# MORPHOLOGICAL OPERATIONS

Ron Michael V. Acda

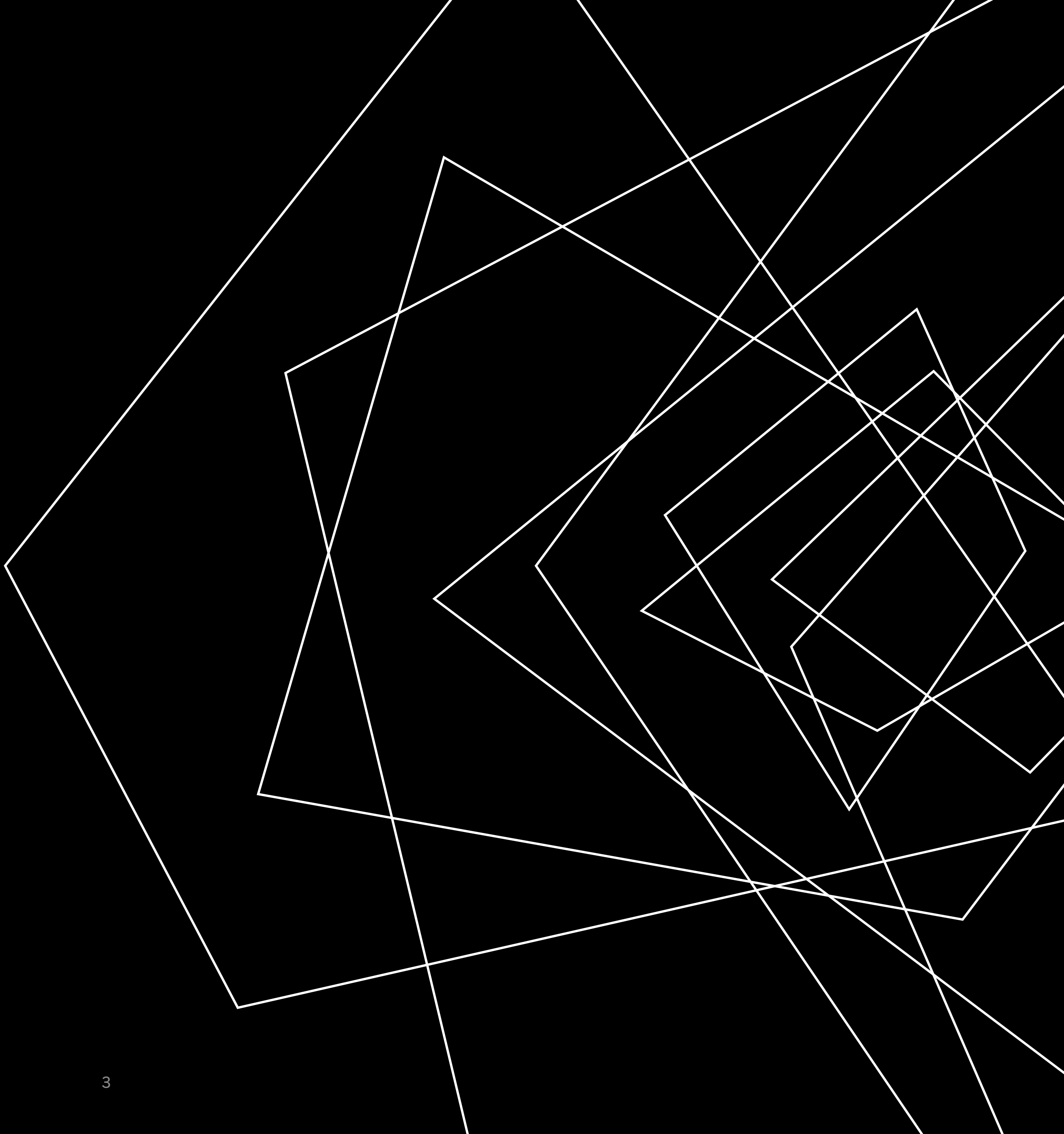
2019-03839

## OBJECTIVES

1. Predict the effects of morphological dilation and erosion of simple patterns using simple structural elements and explain the result.
2. Perform and demonstrate morphological operations in real-life images.

## OBJECTIVES

1. Predict the effects of morphological dilation and erosion of simple patterns using simple structural elements and explain the result.
2. Perform and demonstrate morphological segmentation in images.



## MORPHOLOGICAL DILATION

A dilation of a grid A with a structural element B is mathematically defined as:

$$A \oplus B = \{z | \widehat{B}_z \cap A \neq 0\}$$

where $z \in I$ (I is an integer grid, which is a Numpy 2D array in our case), $\widehat{B}$ is the reflection about a specified origin in the structural element, $\widehat{B}_z$ means we “slide” or translate $\widehat{B}$ across the integer grid.

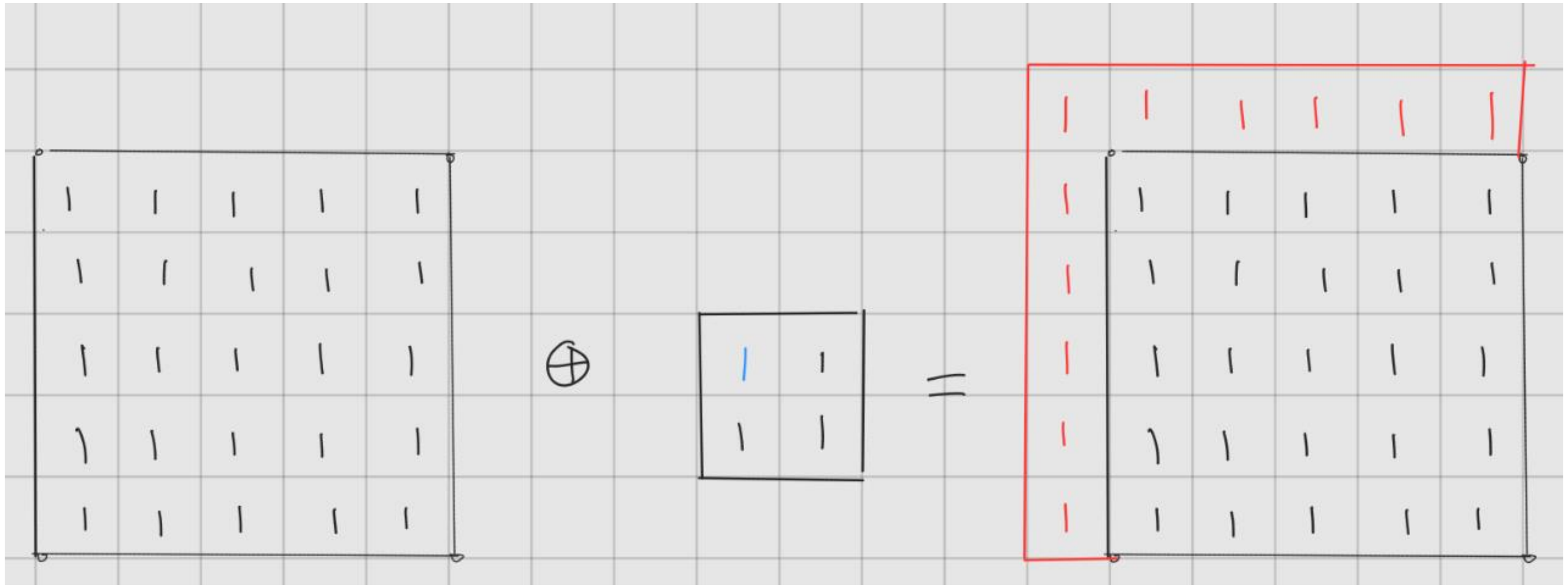
Some properties:

1. $A \oplus B = B \oplus A$ (commutativity). This means that A can be used as the structural element to dilate B and vice-versa, without affecting the result.
2. $(A \oplus B) \oplus C = A \oplus (B \oplus C)$ (associativity)

This can better be explained using examples.

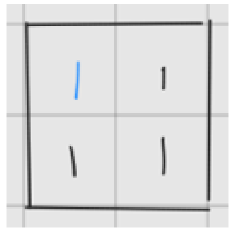
Note: Since there are 4 x 3 x 2 = 24 patterns (12 for dilation and 12 for erosion) I will not try to explain my thought process for all patterns because this will be extremely cumbersome. I will explain only a handful, but I’ll show the result for all of them.

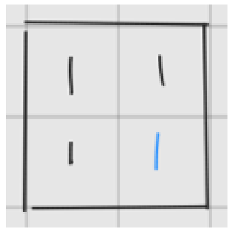
# MORPHOLOGICAL DILATION

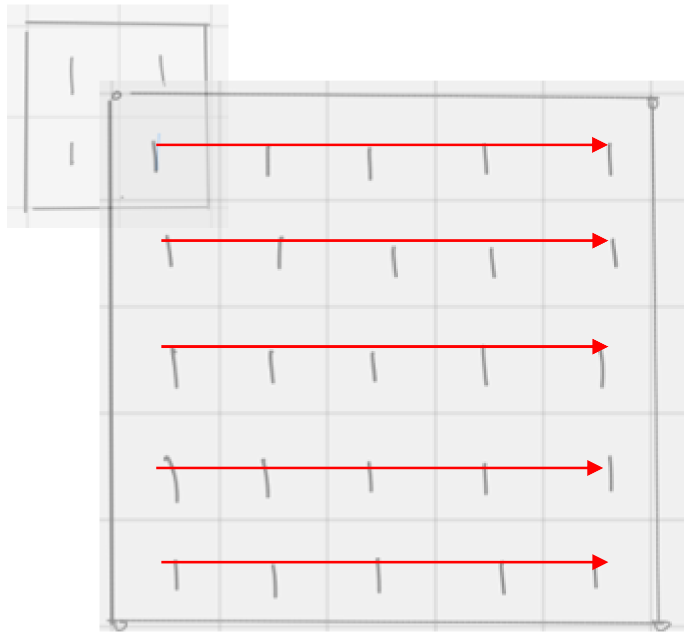


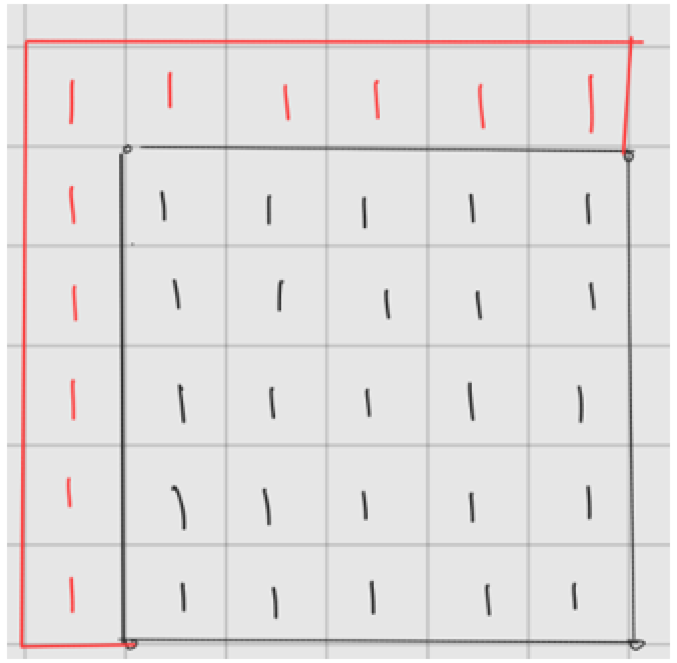
Here, A is a 5x5 array of 1s and the structural element is a 2x2 array of 1s, with the origin at the upper left box (blue). From the definition of dilation, the result should be a 6x6 array, obtained by the padding of 1s at the left and top sides of the original array.

# MORPHOLOGICAL DILATION


$B$

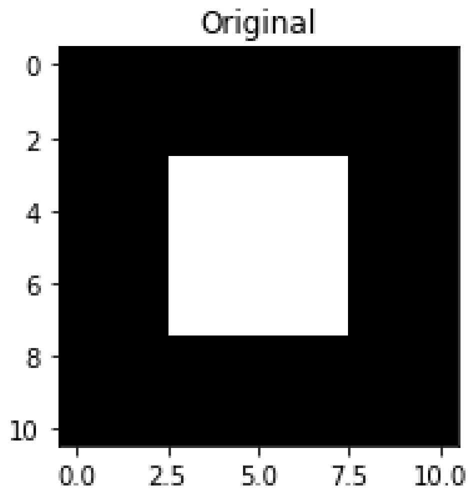

$\hat{B}$

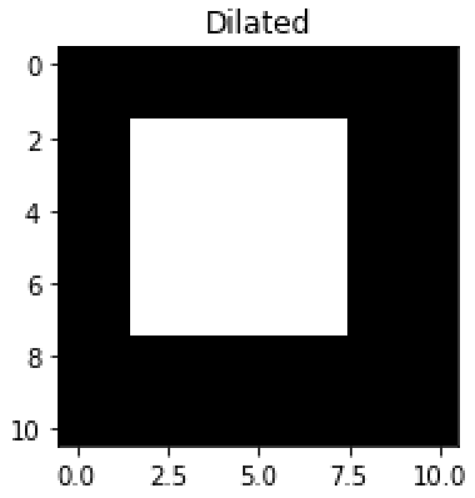

$A$

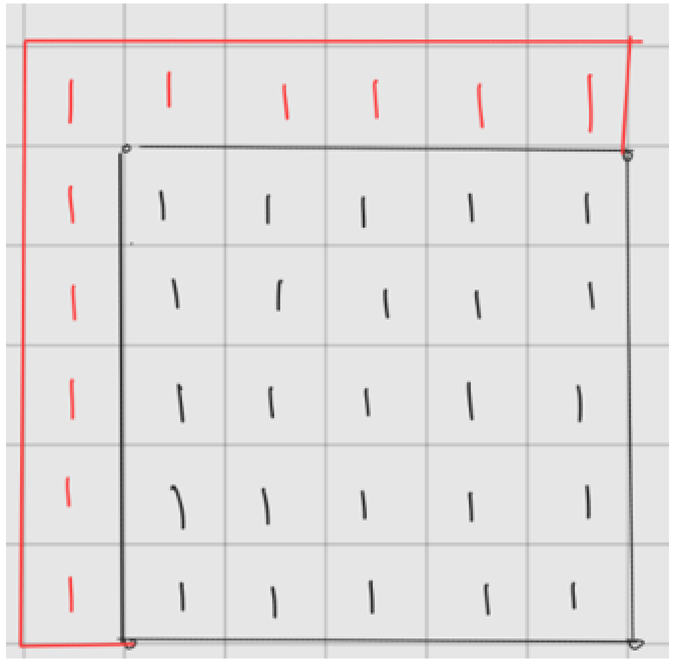

$A \oplus B$

Explanation: We flip B with respect to the origin and slide the resulting array using the location of the origin as the anchor point. The final array is produced by the superimposition of $\hat{B}$ arrays for every point in A.

# MORPHOLOGICAL DILATION



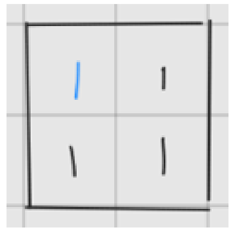


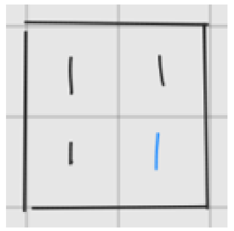


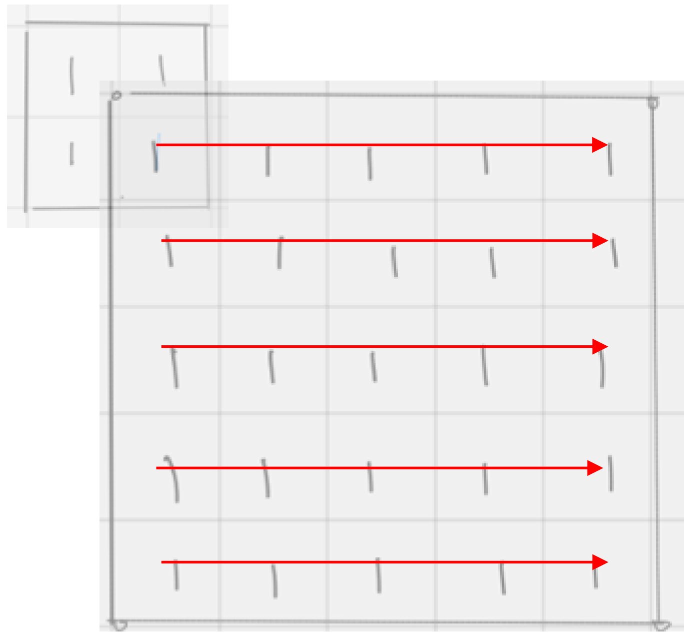
$A \oplus B$ (predicted)

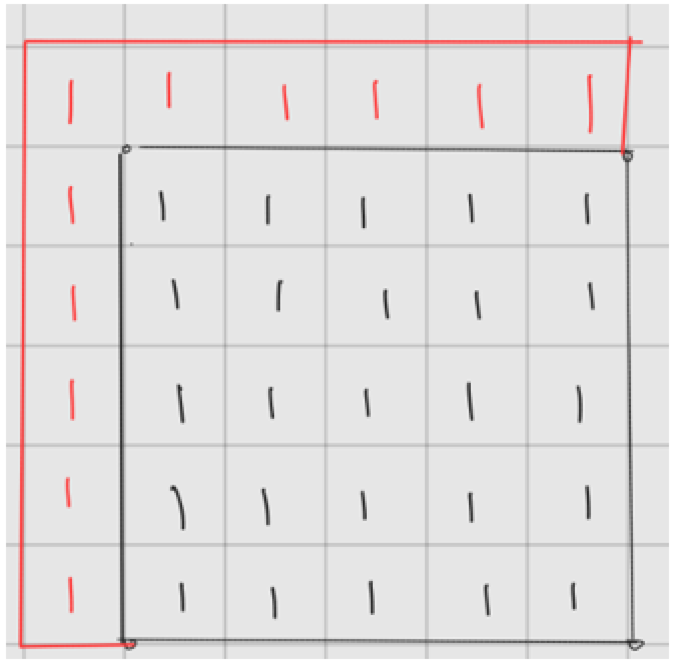
Note that the predicted result agrees with the Python implementation of binary dilation using scipy.ndimage.binary_dilation

# MORPHOLOGICAL DILATION


$B$


$\hat{B}$

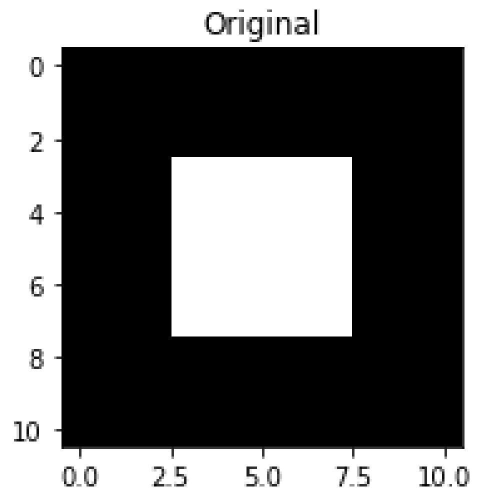

$A$

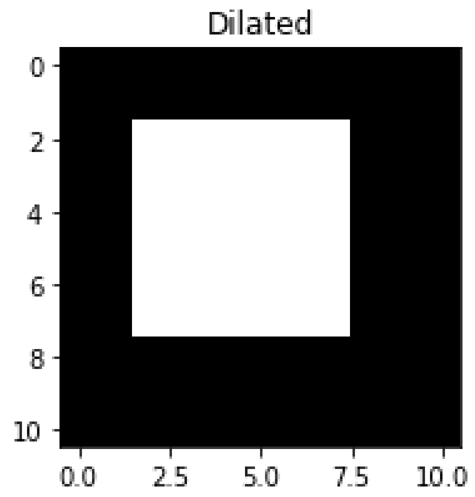

$A \oplus B$

Explanation: We flip B with respect to the origin and slide the resulting array using the location of the origin as the anchor point. The final array is produced by the superimposition of $\hat{B}$ arrays for every point in A.

