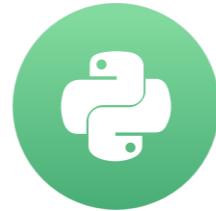


Jump into filtering

IMAGE PROCESSING IN PYTHON

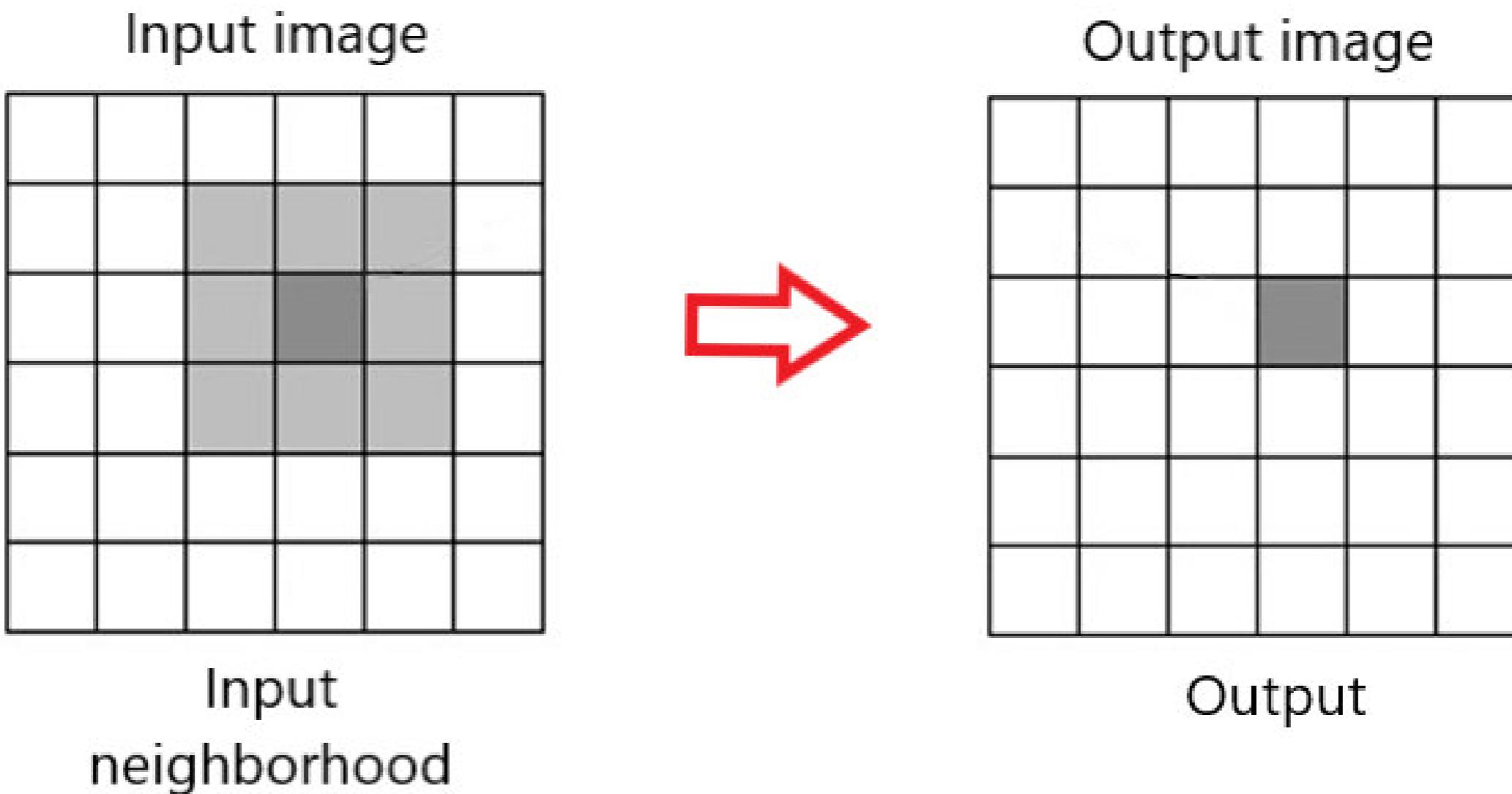


