UT Southwestern Medical Center
Lyda Hill Department of Bioinformatics

BioHPC

Image Processing With Python

[web] portal.biohpc.swmed.edu
[email] biohpc-help@utsouthwestern.edu

Updated for 2021-12-15

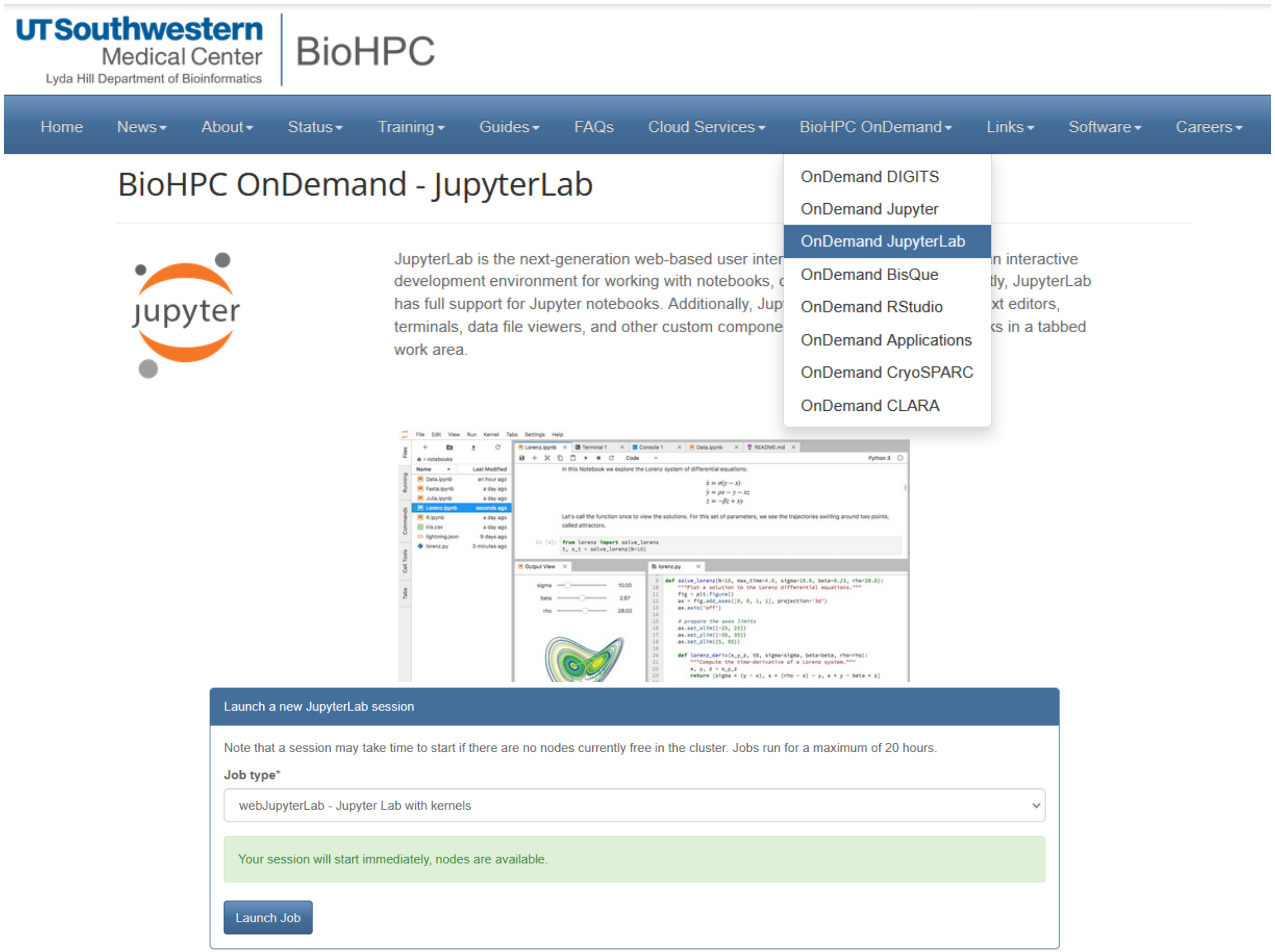
Notebooks for this session can be found on the training page.
https://portal.biohpc.swmed.edu/content/training/training-slides/

- **Ipython**
  - **.ipynb*
  - The notebook format itself – similar to a MATLAB .mlx Live Editor file.
- **Jupyter**
  - The server which provides the interface for you.
  - Runs using your installed Python kernel.
- **JupyterLab**
  - Like a nicer Jupyter – a little bit better for data exploration.

