COMP 2211 Exploring Artificial Intelligence

More Image Processing

Huiru Xiao

Materials Provided by Dr. Desmond Tsoi
Department of Computer Science & Engineering
The Hong Kong University of Science and Technology, Hong Kong SAR, China

Image Cropping

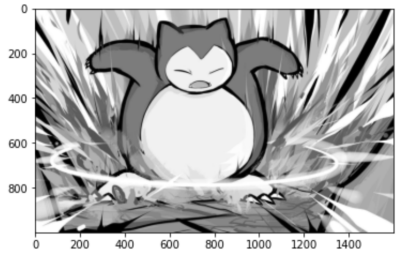
- Sometimes, you may want to crop the region of interest (ROI) for further processing.
- For instance, in a face detection application, you may want to drop the face from an image.
- To crop an image, you can use the same method as numpy array slicing.
- To slice an array, you need to specify the start and end index of the first as well as the second dimension.

Syntax

```
croppedImg = sourceImg[start_row:end_row, start_col:end_col]
```

```python
# Assume Google Drive has been mounted, \& the path
# has been added for interpreter to search

# Import all the required libraries
import cv2
import matplotlib.image as mpimg
import matplotlib.pyplot as plt

img = mpimg.imread('snorlax.png')  # Read the image

# Convert the color image to gray and show it
grayImg = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)

# Show the image
plt.figure()
plt.imshow(grayImg, cmap='gray', vmin=0, vmax=1)

# Obtain part of the image
croppedImg = grayImg[200:733, 300:1100]

# Show the image
plt.figure()
plt.imshow(croppedImg, cmap='gray', vmin=0, vmax=1)
```

```
<matplotlib.image.AxesImage at 0x7f1491def690>
```



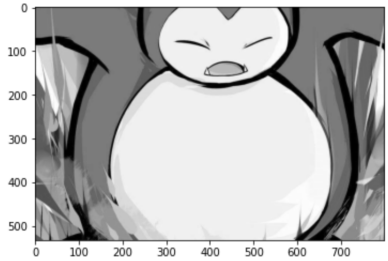


Image Padding

- Image padding introduces new pixels around the edges of an image.
- Types of padding
  - Constant padding
  - Reflection padding
  - Replication padding

0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	1	2	3	0	0
0	0	4	5	6	0	0
0	0	7	8	9	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0

BORDER_CONSTANT

5	4	4	5	6	6	5
2	1	1	2	3	3	2
2	1	1	2	3	3	2
5	4	4	5	6	6	5
8	7	7	8	9	9	8
8	7	7	8	9	9	8
5	4	4	5	6	6	5

BORDER_REFLECT

1	1	1	2	3	3	3
1	1	1	2	3	3	3
1	1	1	2	3	3	3
4	4	4	5	6	6	6
7	7	7	8	9	9	9
7	7	7	8	9	9	9
7	7	7	8	9	9	9

BORDER_REPLICATE

Image Padding

- To pad an image, you need to first import cv2
  `import cv2`
- Then use copyMakeBorder() method of the cv2 module.

Syntax

```
cv2.copyMakeBorder(src, top, bottom, left, right, borderType, value)
```

Parameters:

- src: Source image
- top: The border width in number of pixels in top direction
- bottom: The border width in the number of pixels in bottom direction
- left: The border width in the number of pixels in left direction
- right: The border width in the number of pixels in the right direction
- borderType: The kind of border to be added
  - cv2.BORDER_CONST
  - cv2.BORDER_REFLECT
  - cv2.BORDER_REPLICATE
- value (optional): The color of border if border type is cv2.BORDER_CONSTANT
- Returns the resulting image

```
# Assume Google Drive has been mounted
# \& the path has been added for
# interpreter to search

# Import all the required libraries
import cv2
import matplotlib.image as mpimg
import matplotlib.pyplot as plt

# Read the image
img = mpimg.imread('snorlax.png')

# Create subplots
fig = plt.figure(figsize=(12,9))
fig.tight_layout();
fig.subplots_adjust(wspace=0.2, hspace=0.2)

# Add the image to the first row, first col
ax1= fig.add_subplot(2, 2, 1)
ax1.title.set_text('Original')
grayImg = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
plt.imshow(grayImg, cmap='gray', vmin=0, vmax=1)
```

```
padImgConst = cv2.copyMakeBorder(grayImg,
50, 50, 50, 50,
cv2.BORDER_CONSTANT, 128)
# Add the image to the first row, second col
ax2 = fig.add_subplot(2, 2, 2)
ax2.title.set_text('BORDER_CONSTANT')
plt.imshow(padImgConst, cmap='gray',
vmin=0, vmax=1)

padImgRef = cv2.copyMakeBorder(grayImg,
300, 300, 300, 300,
cv2.BORDER_REFLECT)
# Add the image to the second row, first col
ax3 = fig.add_subplot(2, 2, 3)
ax3.title.set_text('BORDER_REFLECT')
plt.imshow(padImgRef, cmap='gray',
vmin=0, vmax=1)

padImgRep = cv2.copyMakeBorder(grayImg,
300, 300, 300, 300,
cv2.BORDER_REPLICATE)
# Add the image to the second row, second col
ax4 = fig.add_subplot(2, 2, 4)
ax4.title.set_text('BORDER_REPLICATE')
plt.imshow(padImgRep, cmap='gray',
vmin=0, vmax=1)
```

