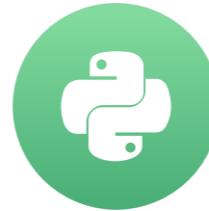


# Jump into filtering

IMAGE PROCESSING IN PYTHON

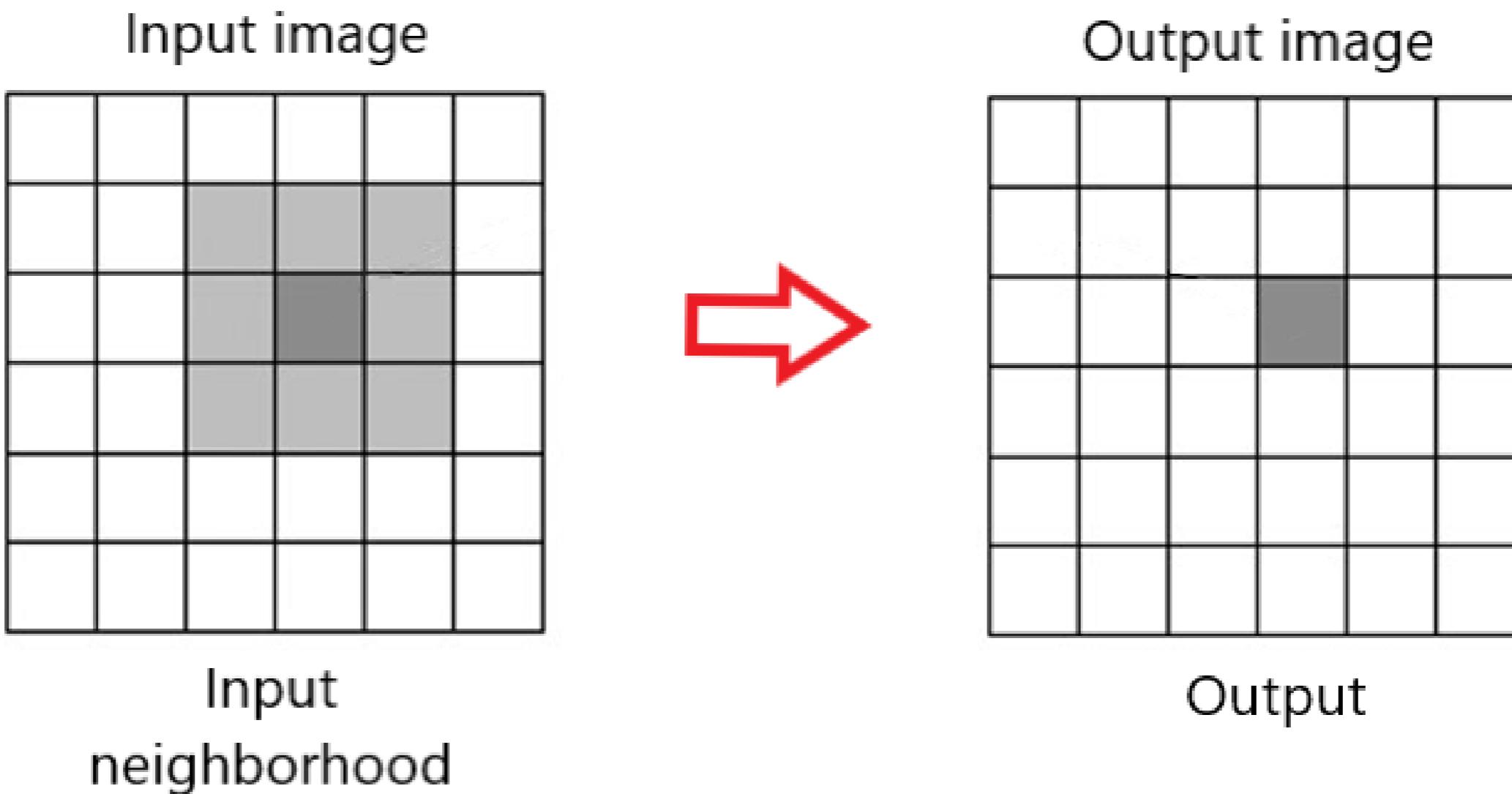


**Rebeca Gonzalez**  
Data Engineer

# Filters

- Enhancing an image
- Emphasize or remove features
- Smoothing
- Sharpening
- Edge detection

# Neighborhoods

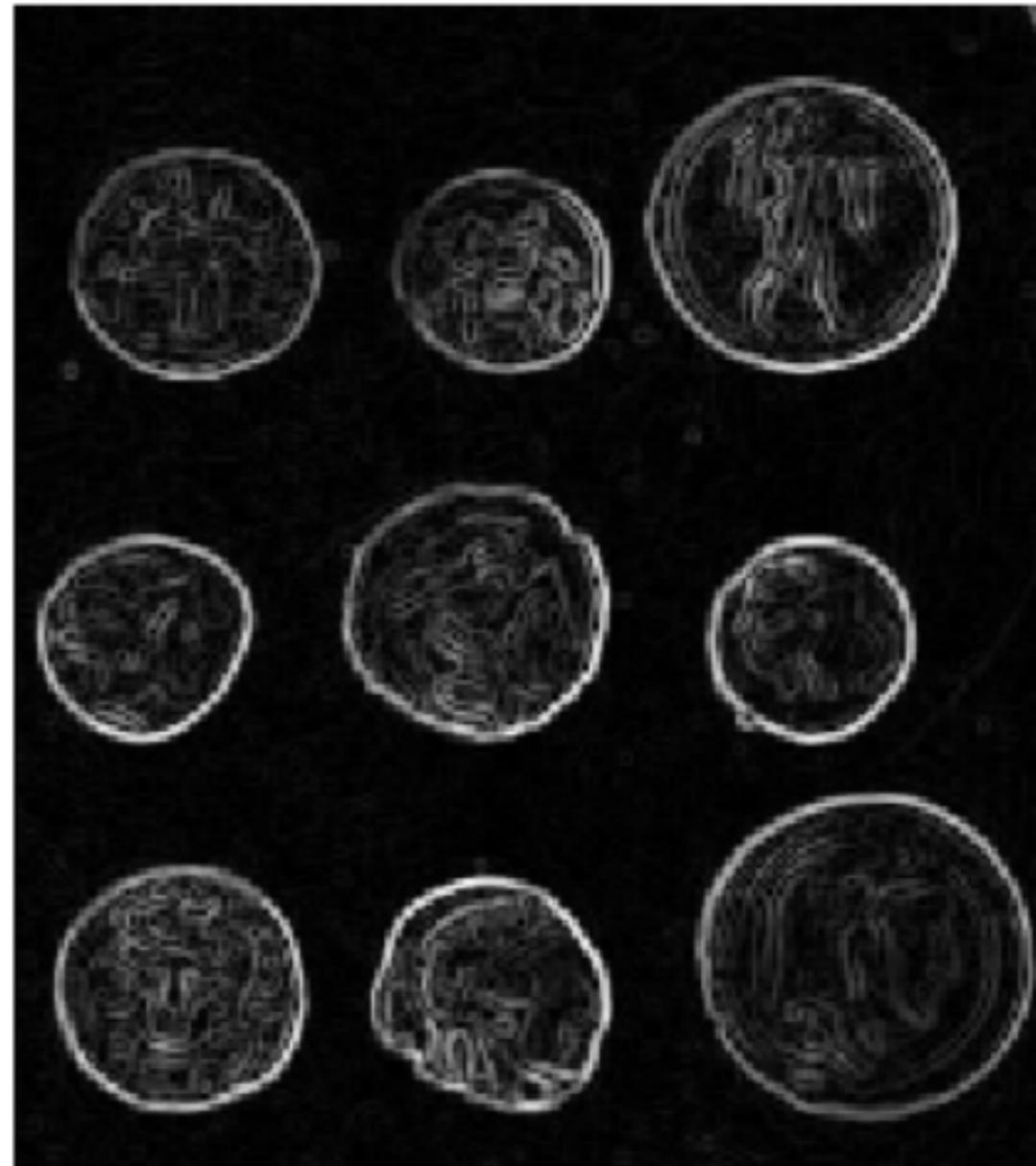


# Edge detection

Original



Edges with Sobel

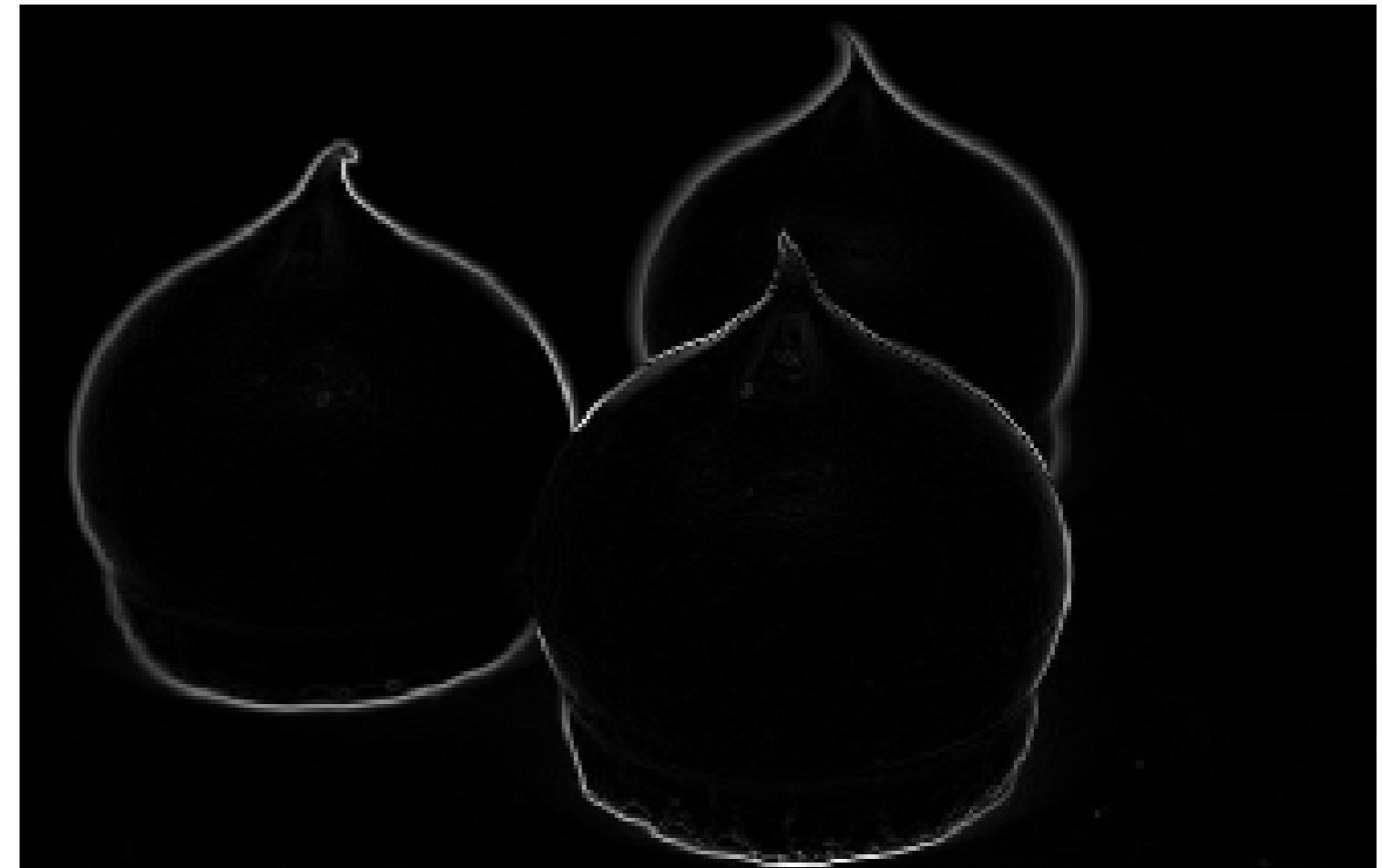


# Edge detection

Original chocolate kisses



Edges with Sobel



# Edge detection

## Sobel

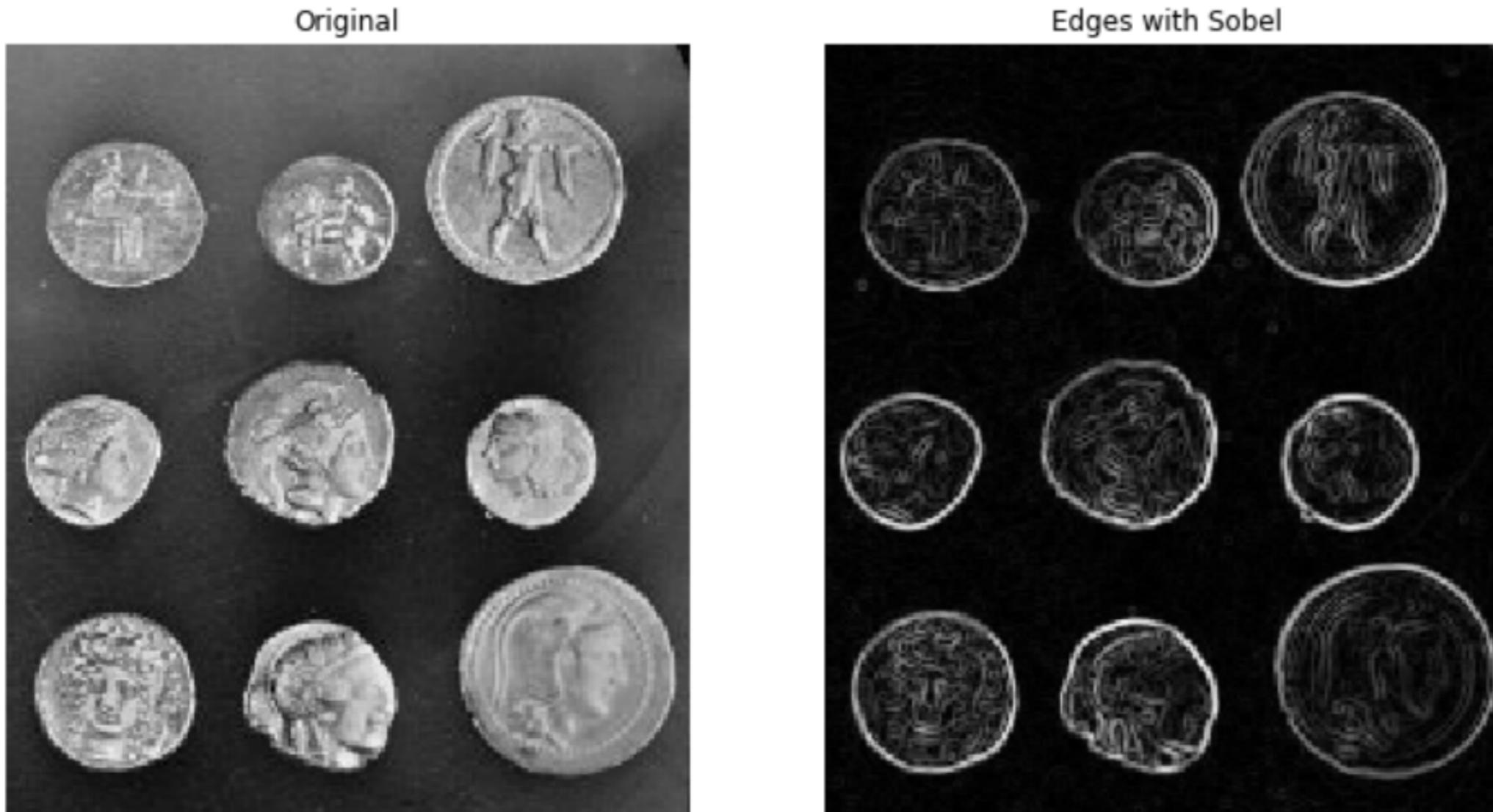
```
# Import module and function
from skimage.filters import sobel

# Apply edge detection filter
edge_sobel = sobel(image_coins)

# Show original and resulting image to compare
plot_comparison(image_coins, edge_sobel, "Edge with Sobel")
```

# Edge detection

## Sobel



# Comparing plots

```
def plot_comparison(original, filtered, title_filtered):  
  
    fig, (ax1, ax2) = plt.subplots(ncols=2, figsize=(8, 6), sharex=True,  
                                  sharey=True)  
    ax1.imshow(original, cmap=plt.cm.gray)  
    ax1.set_title('original')  
    ax1.axis('off')  
    ax2.imshow(filtered, cmap=plt.cm.gray)  
    ax2.set_title(title_filtered)  
    ax2.axis('off')
```

# Gaussian smoothing

This technique is typically used to blur an image or to reduce noise.