Python image processing resources available on the internet:

- https://www.numerical-tours.com/python/
- https://towardsdatascience.com/image-data-analysis-using-python-edddfdf128f4
- https://medium.com/analytics-vidhya/image-processing-with-python-applications-in-machine-learning-17d7aac6bc97

Getting a JupyterLab environment



Python Libraries + Modules used in this training

All should already be installed with JupyterLab OnDemand.

Docs:

- *os : https://docs.python.org/3/library/os.html*
- *matplotlib : https://matplotlib.org/*
- *scipy :*
  - General : *https://docs.scipy.org*
  - *ndimage : https://docs.scipy.org/doc/scipy/reference/ndimage.html*
- *skimage : https://scikit-image.org/*
- *sklearn : https://scikit-learn.org/stable/*
- *numpy :*
  - General *: https://numpy.org/doc/stable/index.html*
  - *ndarrays : https://numpy.org/doc/stable/reference/arrays.ndarray.html#id1*

Basic Image Processing Concepts

- Image processing can mean multiple things:
  - **Image conditioning**
    - Changing the quality of the data at a low level
    - Filtering, interpolation
    - Data -> Data
  - **Image analysis**
    - Deriving other, abstract features from your data
    - Histograms, mean/variance of a region
    - Data -> Information
  - **Image interpretation**
    - Understanding the content of an image
    - Data -> knowledge

Today's focus

Briefly touched on

Images as Arrays

Python and various modules/packages tend to think of images as arrays of different sorts.

- Lists of lists (Python itself)
- *ndarray (scipy)*
- *xarray (xarray)*
- *DataFrames (pandas)*

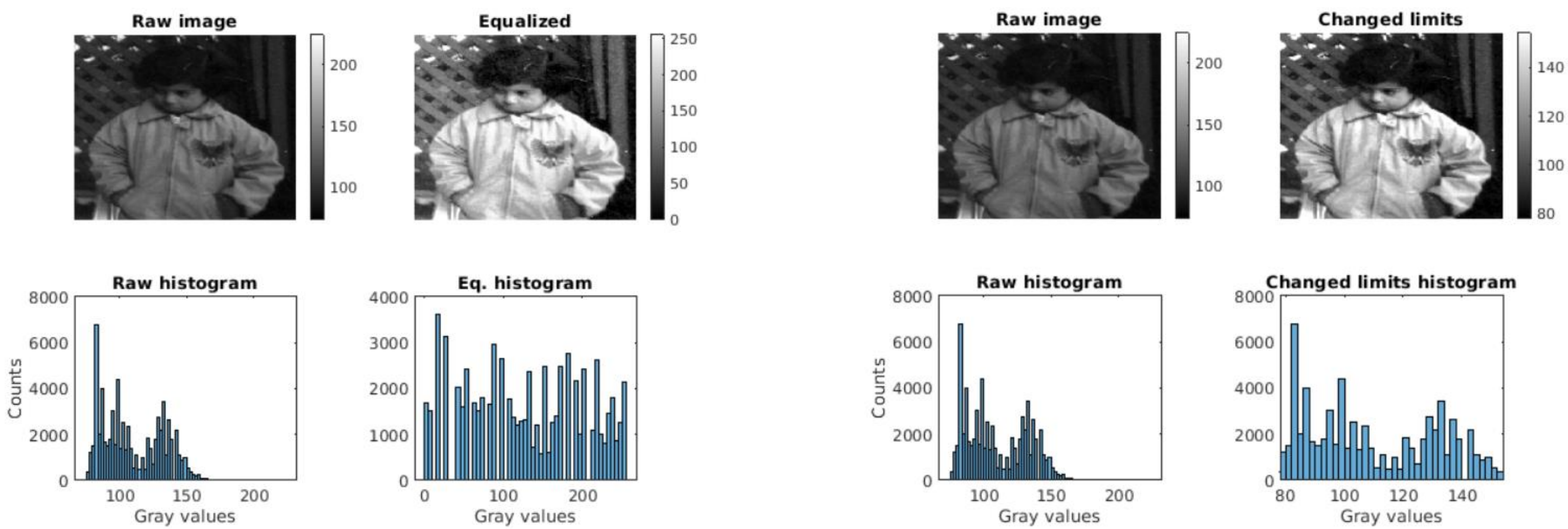
Common data types for image pixels:

- **bool** (binary/Boolean) – [0,1]
- **int** (signed integer) – any whole number.
- **float** (double-precision floating point) – Decimal numbers (e.g. 2.2251e-308, 0.4, 0.333333…)
- **uint8** (unsigned 8-bit) – [0,255]
- **uint16** (unsigned 16-bit) – [0,65535]

You'll frequently move between different data types.
You should know what the 'natural range' of your data type is.

Image Presentation – Changing the Data vs Changing the Display

You often have the choice to change your data or your display – when you can, change your display so you don't affect your data.



See Exercise 1 for more.

Python Array Indexing

- Python starts counting indexes from 0, and arranges coordinates like C does (row-major)
  - Compare/contrast to MATLAB, which is Fortran-like.
  - **This is a common source of errors – be mindful!**

- Can also count by ‘backward index’

letter_list = [“A”, ”B”, ”C”, ”D”, ”E”, ”F”, ”G”, ”H”, ”I”]

+ Index	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Entry	A	B	C	D	E	F	G	H	I
- Index	**-9**	**-8**	**-7**	**-6**	**-5**	**-4**	**-3**	**-2**	**-1**

- Slice indexes are defined by [START:STOP] or [START:STOP:STRIDE]
  - STOP index is ***NOT*** included
    - This is so that **letter_list_2 = letter_list[0:length(letter_list)]** is sensible**.**
  - Stride can be positive or negative, determining the counting direction.

Python Array Indexing

[start:stop:stride]

letter_list[-8:5]

letter_list[:5]

letter_list[0:5]

+ Index	0	1	2	3	4	5	6	7	8
Entry	A	B	C	D	E	F	G	H	I
- Index	-9	-8	-7	-6	-5	-4	-3	-2	-1

letter_list[1:-1]

letter_list[-1:1:-1]

letter_list[::-1]

letter_list[:]

letter_list[-5:-8:-1]

letter_list[-5:-8:1]

Somewhat like for-loops in other languages, but it's easier to define complex behavior.

Defining a range of indexes to loop:

```
image_names = []
for ind in range(len(filelist)):
    filename = filelist[ind]
    if filename[-4:]=='.tif':
        image_names.append(filename)
```

Looping an iterable:

```
image_names = []
for filename in filelist:
    if filename[-4:]=='.tif':
        image_names.append(filename)
```

List Comprehension:

```
image_names = [filename \
               for filename in filelist \
               if filename[-4:]=='.tif']
```

List Comprehension - A little more complex, but a lot more efficient.

```
new_list = [(<expression>) for (<item> in <iterable>) if (<condition> == True)]
```

<expression> and **<condition>** will usually involve **<item>**, though this is not required.

Generate new data:

```
# Getting the squares of numbers...
print( [(x**2) for x in range(0,5) ] )
```

```
# Only getting the squares of originally even numbers.
print( [(x**2) for x in range(0,5) if (x % 2 == 0)])
```

Filter existing data:

```
image_names = [filename for filename in filelist if filename[-4:]=='.tif']
```

Multi-dimensional arrays – Lists of Lists

Python counts in ‘row-major’ ordering, and orders dimensions like C does.
- Multidimensional arrays are ‘lists of lists’

```
my_arr=[['A','F','K','P'],['B','G','L','Q'],['C','H','M','R'],['D','I','N','S'],['E','J','O','T']]
```