Rebeca Gonzalez
Data Engineer

Filters

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

Neighborhoods

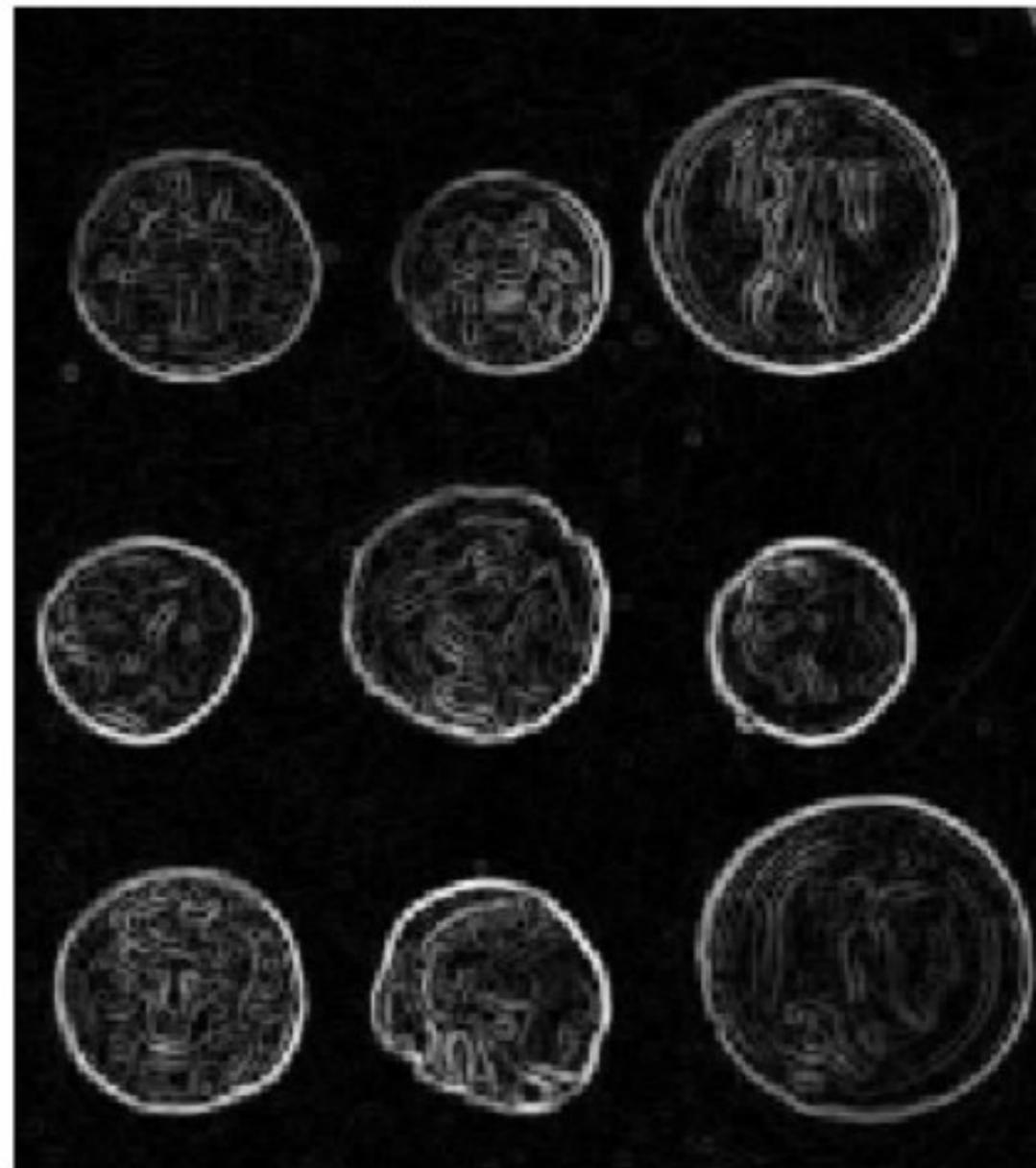