**Note: The direction of the arrows does not matter, so long as the blue anchor covers every point in A.**

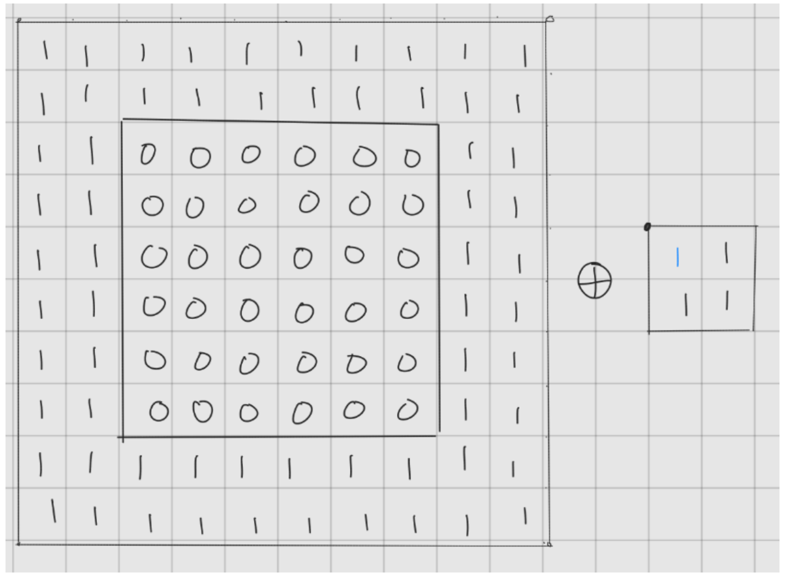
# MORPHOLOGICAL DILATION

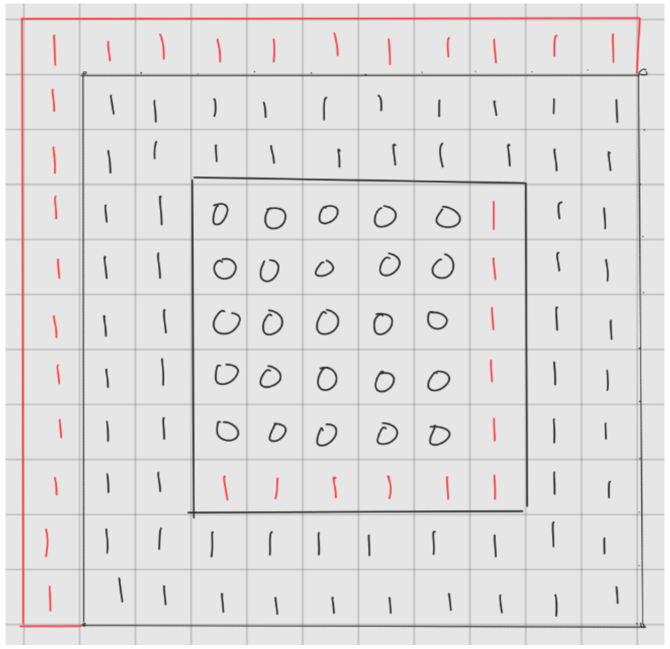




```
import numpy as np

from scipy import ndimage

struc1 = np.array([[1,1], [1,1]])

pattern1 = np.zeros((11,11))

pattern1[3:8, 3:8] = 1 #Create a 5x5 grid

#Set the origin at the upper left box of the 2x2
structural element B

dilated = ndimage.binary_dilation(pattern1, struc1,
origin=(0,0))
```

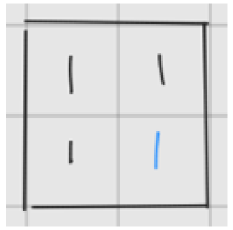
## MORPHOLOGICAL DILATION

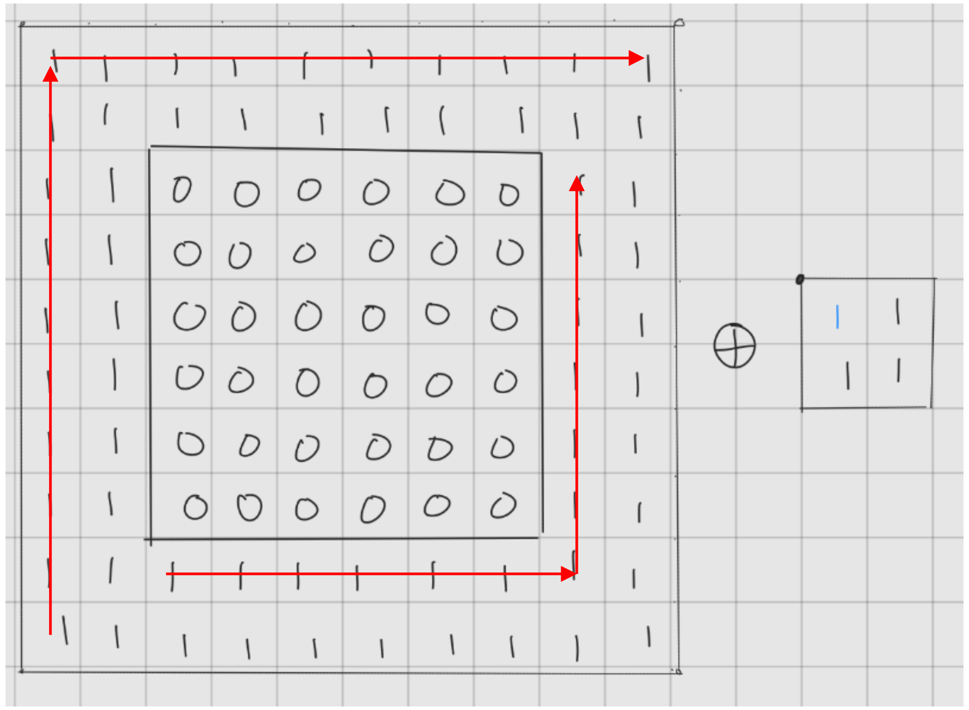


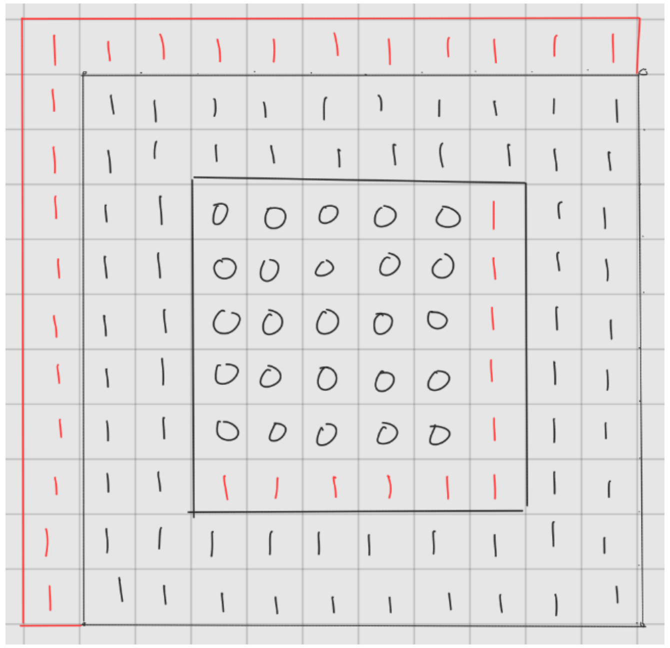


Using the same structural element, we predict that the resulting array has padded 1s on the left and top sides. Moreover, the size of the hole decreases since 1s are padded on its right and bottom sides.

# MORPHOLOGICAL DILATION


$\hat{B}$

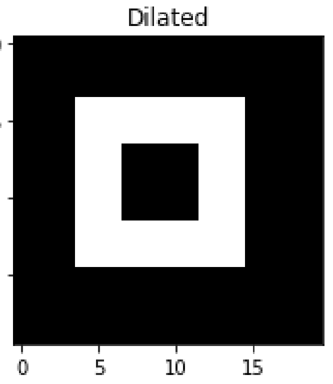

$A$

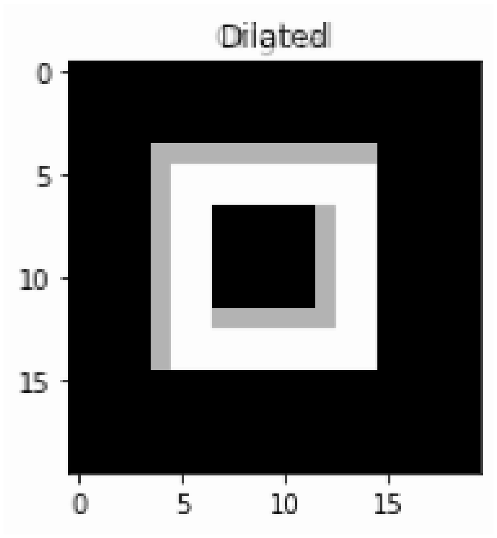

$A \oplus B$

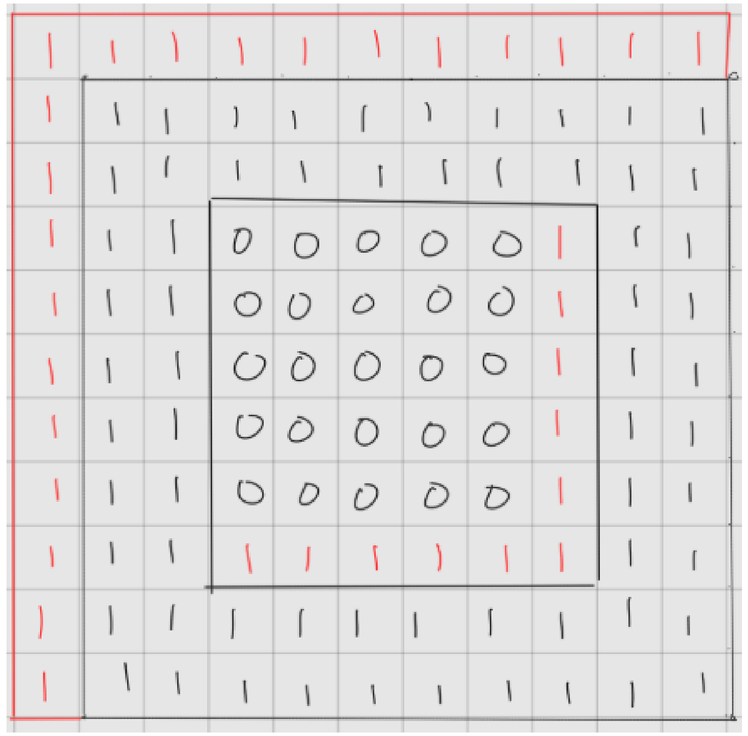
The movement of the reflected anchor (blue) across the red arrows causes the addition of the new, red 1s on the original array A. Other arrows were not shown since they do not change the original array.

## MORPHOLOGICAL DILATION

$A \oplus B$ (





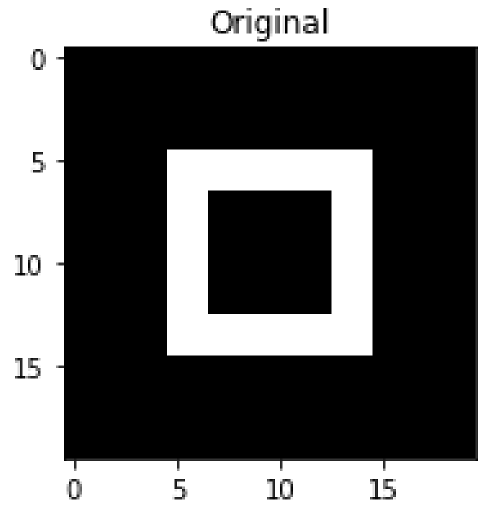


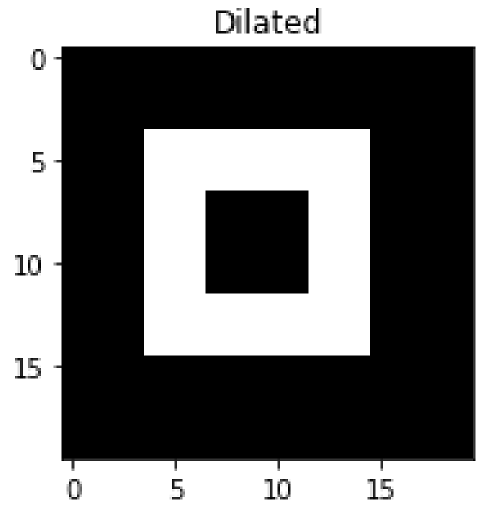
| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 |
| 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 |
| 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 |
| 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 |
| 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

$A \oplus B$ (predicted)

Note that the predicted result agrees with the Python implementation of binary dilation using scipy.ndimage.binary_dilation

## MORPHOLOGICAL DILATION

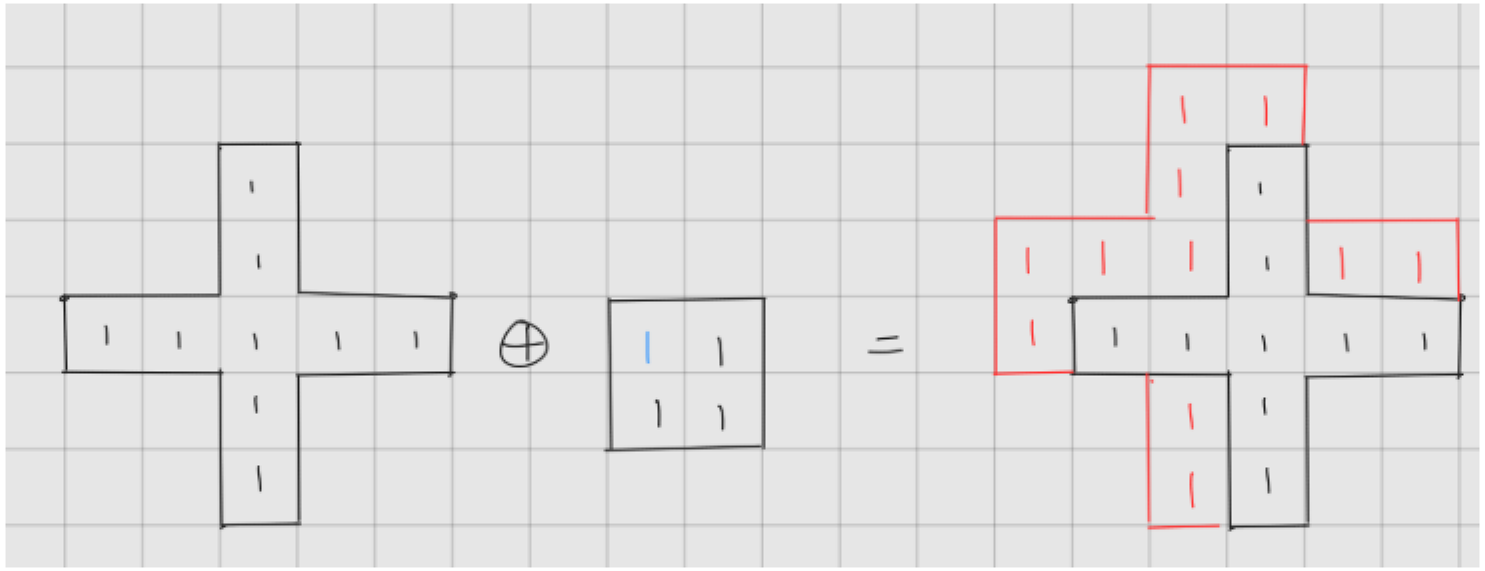




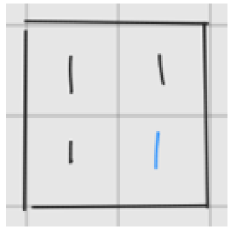
```
import numpy as np

from scipy import ndimage

struc1 = np.array([[1,1], [1,1]])

#Setup the 10x10 hollow grid, 2 boxes thick.
pattern2 = np.zeros((20,20))
pattern2[5:15, 5:15] = 1
pattern2[7:13, 7:13] = 0

#Set the origin at the upper left box of the 2x2
structural element B
dilated = ndimage.binary_dilation(pattern2, struc1,
origin=(0,0))
```

## MORPHOLOGICAL DILATION

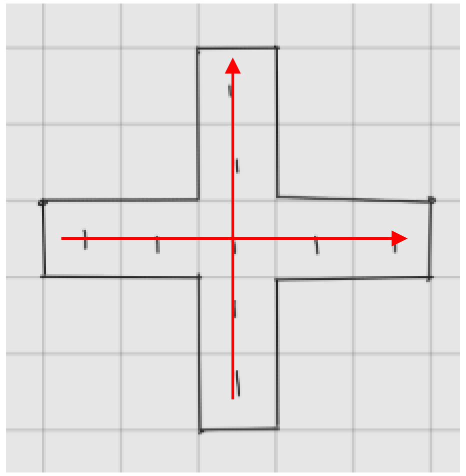
For a cross-shaped array, we predict the following:



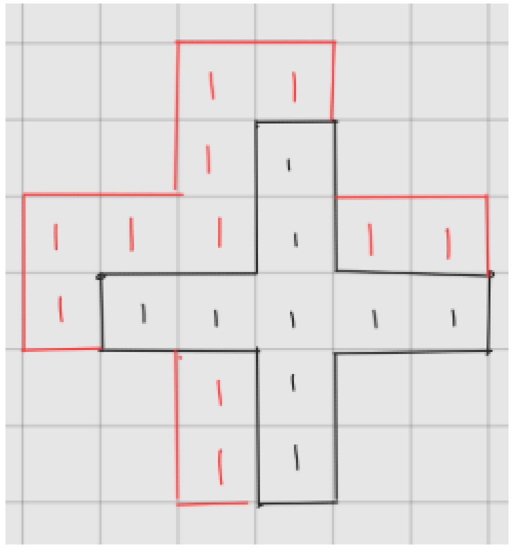
# MORPHOLOGICAL DILATION



$\hat{B}$



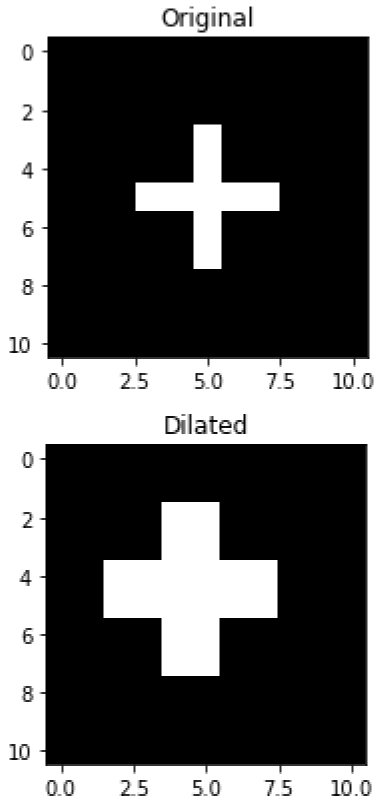
$A$

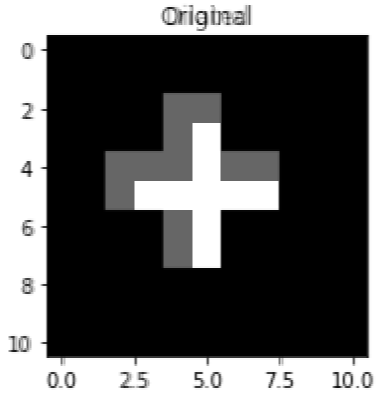


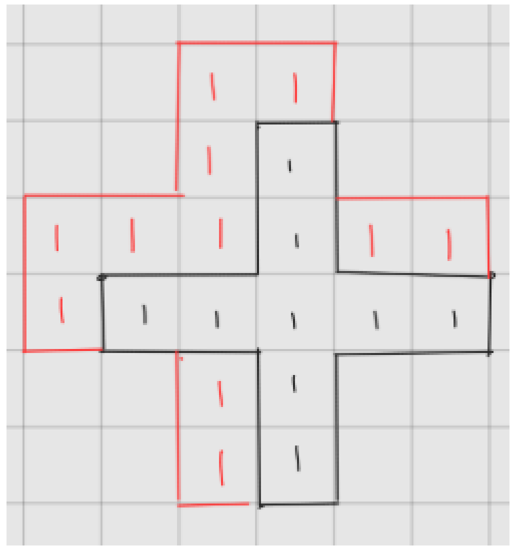
$A \oplus B$

The superimposition of the anchor of $\widehat{B}$ on every point in A adds something to A. If we visualize this, the resulting array should look like the one shown on $A \oplus B$

# MORPHOLOGICAL DILATION



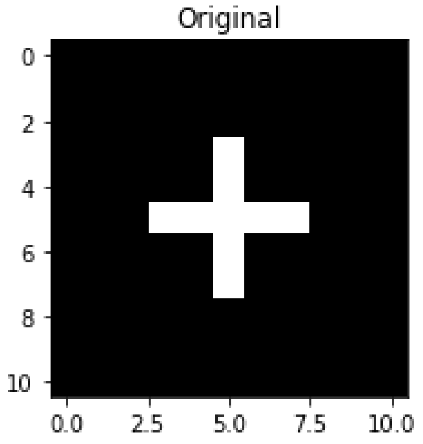


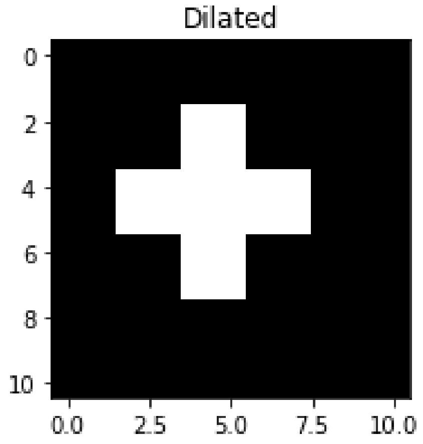


$A \oplus B$ (predicted)

Note that the predicted result agrees with the Python implementation of binary dilation using scipy.ndimage.binary_dilation

## MORPHOLOGICAL DILATION

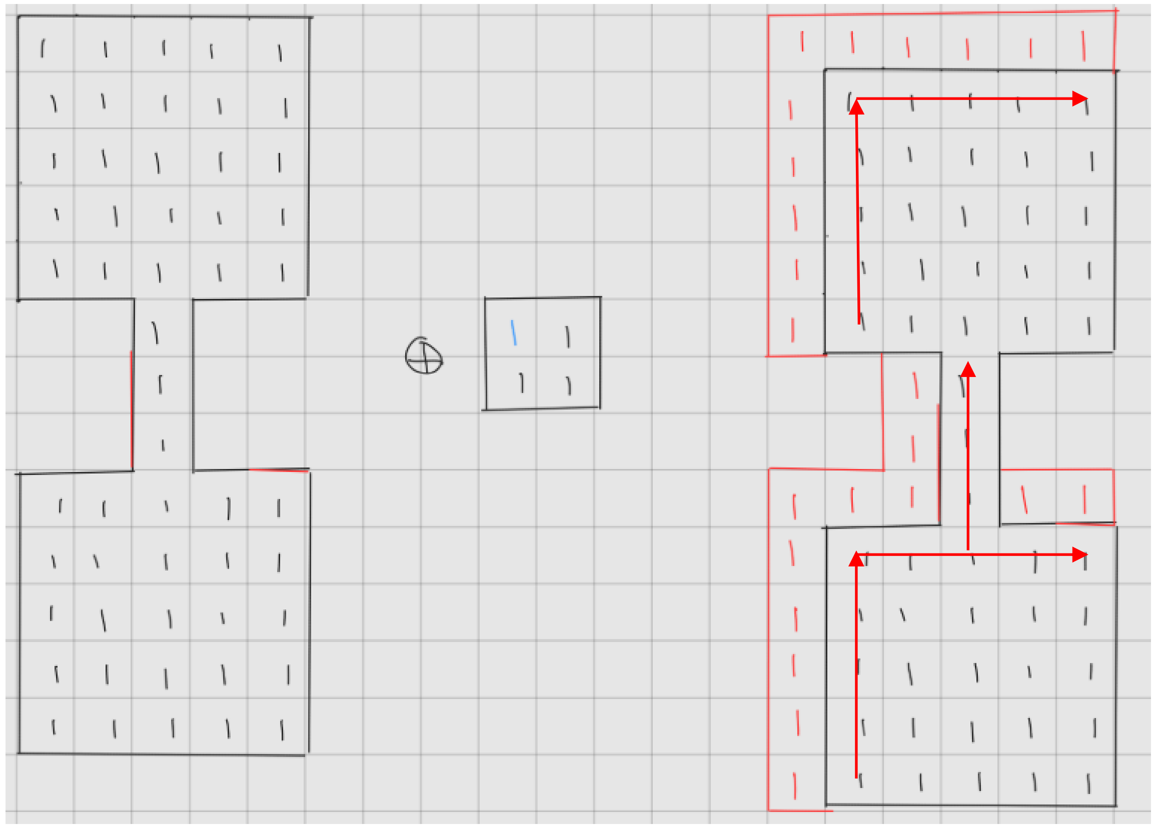




```
import numpy as np

from scipy import ndimage

struc1 = np.array([[1,1], [1,1]])

#Setup the cross pattern.
pattern3 = np.zeros((11,11))
pattern3[5,3:8] = 1
pattern3[3:8,5] = 1

#Set the origin at the upper left box of the 2x2
structural element B
dilated = ndimage.binary_dilation(pattern3, struc1,
origin=(0,0))
```

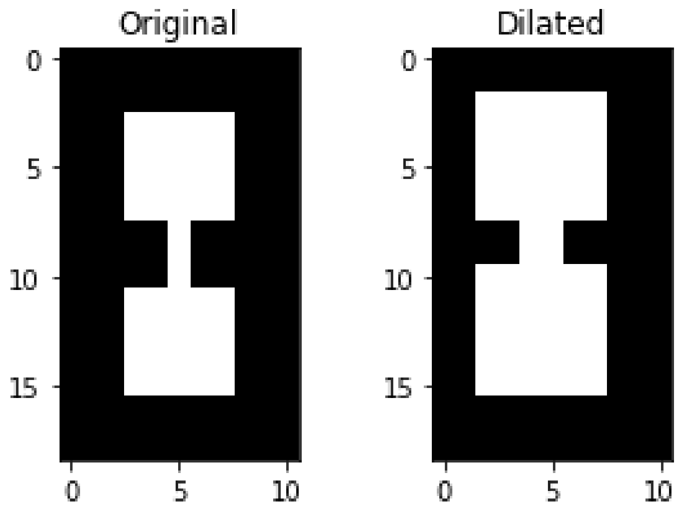
## MORPHOLOGICAL DILATION

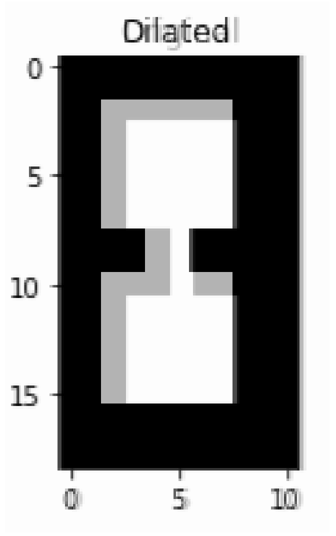
For a dumbbell-shaped array, we predict the following:

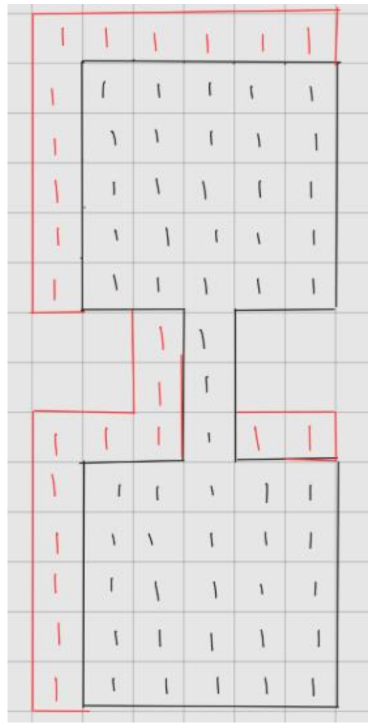


The 3x1 link now becomes 3x2, and 1s are added on the left and top sides of the 5x5 squares.

## MORPHOLOGICAL DILATION



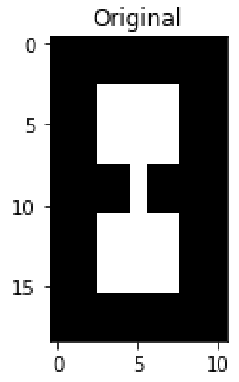


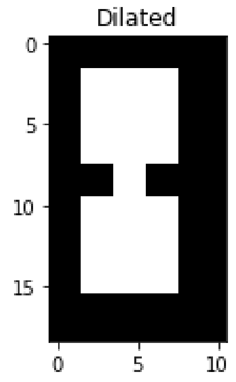


$A \oplus B$ (predicted)

Note that the predicted result agrees with the Python implementation of binary dilation using scipy.ndimage.binary_dilation

## MORPHOLOGICAL DILATION



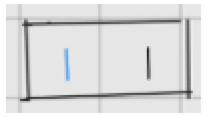


```
import numpy as np

from scipy import ndimage

struc1 = np.array([[1,1], [1,1]])

#Setup the dumbbell pattern.
pattern4 =  np.zeros((19, 11))
pattern4[3:8,3:8 ] = 1
pattern4[11:16,3:8 ] = 1
pattern4[8:11,5] = 1

#Set the origin at the upper left box of the 2x2
structural element B
dilated = ndimage.binary_dilation(pattern4, struc1,
origin=(0,0))
```

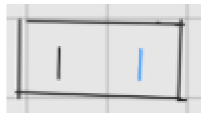
Results agree with prediction

## MORPHOLOGICAL DILATION

For a 1x2 structural element, with the origin on the left box, we have:

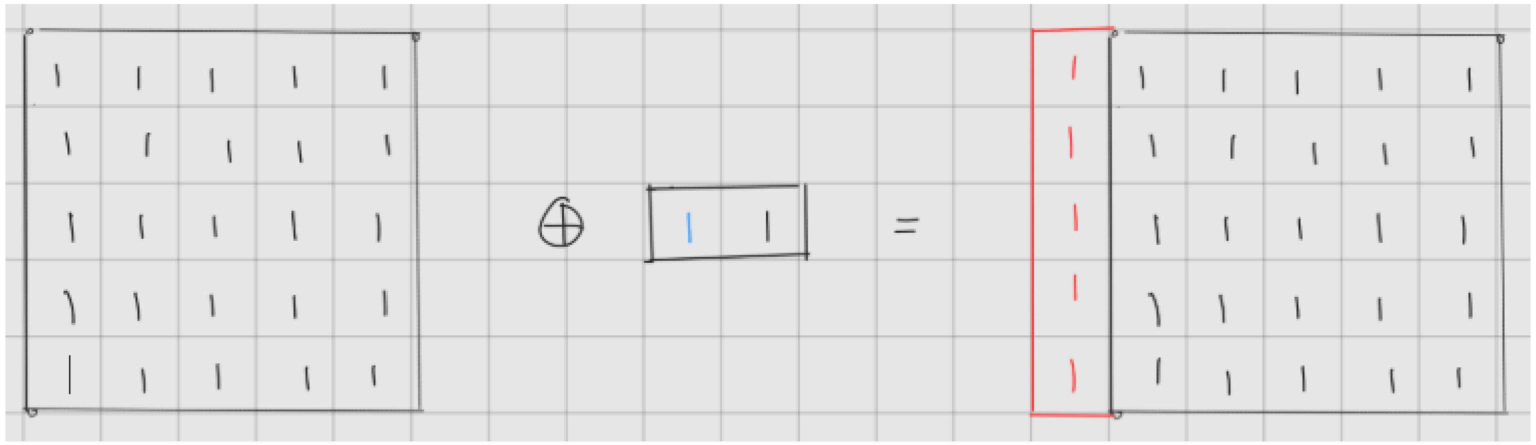


$B$

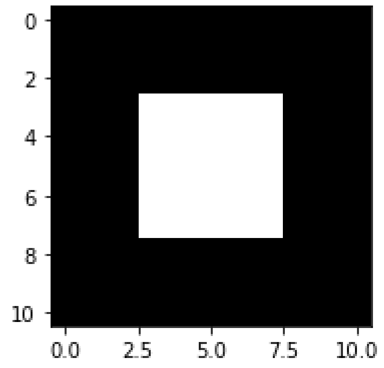


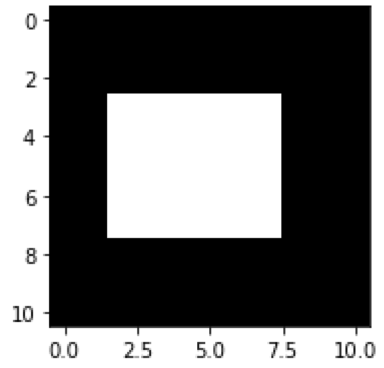
$\hat{B}$

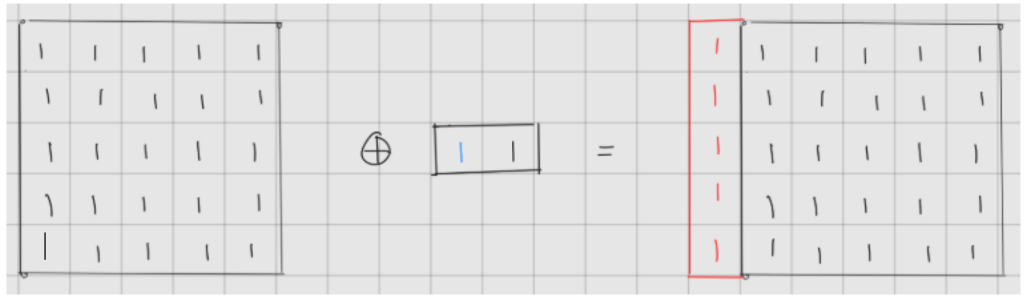
This $\hat{B}$ is now used to slide across A, with the blue 1 as the anchor.