BW		-4	-3	-2	-1
	FW	0	1	2	3
-5	0	A	F	K	P
-4	1	B	G	L	Q
-3	2	C	H	M	R
-2	3	D	I	N	S
-1	4	E	J	O	T

my_arr[2][1:3] → | H | M |

my_arr[-1][:] → | E | J | O | T |

multi-dimensional arrays – numpy arrays:

Numpy simplifies the representation a little:

```
import numpy as np
my_ndarr = np.array(my_arr)
```

BW		-4	-3	-2	-1
	FW	0	1	2	3
-5	0	A	F	K	P
-4	1	B	G	L	Q
-3	2	C	H	M	R
-2	3	D	I	N	S
-1	4	E	J	O	T

my_ndarr[2,1:3] →

H	M

my_ndarr[-1,:] →

E	J	O	T

Segmentation – Separating an image into parts

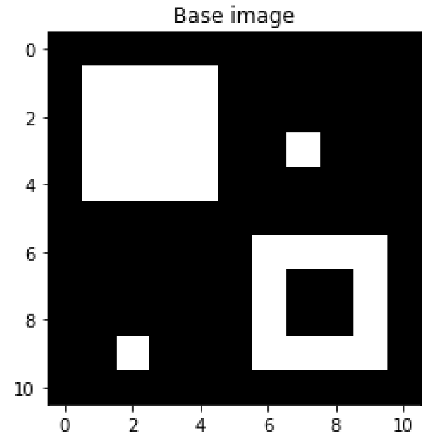
- Most basic: Foreground/background
  - Bright or dark background with a dark or bright foreground, respectively.
  - Choose a cutoff value, threshold.
  - Global thresholds can work, but can miss important elements

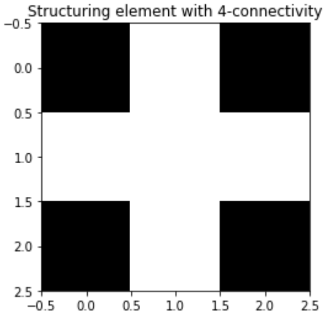
- More complex:
  - Adaptive thresholds
  - Texture clustering
  - Machine learning methods

Mathematical Morphology

- *Structuring element: A binary array of odd size (so that it has a 'center'), or with an explicitly defined 'center' element.*

- *Element is moved over all pixels in an image and a set-theoretic question is asked of the relationship between the structuring element and the image.*
  - *'Fit' : **All of the True elements of the strel** are on top of True elements in the image.*
  - *'Hit': **Any of the True elements of the strel** are on top of True elements in the image.*
  - *'Miss': **None of the True elements of the strel** are on top of True element in the image.*

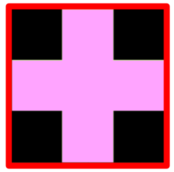




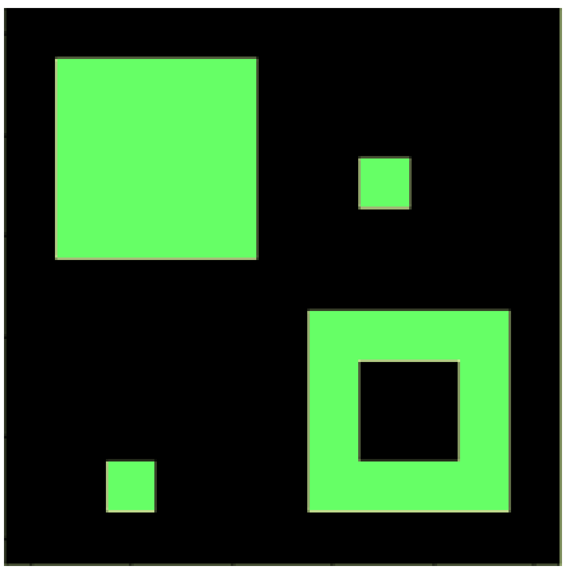
https://www.cs.auckland.ac.nz/courses/compsci773s1c/lectures/ImageProcessing-html/topic4.htm
https://homepages.inf.ed.ac.uk/rbf/HIPR2/morops.htm