The Gaussian filter will blur edges and reduce contrast.

Original



Blurred with Gaussian filter



# Gaussian smoothing



# Gaussian smoothing

```
# Import the module and function
from skimage.filters import gaussian

# Apply edge detection filter
gaussian_image = gaussian(amsterdam_pic, multichannel=True)
    It is set to True if the image is colored, otherwise it needs to be set to False.

# Show original and resulting image to compare
plot_comparison(amsterdam_pic, gaussian_image, "Blurred with Gaussian filter")
```

# Gaussian smoothing

Original



Blurred with Gaussian filter



# Gaussian smoothing

Original



Blurred with Gaussian filter



# Let's practice!

IMAGE PROCESSING IN PYTHON

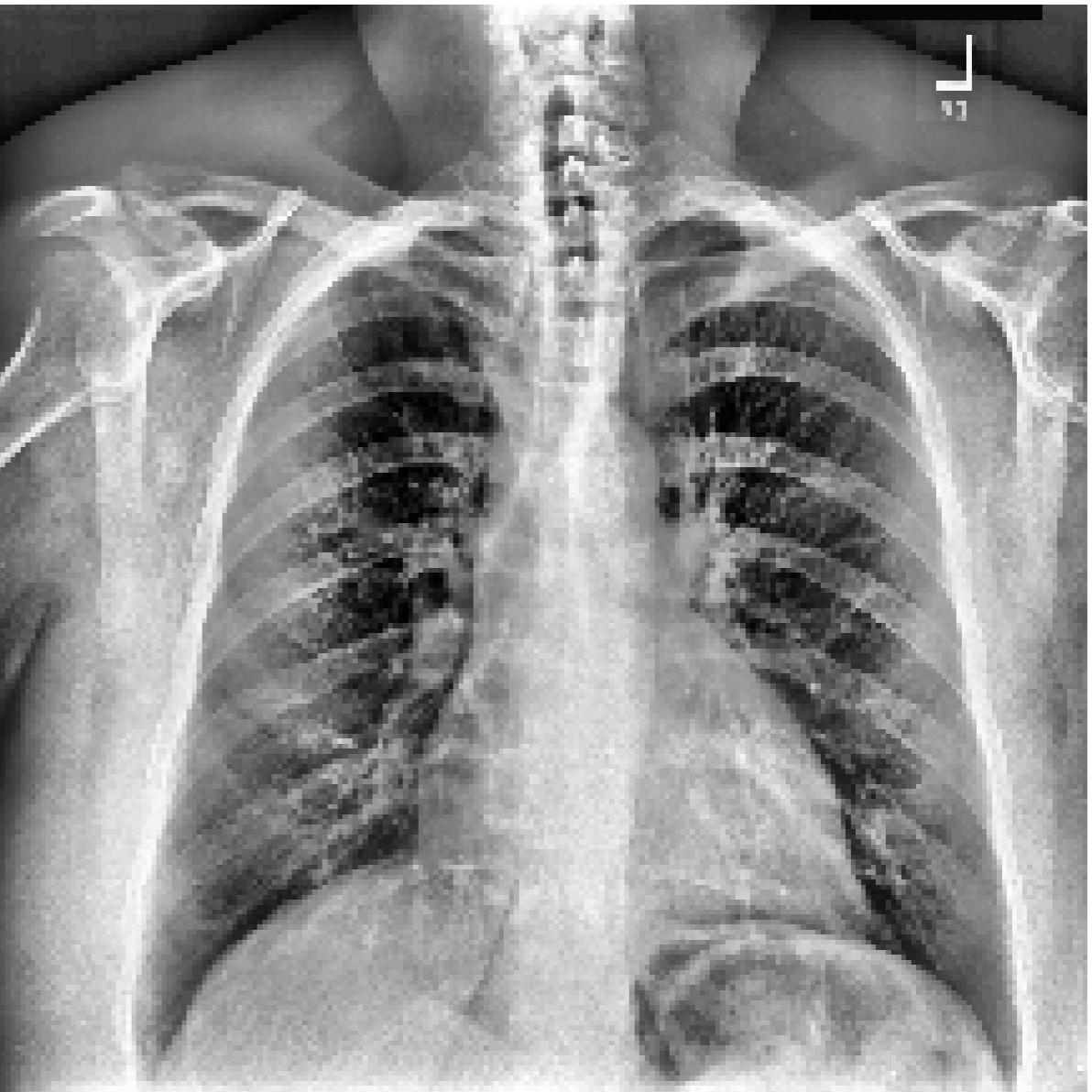
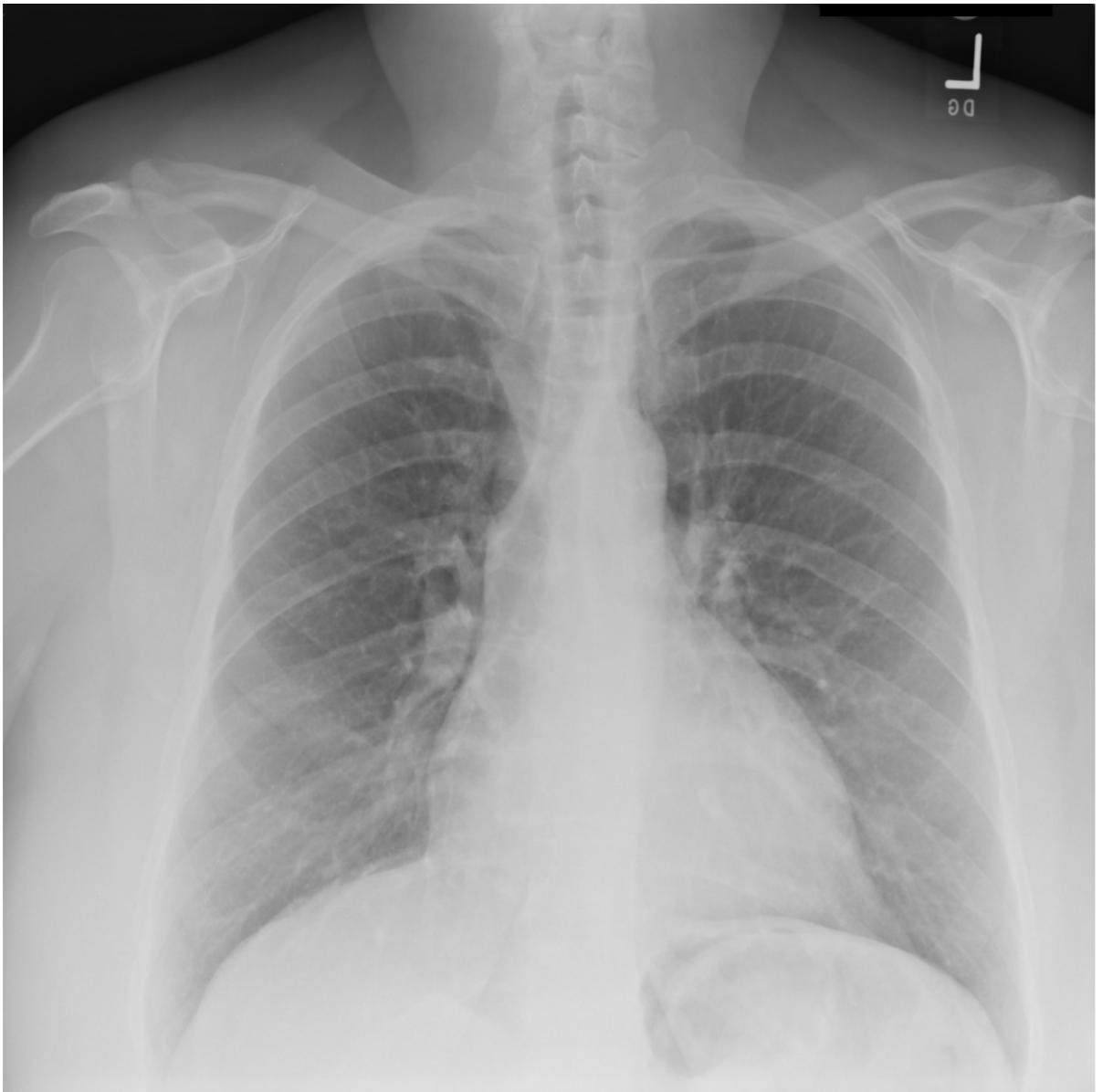
# Contrast enhancement

IMAGE PROCESSING IN PYTHON



**Rebeca Gonzalez**  
Data engineer

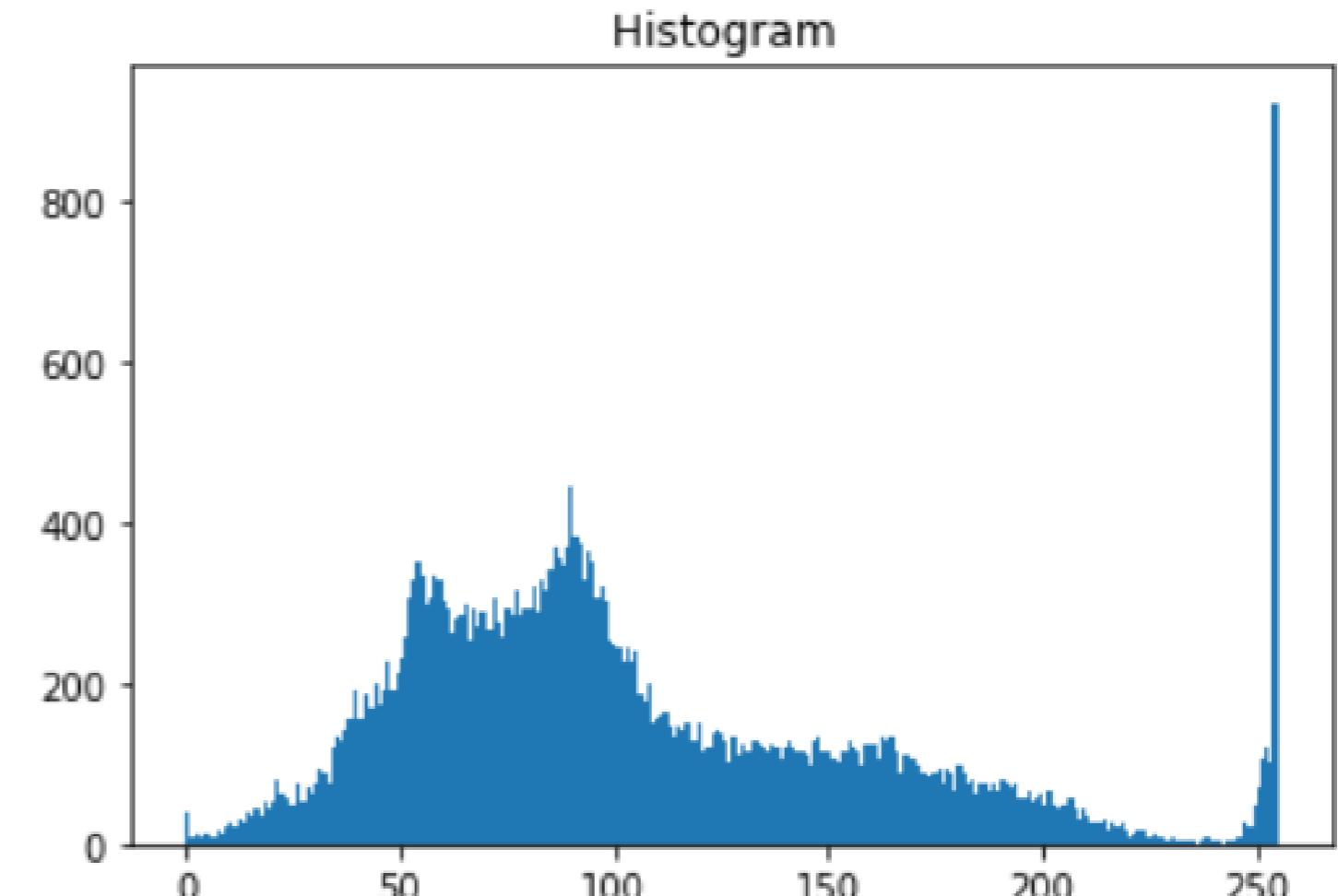
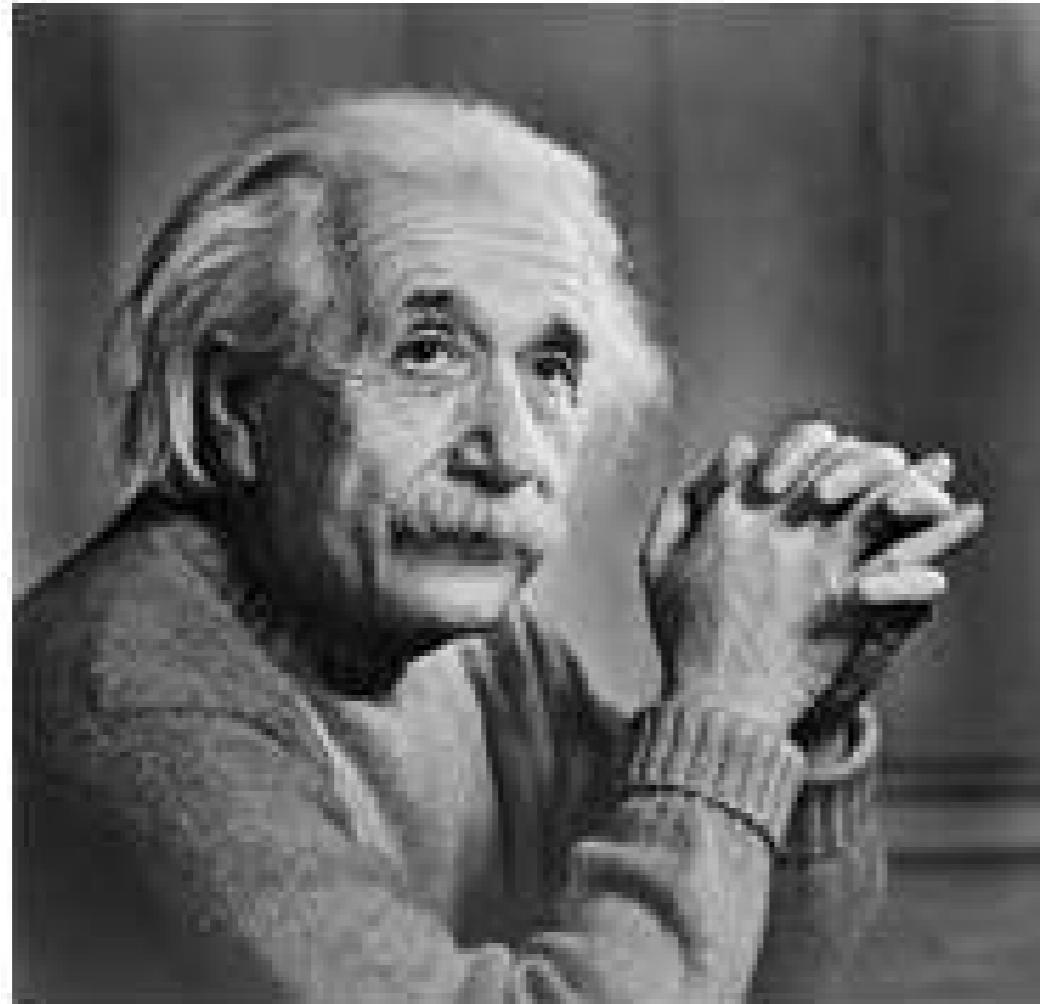
# Contrast enhancement



# Contrast

The contrast is the difference between the maximum and minimum pixel intensity in the image.

