Image Processing
Homework 1

Aadil Anil Kumar
Otmane Sabir

10/2/2020

Introduction

The first homework assignment required us to implement two morphological operations - opening & closing - while following certain guidelines which could be summarized to the following list:

1. Writing a command line program that performs either an erosion or a dilation of a given two-dimensional gray-scale image 'f' with respect to a (symmetric, odd-sized) structuring element.
2. Output a single CSV file for the eroded/dilated image in the same format and same definition and value domains as the input image 'f'.
3. Do meaningful (motivated from a real-world perspective) openings and closings on at least three photographs or 2D images of our choice. The three chosen problems should be very different in nature and write in the a description of each of the three problems, purpose of the operations to be applied, choice of structuring elements, and provide the output as graphics file.

Contents

1 Dilation & Erosion

The realm of morphological operations is based off of certain primitive operations such as dilation and erosion. In the simplest of definitions, dilation adds pixels to the boundaries of objects in an image, while erosion removes pixels on object boundaries. The structuring element defines the range and size of pixels which were added/removed.
As we have previously seen in the lecture we can define grayscale erosion with the following :

Denoting an image by $f(x)$ and the grayscale structuring element by $b(x)$, where B is the space that $b(x)$ is defined, the grayscale erosion of f by b is given by :

$$(f \ominus b)(x) = \inf_{y \in B}[f(x+y) - b[y)].$$

where inf denotes the infimum.

While dilation is defined by :

Denoting an image by $f(x)$ and the grayscale structuring element by $b(x)$, where B is the space that $b(x)$ is defined, the grayscale erosion of f by b is given by :

$$(f \ominus b)(x) = \sup_{y \in B}[f(x+y) - b[y)].$$

where sup denotes the supermum.

2 Implementation

We can see by examining the expressions for the morphological dilation or erosion that we're required to get the minimum/maximum of a pixel within the structuring element. Therefore, we first started by implement 2 function $minVal\&maxVal$ which set the value of the output pixel is the max/min value of the pixels that are in neighborhood the size of the SE within the input image. The following algorithm $maxVal$ gets the maximum value which we'll use for the dilation. This function receives the matrix, the current x and y indices we're checking, the maximum value present in this matrix and the structuring element. The $minVal$ algorithm follows the same logic except we check if the

current value is less than the stored one and not bigger and also gets the largest value present in the matrix instead of the smallest.

2.1 Dilation algorithm

Data: matrix, se, maxi val, a, b
Result: the min value of the pixels that are in neighborhood the size of the SE
initialization;
$rows \leftarrow$ number of rows in se;
$cols \leftarrow$ number of cols in se;
$mRows \leftarrow$ number of rows in matrix;
$mCols \leftarrow$ number of cols in matrix;
while $i \leq rows - 1$ **do**
  while $j \leq cols - 1$ **do**
    if $se[i][j] == 1$ **then**
      if $0 \geq b \leq mRows - 1$ **then**
        if $matrix[a][b] \geq maxi$ **then**
          $maxi = matrix[a][b]$;
        else
          continue;
        end
        $b++$
      else
        do nothing;
      end
    else
      break;
    end
  end
  if $0 \geq a \leq mCols - 1$ **then**
    $a++$;
    reset b to initial value;
  else
  end
end
return $maxi$

Algorithm 1: Algorithm to find maximum for dilation

After we're able to extract the maximum value within the neighboring area of a pixel then we can write our dilation algorithm. Again, the erosion is the same except we use the $minVal$ algorithm.