# MORPHOLOGICAL DILATION



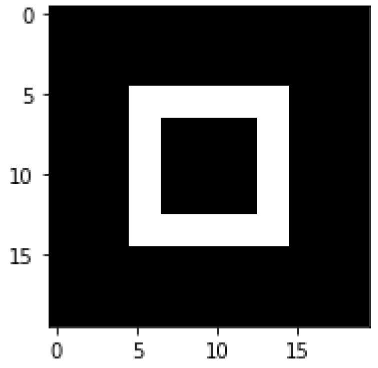


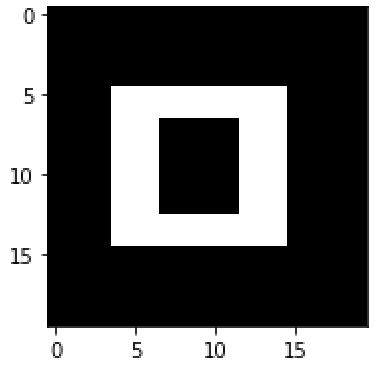


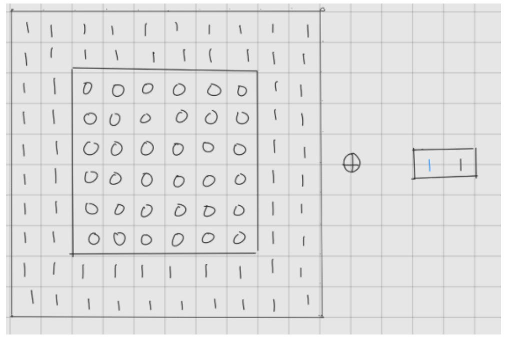
Results agree with prediction

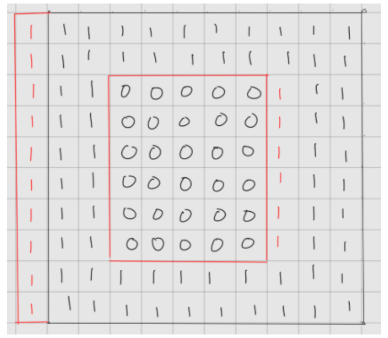
```
import numpy as np

from scipy import ndimage

struc2 = np.array([[True, True]]) #Booleans can also
be used

pattern1 = np.zeros((11,11))

pattern1[3:8, 3:8] = 1 #Create a 5x5 grid

#Set the origin at the upper left box of the 2x2
structural element B

dilated = ndimage.binary_dilation(pattern1, struc2,
origin=(0,0))
```

## MORPHOLOGICAL DILATION





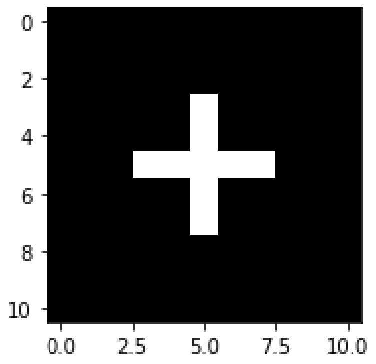
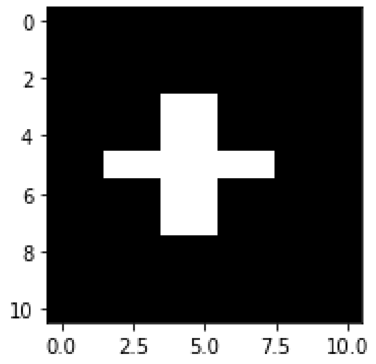
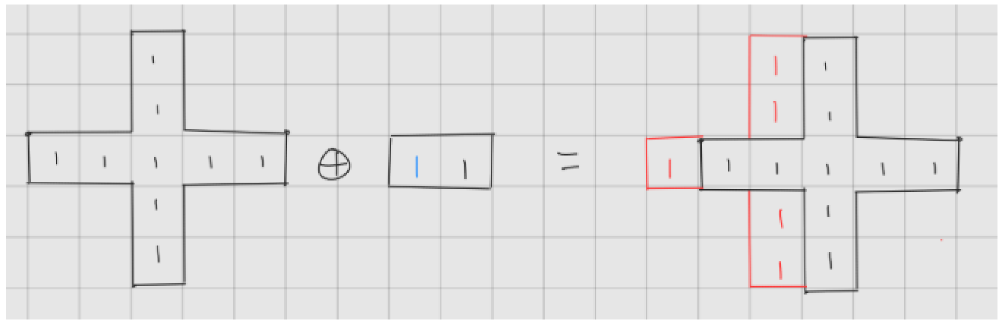




Results agree with prediction

```
import numpy as np

from scipy import ndimage

struc2 = np.array([[True, True]]) #Booleans can also
be used

#Setup the 10x10 hollow grid, 2 boxes thick.
pattern2 = np.zeros((20,20))
pattern2[5:15, 5:15] = 1
pattern2[7:13, 7:13] = 0

#Set the origin at the upper left box of the 2x2
structural element B
dilated = ndimage.binary_dilation(pattern2, struc2,
origin=(0,0))
```

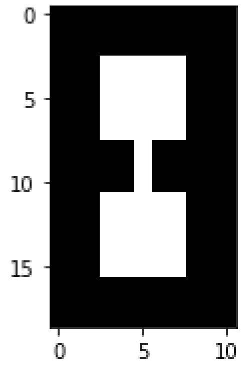
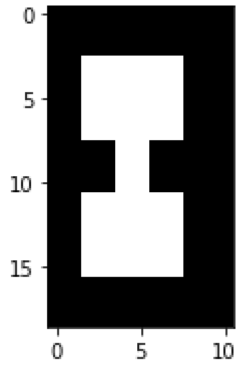
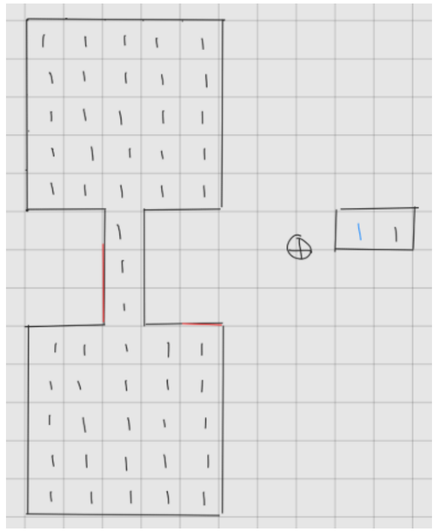
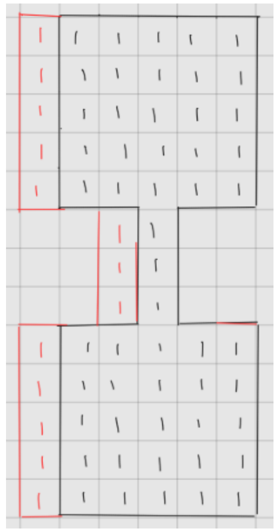
# MORPHOLOGICAL DILATION

Results agree with prediction

```
import numpy as np

from scipy import ndimage

struc2 = np.array([[True, True]]) #Booleans can also
be used

#Setup the cross pattern.
pattern3 = np.zeros((11,11))
pattern3[5,3:8] = 1
pattern3[3:8,5] = 1

#Set the origin at the upper left box of the 2x2
structural element B
dilated = ndimage.binary_dilation(pattern3, struc2,
origin=(0,0))
```

# MORPHOLOGICAL DILATION

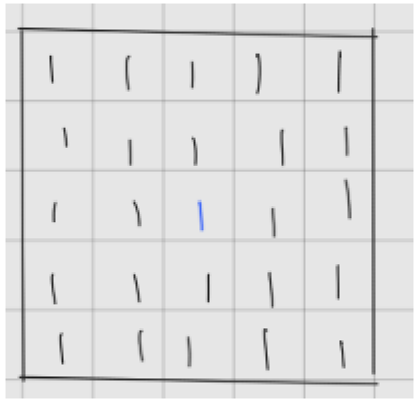





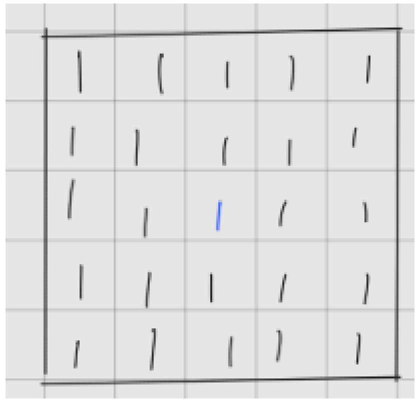
Results agree with prediction

```
import numpy as np

from scipy import ndimage

struc2 = np.array([[True, True]]) #Booleans can also
be used

#Setup the dumbbell pattern.
pattern4 =  np.zeros((19, 11))
pattern4[3:8,3:8 ] = 1
pattern4[11:16,3:8 ] = 1
pattern4[8:11,5] = 1

#Set the origin at the upper left box of the 2x2
structural element B
dilated = ndimage.binary_dilation(pattern4, struc2,
origin=(0,0))
```

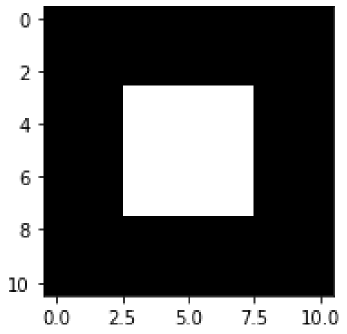
## MORPHOLOGICAL DILATION

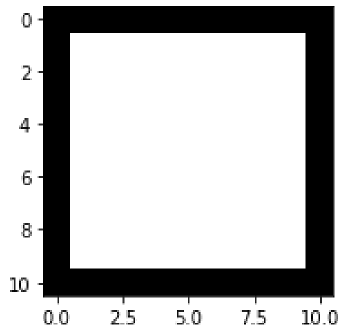
For a 5x5 structural element with the origin on the center box, we have $B = \widehat{B}$

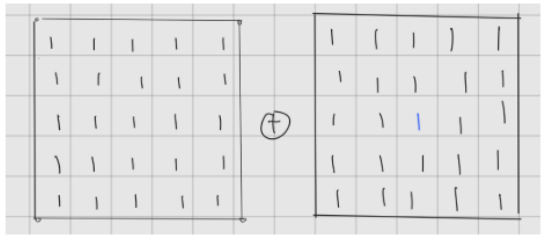

$B$

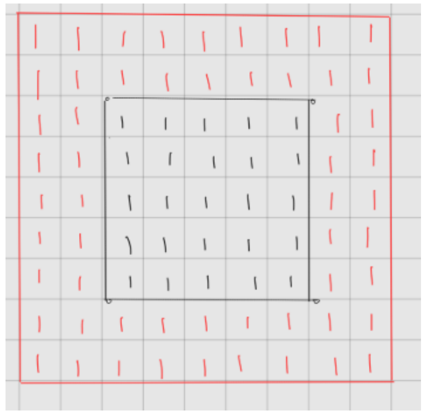

$\widehat{B}$

## MORPHOLOGICAL DILATION





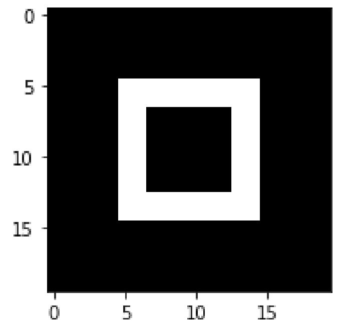


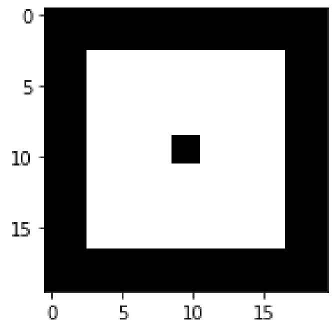


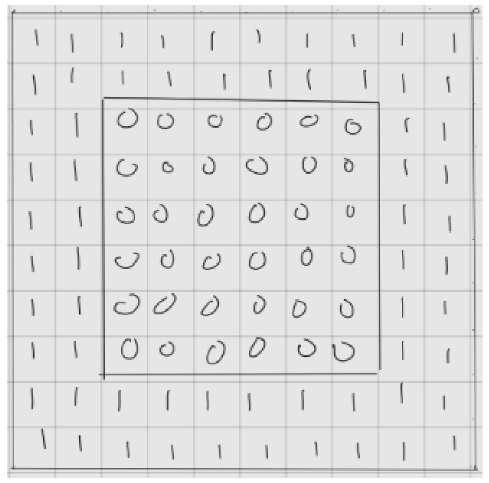
```
import numpy as np

from scipy import ndimage

struc3 = np.ones((5,5)) #Booleans can also be used

pattern1 = np.zeros((11,11))

pattern1[3:8, 3:8] = 1 #Create a 5x5 grid

#The default origin is the center, so I don‘t need
to put in an additional origin argument in the
function anymore.

dilated = ndimage.binary_dilation(pattern1, struc3)
```

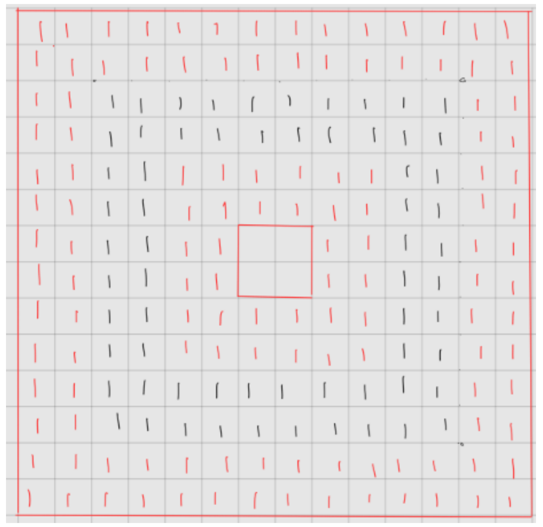
Results agree with prediction

## MORPHOLOGICAL DILATION





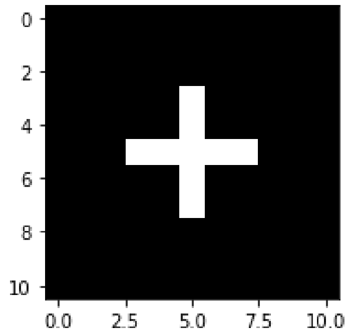


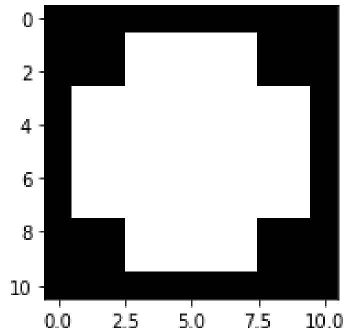


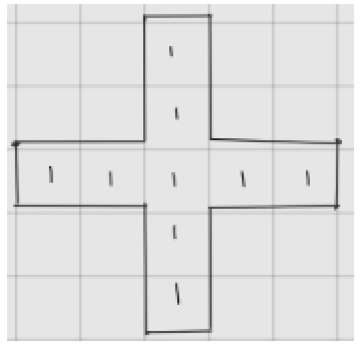
Results agree with prediction

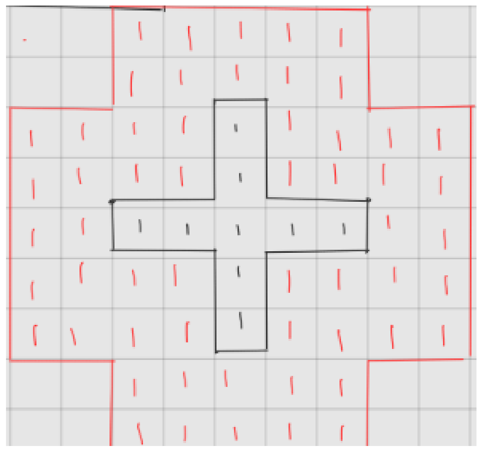
```
import numpy as np

from scipy import ndimage

struc3 = np.ones((5,5)) #Booleans can also be used

#Setup the 10x10 hollow grid, 2 boxes thick.
pattern2 = np.zeros((20,20))
pattern2[5:15, 5:15] = 1
pattern2[7:13, 7:13] = 0

#The default origin is the center, so I don't need to put in an additional origin argument in the function anymore.

dilated = ndimage.binary_dilation(pattern2, struc3)
```

# MORPHOLOGICAL DILATION





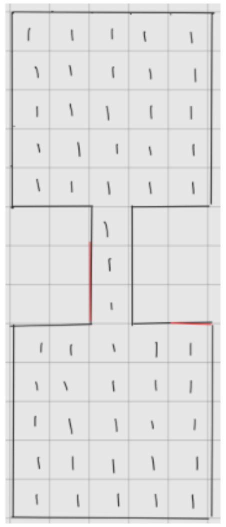


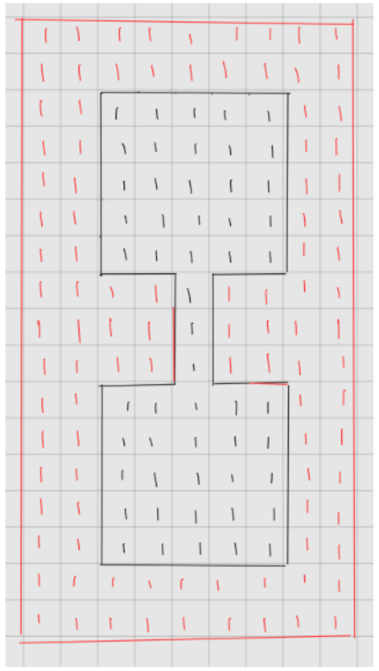


```
import numpy as np

from scipy import ndimage

struc3 = np.ones((5,5)) #Booleans can also be used

#Setup the cross pattern.
pattern3 = np.zeros((11,11))
pattern3[5,3:8] = 1
pattern3[3:8,5] = 1

#The default origin is the center, so I don‘t need
to put in an additional origin argument in the
function anymore.

dilated = ndimage.binary_dilation(pattern3, struc3)
```

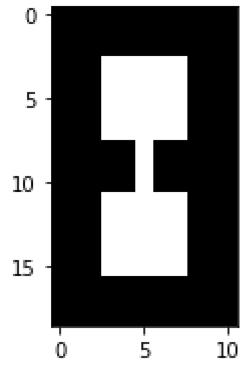
Results agree with prediction

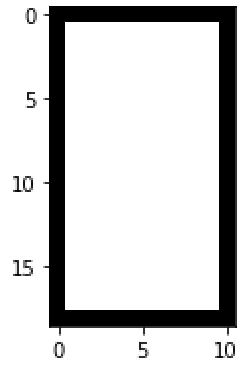
# MORPHOLOGICAL DILATION





Results agree with prediction





```
import numpy as np

from scipy import ndimage

struc3 = np.ones((5,5)) #Booleans can also be used

#Setup the dumbbell pattern.
pattern4 =  np.zeros((19, 11))
pattern4[3:8,3:8 ] = 1
pattern4[11:16,3:8 ] = 1
pattern4[8:11,5] = 1

#The default origin is the center, so I don‘t need
to put in an additional origin argument in the
function anymore.

dilated = ndimage.binary_dilation(pattern1, struc3)
```

## MORPHOLOGICAL EROSION

An erosion of a grid A with a structural element B is mathematically defined as:

$$A \ominus B = \{z | B_z \subseteq A\}$$

where $z \in I$ (I is an integer grid, which is a Numpy 2D array in our case)

Some properties:

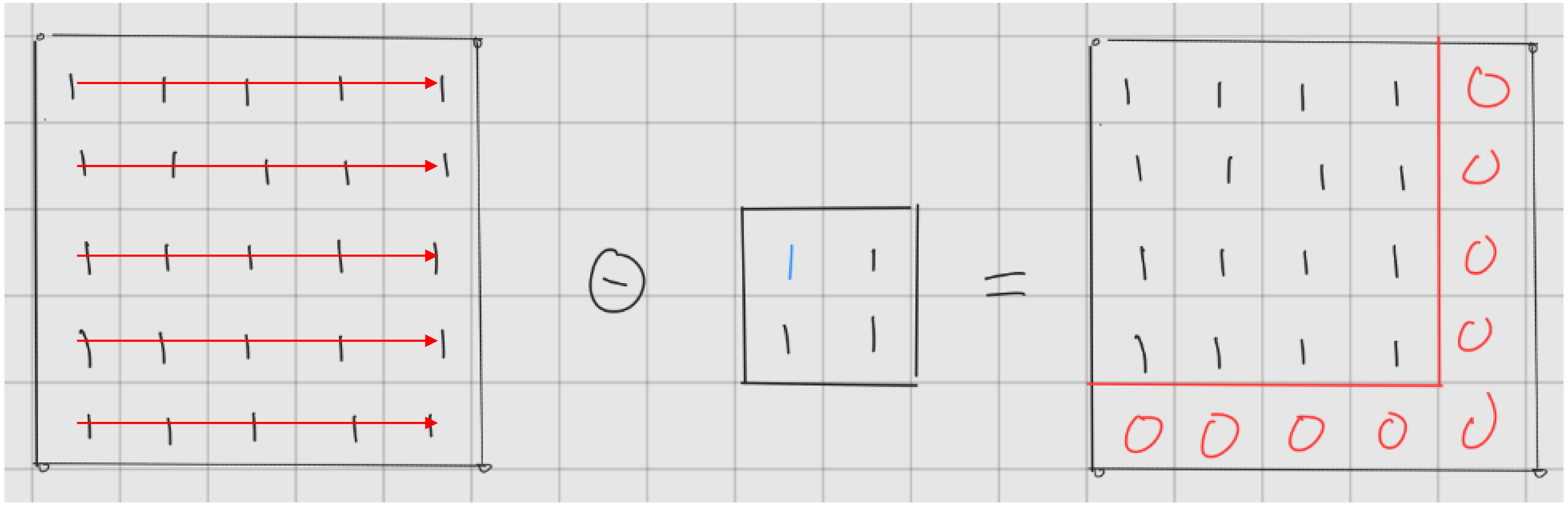
1. $A \ominus B \neq B \ominus A$ (non-commutativity)
2. $(A \ominus B) \ominus C = A \ominus (B \ominus C)$ (associativity)

This can better be explained using examples.