Edge detection

Original



Edges with Sobel

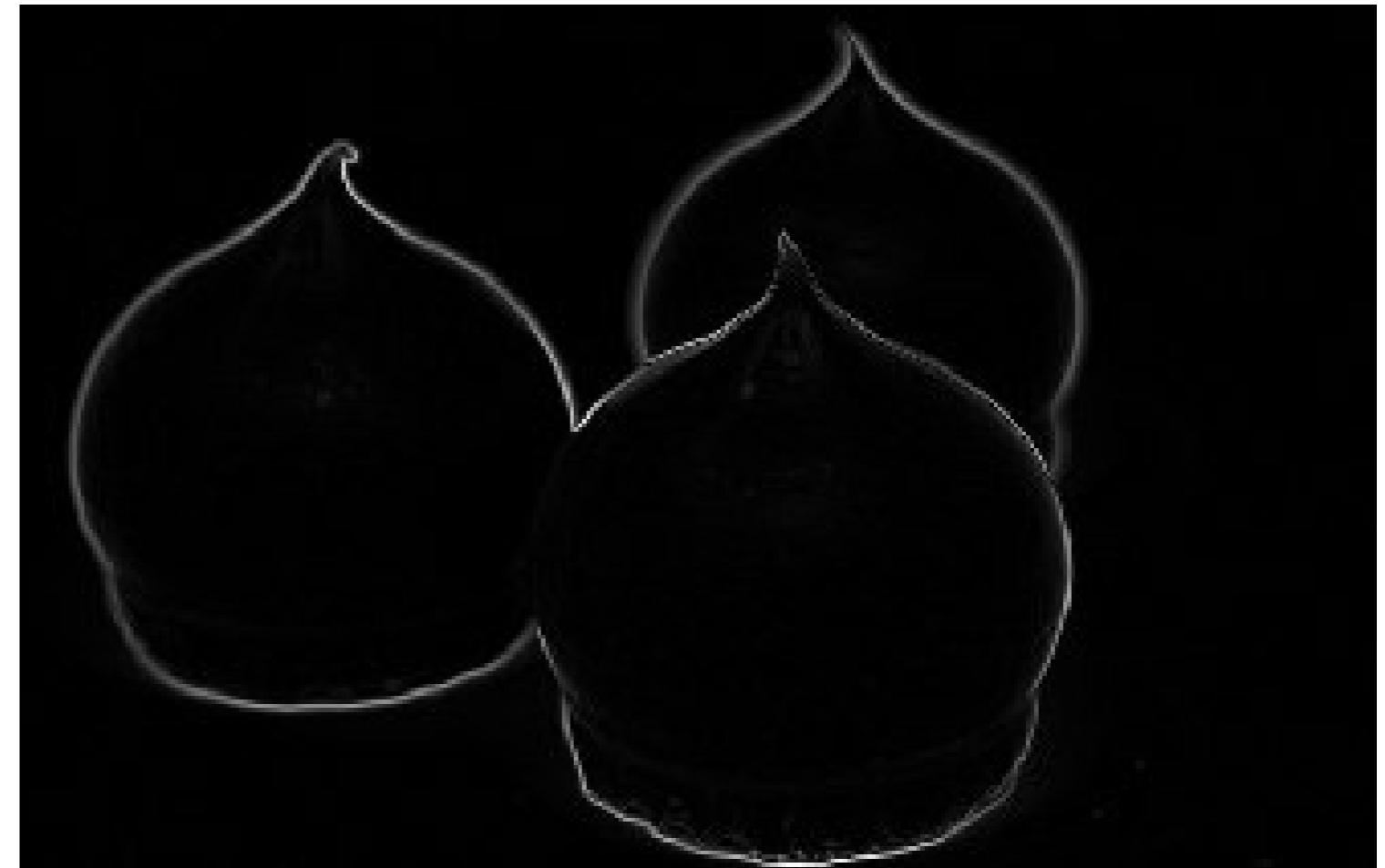


Edge detection