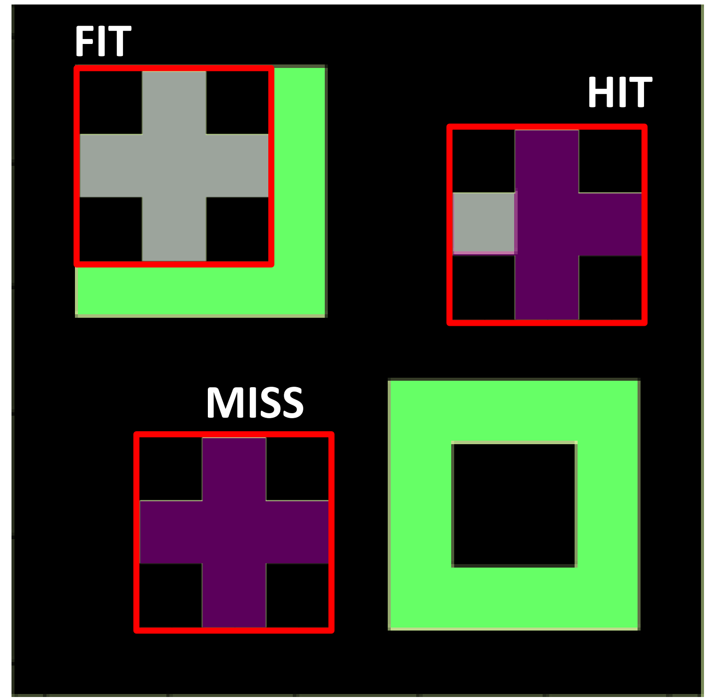
Mathematical Morphology – Fit, Hit, and Miss

Structuring Element



Base Image





Mathematical Morphology – Basic Binary

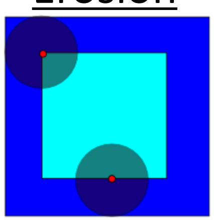
- Erosion: All points in the image where the structuring element ‘fits’, but **not** where it ‘hits’.
  - Tends to make structures smaller.

- Dilation: All points in the image where the structuring element hits (which includes where it fits)
  - Tends to make structures larger and smoother.

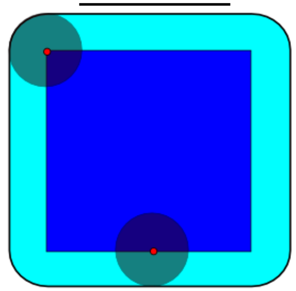
- Opening: Erosion of an image by a strel, followed by a dilation with that same strel.
  - Tends to ‘round off’ convex parts of images and remove fine detail.
  - Removes structures which are SMALLER than the strel and BRIGHTER than their surroundings.

- Closing: Dilation followed by erosion.
  - Tends to result in concave structures being smoothed out.
  - Connects slightly-separated bright structures, and removes dark structures smaller than the strel.