## Histograms for contrast enhancement

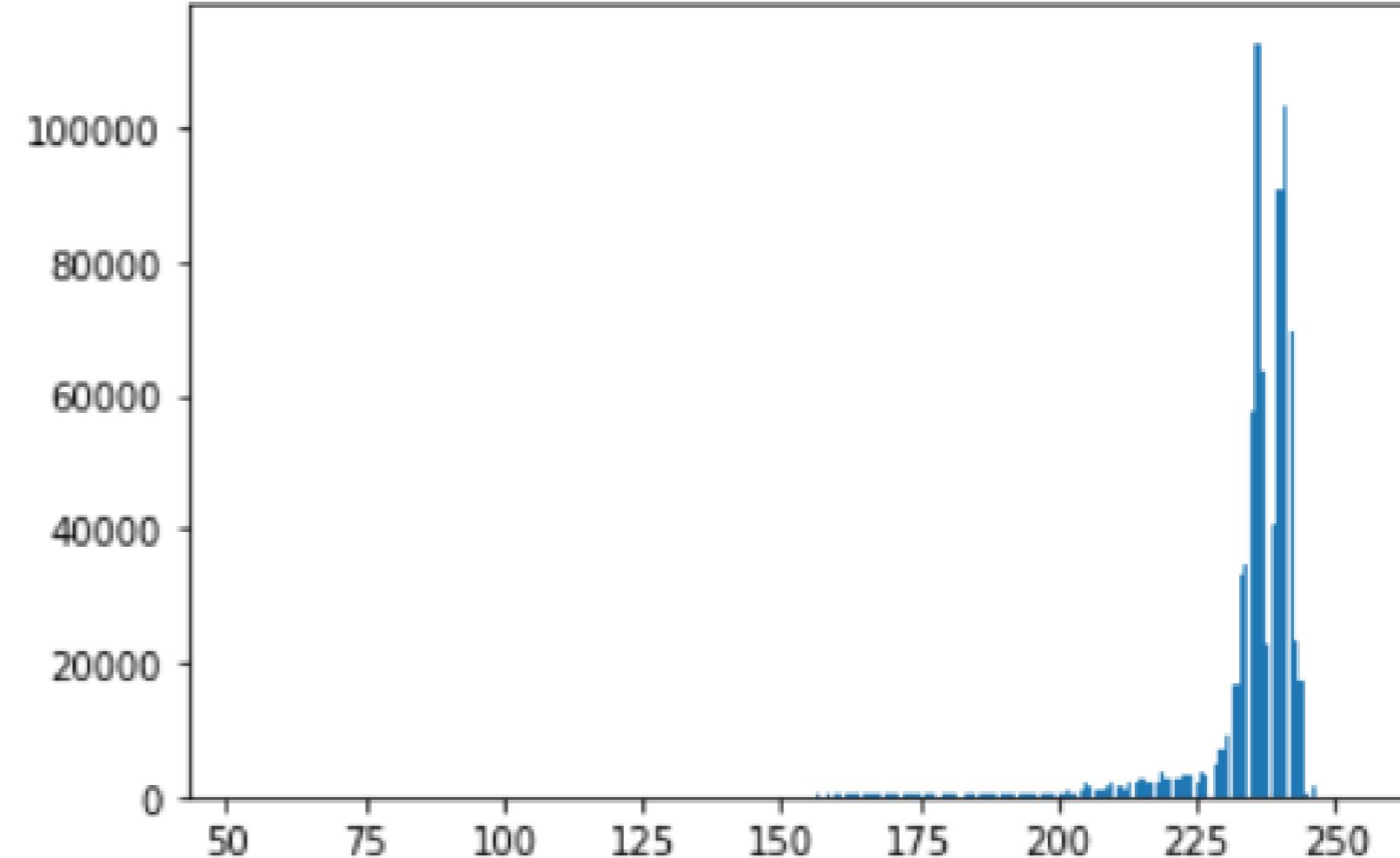


# Contrast

Low contrast image - light



Histogram of image



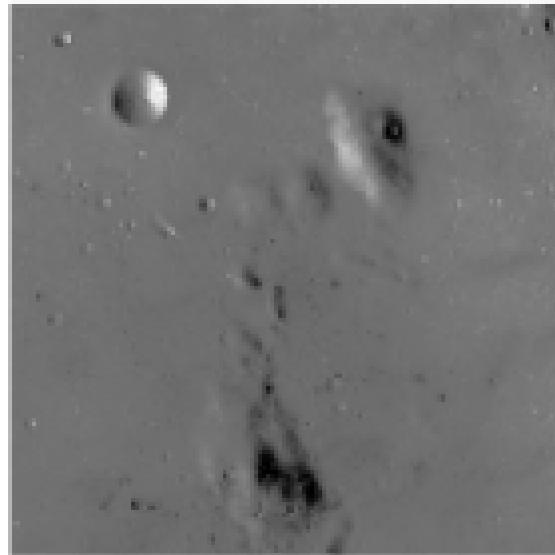
# Enhance contrast

- Contrast stretching
- Histogram equalization

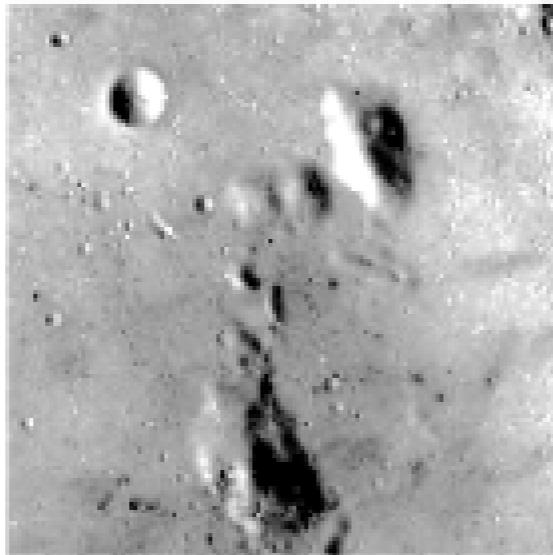
# Types

- Histogram equalization
- Adaptive histogram equalization
- Contrast Limited Adaptive Histogram Equalization (CLAHE)

Low contrast image



Contrast stretching



Histogram equalization



Adaptive equalization



# Histogram equalization

Original



Histogram Equalization



# Histogram equalization

Original



# Histogram equalization

```
from skimage import exposure

# Obtain the equalized image
image_eq = exposure.equalize_hist(image)

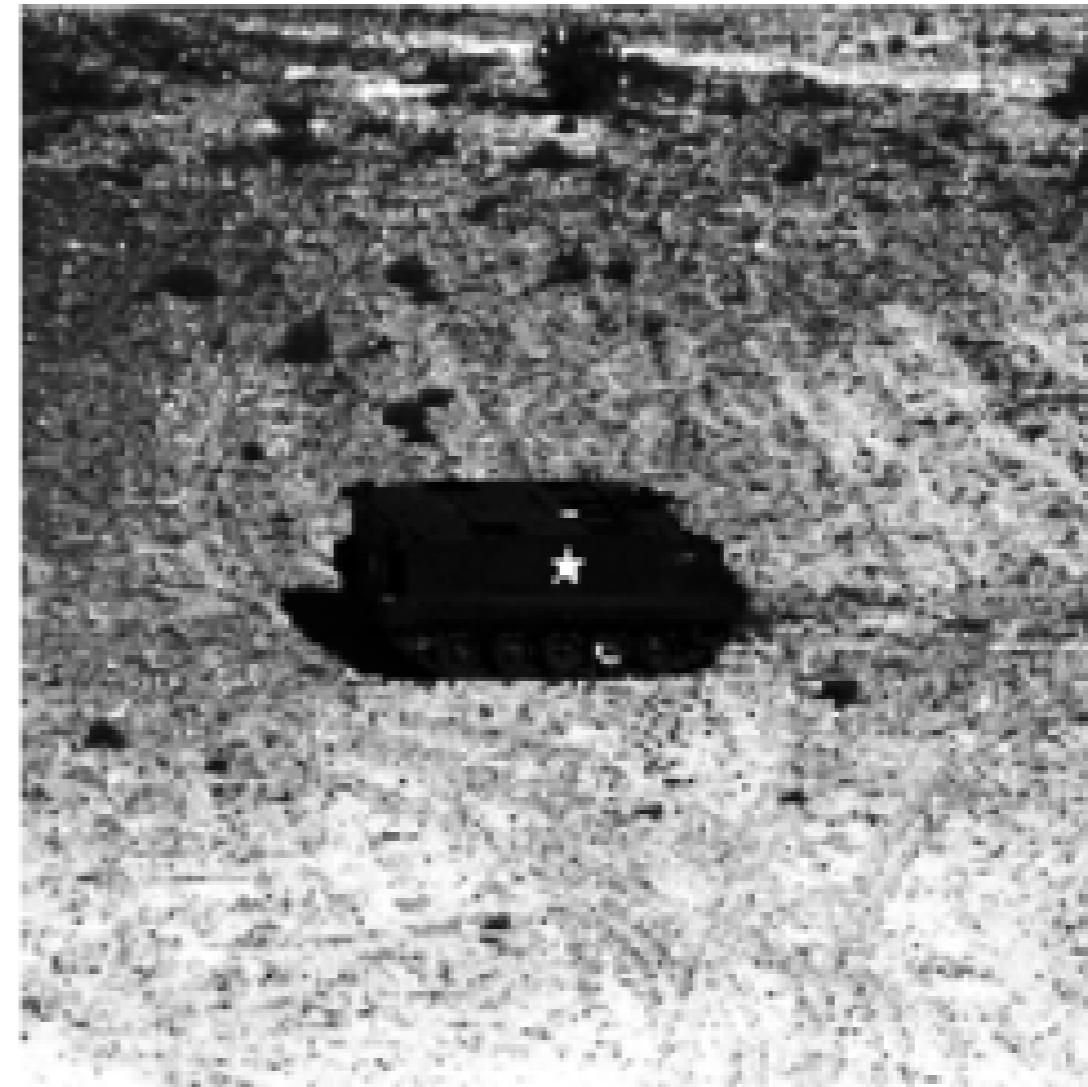
# Show original and result
show_image(image, 'Original')
show_image(image_eq, 'Histogram equalized')
```

# Histogram equalization

Original



Histogram Equalization



# Adaptive Equalization

- Contrastive Limited Adaptive Histogram Equalization

Original



Adaptive Equalization



This method computes several histograms, each corresponding to a distinct part of the image, and uses them to redistribute the lightness values of the image histogram.

# Contrastive Limited Adaptive Equalization

Original



Histogram Equalization Adaptive Equalization



The adaptive method is not that intense, so it looks more natural. This is because it is not taking the global histogram of the entire image, but operates on small regions called tiles or neighborhoods.

# CLAHE in scikit-image

```
from skimage import exposure

# Apply adaptive Equalization
image_adapteq = exposure.equalize_adapthist(image, clip_limit=0.03)
    This clipping limit, is normalized between 0 and 1 (higher values give more contrast).

# Show original and result
show_image(image, 'Original')
show_image(image_adapteq, 'Adaptive equalized')
```

# CLAHE in scikit-image

Original



Adaptive Equalization

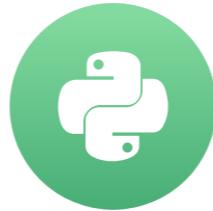


# Let's practice!

IMAGE PROCESSING IN PYTHON

# Transformations

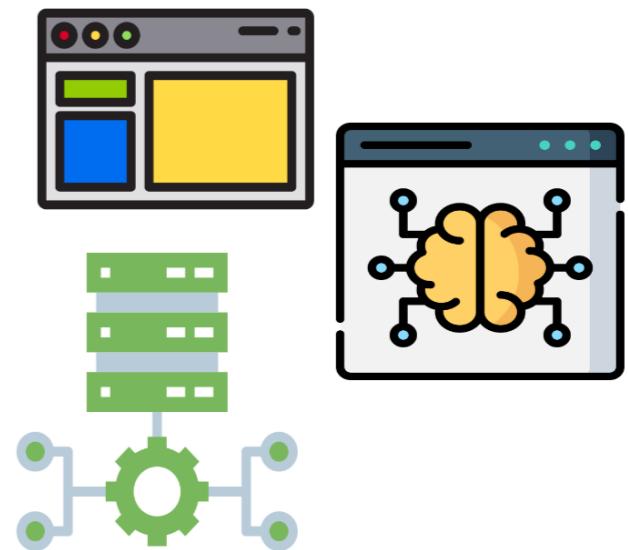
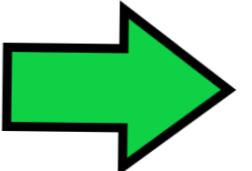
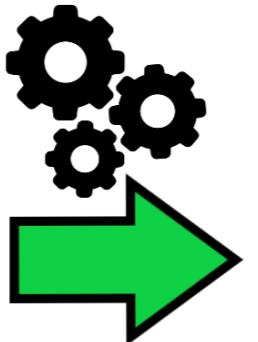
IMAGE PROCESSING IN PYTHON



**Rebeca Gonzalez**  
Data Engineer

# Why transform images?

- Preparing images for classification Machine Learning models
- Optimization and compression of images
- Save images with same proportion

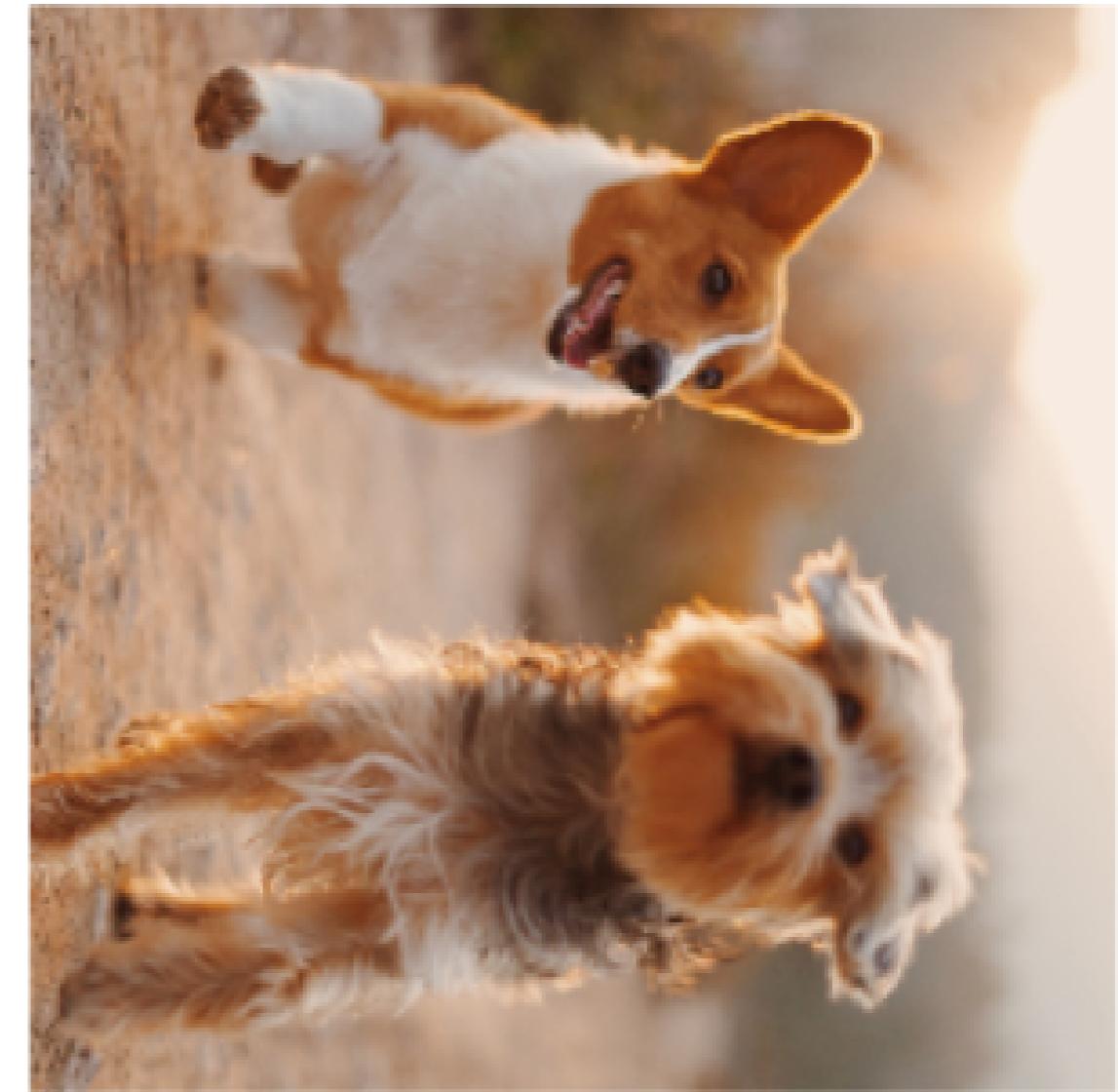


# Rotating

Original



Rotated 90 degrees clockwise



# Rotating

Original



Rotated 90 degrees anticlockwise



# Rotating clockwise

```
from skimage.transform import rotate

# Rotate the image 90 degrees clockwise
image_rotated = rotate(image, -90)

show_image(image_rotated, 'Original')
show_image(image_rotated, 'Rotated 90 degrees clockwise')
```