Original chocolate kisses



Edges with Sobel



Edge detection

Sobel

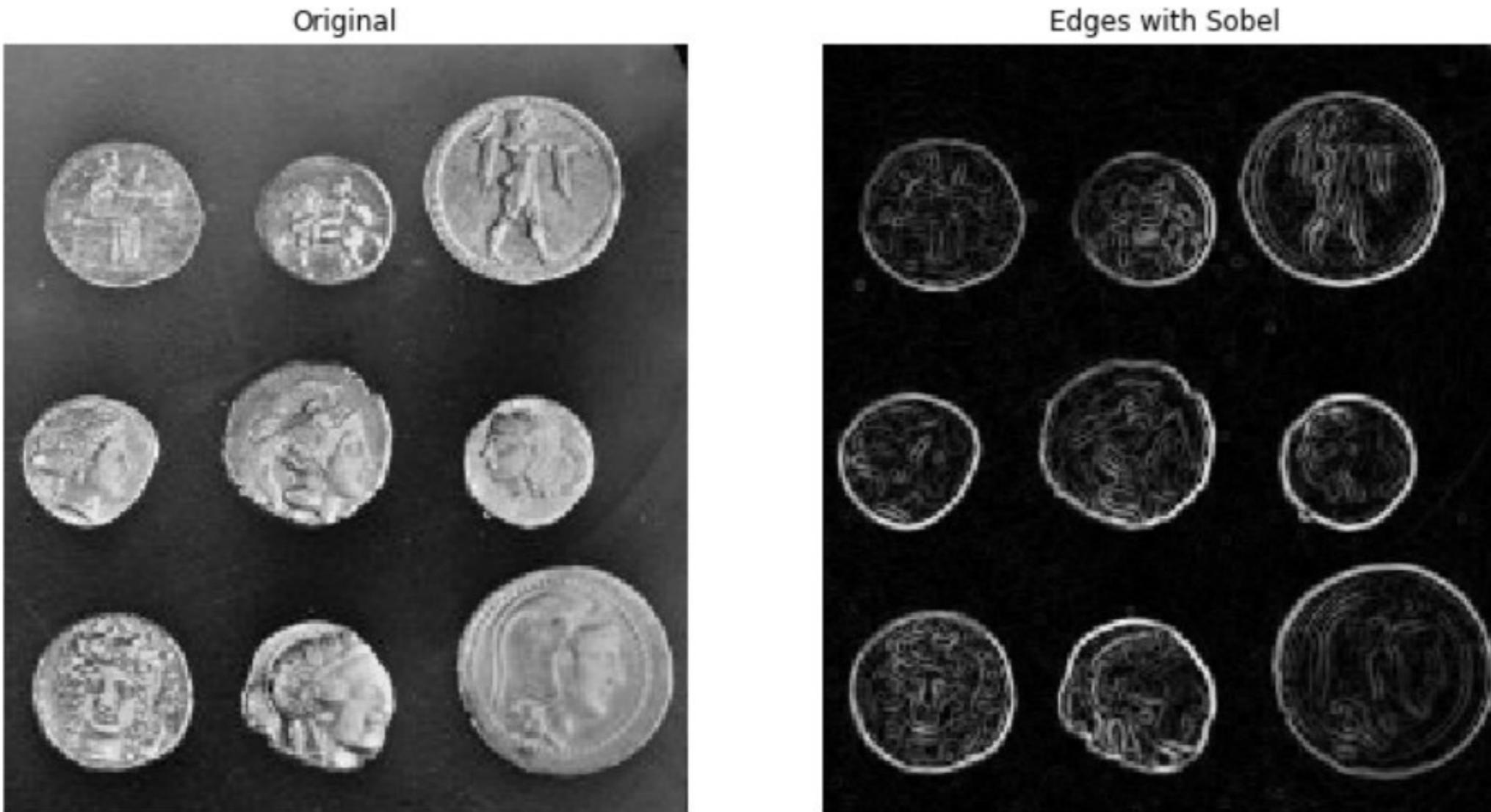
```
# Import module and