Example

```
<matplotlib.image.AxesImage at 0x7fc14535b410>
```

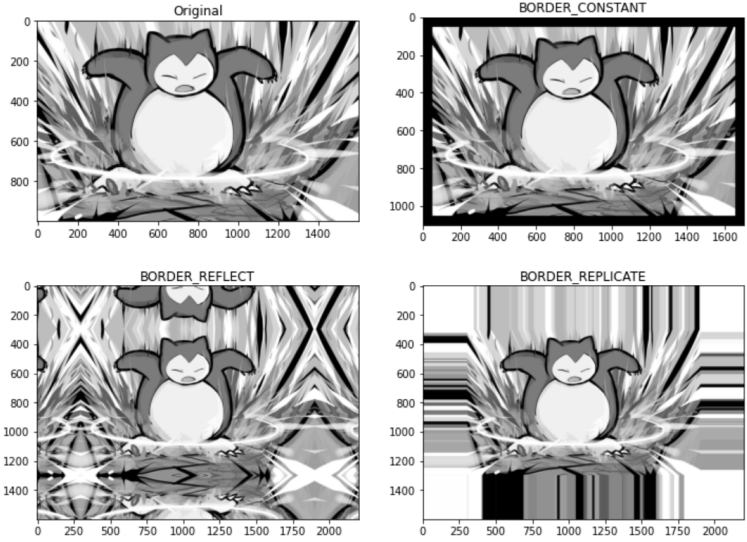


Image Histogram

- An image histogram is a graphical representation of the number of pixels in an image as a function of their intensity.
- To find the histogram of an image, you need to first import cv2

```
import cv2
```

- Then use calcHist() method of the cv2 module.

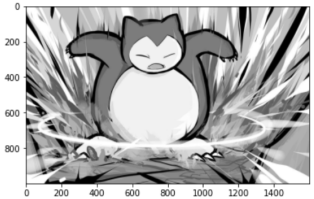
Syntax

```
cv2.calcHist(images, channels, mask, histSize, ranges[, hist[, accumulate]])
```

Parameters:

- image: Image of type uint8 or float32 represented as "[img]"
- channels: It is the index of channel for which we calculate histogram. For grayscale image, its value is [0] and color image, you can pass [0], [1], or [2] to calculate histogram of each channel respectively.
- mask: mask image. To find histogram of full image, it is given as 'None".
- histSize: This represents the number of bins. For full scale, we pass [256]
- ranges: This is the range of intensities. Normally, it is [0,256].
- Return value: Histogram of the image.

```python
# Assume Google Drive has been mounted, the path has been added for interpreter to search

# Import all the required libraries
import cv2
import matplotlib.image as mpimg
import matplotlib.pyplot as plt

img = mpimg.imread('snorlax.png')  # Read the image

# Convert the color image to gray and show it
grayImg = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
plt.figure()
plt.imshow(grayImg, cmap='gray', vmin=0, vmax=1)

# Convert pixel values from [0,1] to [0,255]
grayImgUint = grayImg*255
grayImgUint = grayImgUint.astype(np.uint8)

# Calculate histogram
hist = cv2.calcHist([grayImgUint], [0], None, [256], [0,255])
plt.figure()
plt.plot(hist)  # Plot and show the histogram
plt.show()
```



Brightness Adjustment

- To adjust the brightness of an image using OpenCV, you need to first import cv2

```
import cv2
```

- Then use convertScaleAbs() method of the cv2 module.

Syntax

```
cv2.convertScaleAbs(image, alpha = 1, beta = 0)
```

convertScaleAbs does the following:

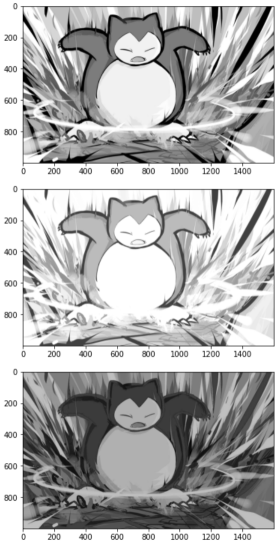
```
Inew(x,y) = min(alpha * I(x,y) + beta, 255)
Inew(x,y) = max(Inew(x,y), 0)
```

Parameters:

- image: Image to be processed in n-dimensional array
- alpha: The scale factor. It is 1 by default
- beta: The delta added to the scaled values. It is 0 by default.
- Return value: Converted image.

```
# Assume Google Drive has been mounted, the path has been added for interpreter

# Import all the required libraries
import cv2
import matplotlib.image as mpimg
import matplotlib.pyplot as plt

img = mpimg.imread('snorlax.png')  # Read the image

# Convert the color image to gray and show it
grayImg = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
plt.figure(); plt.imshow(grayImg, cmap='gray', vmin=0, vmax=1)

# Convert pixel values from [0,1] to [0,255]
grayImgUint = grayImg*255
grayImgUint = grayImgUint.astype(np.uint8)

# Produce an image that is brighter than the original
brighterImg = cv2.convertScaleAbs(grayImgUint, alpha = 1.0, beta = 64)
plt.figure(); plt.imshow(brighterImg, cmap='gray', vmin=0, vmax=255)

# Produce an image that is dimmer than the original
dimmerImg = cv2.convertScaleAbs(grayImgUint, alpha = 1.0, beta = -64)
plt.figure(); plt.imshow(dimmerImg, cmap='gray', vmin=0, vmax=255)
```

```
<matplotlib.image.AxesImage at 0x7fc145737b90>
```



Gamma Correction

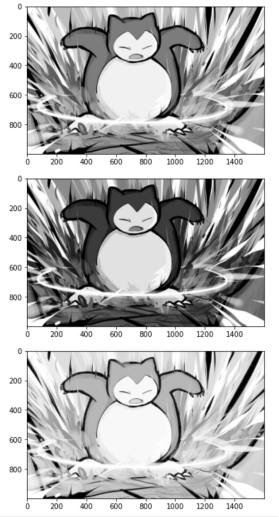
- Gamma defines the relationship between a pixel's numerical value and its actual luminance.

- Without gamma, shades captured by digital cameras would not appear as they did to our eyes (on a standard monitor).
- Gamma is also referred to as gamma correction, gamma encoding or gamma compression, but these all refer to a similar concept.
- A gamma encoded image has to have "gamma correction" applied when it is viewed – which effectively converts it back into light from the original scene.

Gamma correction can be performed by adjusting gamma value (γ).

- $\gamma < 1$ will make the image appear darker
- $\gamma > 1$ will make the image appear lighter
- $\gamma = 1$ will have no effect on the input image

```
# Assume Google Drive has been mounted, the path has been added for interpreter
<matplotlib.image.AxesImage at 0x7f13523ee590>

# Import all the required libraries
import cv2; import numpy as np
import matplotlib.image as mpimg; import matplotlib.pyplot as plt

img = mpimg.imread('snorlax.png') # Read the image

# Convert the color image to gray and show it
grayImg = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
plt.figure(); plt.imshow(grayImg, cmap='gray', vmin=0, vmax=1)
# Convert pixel values from [0,1] to [0,255]
grayImgUint = grayImg*255; grayImgUint = grayImgUint.astype(np.uint8)

# Prepare look-up-table and perform gamma correction
gamma = 0.5; invGamma = 1/gamma
table = [((i / 255) ** invGamma) * 255 for i in range(256)]
table = np.array(table, np.uint8)
processedImg1 = cv2.LUT(grayImgUint, table)
plt.figure(); plt.imshow(processedImg1, cmap='gray', vmin=0, vmax=255)

# Prepare look-up-table and perform gamma correction
gamma = 2.2; invGamma = 1/gamma
table = [((i / 255) ** invGamma) * 255 for i in range(256)]
table = np.array(table, np.uint8)
processedImg2 = cv2.LUT(grayImgUint, table)
plt.figure(); plt.imshow(processedImg2, cmap='gray', vmin=0, vmax=255)
```



Histogram Equalization

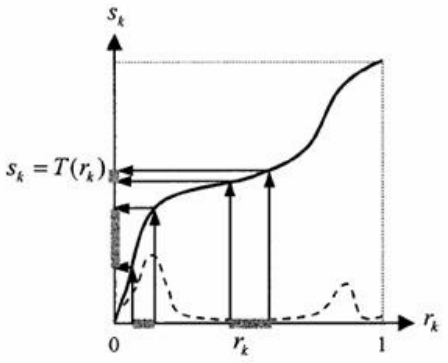
- Histogram equalization is another technique used to improve contrast of images.
- The idea is to spread out the most frequent intensity values.
- Algorithm