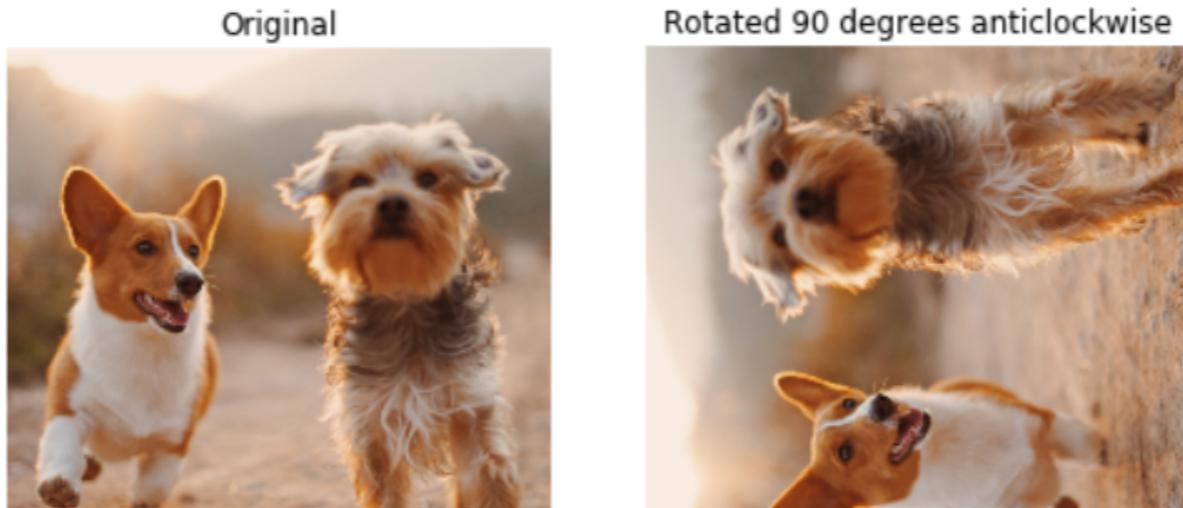


# Rotating anticlockwise

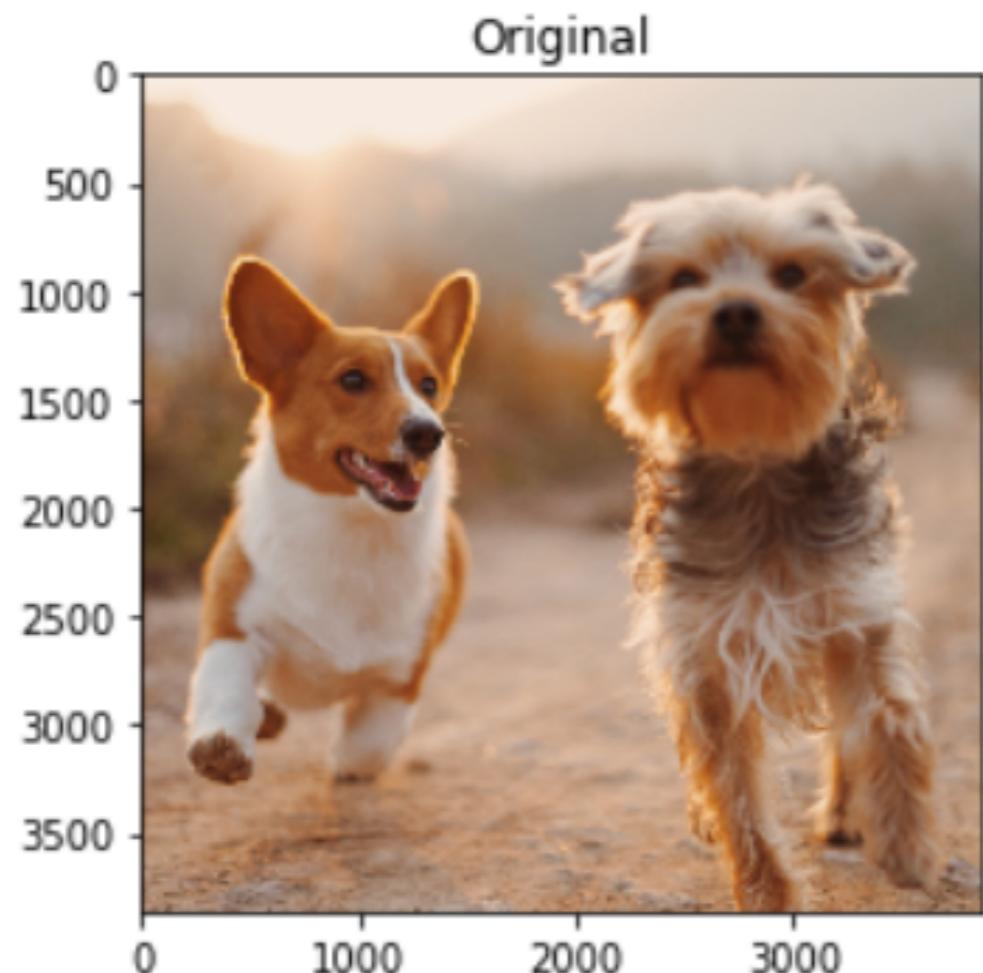
```
from skimage.transform import rotate

# Rotate an image 90 degrees anticlockwise
image_rotated = rotate(image, 90)

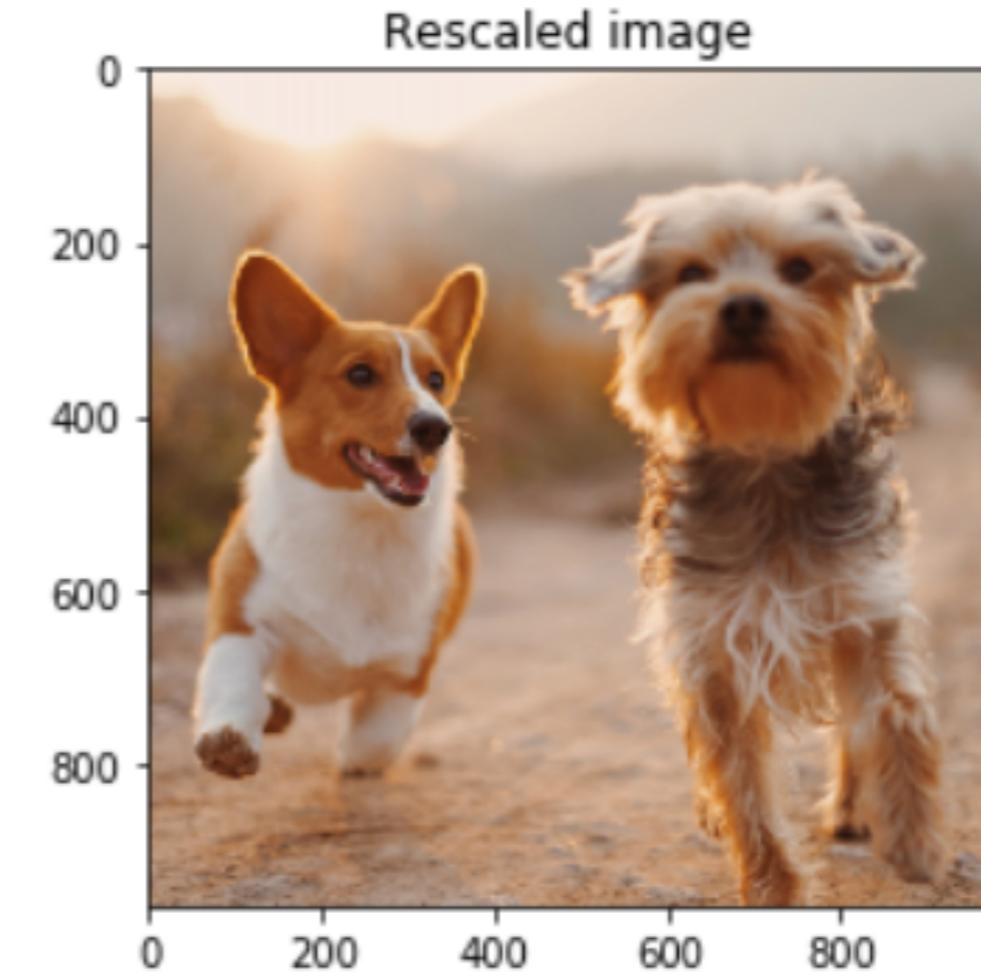
show_image(image, 'Original')
show_image(image_rotated, 'Rotated 90 degrees anticlockwise')
```



# Rescaling



Original size



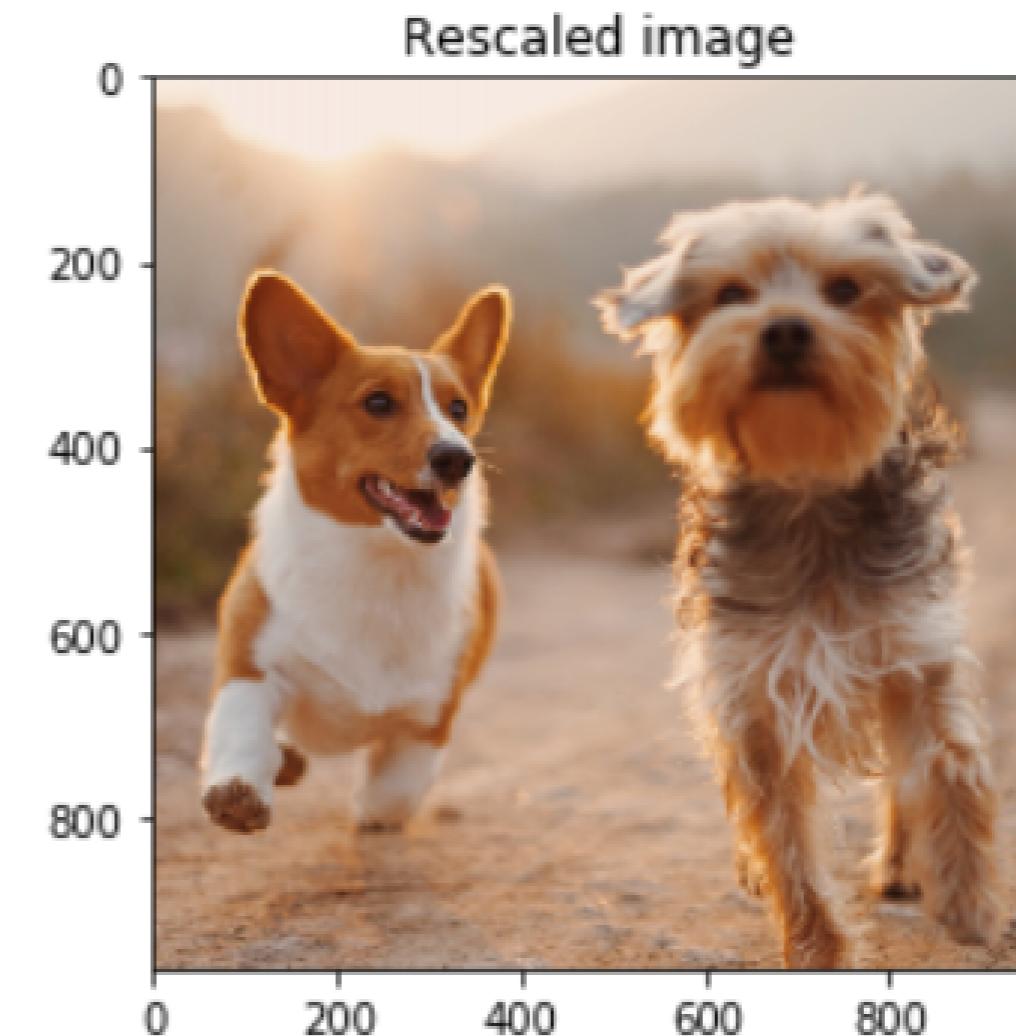
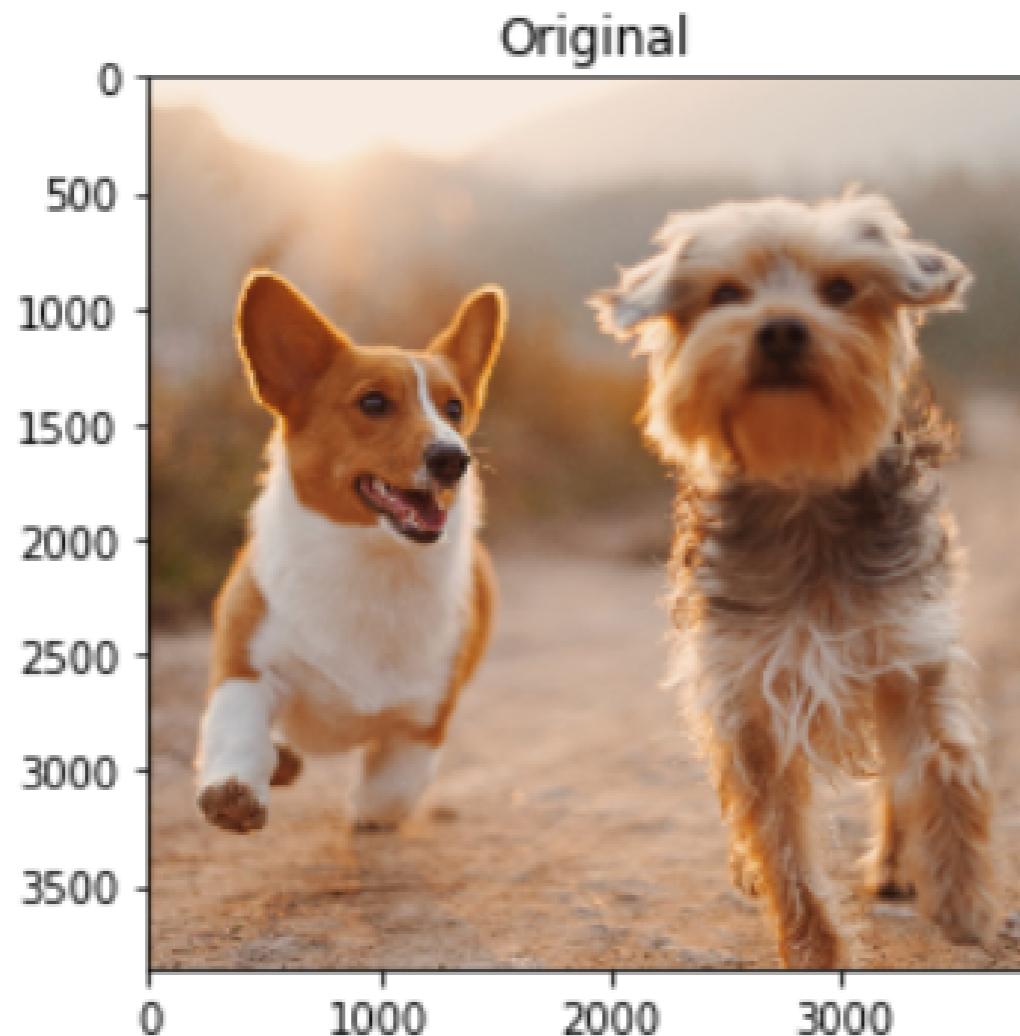
Rescaled size