function
from skimage.filters import sobel

# 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

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)

# 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

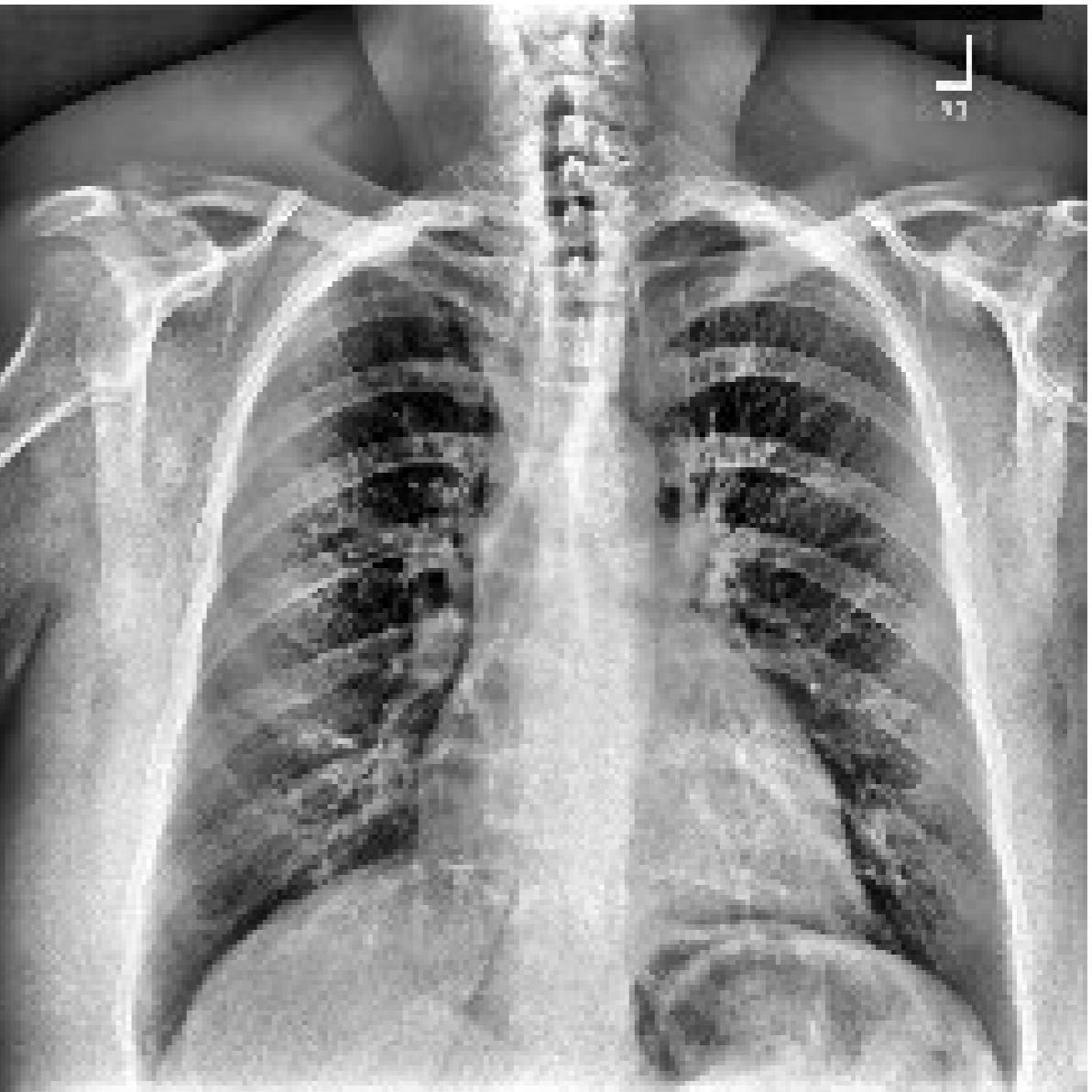