1. Compute the histogram, H, of the image
2. Compute the cumulative histogram, C, of the image

$$C(i) = \sum_{j=0}^{i} H(j)$$

3. Map old intensity value to new intensity value as follows:

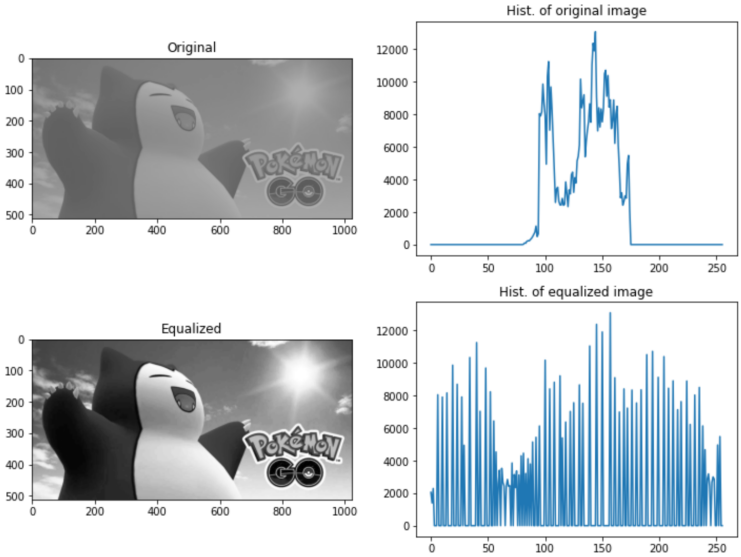
$$I_{new} = C(I)$$



```
# Assume Google Drive has been mounted, the path has been added for interpreter to search

# Import all the required libraries
import cv2
import matplotlib.image as mpimg
import matplotlib.pyplot as plt

# Read the image
img = mpimg.imread('snorlax-low-contrast.png')

# Prepare subplots
fig = plt.figure(figsize=(12,9))
fig.tight_layout();
fig.subplots_adjust(wspace=0.2, hspace=0.2)

# Add the image to the first row, first col
gImg = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
ax1 = fig.add_subplot(2, 2, 1)
ax1.title.set_text("Original")
plt.imshow(gImg, cmap='gray', vmin=0, vmax=1)

# Convert pixel values from [0,1] to [0,255]
gImgUint = gImg*255
gImgUint = gImgUint.astype(np.uint8)
# Produce histogram
hist1 = cv2.calcHist([gImgUint],
[0], None, [256], [0,255])
```

```
# Add the image to the first row, second col
ax2 = fig.add_subplot(2, 2, 2)
ax2.title.set_text('Hist. of original image')
plt.plot(hist1);

# Produce cumulative histogram
cumHist = np.cumsum(hist1)
cumMax = np.max(cumHist)
table = np.array((cumHist/np.max(cumHist))*255,
np.uint8)
equalizedImg = cv2.LUT(gImgUint, table)
# Add the image to the second row, first col
ax3 = fig.add_subplot(2, 2, 3)
ax3.title.set_text("Equalized")
plt.imshow(equalizedImg, cmap='gray',
vmin=0, vmax=255)

# Produce histogram
hist2 = cv2.calcHist([equalizedImg],
[0], None, [256], [0,255])
# Add the image to the second row, second col
ax4 = fig.add_subplot(2, 2, 4)
ax4.title.set_text('Hist. of equalized image')
plt.plot(hist2);
```

Example


Original
Hist. of original image
Equalized
Hist. of equalized image

Histogram Equalization

- In fact, histogram equalization can be performed using OpenCV function. To do so, you need to first import cv2

```
import cv2
```

- Then use equalizeHist() method of the cv2 module.

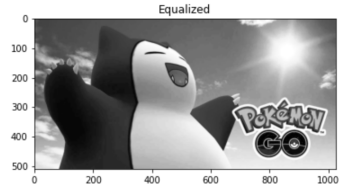
Syntax

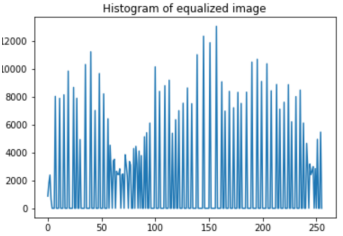
```
equalizeHist(source)
```

Parameters:

- source: input image array
- Return value: Equalized image

```
# Assume Google Drive has been mounted, the path has been added for interpreter to search

# Import all the required libraries
import cv2; import numpy as np
import matplotlib.image as mpimg; import matplotlib.pyplot as plt

img = mpimg.imread('snorlax-low-contrast.png')  # Read the image

# Prepare subplots
fig = plt.figure(figsize=(12,9))
fig.set_figwidth(6); fig.tight_layout()
fig.subplots_adjust(wspace=0.2, hspace=0.2)

# Convert pixel values from [0,1] to [0,255]
grayImg = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
grayImgUint = grayImg*255
grayImgUint = grayImgUint.astype(np.uint8)

# Perform histogram equalization
equalizedImg = cv2.equalizeHist(grayImgUint)
ax1 = fig.add_subplot(2, 1, 1); ax1.title.set_text('Equalized')
plt.imshow(equalizedImg, cmap='gray', vmin=0, vmax=255)

# Produce histogram
hist = cv2.calcHist([equalizedImg], [0], None, [256], [0,255])
ax2 = fig.add_subplot(2, 1, 2)
ax2.title.set_text('Histogram of equalized image'); plt.plot(hist)
```





Interesting Kernels

Original image

0	0	0
0	1	0
0	0	0

Identity kernel

Resulting image

Convolve the original image with the kernel 5 times, we still get back the identical image.