Note: Since there are 4 x 3 x 2 = 24 patterns (12 for dilation and 12 for erosion) I will not try to explain my thought process for all patterns because this will be extremely cumbersome. I will explain only a handful, but I'll show the result for all of them.

# MORPHOLOGICAL EROSION

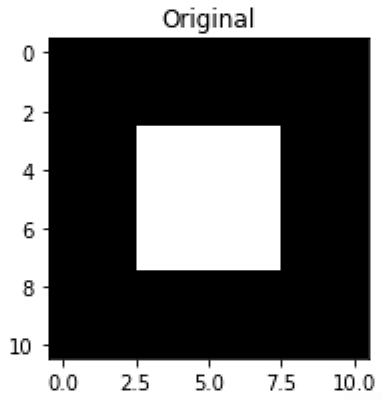
From the given definition of erosion, I make the following prediction

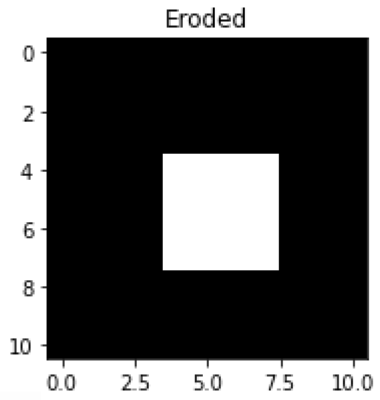


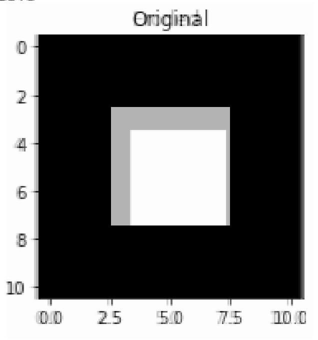
1. Superimpose the structural element B (2x2 array, origin at upper left) on every pixel of A, with the upper left as the anchor (blue)
2. If B is completely in A, then the value at that anchor is 1, else, it becomes 0.

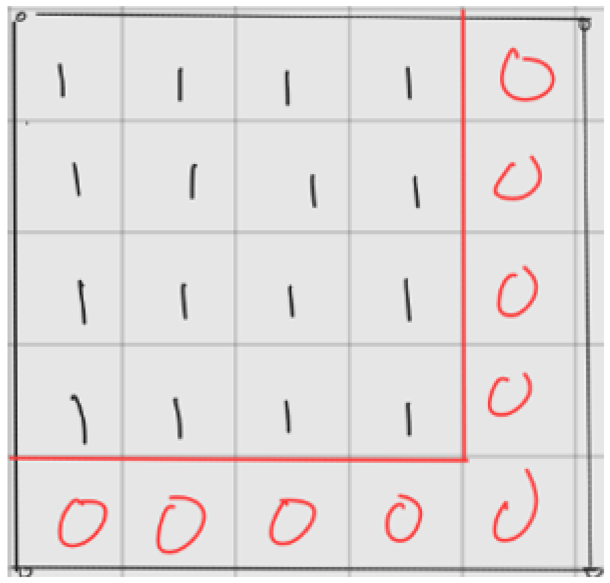
## MORPHOLOGICAL EROSION

However, the Python result is a little bit different. The top and left sides are eroded, not the bottom and the right as predicted.









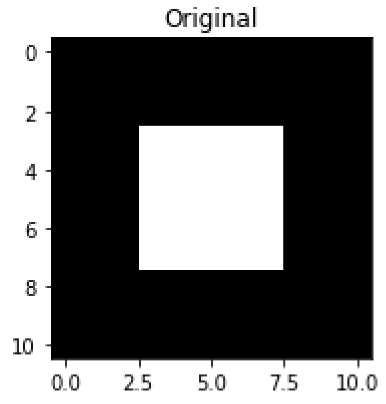
Predicted

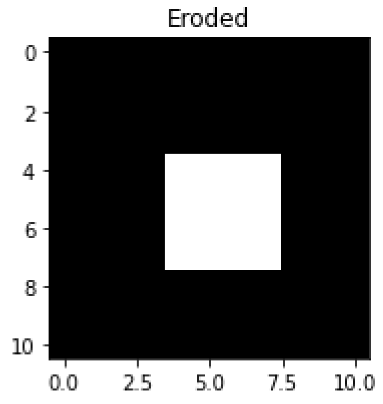
It is possible that the function scipy.ndimage.binary_erosion implements:

$$A \ominus B = \{z | (\hat{B})_z \subseteq A\}$$

Indeed, I think this must be the case, as the predictions and the actual result always seem to differ by the structural element having been reflected about the origin, as we can see from the next patterns.

## MORPHOLOGICAL EROSION

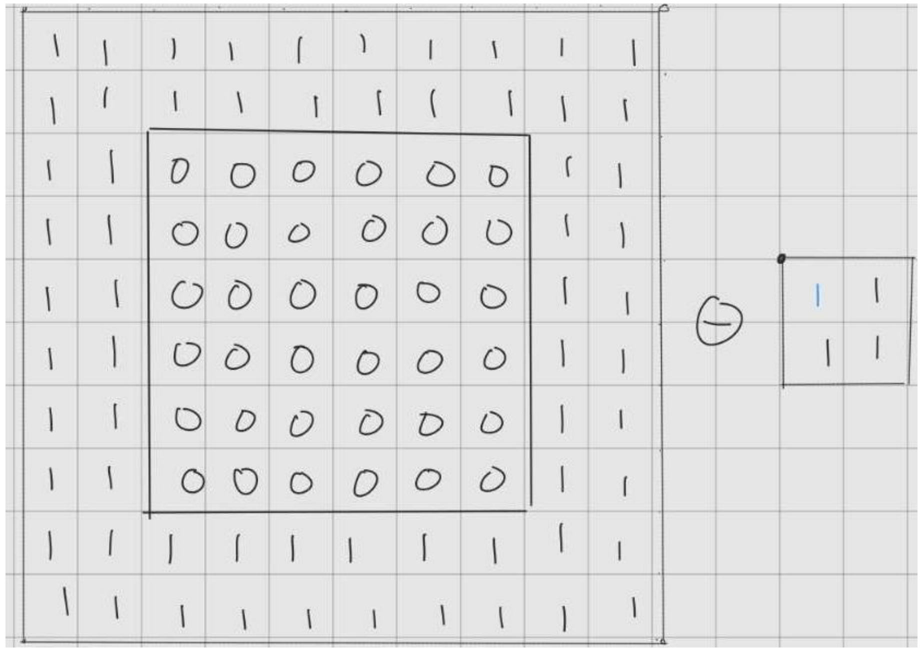




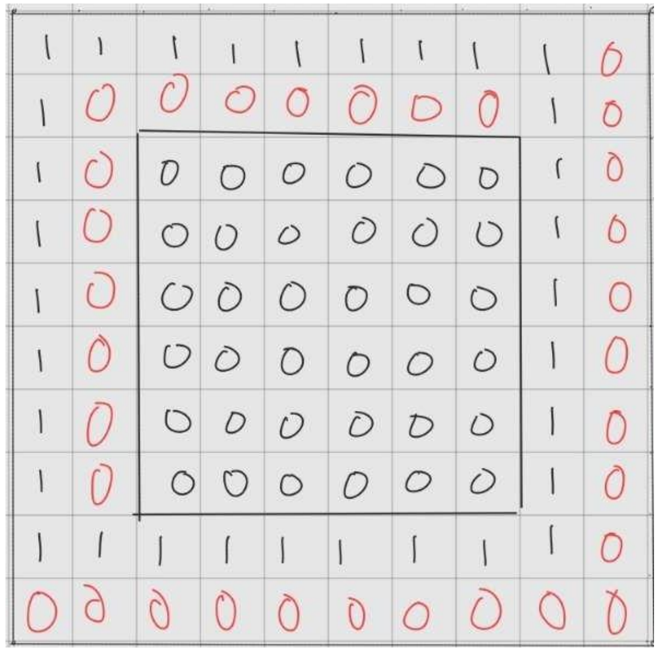
```
import numpy as np

from scipy import ndimage

struc1 = np.array([[1,1], [1,1]])

pattern1 = np.zeros((11,11))

pattern1[3:8, 3:8] = 1 #Create a 5x5 grid

#Set the origin at the upper left box of the
2x2 structural element B

dilated = ndimage.binary_erosion(pattern1,
struc1, origin=(0,0))
```

## MORPHOLOGICAL EROSION

From the given definition of erosion, I make the following prediction



$$\begin{matrix}1&1&1&1&1&1&1&1&1&1\\1&1&1&1&1&1&1&1&1&1\\1&1&0&0&0&0&0&0&1&1\\1&1&0&0&0&0&0&0&1&1\\1&1&0&0&0&0&0&0&1&1\\1&1&0&0&0&0&0&0&1&1\\1&1&0&0&0&0&0&0&1&1\\1&1&0&0&0&0&0&0&1&1\\1&1&1&1&1&1&1&1&1&1\\1&1&1&1&1&1&1&1&1&1\end{matrix} \ominus \begin{matrix}1&1\\1&1\end{matrix}$$

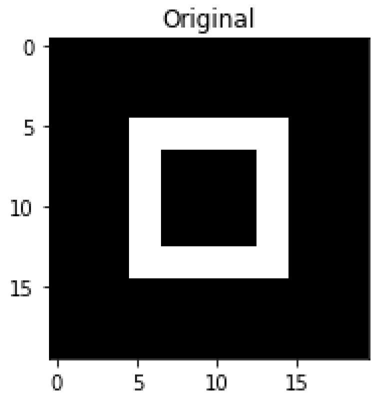


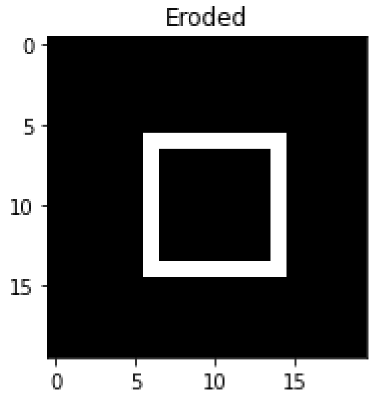
$$\begin{matrix}1&1&1&1&1&1&1&1&1&0\\1&0&0&0&0&0&0&0&1&0\\1&0&0&0&0&0&0&0&1&0\\1&0&0&0&0&0&0&0&1&0\\1&0&0&0&0&0&0&0&1&0\\1&0&0&0&0&0&0&0&1&0\\1&0&0&0&0&0&0&0&1&0\\1&0&0&0&0&0&0&0&1&0\\1&1&1&1&1&1&1&1&1&0\\0&0&0&0&0&0&0&0&0&0\end{matrix}$$

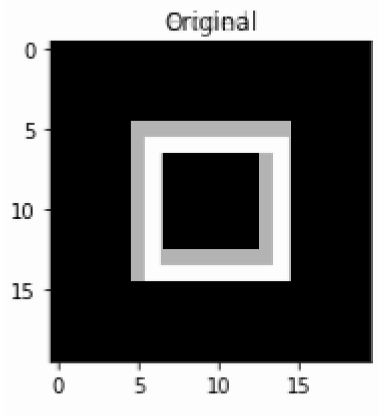
1. Superimpose the structural element B (2x2 array, origin at upper left) on every pixel of A, with the upper left as the anchor (blue)
2. If B is completely in A, then the value at that anchor is 1, else, it becomes 0.
3. Note that the top and the left of the inner square become zeros. The right and the bottom sides of the larger square become zeros also.

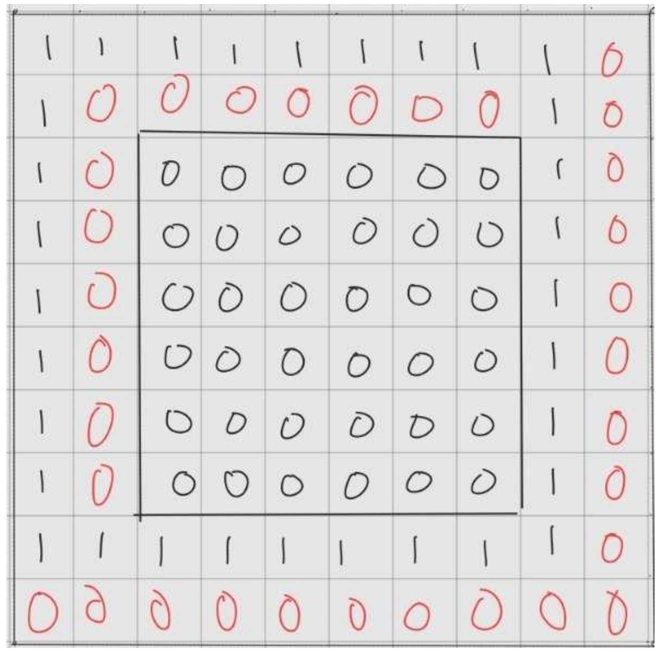
# MORPHOLOGICAL EROSION

Yet again, it seems that the Python implementation reflects B about the origin before superimposing.







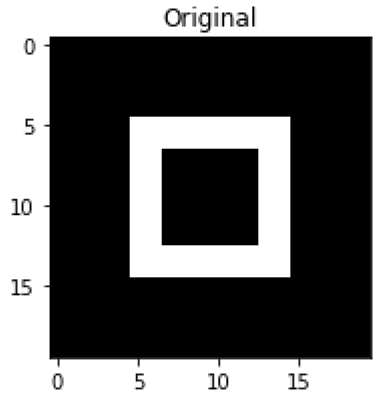


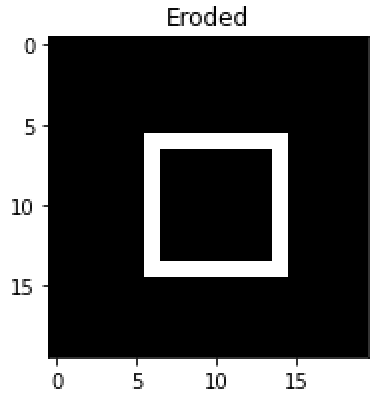
| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Predicted

I superimposed the original and the eroded image. Note that the red zeros (grey area in the superimposed image) are mirror images about the origin.