# Rescaling

## Downgrading

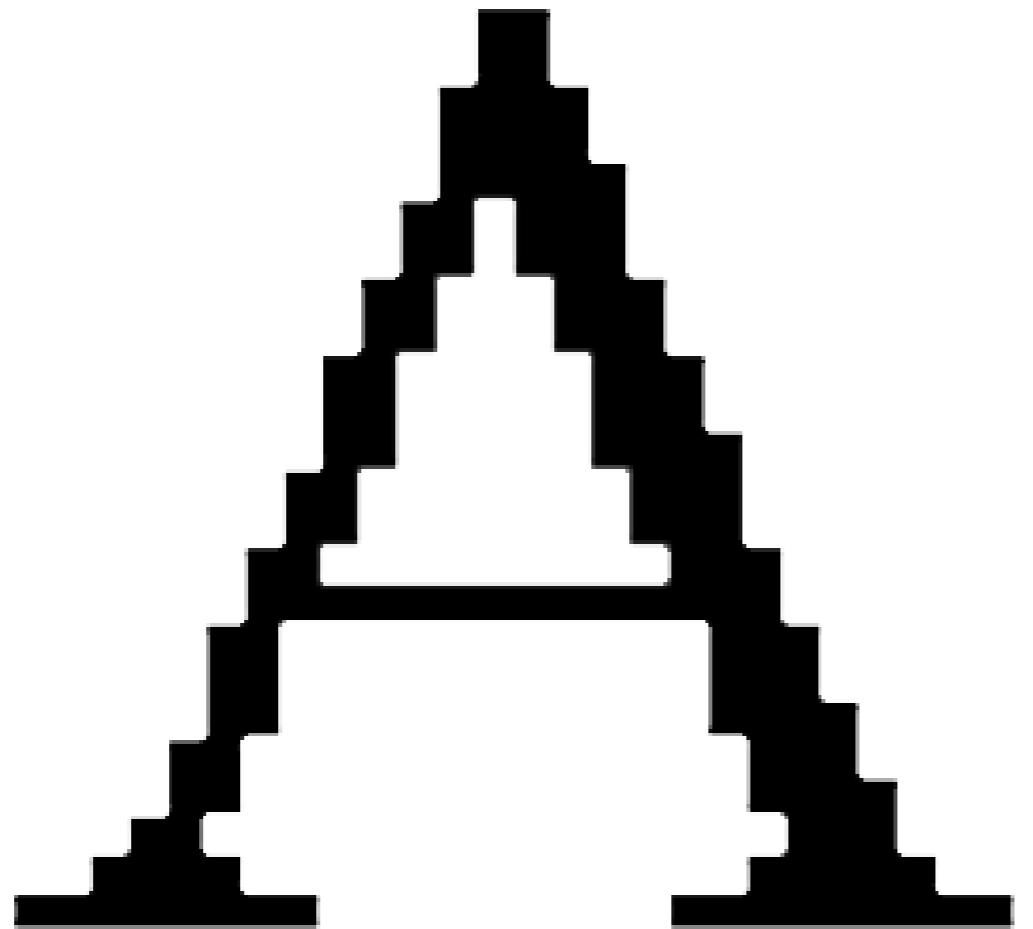
```
from skimage.transform import rescale  
  
# Rescale the image to be 4 times smaller  
image_rescaled = rescale(image, 1/4, anti_aliasing=True, multichannel=True)  
  
show_image(image, 'Original image')  
show_image(image_rescaled, 'Rescaled image')  
  
Setting an anti_aliasing boolean parameter to true specifies if applying  
a filter to smooth the image prior to down-scaling.
```

# Rescaling

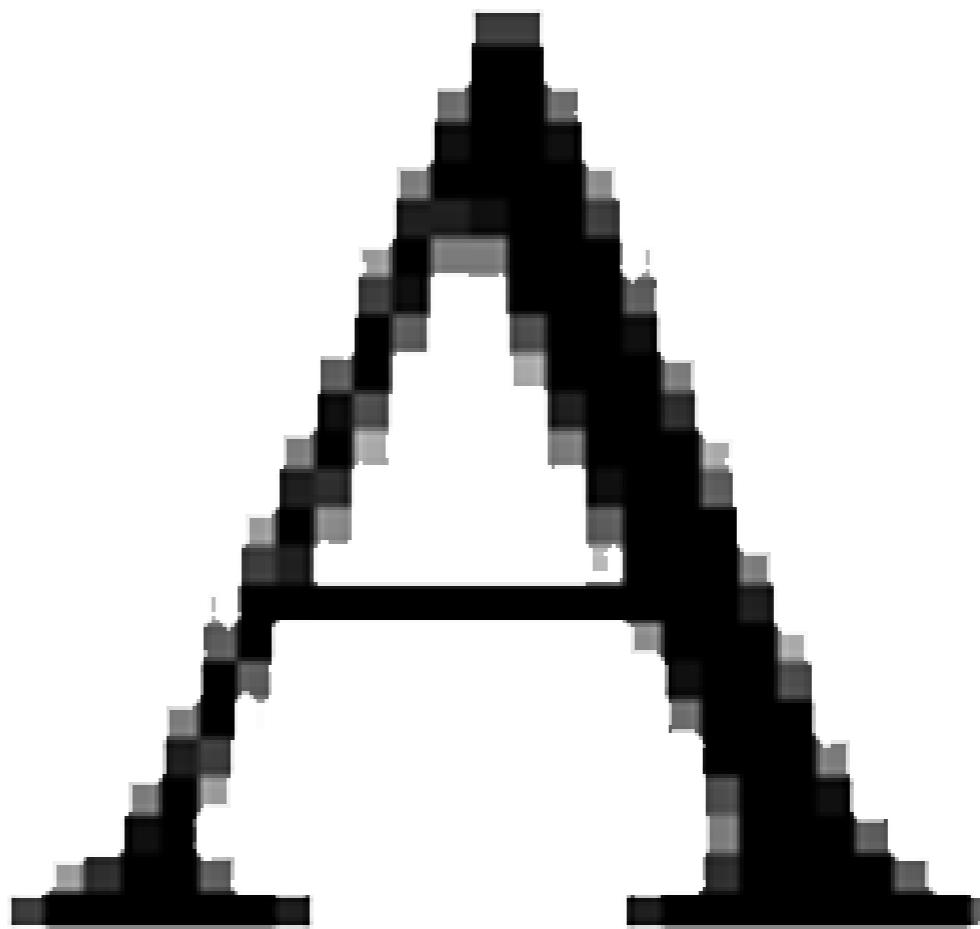


# Aliasing in digital images

In a digital image, aliasing is a pattern or a rippling effect. This happens because the pixelation of the image is poor.

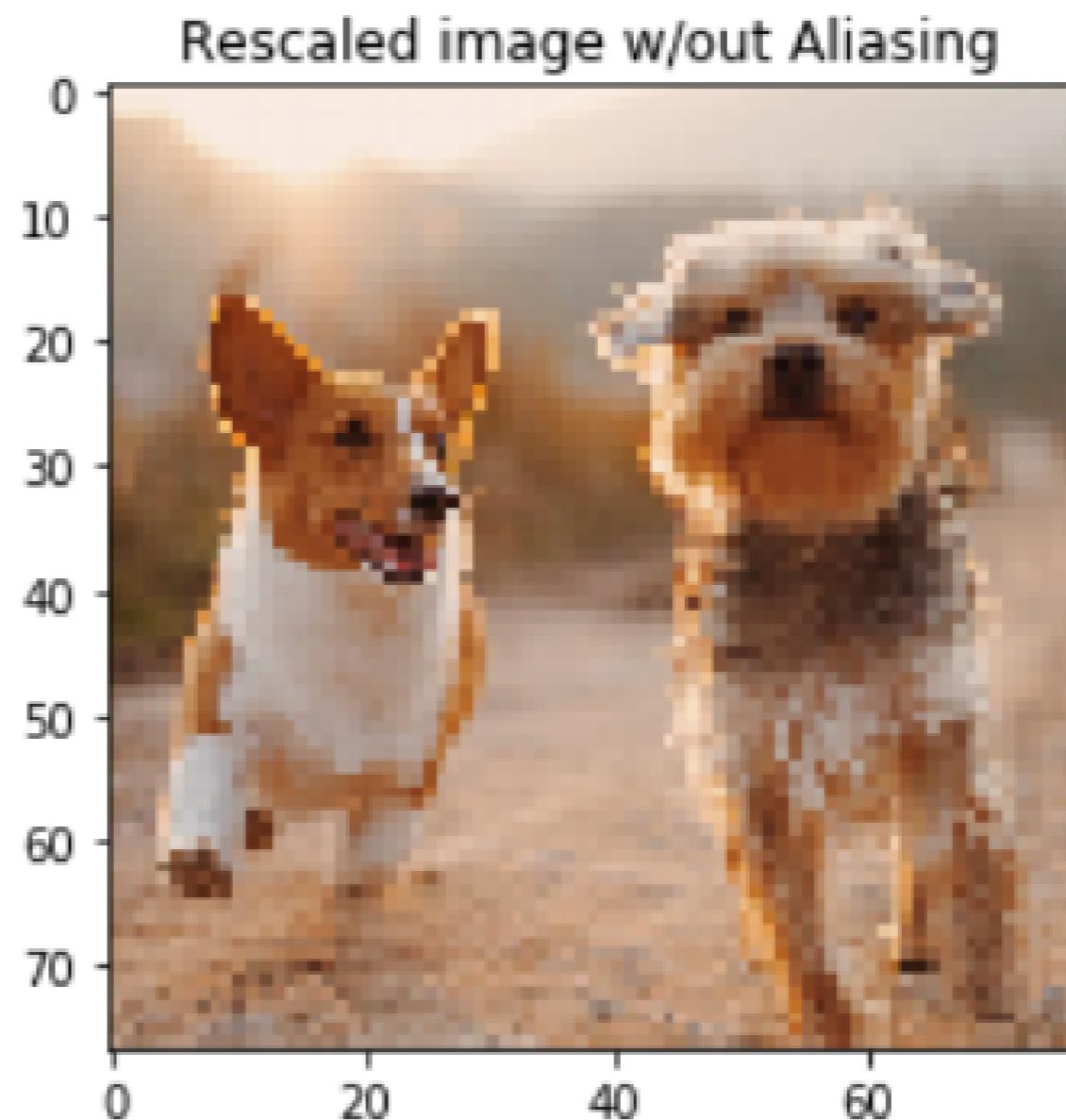
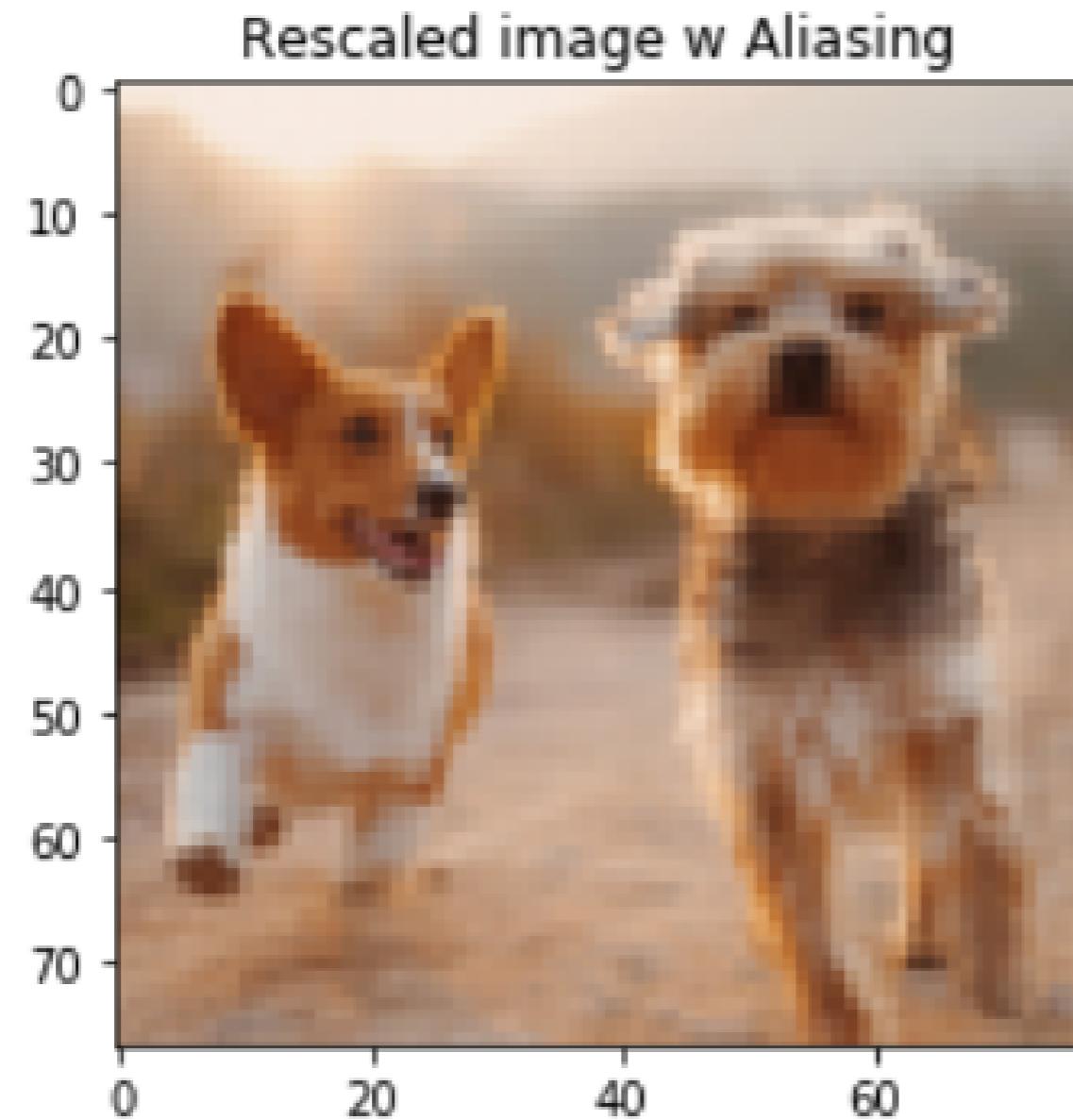