Data: matrix, structuring element
Result: the dilated matrix
initialization;
$ans \leftarrow$ empty 2D array of identical size to matrix;
$mRows \leftarrow$ number of rows in matrix;
$mCols \leftarrow$ number of cols in matrix;
while $i \leq mRows$ **do**
    while $j \leq mCols$ **do**
        $ans[i][j] \leftarrow maxVal(matrix, se, i, j)$;
        increment j;
    end
    increment i;
end

Algorithm 2: Dilation algorithm

2.2 Erosion algorithm

Data: matrix, se, min val, a, b
Result: the min value of the pixels that are in neighborhood the size of the SE
initialization;
$rows \leftarrow$ number of rows in se;
$cols \leftarrow$ number of cols in se;
$mRows \leftarrow$ number of rows in matrix;
$mCols \leftarrow$ number of cols in matrix;
while $i \leq rows - 1$ **do**
  while $j \leq cols - 1$ **do**
    if $se[i][j] == 1$ **then**
      if $0 \geq b \leq mRows - 1$ **then**
        if $matrix[a][b] \leq min$ **then**
          $min = matrix[a][b]$;
        else
          continue;
        end
        $b++$
      else
        do nothing;
      end
    else
      break;
    end
  end
  if $0 \geq a \leq mCols - 1$ **then**
    $a++$;
    reset b to initial value;
  else
  end
end
return *maxi*

Algorithm 3: Algorithm to find minimum for erosion

Data: matrix, structuring element
Result: the dilated matrix
initialization;
$ans \leftarrow$ empty 2D array of identical size to matrix;
$mRows \leftarrow$ number of rows in matrix;
$mCols \leftarrow$ number of cols in matrix;
while $i \leq mRows$ **do**
  while $j \leq mCols$ **do**
    $ans[i][j] \leftarrow minVal(matrix, se, i, j)$ increment j;
  end
  increment i;
end

Algorithm 4: Erosion algorithm

3 Experiments Results

3.1 Erosion using symmetric vs asymmetric SE

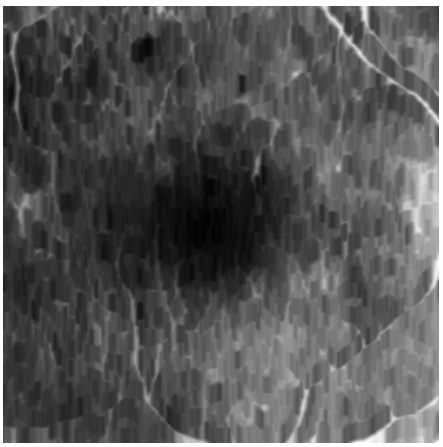

Figure 1: Erosion with Symmetric SE

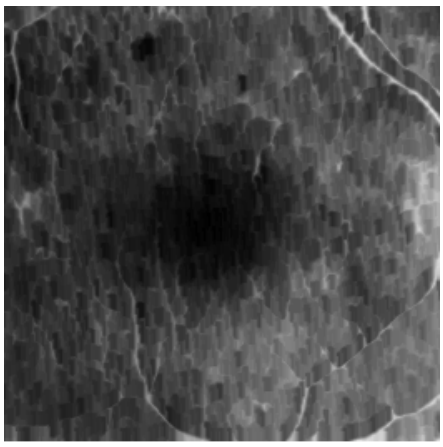

Figure 2: Erosion with Asymmetric SE

Border effects tend to occur when using an asymmetric SE that does not fit the definition domain of the image set. As expected the chosen border handling method does introduce artifacts to the image.
A possible explanation for this could be the fact that the definition domain of the image set decreases when doing the operation due to image borders not being handled correctly.

If you were to observe and compare Figure 1 and 2 closely you can see that the bottom border of Figure 2 contains a row of white space. This is evident when viewing the image with a black background. The effects were more pronounced when doing a comparison of the closing operation, as shown in figure 3 and 4.