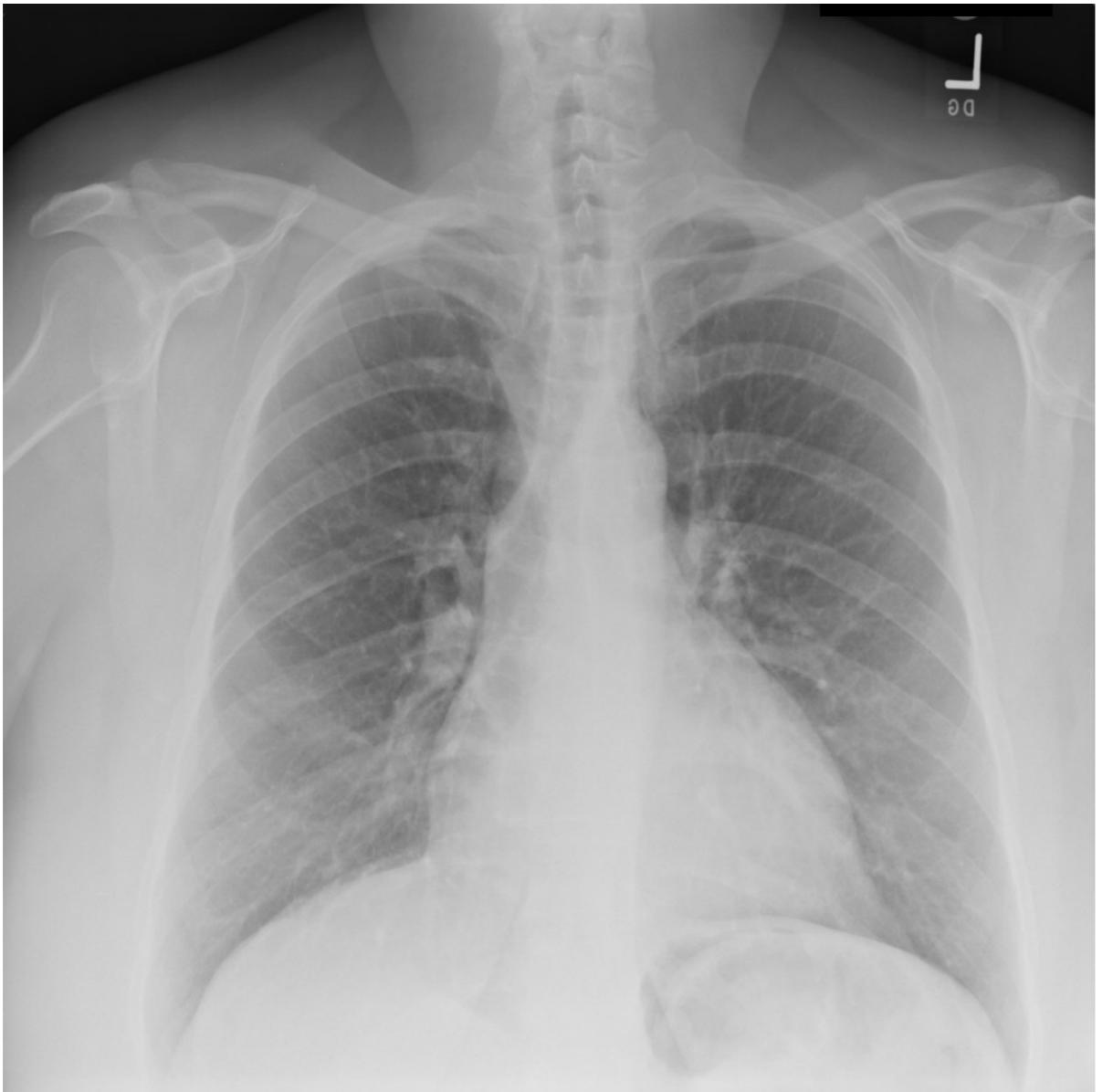
Contrast enhancement

IMAGE PROCESSING IN PYTHON



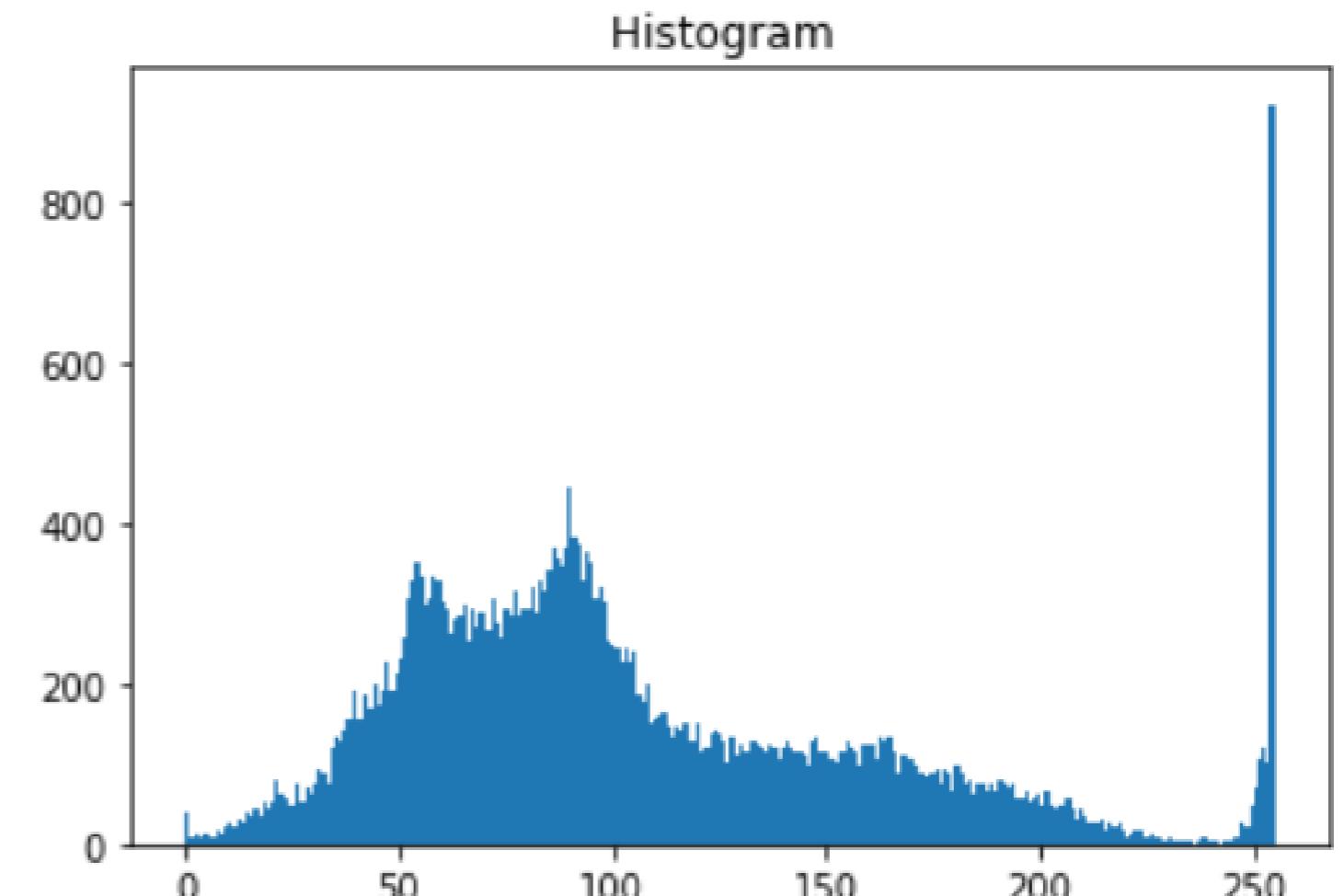