Original image

0	0	0
1	0	0
0	0	0

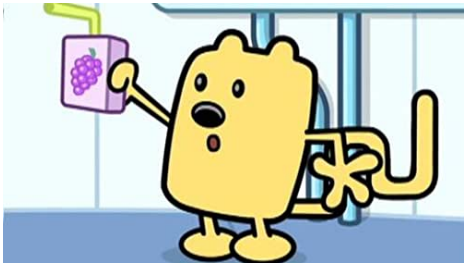
Shifted identity kernel

Resulting image

Convolve the original image with the kernel 5 times, we get back an image shifted by 5 pixels.

Non-linear Filtering

- Non-linear filters are typically more powerful than linear filters
  - Suppression of spikes
  - Edge preserving properties
- Examples:
  - Median filter
  - Morphological filters



Median Filter

- The median filter is often used to remove noise from an image.
- It preserves edges while removing noise. Also, no new gray value is introduced.
- Idea:
  - Consider each pixel in the image in turn and looks at its nearby neighbors to decide whether or not it is representative of its surroundings.
  - It replaces the pixel with the median of those values.
  - The median is calculated by first sorting all the pixel values from the surrounding neighborhood into numerical order and then replacing the pixel being considered with the middle pixel value.
- Example:

123	125	126	130	140
122	124	126	127	135
118	120	150	125	134
119	115	119	123	133
111	116	110	120	130

- Neighborhood values: 124, 126, 127, 120, 150, 125, 115, 119, 123
- Sort the values: 115, 119, 120, 123, 124, 125, 126, 127, 150
- Pick the median value: 124

Median Filter

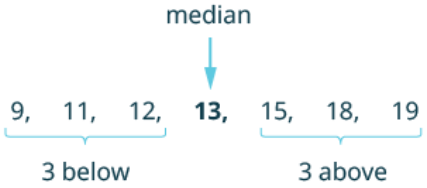
- To perform median filtering, you need to first import cv2
  `import cv2`
- Then use medianBlur() method of the cv2 module.

Syntax

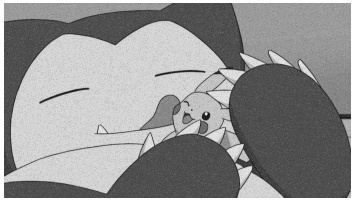
```
cv2.medianBlur(src, kernelSize)
```

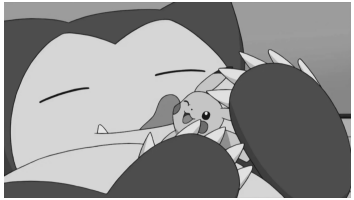
Parameters:

- src: input image that you want to process
- kernelSize: The size of the kernel
- Return value: filtered image.



```
# Assume Google Drive has been mounted, the path has been added for interpreter to search

# Import all the required libraries
import cv2; import numpy as np
import matplotlib.image as mpimg
import matplotlib.pyplot as plt

# Read the image
grayImg = mpimg.imread('snorlax-noisy-result.png')
# Convert the pixel values from [0,1] to [0,255]
grayImgUint = grayImg*255
grayImgUint = grayImgUint.astype(np.uint8)
plt.figure(); plt.imshow(grayImgUint, cmap='gray',
vmin=0, vmax=255)

# Perform median filtering
resultImg = cv2.medianBlur(grayImgUint, 5)

# Show the resulting image
plt.figure();
plt.imshow(resultImg, cmap="gray", vmin=0, vmax=255)
```

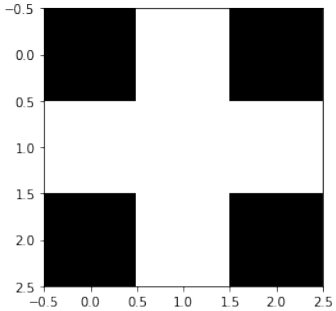




Morphological Filter

- Morphological filters are a set of image processing operations where the shapes of the image's object are manipulated.
- Similar to convolutional kernels, morphological operations utilize a structuring element to transform each pixel of an image to a value based on its neighbors' value.
- An example of structuring element:

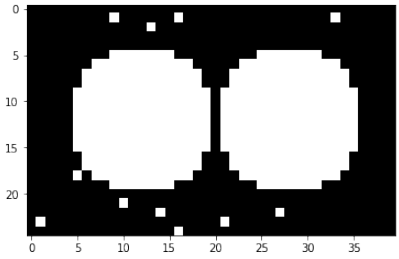
```
import numpy as np

structuring_element = np.array([ [0,1,0],
                                 [1,1,1],
                                 [0,1,0] ], np.uint8)
plt.imshow(structuring_element, cmap='gray');
```