# MORPHOLOGICAL EROSION

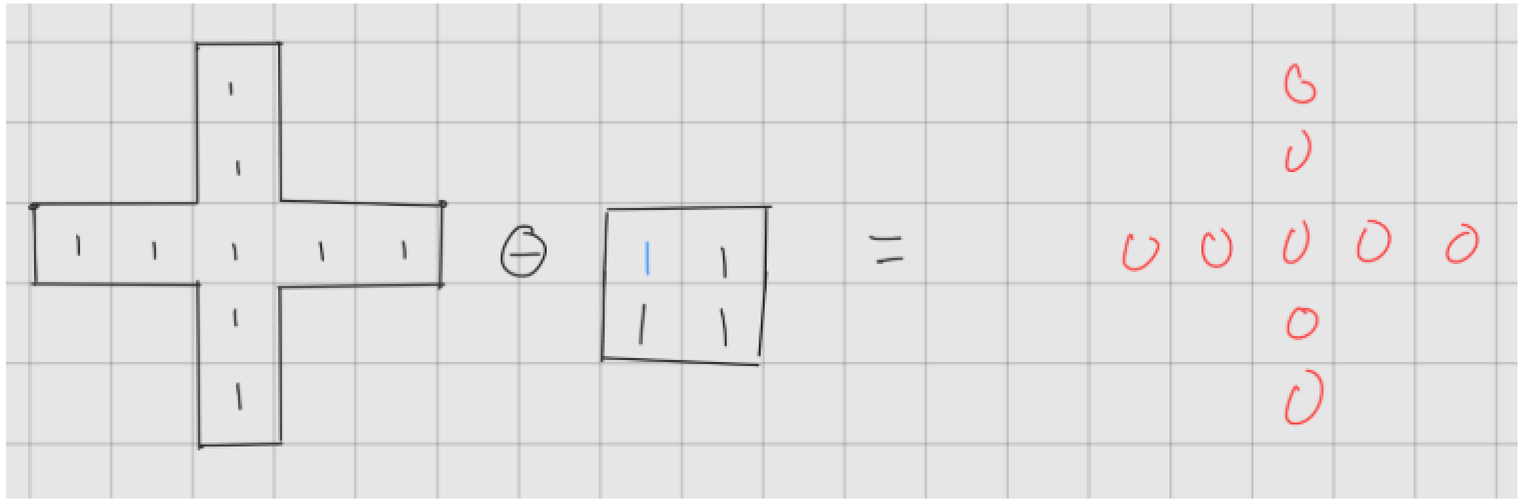




```
import numpy as np
from scipy import ndimage
struc1 = np.ones((2,2))#Booleans can also be
used
#Setup the 10x10 hollow grid, 2 boxes thick.
pattern2 = np.zeros((20,20))
pattern2[5:15, 5:15] = 1
pattern2[7:13, 7:13] = 0
#Set the origin at the upper left box of the
2x2 structural element B
dilated = ndimage.binary_erosion(pattern2,
struc1, origin=(0,0))
```

## MORPHOLOGICAL EROSION

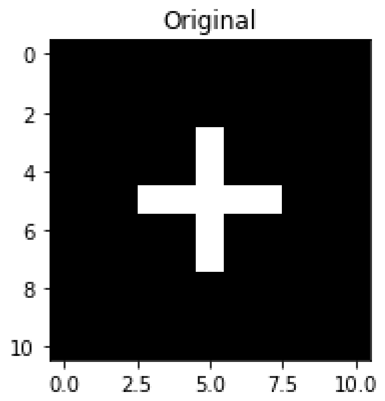
From the given definition of erosion, I make the following prediction

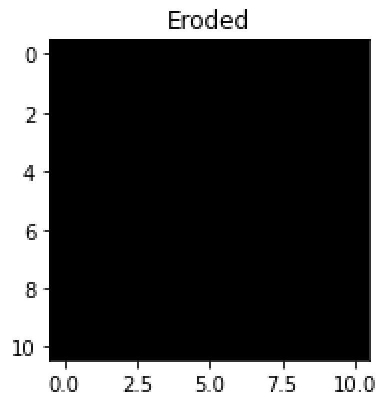


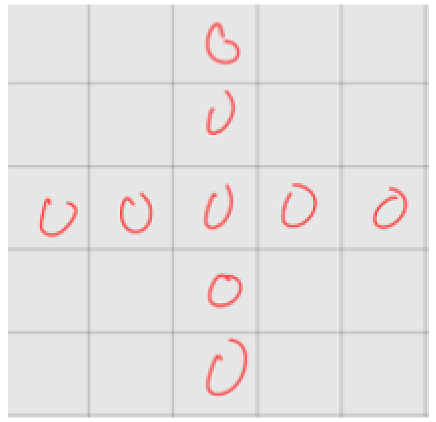
1. Superimpose the structural element B (2x2 array, origin at upper left) on every pixel of A, with the upper left as the anchor (blue)
2. If B is completely in A, then the value at that anchor is 1, else, it becomes 0.
3. Prediction: Nothing remains after erosion!

## MORPHOLOGICAL EROSION

This time, the prediction agrees with the result.





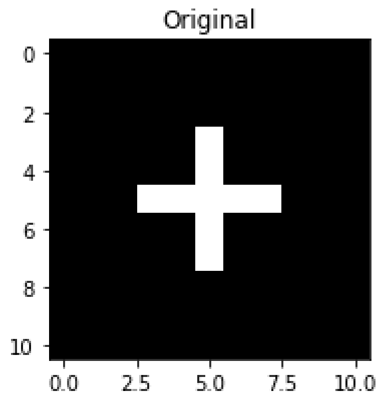


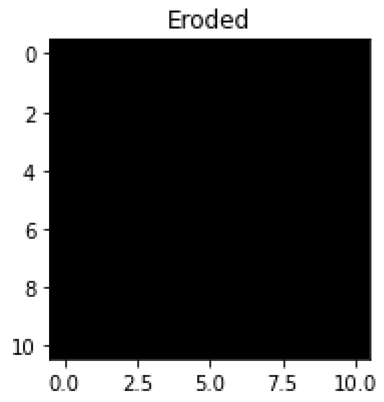
Predicted

It doesn't matter whether we implement $A \ominus B = \{z | B_z \subseteq A\}$ or $A \ominus B = \{z | (\hat{B})_z \subseteq A\}$

## MORPHOLOGICAL EROSION

This time, the prediction agrees with the result.

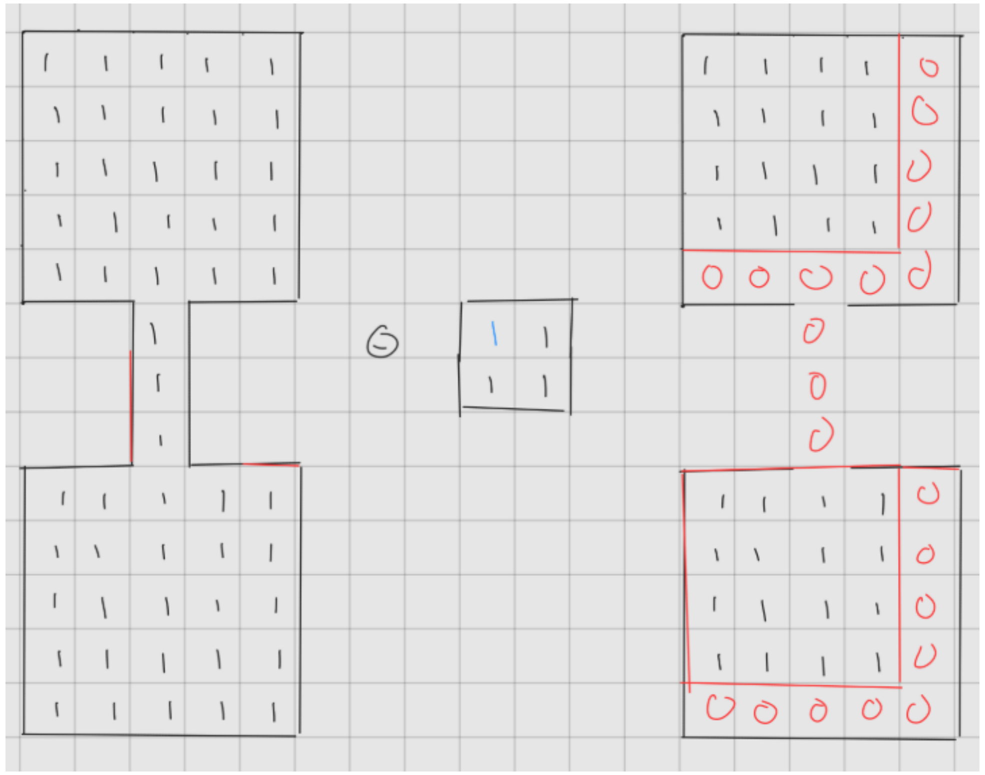




```
import numpy as np
from scipy import ndimage
struc1 = np.ones((2,2))

#Setup the cross pattern.
pattern3 = np.zeros((11,11))
pattern3[5,3:8] = 1
pattern3[3:8,5] = 1

dilated = ndimage.binary_erosion(pattern3, struc1)
```

## MORPHOLOGICAL EROSION

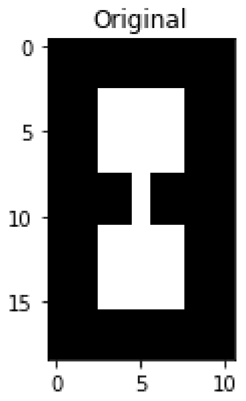
From the given definition of erosion, I make the following prediction

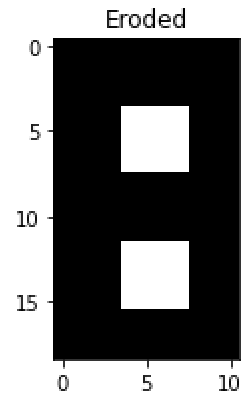


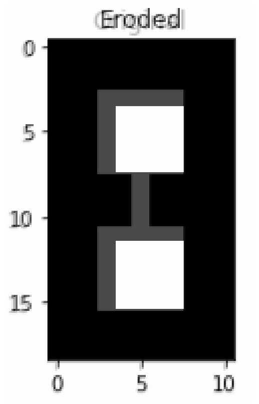
1. Superimpose the structural element B (2x2 array, origin at upper left) on every pixel of A, with the upper left as the anchor (blue)
2. If B is completely in A, then the value at that anchor is 1, else, it becomes 0.

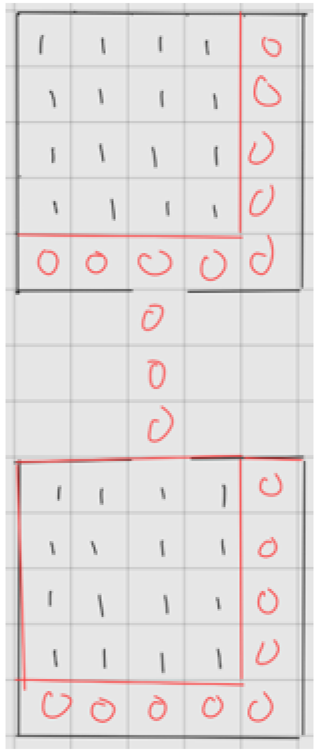
# MORPHOLOGICAL EROSION

Yet again, it seems that the Python implementation reflects B about the origin before superimposing.



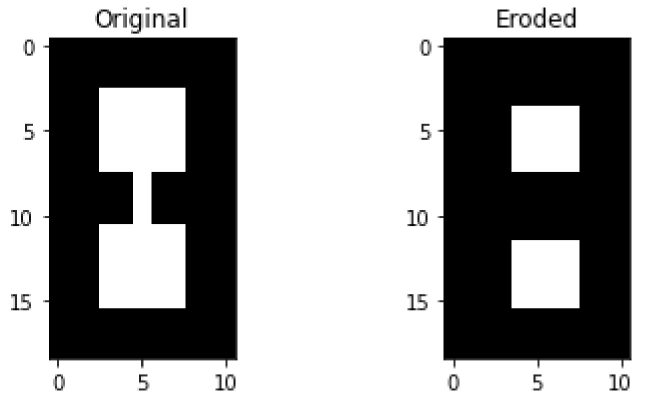





Predicted

I superimposed the original and the eroded image. Note that the red zeros (grey area in the superimposed image) are mirror images about the origin.

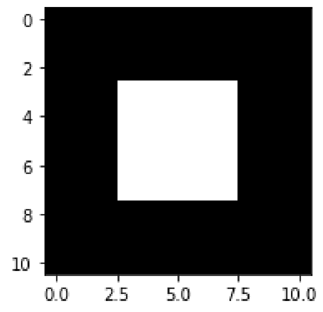
## MORPHOLOGICAL EROSION

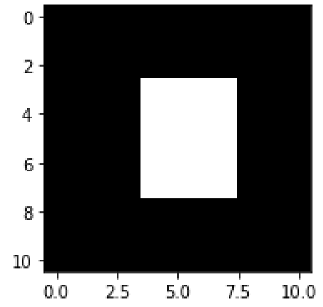


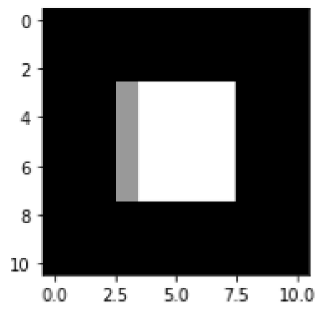
```
import numpy as np

from scipy import ndimage

struc1 = np.array([[1,1], [1,1]])

#Setup the dumbbell pattern.
pattern4 =  np.zeros((19, 11))
pattern4[3:8,3:8 ] = 1
pattern4[11:16,3:8 ] = 1
pattern4[8:11,5] = 1

#Set the origin at the upper left box of the 2x2
structural element B
dilated = ndimage.binary_erosion(pattern4, struc1,
origin=(0,0))
```

## MORPHOLOGICAL EROSION

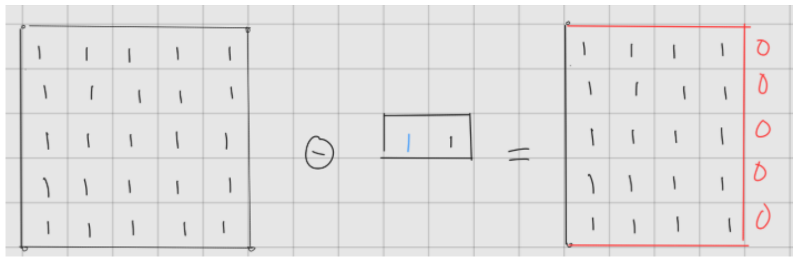
`Structural element is a 1x2 array with origin at the left.`





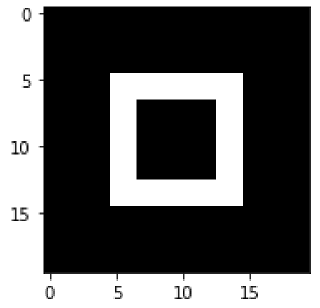


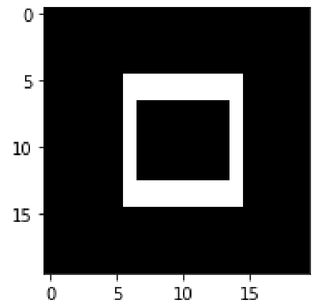
```
import numpy as np

from scipy import ndimage

struc2 = np.array([[True, True]]) #Booleans can also
be used

pattern1 = np.zeros((11,11))

pattern1[3:8, 3:8] = 1 #Create a 5x5 grid

#Set the origin at the upper left box of the 2x2
structural element B

dilated = ndimage.binary_erosion(pattern1, struc2,
origin=(0,0))
```

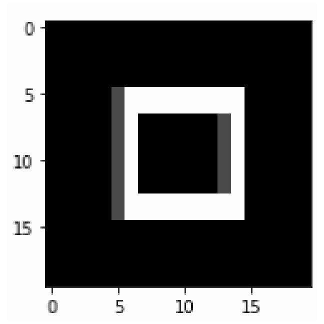


Predicted result is mirrored about origin

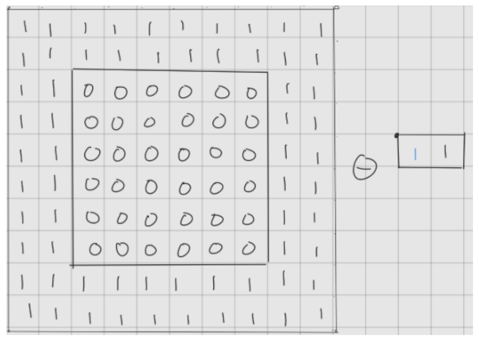
## MORPHOLOGICAL EROSION

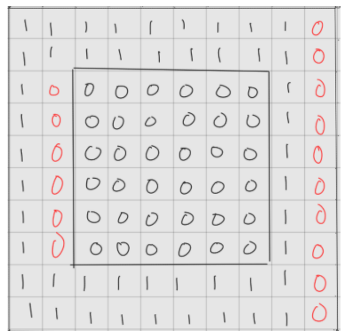






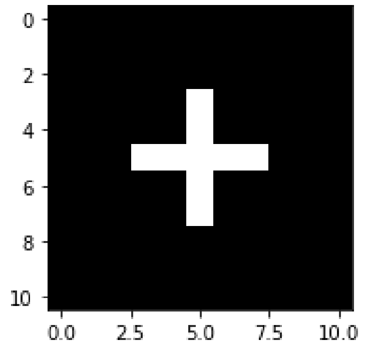
```
import numpy as np

from scipy import ndimage

struc2 = np.array([[True, True]]) #Booleans can also
be used

#Setup the 10x10 hollow grid, 2 boxes thick.
pattern2 = np.zeros((20,20))
pattern2[5:15, 5:15] = 1
pattern2[7:13, 7:13] = 0

#Set the origin at the upper left box of the 2x2
structural element B
dilated = ndimage.binary_erosion(pattern2, struc2,
origin=(0,0))
```

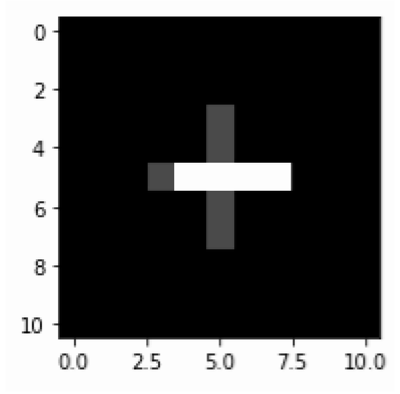


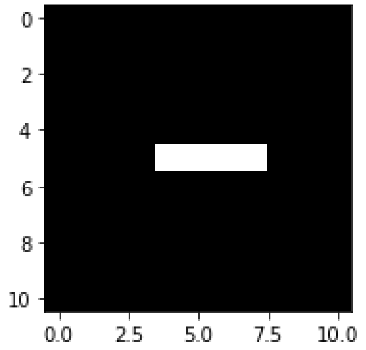


Predicted result is mirrored about the origin

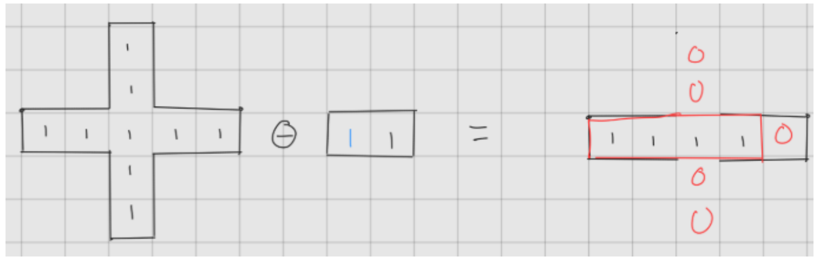
## MORPHOLOGICAL EROSION





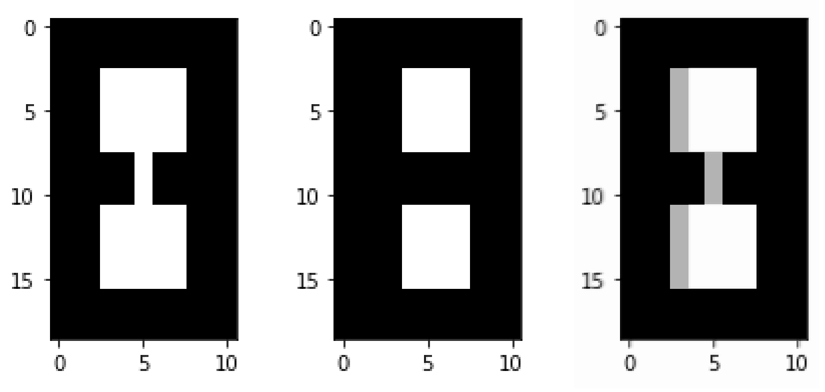


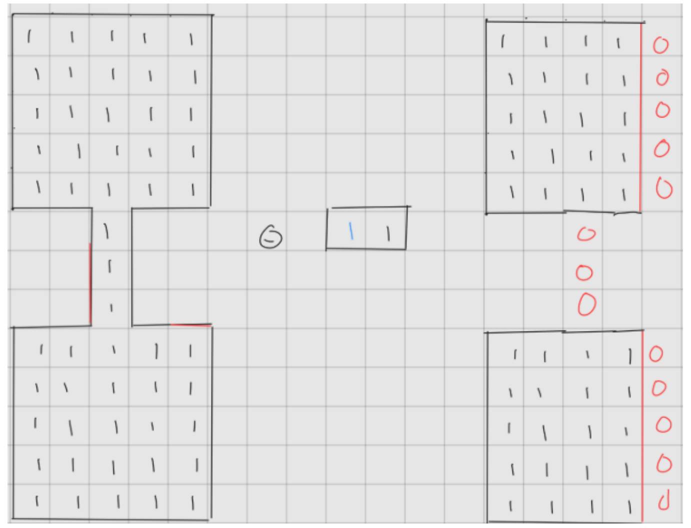
```
import numpy as np

from scipy import ndimage

struc1 = np.array([[1,1], [1,1]])

#Setup the cross pattern.
pattern3 = np.zeros((11,11))
pattern3[5,3:8] = 1
pattern3[3:8,5] = 1

#Set the origin at the upper left box of the 2x2
structural element B
dilated = ndimage.binary_erosion(pattern3, struc1,
origin=(0,0))
```