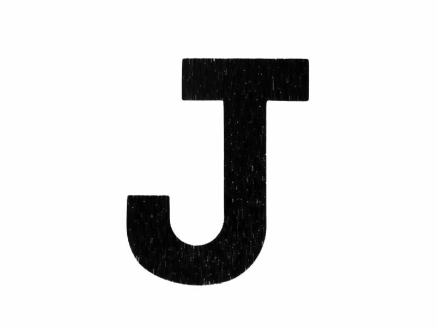

Figure 3: Closing with Symmetric SE

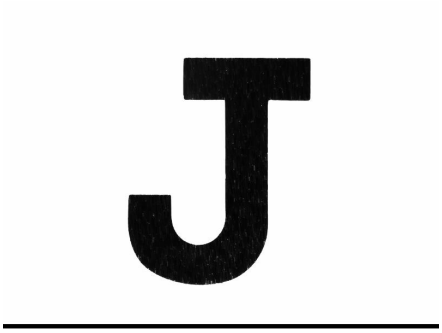

Figure 4: Closing with Asymmetric SE

3.2 Improving the clarity of handwritten digits

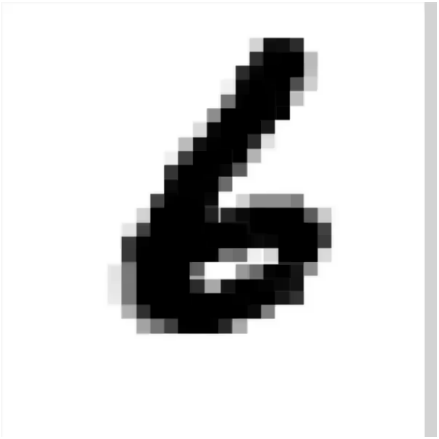

Figure 5: Original Image (602×599)

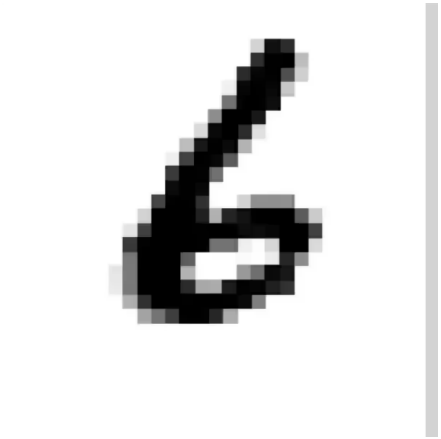

Figure 6: Dilated Six Digit

As visible in Figure 5, the six is not very well defined and specifically the white circle in the middle. This situation could raise problems with the classification of handwritten digits. Therefore, we dilate the image with a structuring element that is a square of size 15 (SE10.txt) in order to obtain we get a sharper six. We chose this structuring element because we noticed that the six was formed using square blots. We decided to use a larger structuring element because the image is very pixelated and we wanted to obtain a clear circle in the center of the six. A possible use case for this would be feature extraction. A classification algorithm could more efficiently extract features that maps the digit as a six.

3.3 Cell segmentation

For these operations we used a parallelogram structuring element (SE11.txt). We chose this shape for this image as there are a lot of circular structures that we wanted to highlight. There is a lot more contrast in the images that had the operations done on them; helping clearly segment each cellular structure. The dilated image has the clearest segmentation between structures. However, the opened image also shows clear segmentation while not having as jarring of a contrast. This helps us achieve clear segmentation of cells in order to assist with counting.

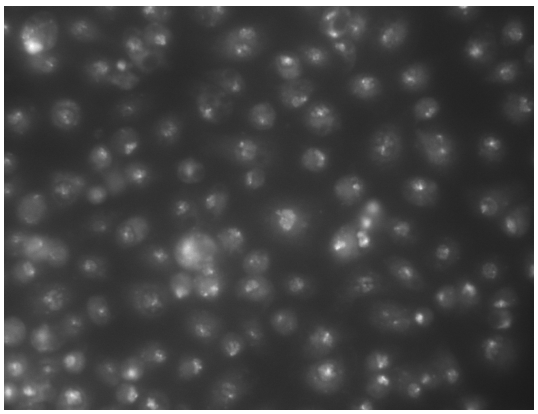

Figure 7: Original image (1344×1024)