Original letter A

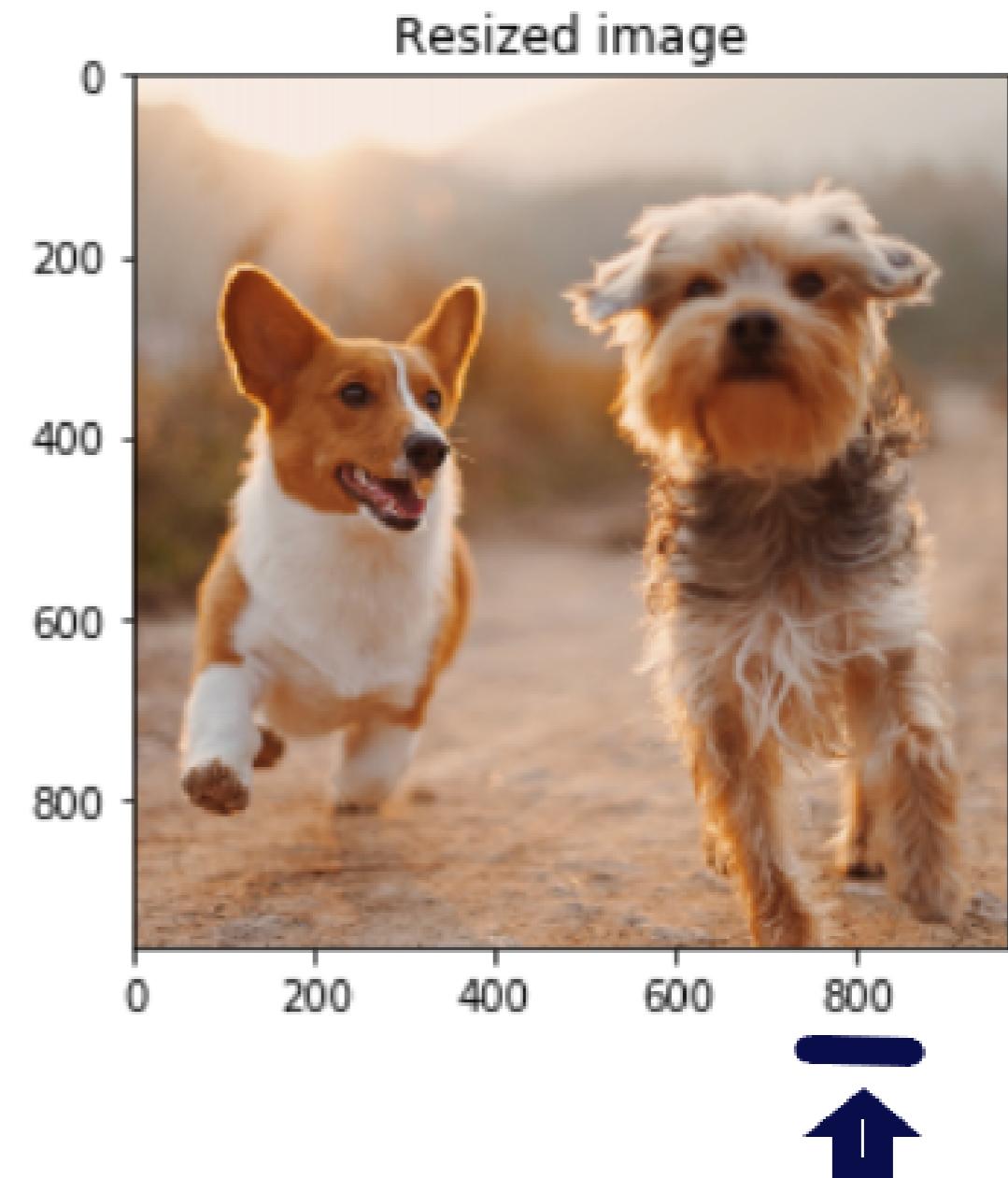
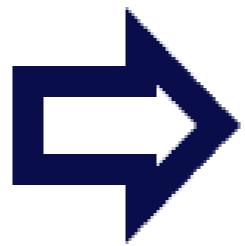
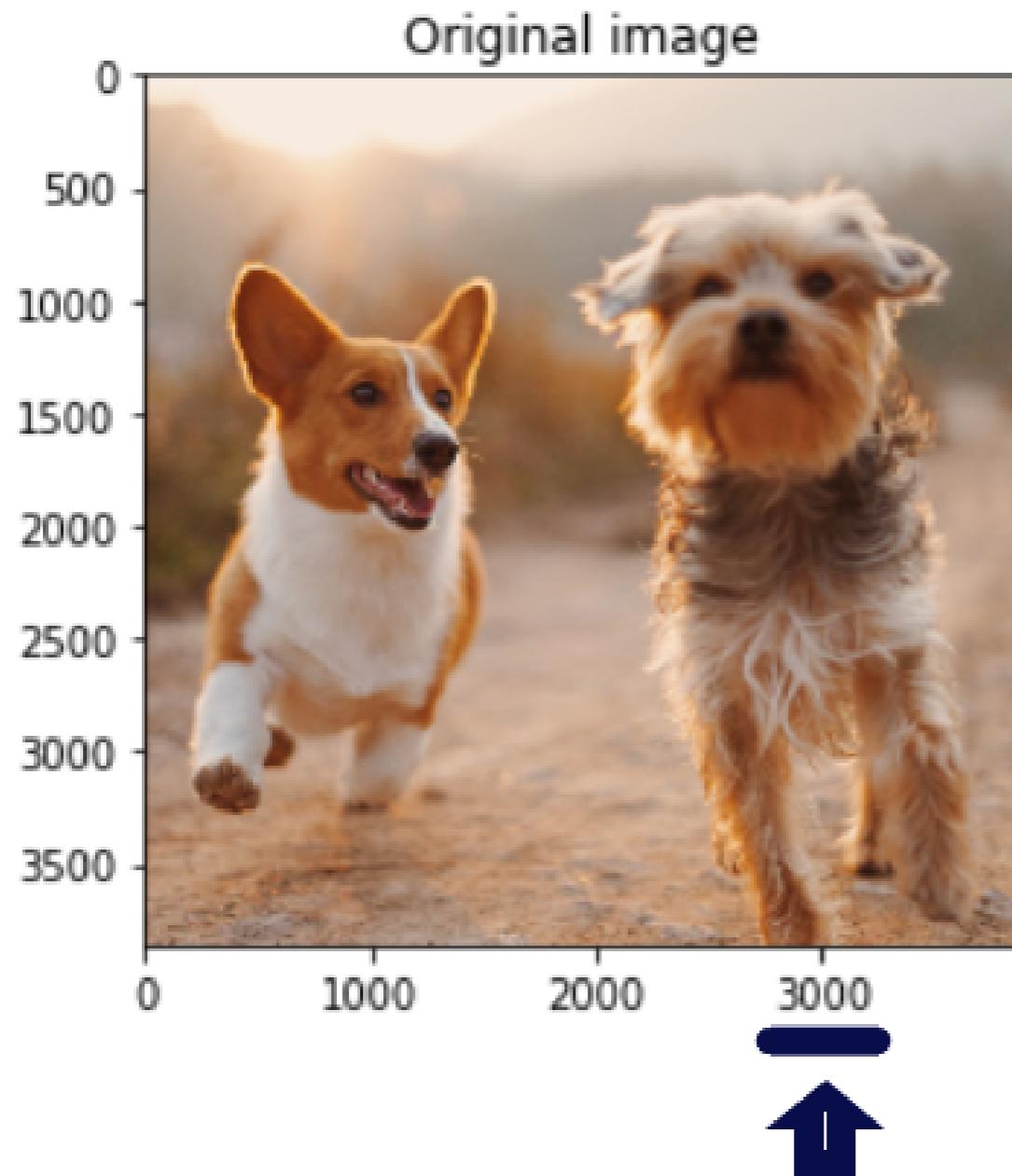


Aliased letter A

# Aliasing in digital images



# Resizing



# Resizing

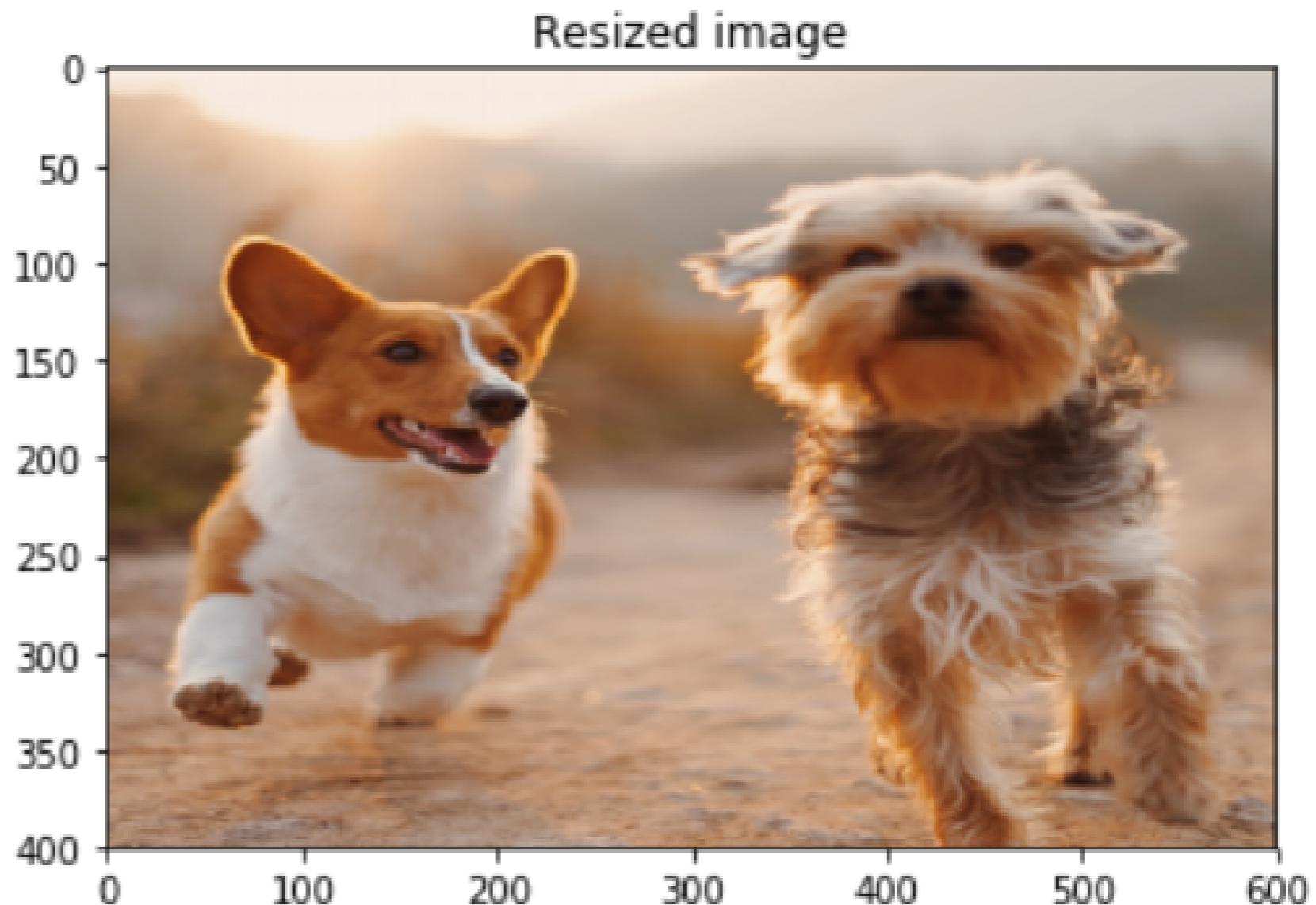
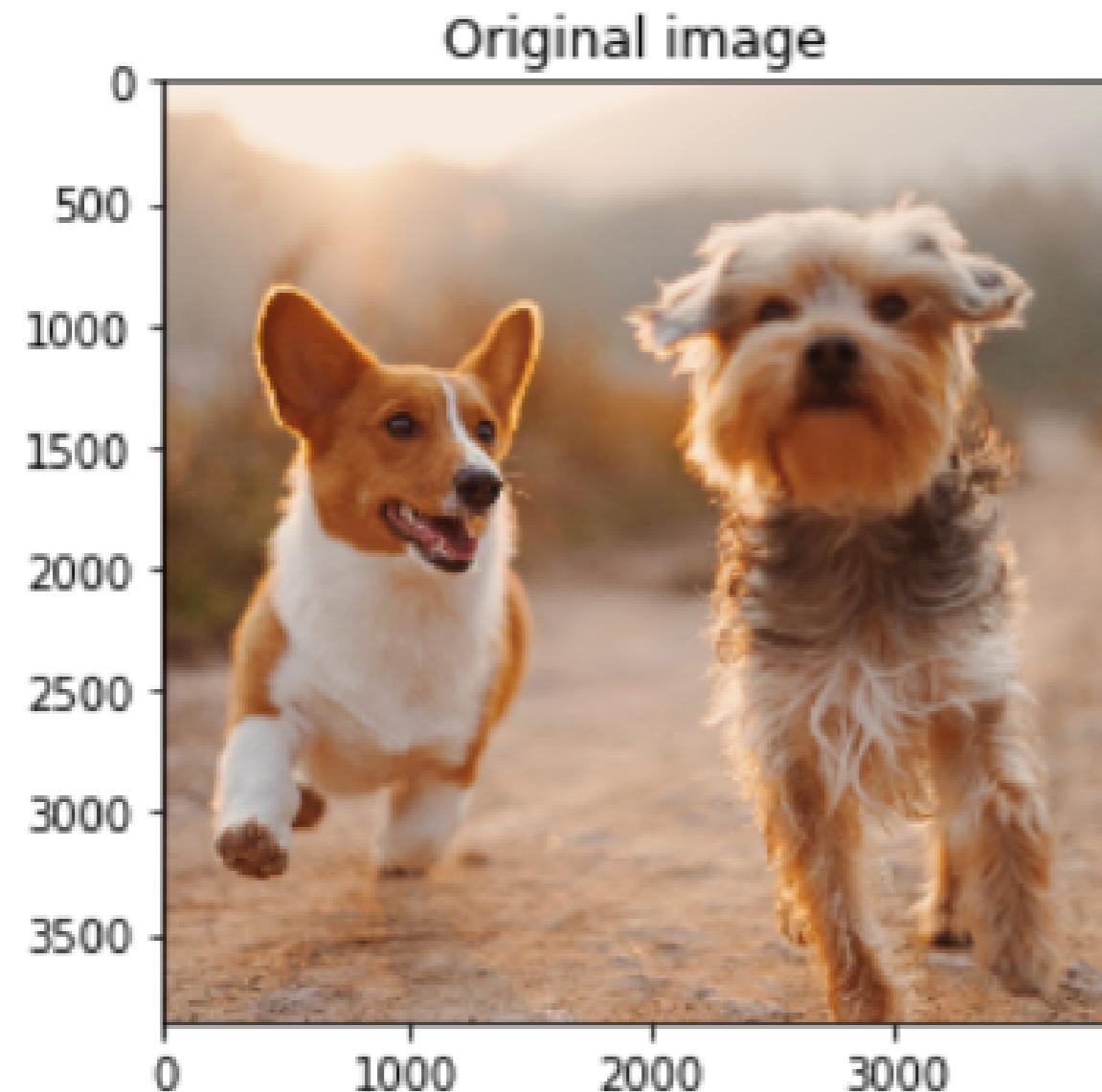
```
from skimage.transform import resize

# Height and width to resize
height = 400
width = 500

# Resize image
image_resized = resize(image, (height, width), anti_aliasing=True)

# Show the original and resulting images
show_image(image, 'Original image')
show_image(image_resized, 'Resized image')
```

