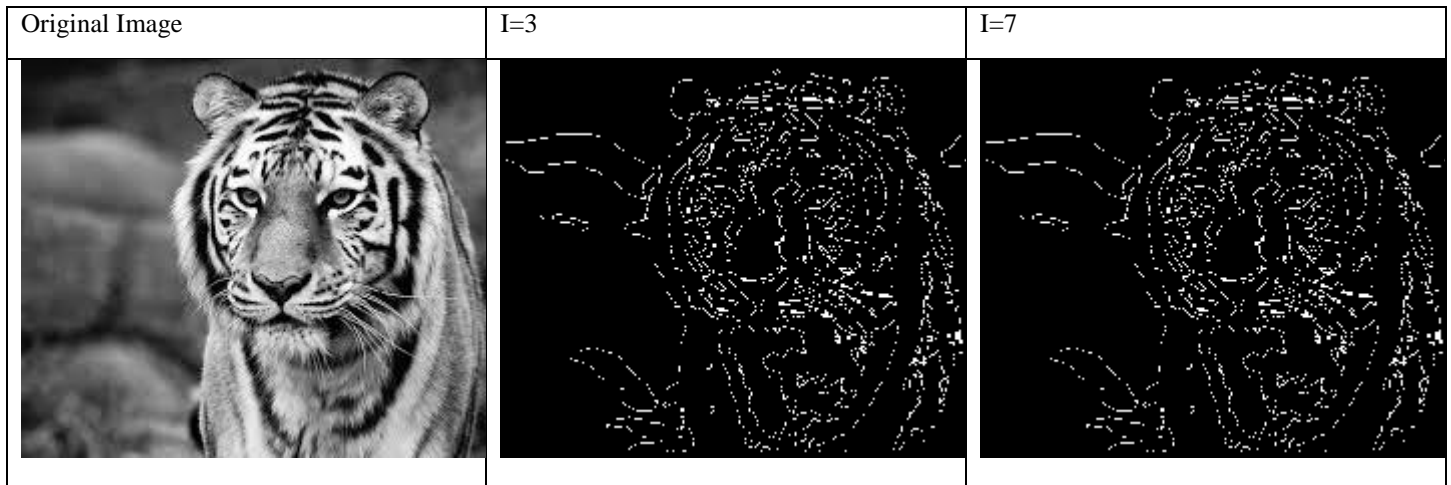
Close-Open



Hit and Miss

Binary





Majority Filter (Median Filter)

In this morphological function, the main purpose is smoothing. I've implemented square window by given kernel size. I've sorted the values and picked the median value. I've implemented zero-padded edges instead of revolving convolution. I've tested with couple of images. I've seen ragged edges become smoother. I've also seen other window shapes applied for different purposes.

Skeletonization by Thinning

In this part of the application, I've used Zhang-Suen Thinning Algorithm. I've applied thinning until there were no possible changes left. Thinning Algorithm is consisting of two passes. Each pass has a list of conditions that will be checked and if the conditions met that means we can remove that pixel. Similar conditions for both steps are:

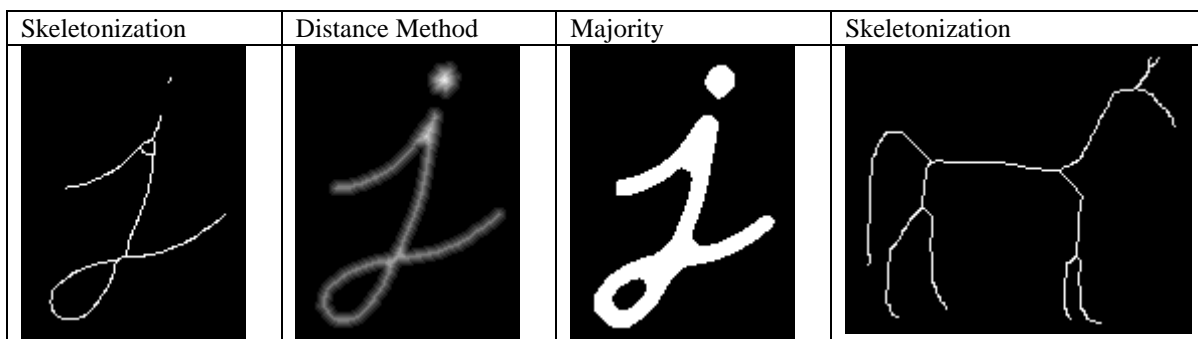
- Is the current pixel white?
- How many white neighbors do I have?
- How many transitions happen around me?

While in the first pass it checks (north, east, south) and (east, south, west) neighbors, in the second pass it checks (north, east, west) and (north, south, west). First pass is removing south east corners and second pass removes north west corners. Even though this algorithm looks very simple, it does thinning job very well and it is very efficient.

Skeletonization by Distance Map

In this part of the application, I've tried to create a distance map. In distance map, I've calculated distance from each white pixel to the closest black pixel. For distance calculation, I've tried Euclidean and Taxicab Method (Manhattan). After distance map is created, we need to extract ridges from the distance map, I've created a simple algorithm for ridge detection. I've extracted each pixel that have two neighbors in across from each other that are smaller than pixels value.

From my readings, a better method for ridge detection would be calculating the major eigenvalue of the Hessian matrix at each pixel.



Observations

- As we increased the size of the kernel,
- For erosion, as we can see from the image, boundaries of the regions of foreground pixels got shirnked in size and area of background pixels became larger.
- Where as in dilation, the foreground pixels grow in size, the wholes within those regions become smaller
- Erosion on Gray scale images rendered more darker image with increase in kernel size
- Dilation on Gray scale image tends to give an image with uniform intensities.
- Opening, Closing, Open-Close and Close-Open had similar effect as in erosion and dilation but are less destructive. They tend to preserve the structure of the image while modifying the intensity values of the foreground and background pixels.
- With increase in number of iterations and kernel size, the time to obtain the desired output also slightly increased.
- For, Hit and Miss we have used different kernels that consist of both 1's and 0's. Each of these kernels are applied to obtain all the four different corners of the image.
- With the increase in number of iterations, there is not much difference in detecting the corners of the image as it renders the corners of the image very effectively in the first iteration itself for both binary and gray scale images.

Graphical User Interface(GUI): We have used tinkercart to implement the. Our GUI allows the user to select the input image, type of operator, kernel size and number of iterations. Output image is also displayed along with input image.