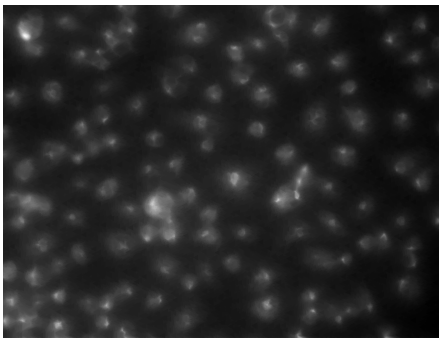

Figure 8: Erosion

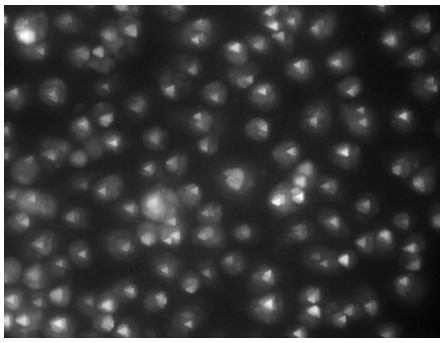

Figure 9: Dilation

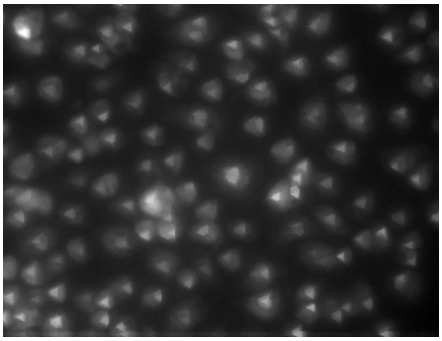

Figure 10: Opening

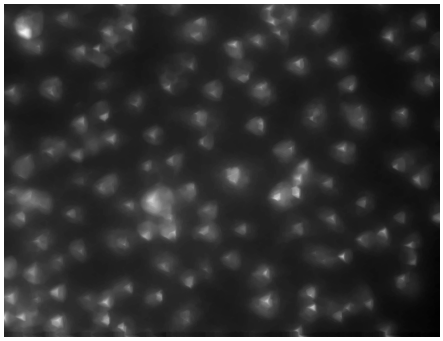

Figure 11: Closing

3.4 Haemoglobin noise filtering and cell filling

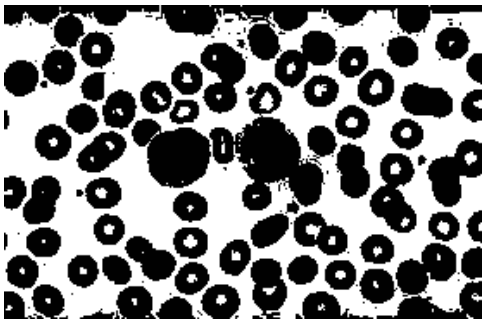

Figure 12: Original image, (277×182)

In order to assist in the counting of the red blood cells present in this image, we would first remove the evident noise then proceed to fill in the holes in the middle of the cells as best as possible.

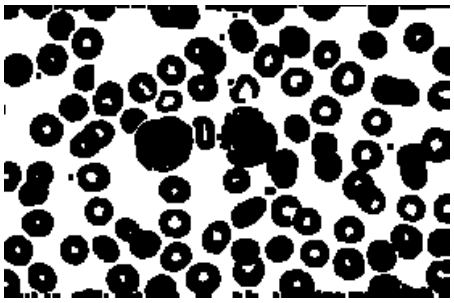

Figure 13: Closing with SE4

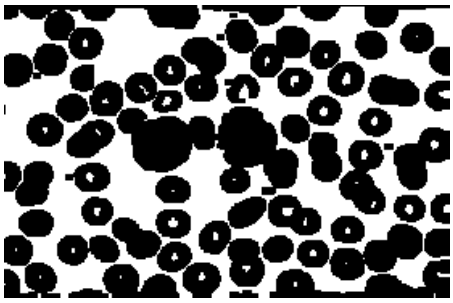

Figure 14: Erosion of Fig.13 (SE3)

We decided to perform a closing operation to remove the small black spots present then fill in some of the white spaces present in the cell. After comparing the results of a size 3 and size 4 square, we found that the latter worked better in removing the noisy elements within this picture. Performing the closing led to the red blood cells not being as defined. In order to fill in the holes and accentuate the features of the cells we decided to perform a erosion on the closed image using a vertical SE of length 3. However, additional operations would need to be done in order to further fill in the center of some of the cells.