# Resizing



# Resizing proportionally

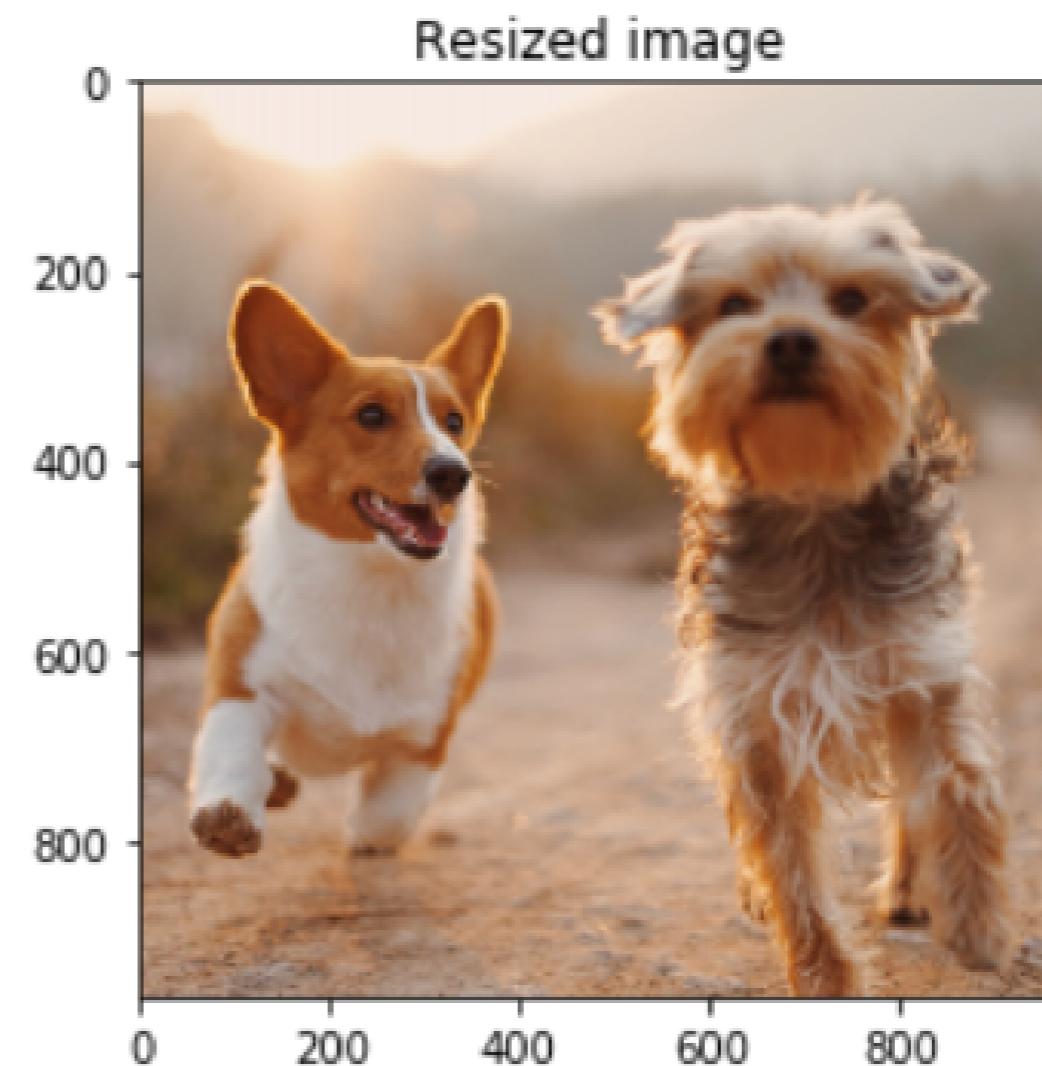
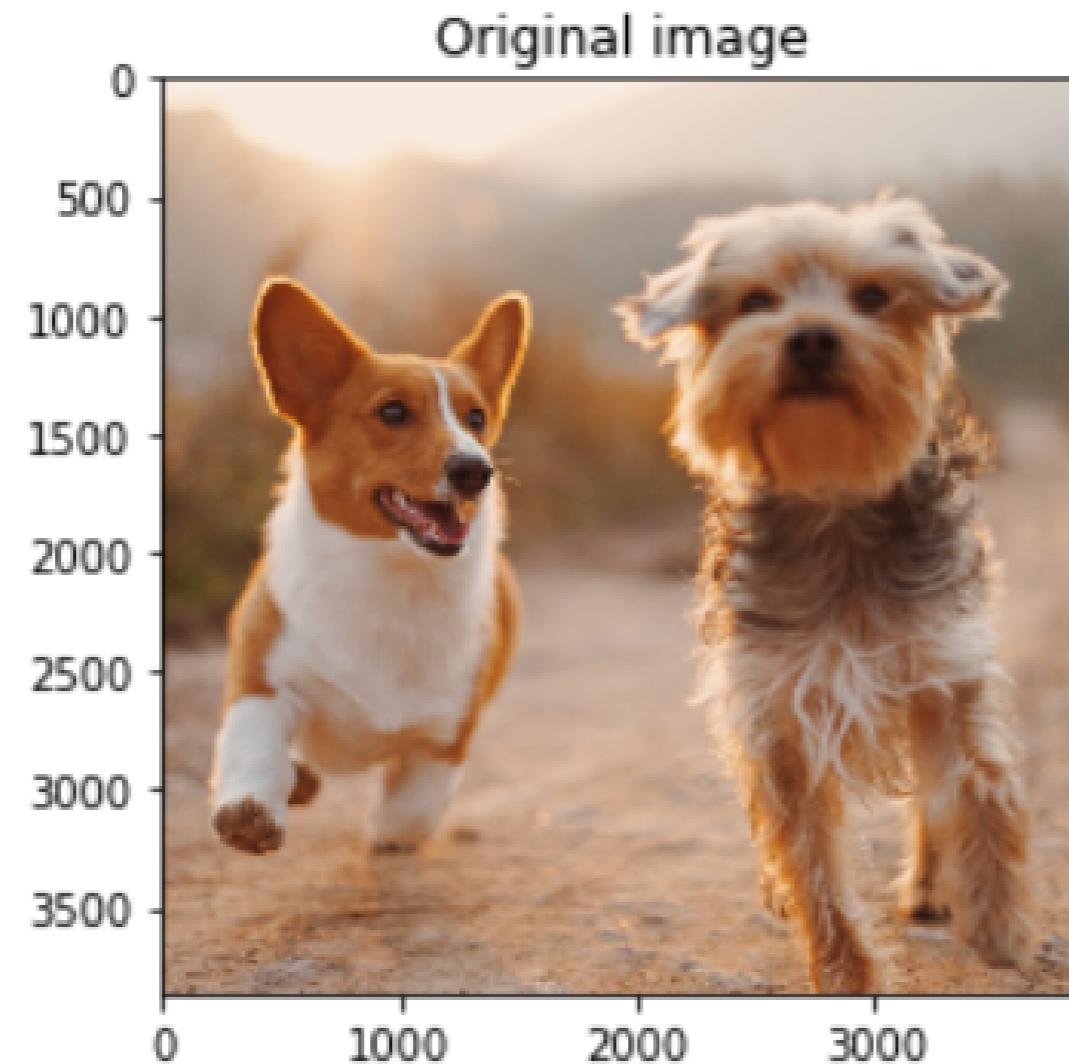
```
from skimage.transform import resize

# Set proportional height so its 4 times its size
height = image.shape[0] / 4
width = image.shape[1] / 4

# Resize image
image_resized = resize(image, (height, width), anti_aliasing=True)

show_image(image_resized, 'Resized image')
```

# Resizing proportionally

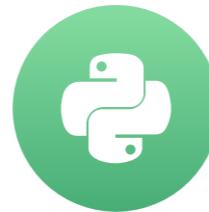


# Let's practice!

IMAGE PROCESSING IN PYTHON

# Morphology

IMAGE PROCESSING IN PYTHON



**Rebeca Gonzalez**  
Data Engineer

# Binary images

original



Thresholded image



# Morphological filtering

- Better for binary images
- Can extend for grayscale

Binary image



Grayscale

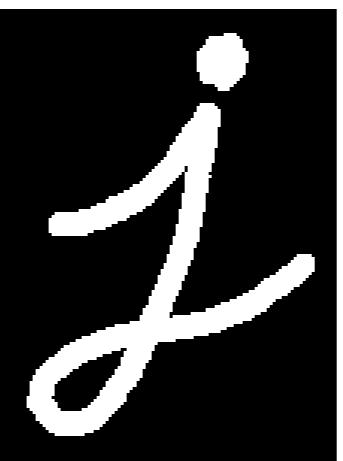


# Morphological operations

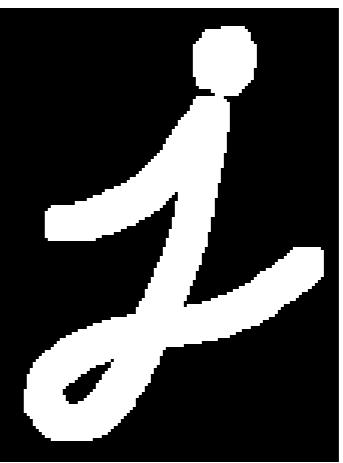
- Dilation
- Erosion

Dilation adds pixels to the boundaries of objects in an image, while erosion removes pixels on object boundaries.

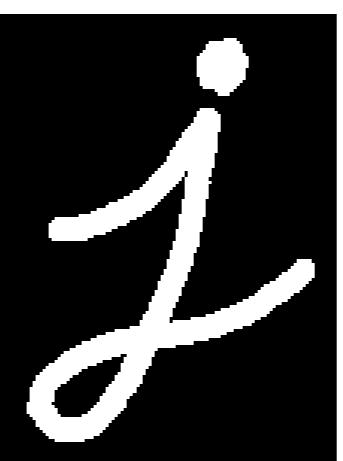
Original



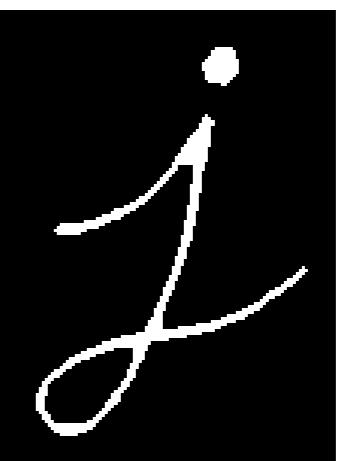
Dilated



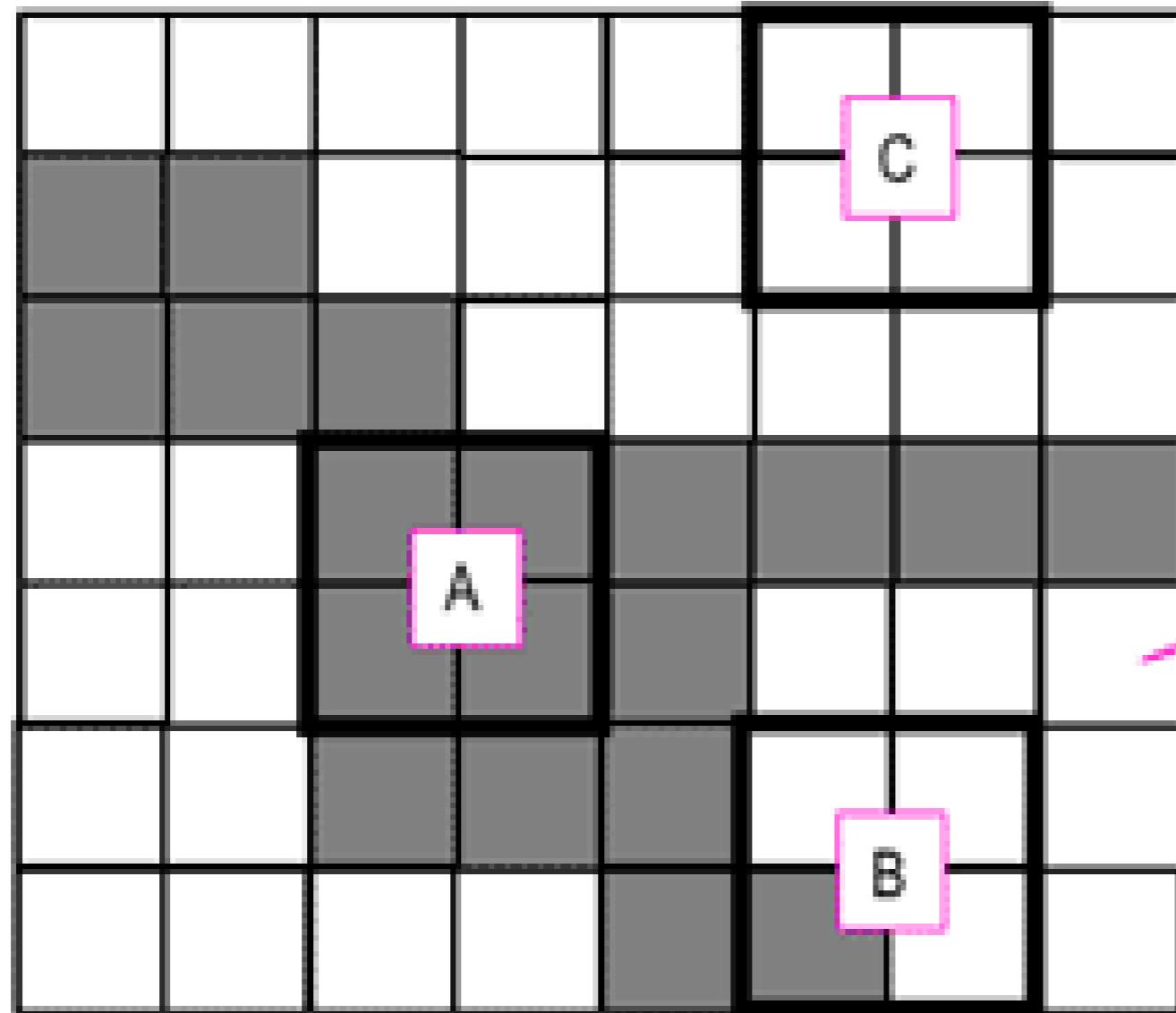
Original



Eroded

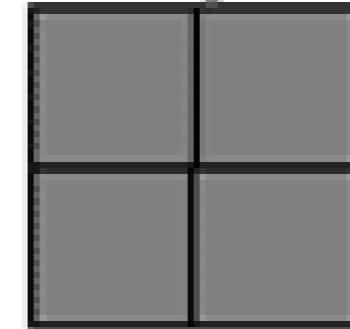


# Structuring element



- A - the structuring element fits the image
- B - the structuring element hits (intersects) the image
- C - the structuring element neither fits, nor hits the image

Structuring element



# Structuring element

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

Square 5x5 element

0	0	1	0	0
0	1	1	1	0
1	1	1	1	1
0	1	1	1	0
0	0	1	0	0

Diamond-shaped 5x5 element

0	0	1	0	0
0	0	1	0	0
1	1	1	1	1
0	0	1	0	0
0	0	1	0	0

Cross-shaped 5x5 element

■ ←→ Origin

1	1	1
1	1	1
1	1	1

Square 3x3 element

# Shapes in scikit-image

```
from skimage import morphology
```

```
square = morphology.square(4)
```

```
[[1 1 1 1]
 [1 1 1 1]
 [1 1 1 1]
 [1 1 1 1]]
```

```
rectangle = morphology.rectangle(4, 2)
```

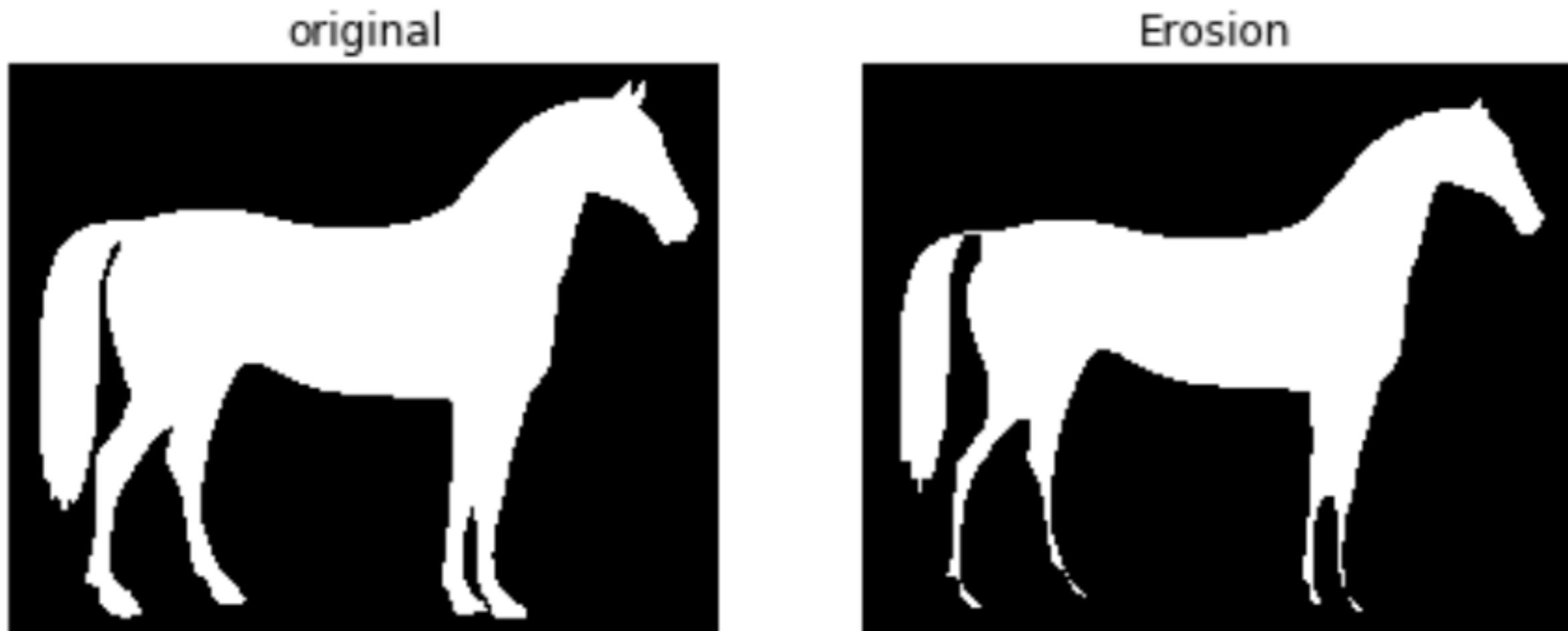
```
[[1 1]
 [1 1]
 [1 1]
 [1 1]]
```

# Erosion in scikit-image

```
from skimage import morphology  
  
# Set structuring element to the rectangular-shaped  
selem = rectangle(12, 6)  
  
# Obtain the eroded image with binary erosion  
eroded_image = morphology.binary_erosion(image_horse, selem=selem)
```