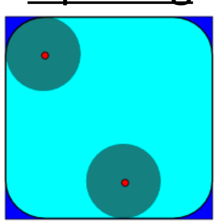
Erosion



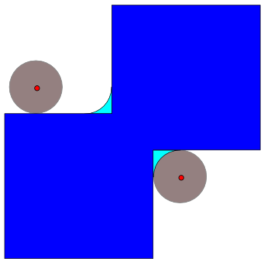
Dilation



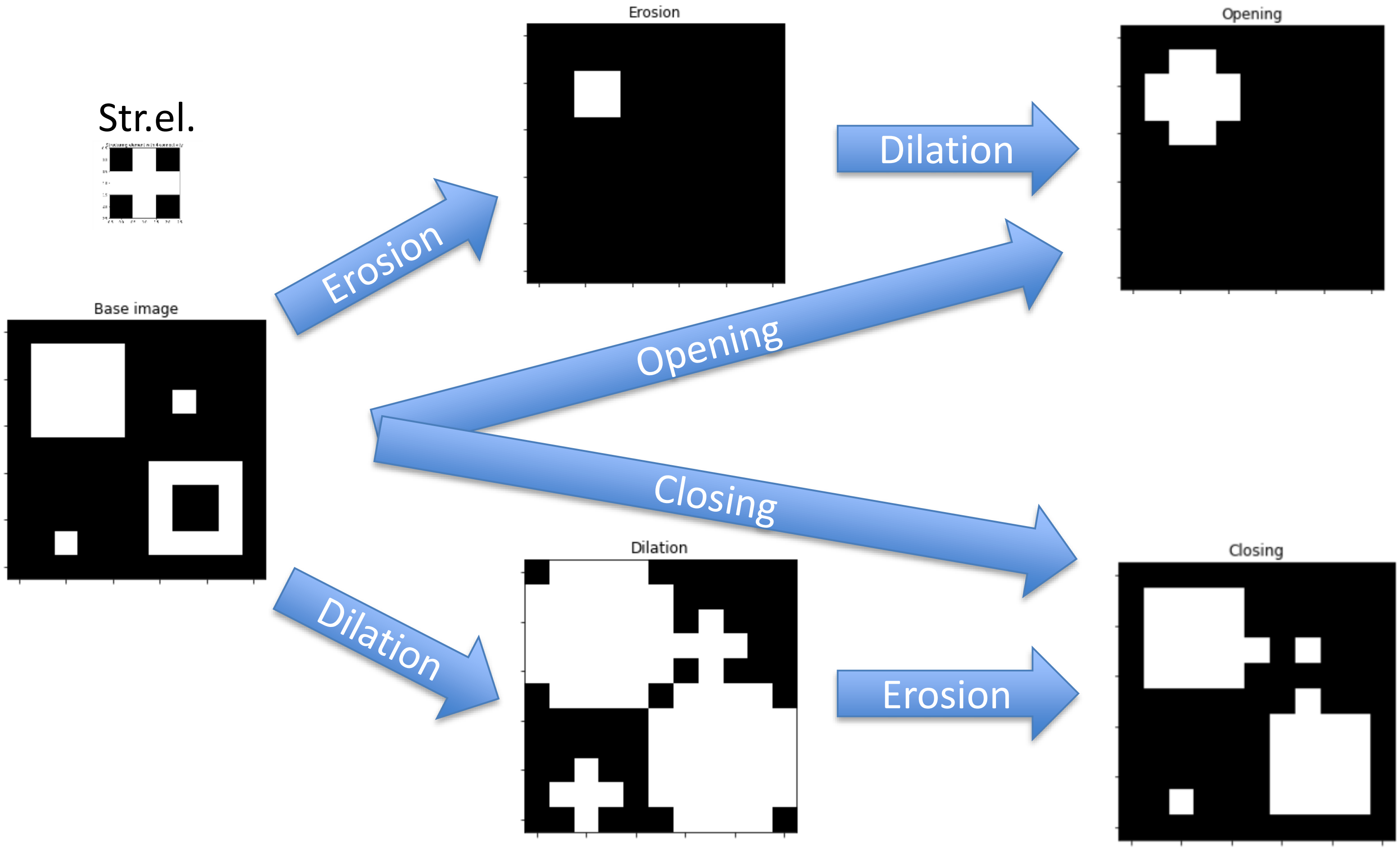
Opening



Closing



Map of the basic MM operations.



Mathematical Morphology - Grayscale

- Grayscale images can be treated similarly, but with a slightly modified interpretation of 'hit or miss'
    - Dilation will result in a pixel taking on the max value defined by the moving window of the strel.
    - Erosion will result in a pixel taking on the min value defined by the moving window of the strel.

Grayscale transforms:

- *Top-hat transform: Difference between an image and its opening.*
    - *Returns elements that are SMALLER than the strel and BRIGHTER than their surroundings*
    - *e.g. define a strel which is a disk of size slightly larger than speckle artifacts – punctate structures.*

- *Bot-hat ('black tophat'): Difference between the closure of an image and the original image.*
    - *Returns elements that are SMALLER than the strel and DARKER than their surroundings.*
    - *e.g. holes/quenched regions.*