3.5 Fingerprint image cleaning

In this experiment we try to remove some of the noise present in this fingerprint image (Figure 15) and sharpen the image further to create a clearer result. There is a bit of noise surrounding the fingerprint so we decided to use a square of size 3 (SE1.txt) in order to remove this noise. After performing an erosion we get an image without the white squares (Figure 16), however it also leads to a darker fingerprint. In order to further define the print lines we decided to perform an erosion which produces a far better result (Figure 17) leave us with a clear and defined fingerprint.

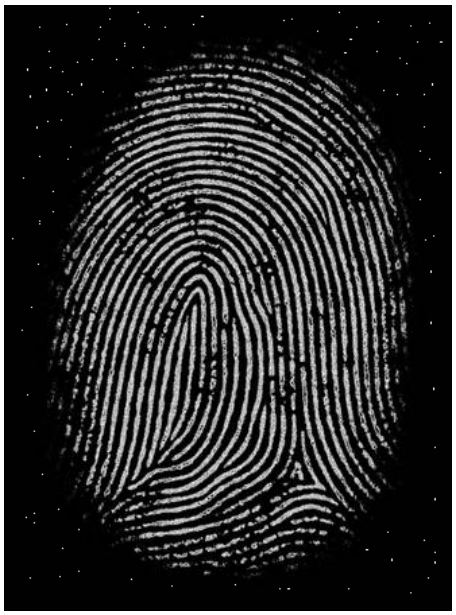

Figure 15: Original image, (332×450)

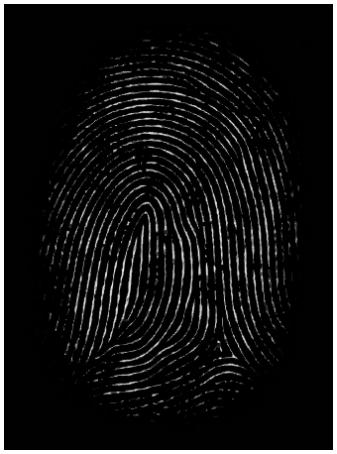

Figure 16: Closing with SE4

Figure 17: Erosion of Fig.16 (SE3)

4 Comparison

After writing down the algorithms their time complexities we figured these operations are slow to compute in practice especially for large structuring elements. We then decided to compared our implementation to the OpenCV erode and dilate functions and found that on average our implementation processes a 512x512 image with a structuring element of size 3x3 in $2.4344s$ while the OpenCV function preforms it in $0.0001242s$ which made us realize there must be plenty of more efficient approaches.

5 Task Distribution

Regarding the task distribution, we decided to split up the work evenly. These were our tasks.
Otmane Sabir :

1. Wrote function for finding minimum and wrote erosion
2. Wrote function for opening
3. Ran test 1-7 and ran the fingerprint and cell tests
4. Wrote 2 chapters of the report.

Aadil Anil Kumar :

1. Wrote function for finding maximum and wrote dilation
2. Wrote function for dilation
3. Ran the rest of the test and 3 experiments
4. Wrote 2 chapters of the report.

References

[1] Erosion (morphology),
`https://en.wikipedia.org/wiki/Erosion_(morphology)`

[2] Dilation (morphology),
`https://en.wikipedia.org/wiki/Dilation_(morphology)`

[3] Types of Morphological Operations,
`https://www.mathworks.com/help/images/morphological-dilation-and-erosion.html`

[4] Morphology,
`https://homepages.inf.ed.ac.uk/rbf/HIPR2/morops.htm`

[5] Eroding and dilating,
https://docs.opencv.org/2.4/doc/tutorials/imgproc/erosion$_d$*ilatation*/*erosion*$_d$*ilatation.html*

[6] Nick Efford. *Digital Image Processing: A Practical Introduction Using Java*
Pearson Education, 2000.