# Erosion in scikit-image

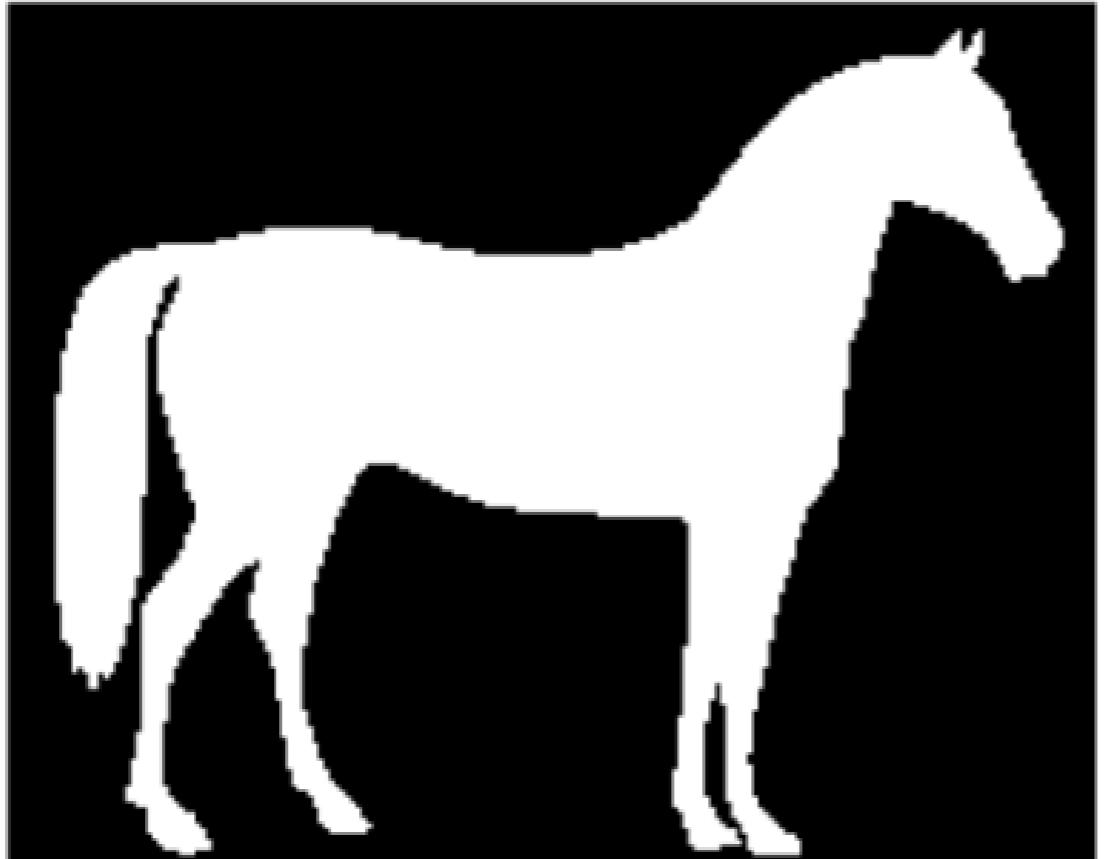
```
# Show result  
plot_comparison(image_horse, eroded_image, 'Erosion')
```



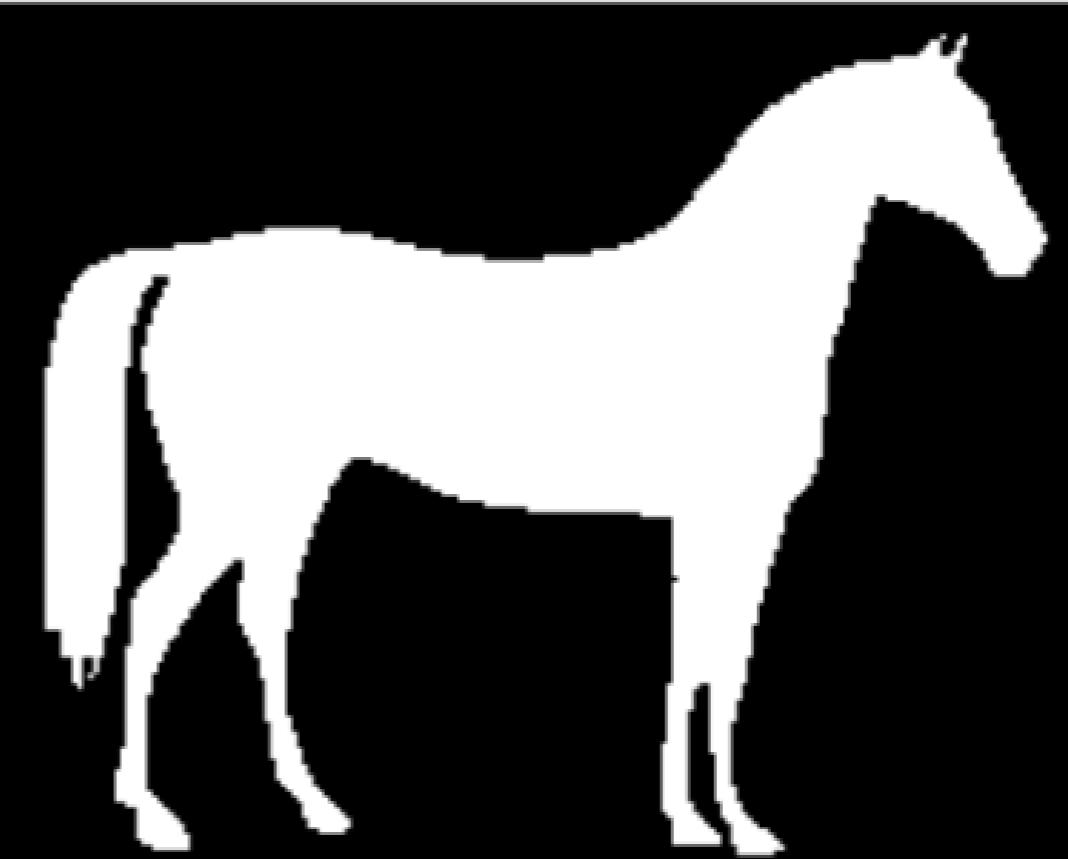
# Binary erosion with default selem

```
# Binary erosion with default selem  
eroded_image = morphology.binary_erosion(image_horse)
```

original



Erosion

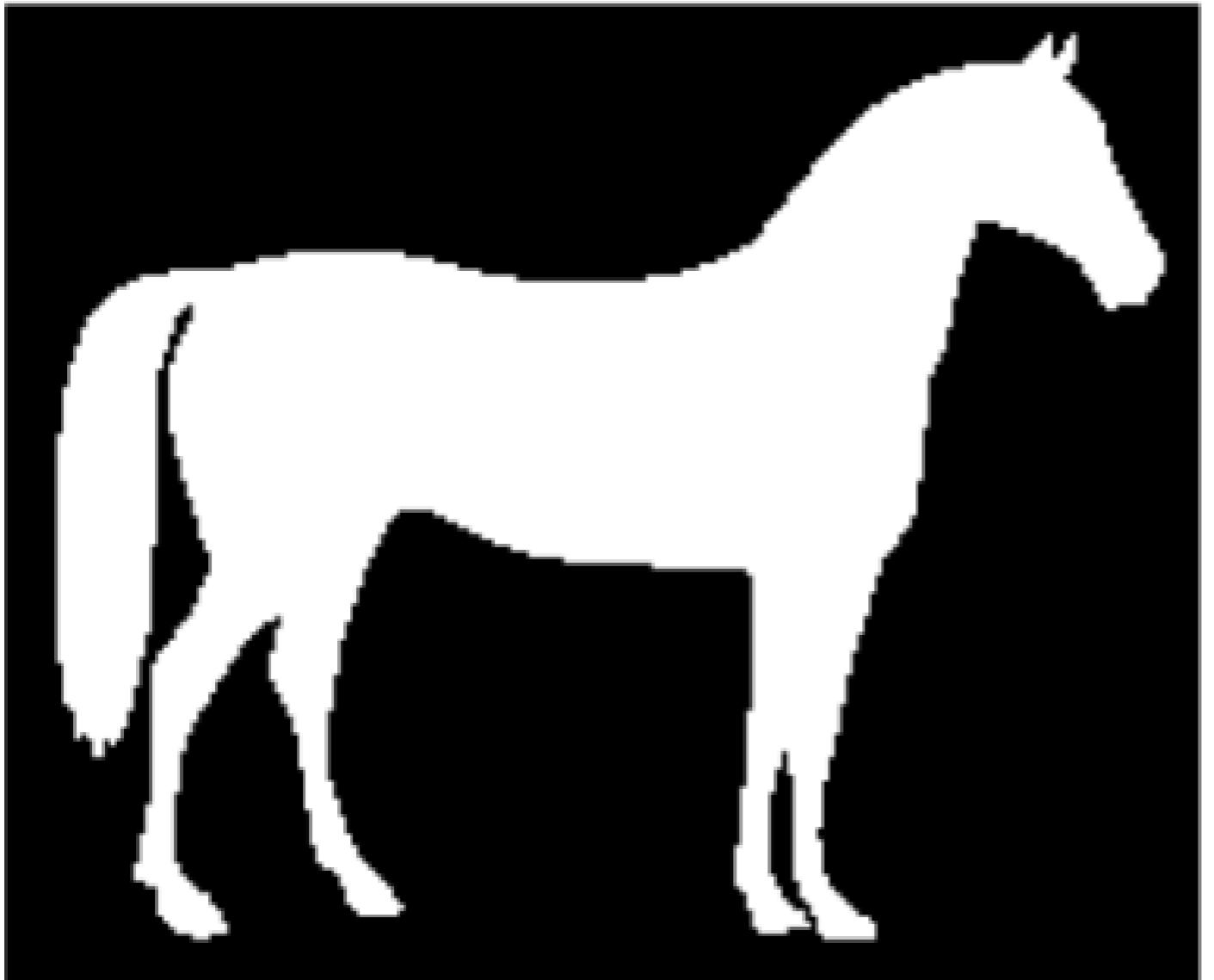


# Dilation in scikit-image

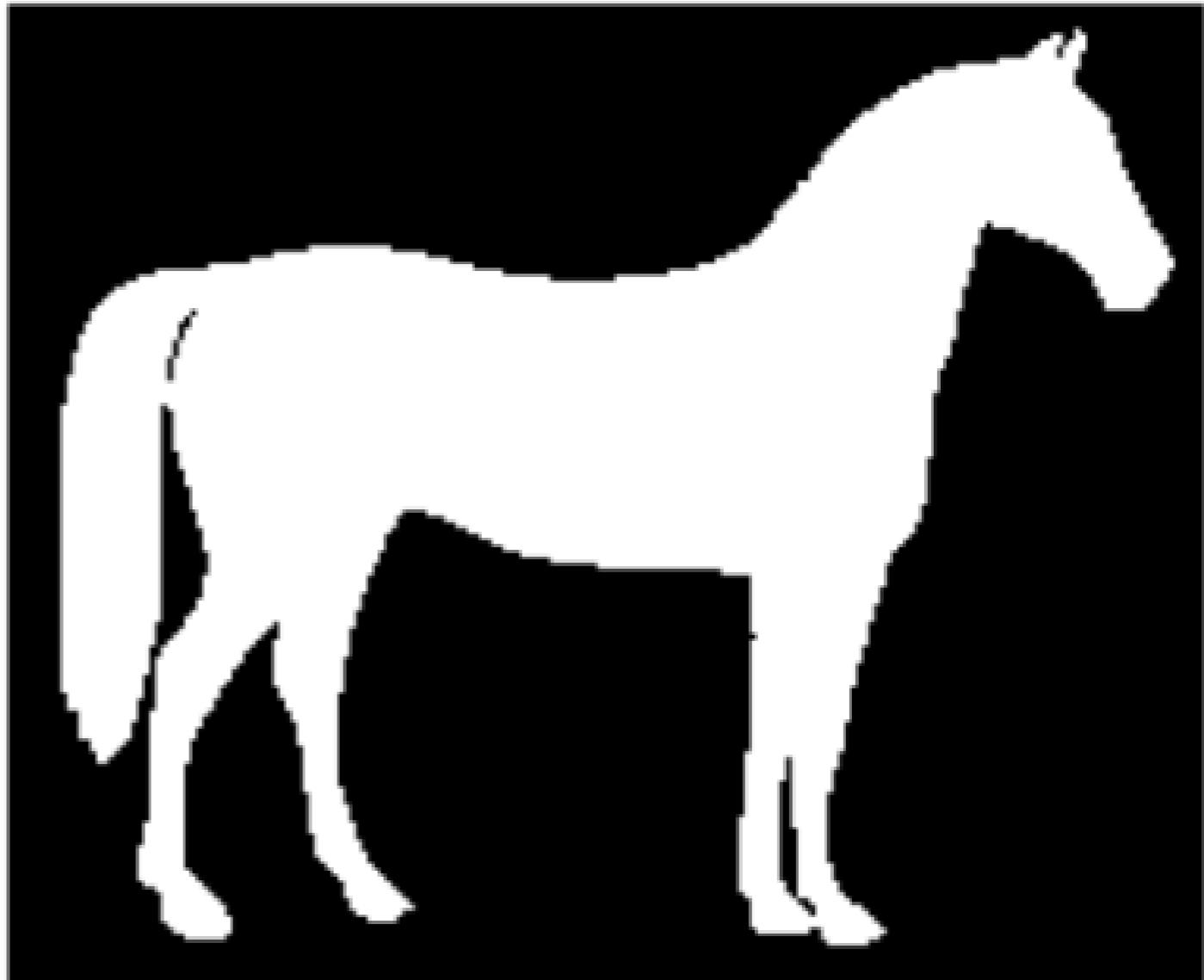
```
from skimage import morphology  
  
# Obtain dilated image, using binary dilation  
dilated_image = morphology.binary_dilation(image_horse)  
  
# See results  
plot_comparison(image_horse, dilated_image, 'Erosion')
```

# Dilation in scikit-image

original



Dilation



# Let's practice!

## IMAGE PROCESSING IN PYTHON

A very interesting use of computer vision in real-life solutions is performing Optical Character Recognition (OCR) to distinguish printed or handwritten text characters inside digital images of physical documents. We can improve the handwritten letter so that it's easier to classify.