Morphological Filter

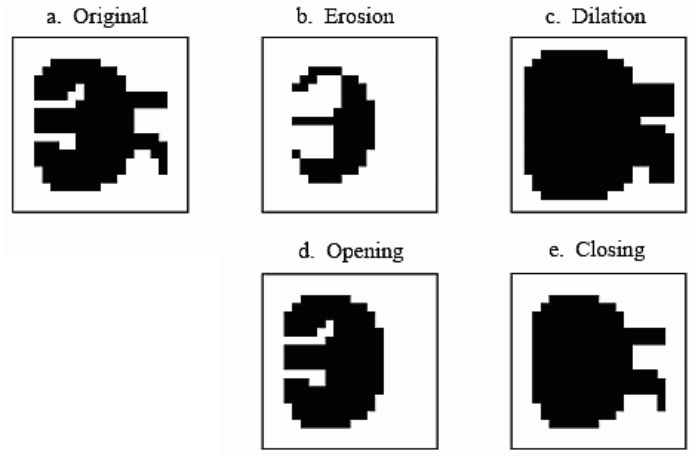
- To demonstrate how morphological filter work, let us create two adjacent circles with random noise on its background.

```
from skimage.draw import disk
import numpy as np

circle_image = np.zeros((25, 40))
circle_image[disk((12, 12), 8)] = 1
circle_image[disk((12, 28), 8)] = 1
for x in range(20):
   circle_image[np.random.randint(25),
                np.random.randint(40)] = 1
plt.imshow(circle_image, cmap='gray');
```



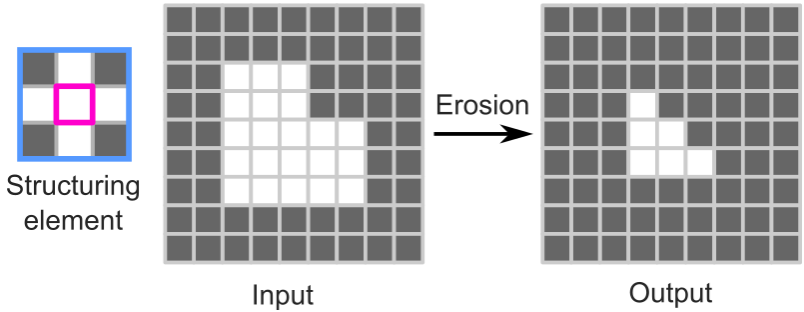
Examples

- Erosion
- Dilation
- Opening
- Closing



Erosion Filter

- Erosion is used for shrinking of element in input image by using the structuring element.
- The pixel values are retained only when the structuring element is completely contained inside input image. Otherwise, it gets deleted or eroded.



Erosion Filter

- To perform erosion, you need to first import cv2

```
import cv2
```

- Then use erode() method of the cv2 module.

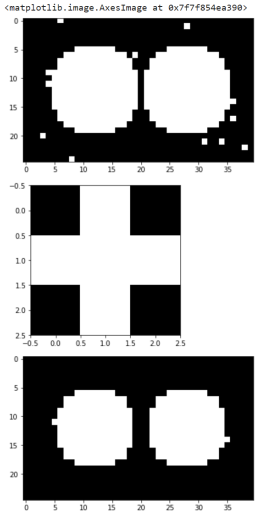
Syntax

```
cv2.erode(src, kernel, dst, anchor, iterations, borderType, borderValue)
```

Parameters:

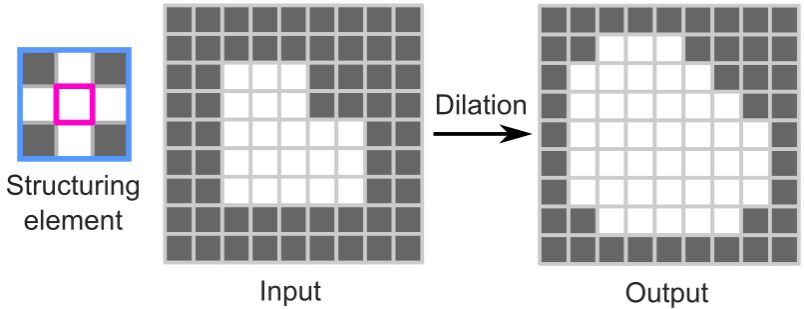
- src: input image that you want to erode
- kernel: A structuring element used for erosion
- dst: Output image
- anchor: Integer representing anchor point and it's default value Point is (-1,-1) which means that the anchor is at the kernel center.
- borderType: cv2.BORDER_CONSTANT, cv2.BORDER_REFLECT, etc.
- iterations: Number of times erosion is applied.
- borderValue: It is border value in case of a constant border.
- Return value: filtered image.

```python
import cv2
import matplotlib.pyplot as plt

# Show circle\_image
plt.figure(); plt.imshow(circle_image, cmap='gray')
# Show structuring element
plt.figure(); plt.imshow(structuring_element, cmap='gray');
# Perform erosion filter
eroded_img = cv2.erode(circle_image, structuring_element)
# Show the resulting image
plt.figure(); plt.imshow(eroded_img, cmap='gray')
```



Dilation Filter

- Dilation is used for expanding of element in input image by using the structuring element.
- The pixel values are "on" only when the structuring element has overlapped with the input image. Otherwise, the pixel values are "off".