Predicted result is mirrored about the origin

## MORPHOLOGICAL EROSION

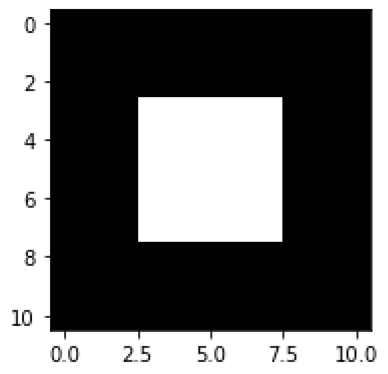
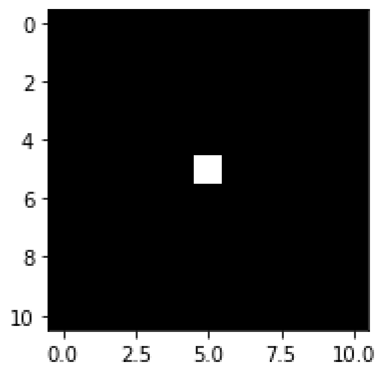
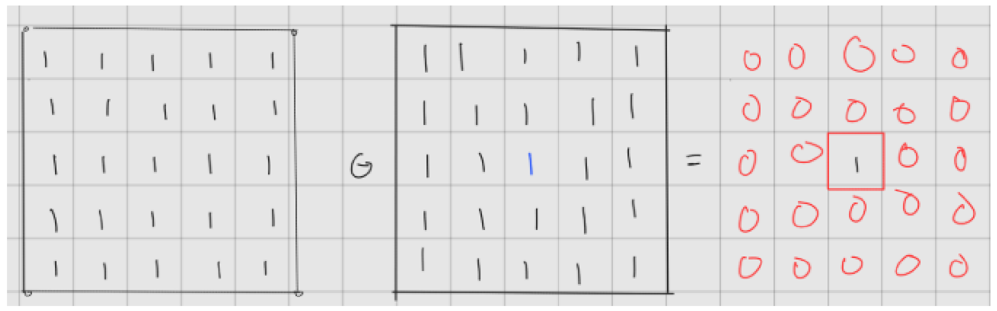




Predicted result is mirrored about the origin

```
import numpy as np

from scipy import ndimage

struc2 = np.array([[True, True]]) #Booleans can also be used

#Setup the dumbbell pattern.
pattern4 =  np.zeros((19, 11))
pattern4[3:8,3:8 ] = 1
pattern4[11:16,3:8 ] = 1
pattern4[8:11,5] = 1

#Set the origin at the upper left box of the 2x2 structural element B
dilated = ndimage.binary_erosion(pattern4, struc2, origin=(0,0))
```

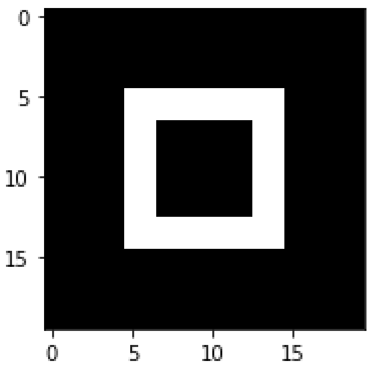
# MORPHOLOGICAL DILATION

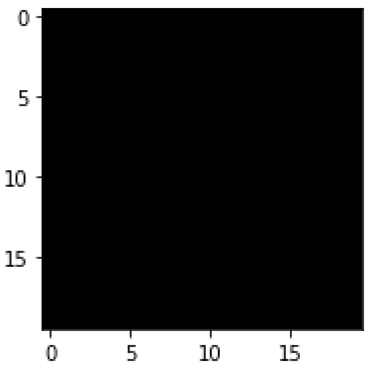




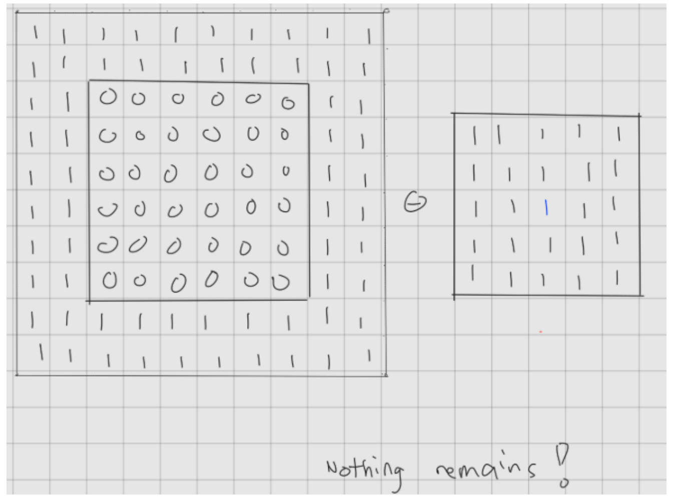
```
import numpy as np

from scipy import ndimage

struc3 = np.ones((5,5)) #Booleans can also be used

pattern1 = np.zeros((11,11))

pattern1[3:8, 3:8] = 1 #Create a 5x5 grid

#The default origin is the center, so I don‘t need
to put in an additional origin argument in the
function anymore.

dilated = ndimage.binary_erosion(pattern1, struc3)
```

Results agree with prediction

## MORPHOLOGICAL DILATION



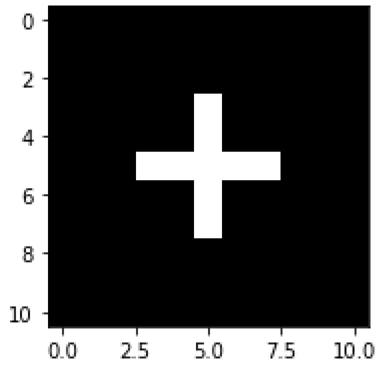


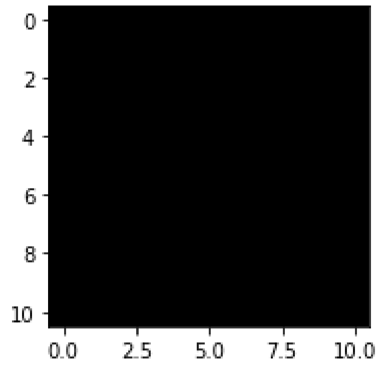


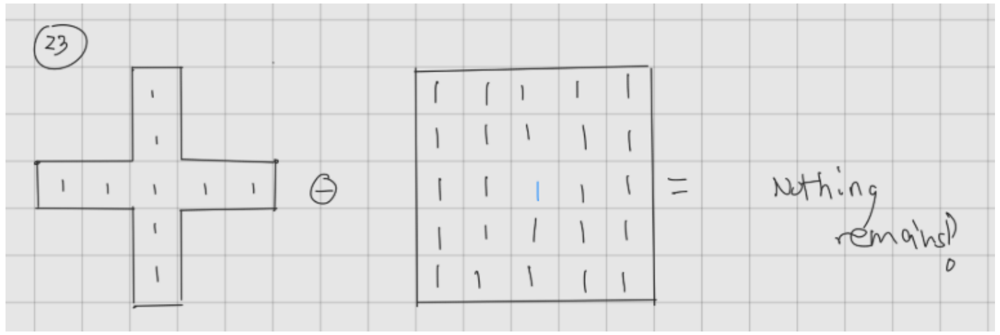
Results agree with prediction

```
import numpy as np

from scipy import ndimage

struc3 = np.ones((5,5)) #Booleans can also be
used

#Setup the 10x10 hollow grid, 2 boxes thick.
pattern2 = np.zeros((20,20))
pattern2[5:15, 5:15] = 1
pattern2[7:13, 7:13] = 0

#The default origin is the center, so I don't
need to put in an additional origin argument
in the function anymore.

dilated = ndimage.binary_erosion(pattern2,
struc3)
```

## MORPHOLOGICAL EROSION



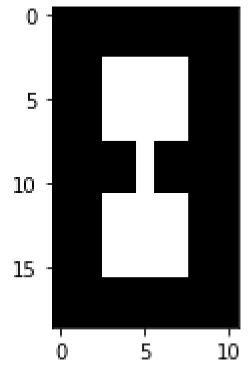


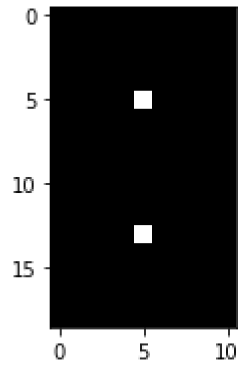


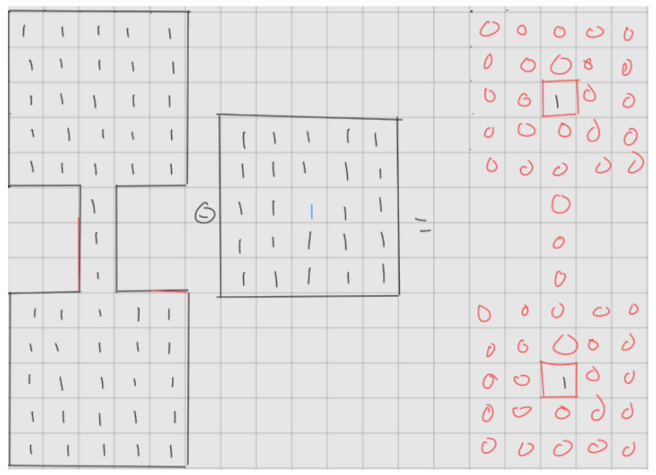
```
import numpy as np

from scipy import ndimage

struc3 = np.ones((5,5)) #Booleans can also be used

#Setup the cross pattern.
pattern3 = np.zeros((11,11))
pattern3[5,3:8] = 1
pattern3[3:8,5] = 1

#The default origin is the center, so I don't need
to put in an additional origin argument in the
function anymore.

dilated = ndimage.binary_erosion(pattern3, struc3)
```

Results agree with prediction

# MORPHOLOGICAL EROSION







Results agree with prediction

```
import numpy as np

from scipy import ndimage

struc3 = np.ones((5,5)) #Booleans can also be used

#Setup the dumbbell pattern.
pattern4 =  np.zeros((19, 11))
pattern4[3:8,3:8 ] = 1
pattern4[11:16,3:8 ] = 1
pattern4[8:11,5] = 1

#The default origin is the center, so I don‘t need
to put in an additional origin argument in the
function anymore.

dilated = ndimage.binary_erosion(pattern1, struc3)
```

## MORPHOLOGICAL EROSION

The fact that the prediction and the results become the same when the origin is at the center (and thus, $\hat{B} = B$) leads me to believe that skimage.ndimage.binary_erosion is defined as $A \ominus B = \{z | (\hat{B})_z \subseteq A\}$

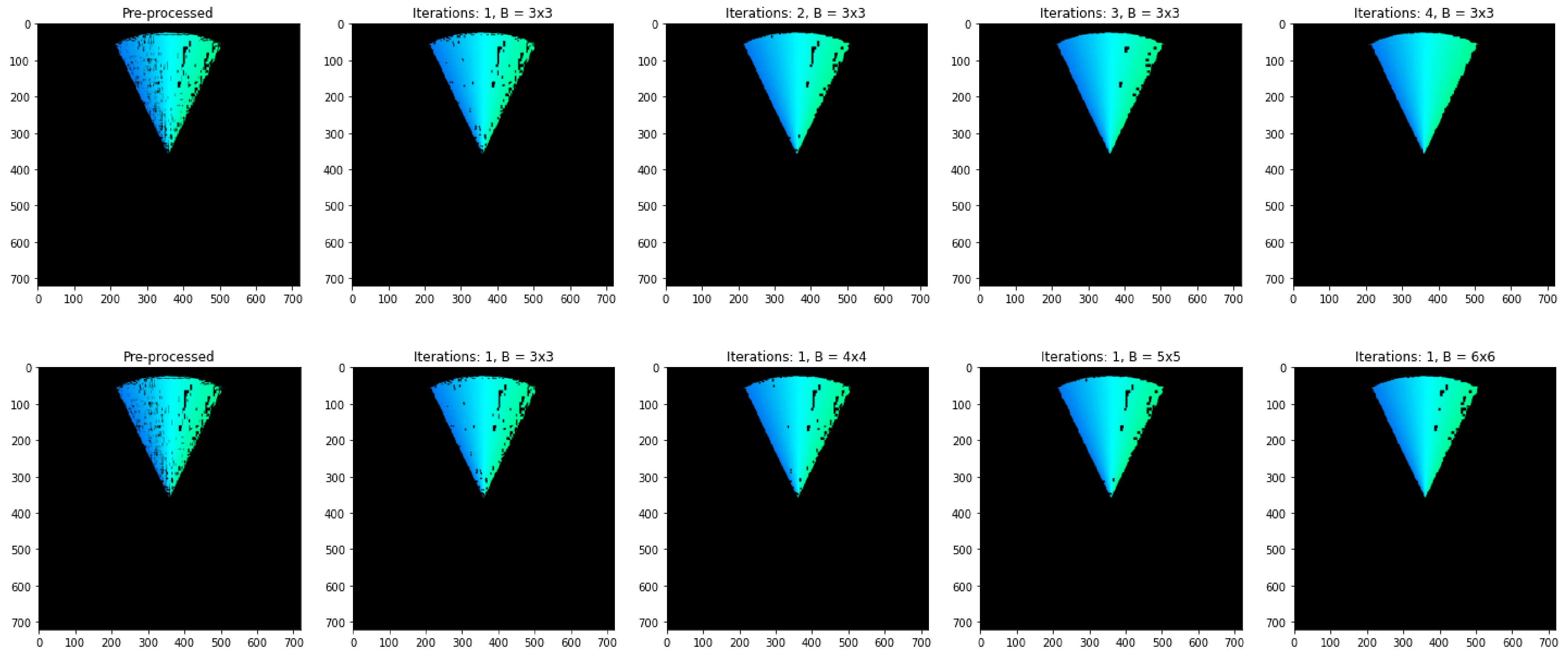
## OBJECTIVES

1. Predict the effects of morphological dilation and erosion of simple patterns using simple structural elements and explain the result.
2. Perform and demonstrate morphological segmentation in images.

## MORPHOLOGICAL CLOSING

Morphological closing is defined as $(A \oplus B) \ominus B$. It can be used to close small holes in an image (by eroding first) without changing the shape of the image drastically (unlike pure erosion which runs the risk of “over-eroding” at the edges) by dilating after.

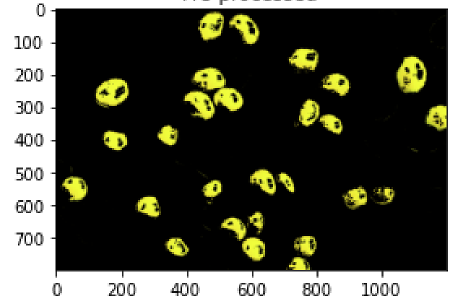
The larger the structural element B, the larger the holes it can close per iteration. Increasing the number of iterations ensures that more holes are closed.

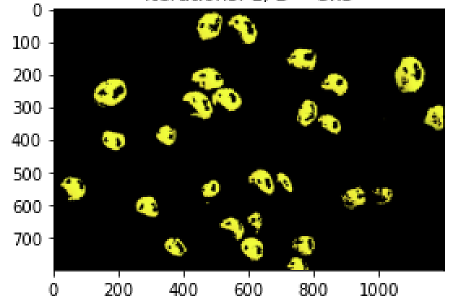


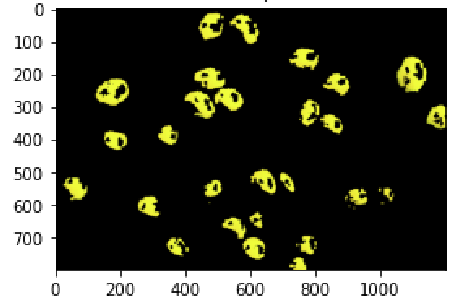
# MORPHOLOGICAL OPENING

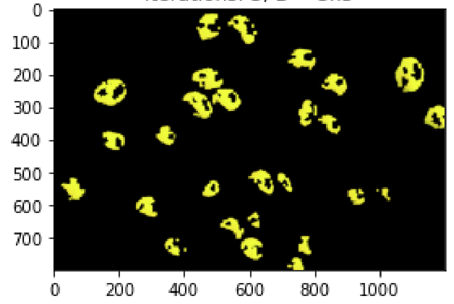
Morphological opening is defined as $(A \ominus B) \oplus B$. It can be used to separate touching regions in an image (by dilating first) without changing the shape of the image drastically (unlike pure erosion which runs the risk of "over-dilating" at the edges) by dilating after.