Dilation Filter

- To perform dilation, you need to first import cv2

```
import cv2
```

- Then use dilate() method of the cv2 module.

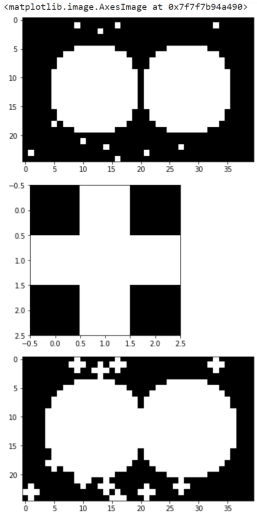
Syntax

```
cv2.dilate(src, kernel, dst, anchor, iterations, borderType, borderValue)
```

Parameters:

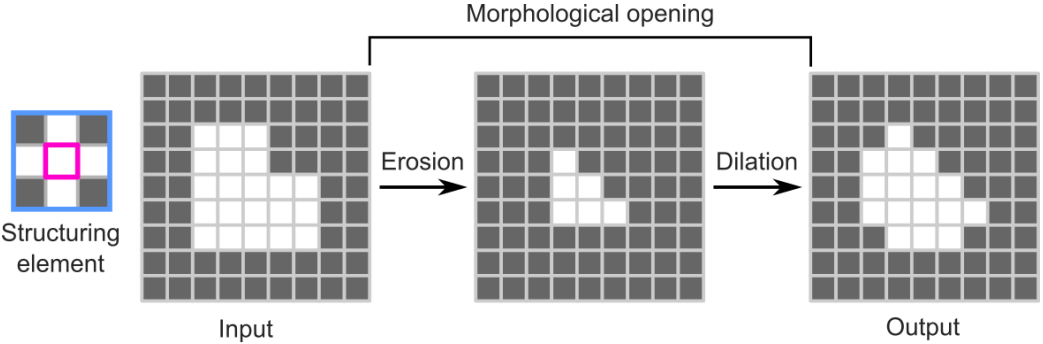
- src: input image that you want to dilate
- kernel: A structuring element used for dilation
- dst: Output image
- anchor: Integer representing anchor point and it's default value Point is (-1,-1) which means that the anchor is at the kernel center.
- borderType: cv2.BORDER_CONSTANT, cv2.BORDER_REFLECT, etc.
- iterations: Number of times dilation is applied.
- borderValue: It is border value in case of a constant border.
- Return value: filtered image.

```python
import cv2
import matplotlib.pyplot as plt

# Show circle\_image
plt.figure(); plt.imshow(circle_image, cmap='gray')
# Show structuring element
plt.figure(); plt.imshow(structuring_element, cmap='gray');
# Perform erosion filter
dilated_img = cv2.dilate(circle_image, structuring_element)
# Show the resulting image
plt.figure(); plt.imshow(dilated_img, cmap='gray')
```



Opening Filter

- Opening filter removed small objects while also maintaining the original shape of the object.
- Opening is done by applying the erosion first, and then applying dilation.



Opening Filter

- To perform opening, you need to first import cv2

```
import cv2
```

- Then use morphologyEx() method of the cv2 module.

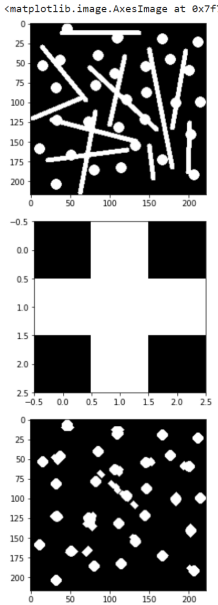
Syntax

```
cv2.morphologyEx(src, op, kernel, dst=None, anchor=None,
                 iterations=None, borderType=None, borderValue=None)
```

Parameters:

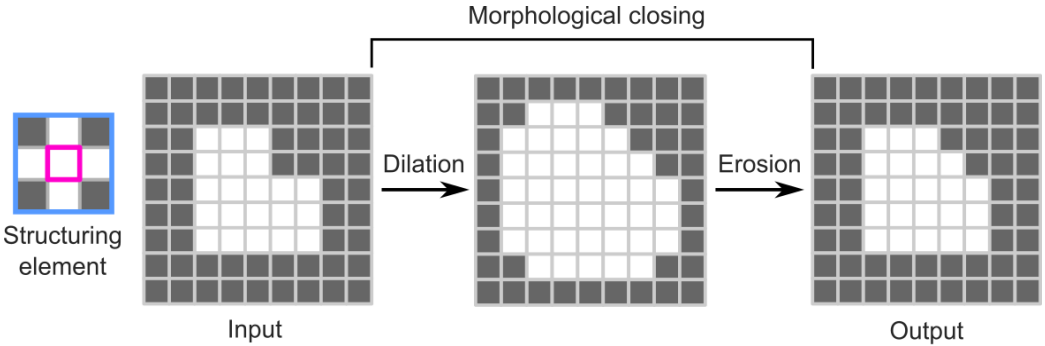
- src: input image that you want to process
- op: Operations (cv2.MORPH_ERODE, cv2.MORPH_DILATE, cv2.MORPH_OPEN, cv2.MORPH_CLOSE)
- kernel: A structuring element used for dilation
- dst: Output image
- anchor: Integer representing anchor point and it's default value Point is (-1,-1) which means that the anchor is at the kernel center.
- iterations: Number of times dilation is applied (e.g. iterations $= 2$, erode$\times 2$, dilate$\times 2$).
- borderType: cv2.BORDER_CONSTANT, cv2.BORDER_REFLECT, etc.
- borderValue: It is border value in case of a constant border.
- Return value: filtered image.

```python
import cv2
import matplotlib.pyplot as plt

img = plt.imread('input-open.png')

# Show input image
plt.figure(); plt.imshow(img, cmap='gray')
# Show structuring element
plt.figure();
plt.imshow(structuring_element, cmap='gray');

# Perform opening filter
opened_img = cv2.morphologyEx(img, cv2.MORPH_OPEN,
                              structuring_element,
                              iterations=4)
# Show the resulting image
plt.figure();
plt.imshow(opened_img, cmap='gray')
```



Closing Filter

- Closing filter removes small holes while also maintaining the original shape of the object.
- Closing is done by applying the dilation first, and then applying erosion.



Closing Filter

- To perform closing, you need to first import cv2

```
import cv2
```

- Then use morphologyEx() method of the cv2 module.

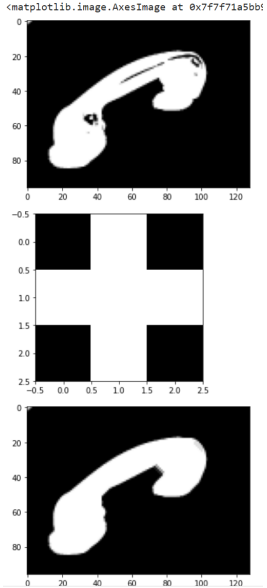
Syntax

```
cv2.morphologyEx(src, op, kernel, dst=None, anchor=None,
                 iterations=None, borderType=None, borderValue=None)
```

Parameters:

- src: input image that you want to process
- op: Operations (cv2.MORPH_ERODE, cv2.MORPH_DILATE, cv2.MORPH_OPEN, cv2.MORPH_CLOSE)
- kernel: A structuring element used for dilation
- dst: Output image
- anchor: Integer representing anchor point and it's default value Point is (-1,-1) which means that the anchor is at the kernel center.
- iterations: Number of times dilation is applied (e.g. iterations $= 2$, dilate$\times 2$, erode$\times 2$).
- borderType: cv2.BORDER_CONSTANT, cv2.BORDER_REFLECT, etc.
- borderValue: It is border value in case of a constant border.
- Return value: filtered image.

```python
import cv2
import matplotlib.pyplot as plt

img = plt.imread('input-close.png')

# Show input image
plt.figure(); plt.imshow(img, cmap='gray')

# Show structuring element
plt.figure();
plt.imshow(structuring_element, cmap='gray',
           vmin=0, vmax=1);

# Perform closing filter
closed_img = cv2.morphologyEx(img, cv2.MORPH_CLOSE,
                              structuring_element,
                              iterations=4)
# Show the resulting image
plt.figure();
plt.imshow(closed_img, cmap='gray')
```



Useful Links

- imread(): https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.imread.html
- imshow(): https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.imshow.html
- imsave(): https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.imsave.html
- cvtColor(): https://docs.opencv.org/3.4/df/d9d/tutorial_py_colorspaces.html
- warpAffine(), getRotationMatrix2D(), resize():
  https://docs.opencv.org/3.4/da/d6e/tutorial_py_geometric_transformations.html
- copyMakeBorder(): https://docs.opencv.org/3.4/dc/da3/tutorial_copyMakeBorder.html
- calcHist(): https://docs.opencv.org/3.4/dd/d0d/tutorial_py_2d_histogram.html
- convertScaleAbs():
  https://docs.opencv.org/3.4/d2/de8/group__core__array.html#ga3460e9c9f37b563ab9dd550c4d8c4e7d
- threshold(): https://docs.opencv.org/3.4/d7/d4d/tutorial_py_thresholding.html
- equalizeHist(): https://docs.opencv.org/3.4/d5/daf/tutorial_py_histogram_equalization.html
- filter2D(), medianBlur(): https://docs.opencv.org/3.4/d4/d13/tutorial_py_filtering.html
- erode(), dilate(), morphologyEx(),
  https://docs.opencv.org/3.4/d9/d61/tutorial_py_morphological_ops.html

That's all!

Any questions?