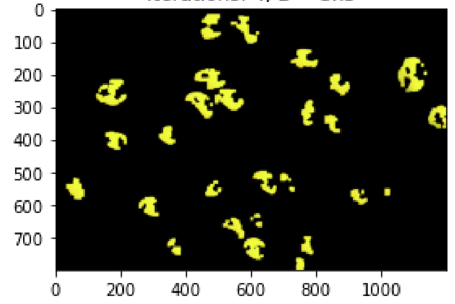
The greater the iterations, the more touching regions that can be separated (at the risk of over-separating or over-eroding). The greater the size of the structural element, the more aggressive the separation becomes (at a constant iteration).

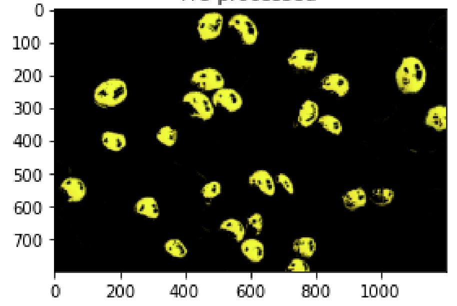

Pre-processed

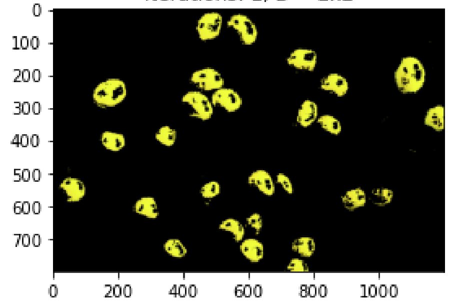

Iterations: 1, B = 3x3

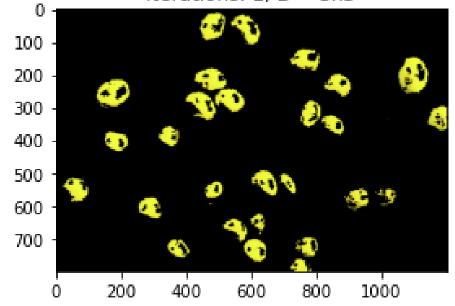

Iterations: 2, B = 3x3

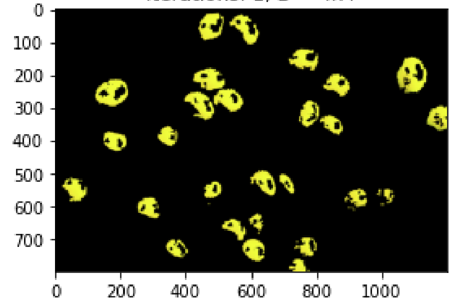

Iterations: 3, B = 3x3

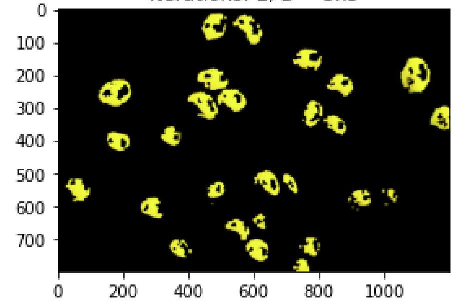

Iterations: 4, B = 3x3


Pre-processed


Iterations: 1, B = 2x2


Iterations: 1, B = 3x3


Iterations: 1, B = 4x4

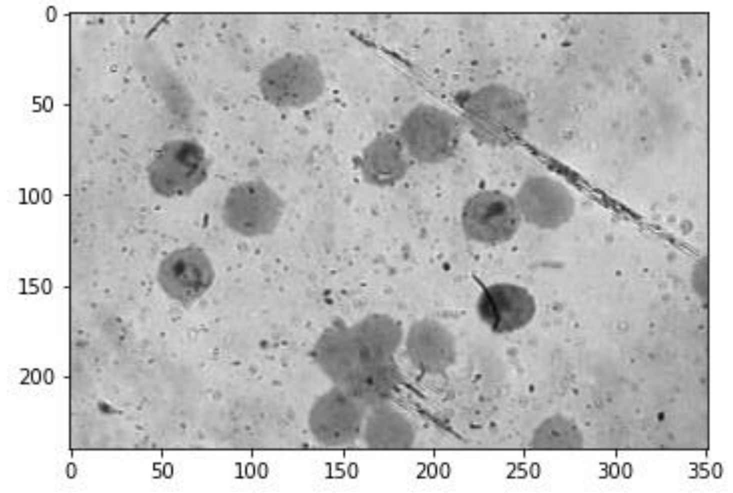

Iterations: 1, B = 5x5

## APPLICATIONS

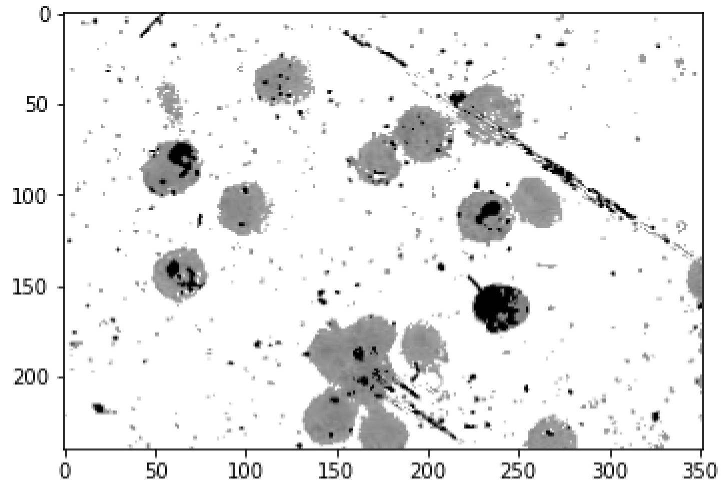
The four basic morphological operations (dilation, erosion, closing, and opening) can be used to remove artifacts and improve image segmentation. Here, I segment an image containing cells infested with malaria.

Step 1: Removal of background by **grayscale thresholding**

Original



Post-threshold process



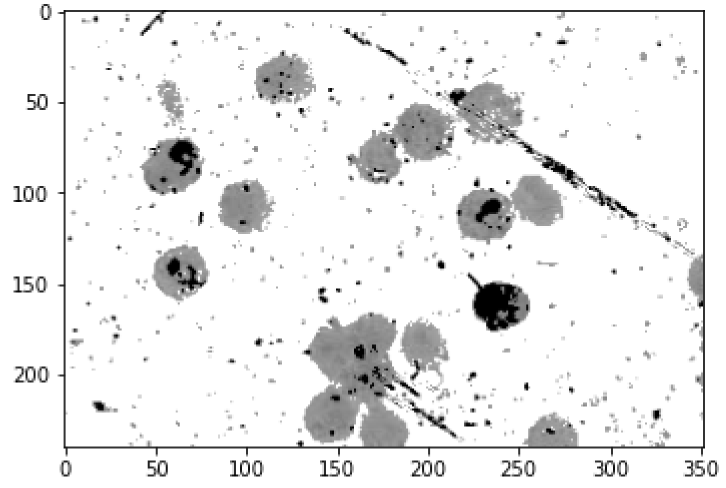
Here, the part of the grayscale image with values between 121 and 164 are extracted.

## APPLICATIONS

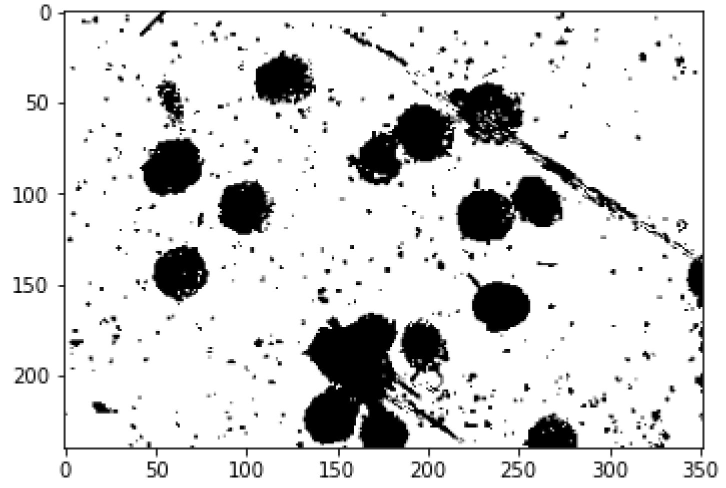
The four basic morphological operations (dilation, erosion, closing, and opening) can be used to remove artifacts and improve image segmentation. Here, I segment an image containing cells infested with malaria.

Step 2: Binarization

Pre-binarized Image



Binarized Image



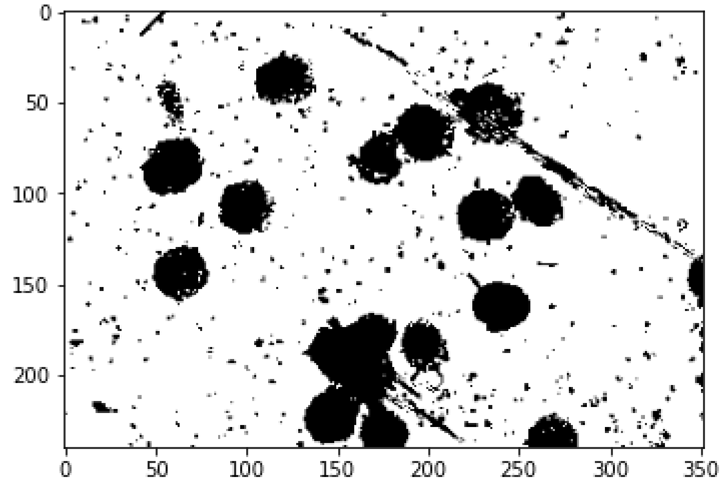
To binarize the image, I used the skimage.filters.threshold_otsu function.

## APPLICATIONS

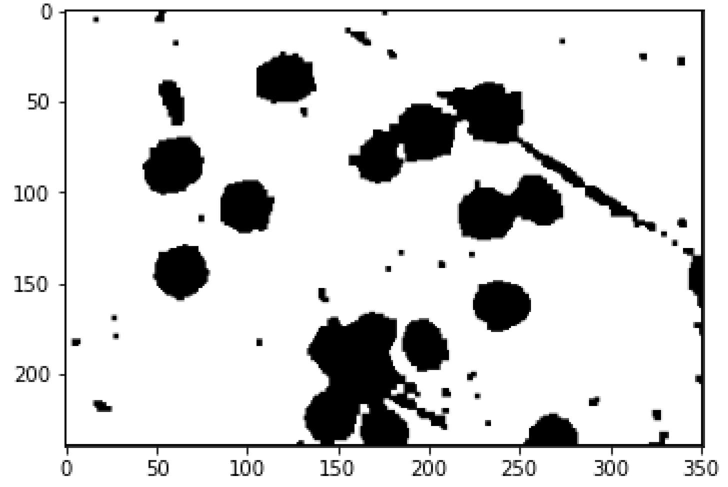
The four basic morphological operations (dilation, erosion, closing, and opening) can be used to remove artifacts and improve image segmentation. Here, I segment an image containing cells infested with malaria.

Step 3: Morphological Cleaning

Preprocessed

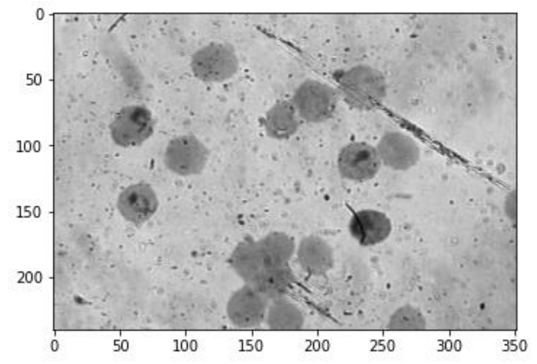


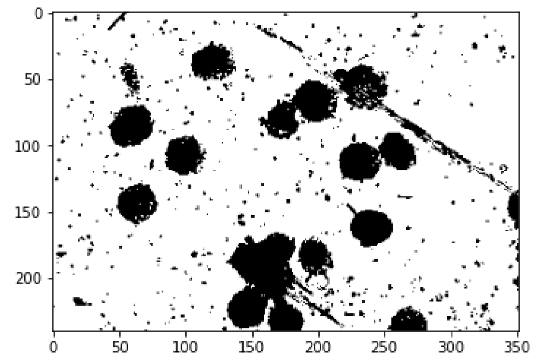
Post-processed



To remove the specks in the image and clean the blobs, I performed morphological opening, then closing.

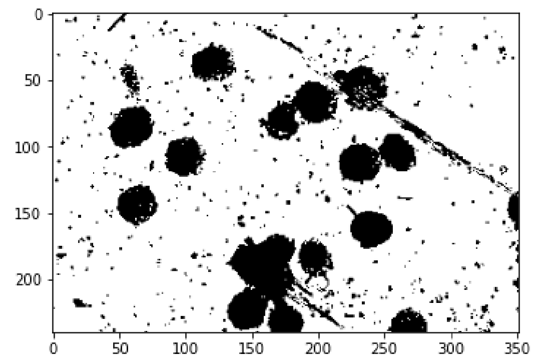
## APPLICATIONS

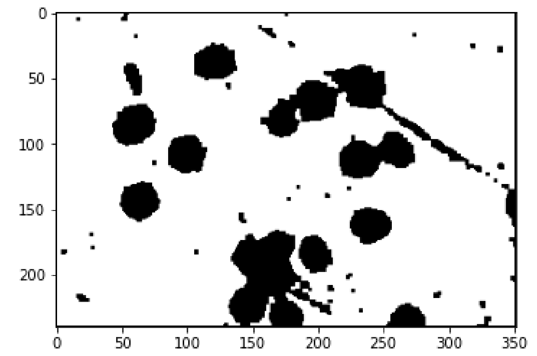




```
from skimage import filters
def gray_thresh(image, low, high):
    image[image<low] = 0
    image[image>high] = 255
    return image
thresholded= gray_thresh(malaria, 121,
164)
val=
filters.threshold_otsu(thresholded[np.isf
inite(thresholded)])
BW = thresholded > val
plt.imshow(BW, cmap='gray')
```

## APPLICATIONS

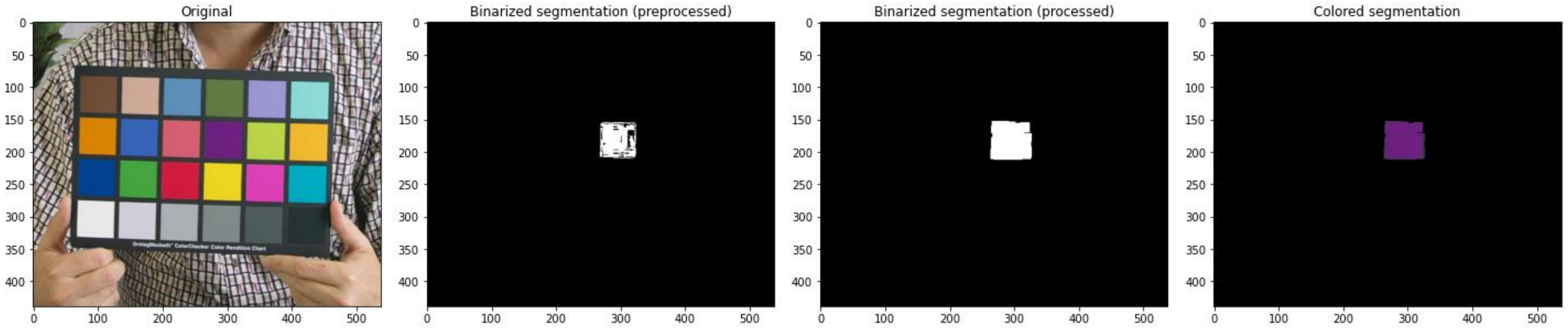




```
from skimage import filters
strel1 = np.ones((3,3))
BW2= ndimage.binary_opening(BW, strel1)
BW3 = ndimage.binary_closing(BW2, strel1)
plt.imshow(BW3, cmap='gray')
```

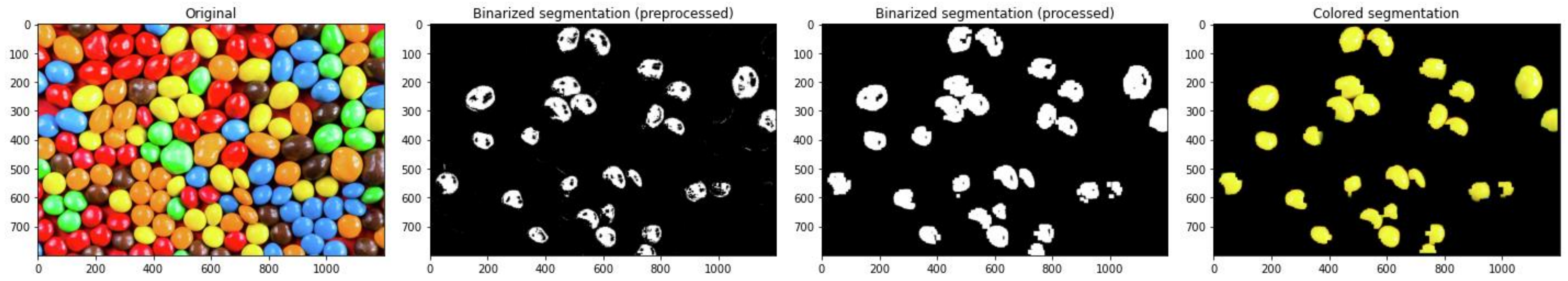
## APPLICATIONS

I applied a series of morphological operations in the segmented images I generated in the previous module, and here are some of the results.



## APPLICATIONS

I applied a series of morphological operations in the segmented images I generated in the previous module, and here are some of the results.


Original
Binarized segmentation (preprocessed)
Binarized segmentation (processed)
Colored segmentation

## SELF-REFLECTION

1. It took me quite a while to realize what reflections and translations mean on a grid array. For me, these operations are quite intuitive in Euclidean space, but not so much at first in grids. It took me an embarrassing amount of time to realize what the origin truly means for an array grid. It was all easy after it clicked, fortunately.
2. Performing 24 morphological operations manually on a graphing paper is exhausting and cumbersome, but I pulled through :D
3. I think I could've experimented with more images.

Self-score: 95/100

## REFERENCES

https://scipy-lectures.org/packages/scikit-image/auto_examples/plot_threshold.html (Otsu thresholding for image binarization)

Color wheel: https://pixabay.com/illustrations/colour-wheel-spectrum-rainbow-1740381/

M&Ms candies: https://www.sfgate.com/politics/article/wokeness-m-and-ms-defeated-17736299.php

Macbeth Color checker from AP157 Module, Automated Feature Extraction Part I

Image of malaria-infected cells are from the AP157 Module, Morphological Operations