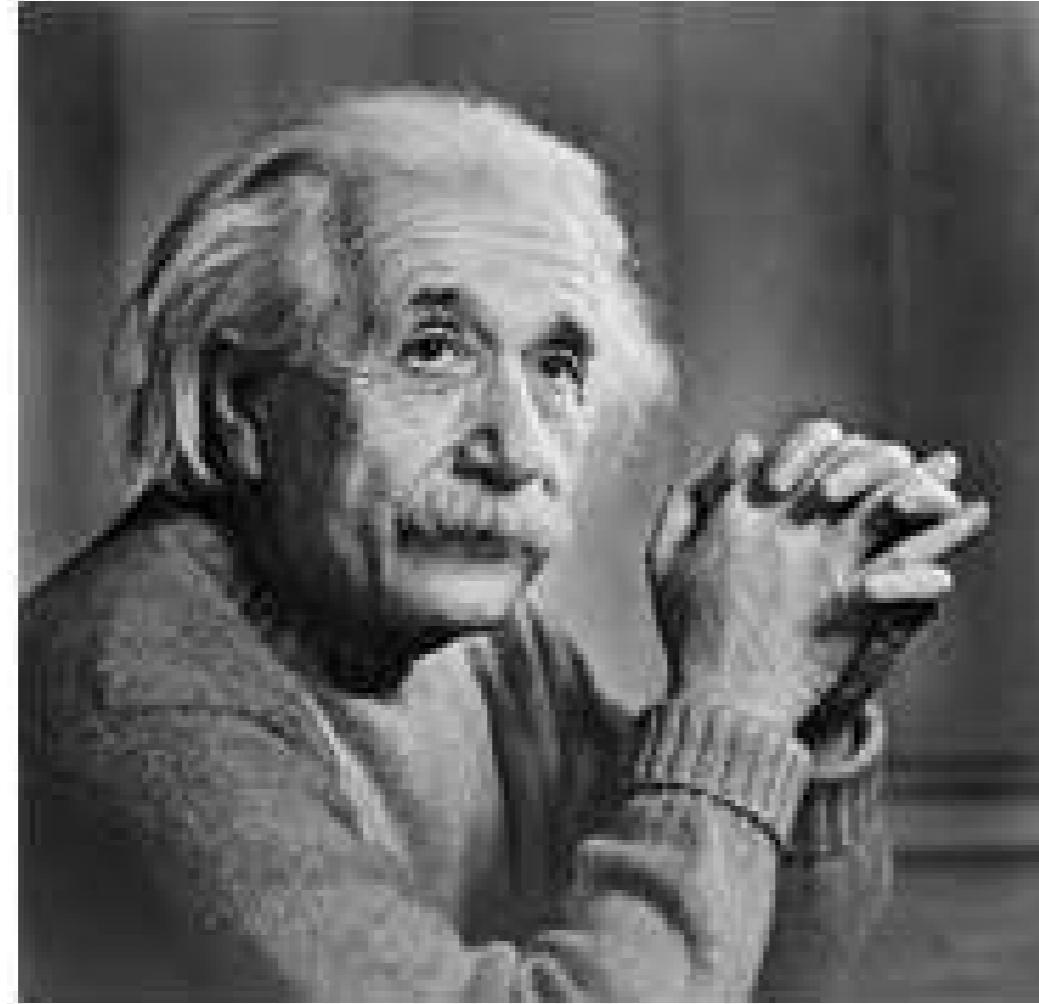
Rebeca Gonzalez
Data engineer

Contrast enhancement



Contrast

Histograms for contrast enhancement

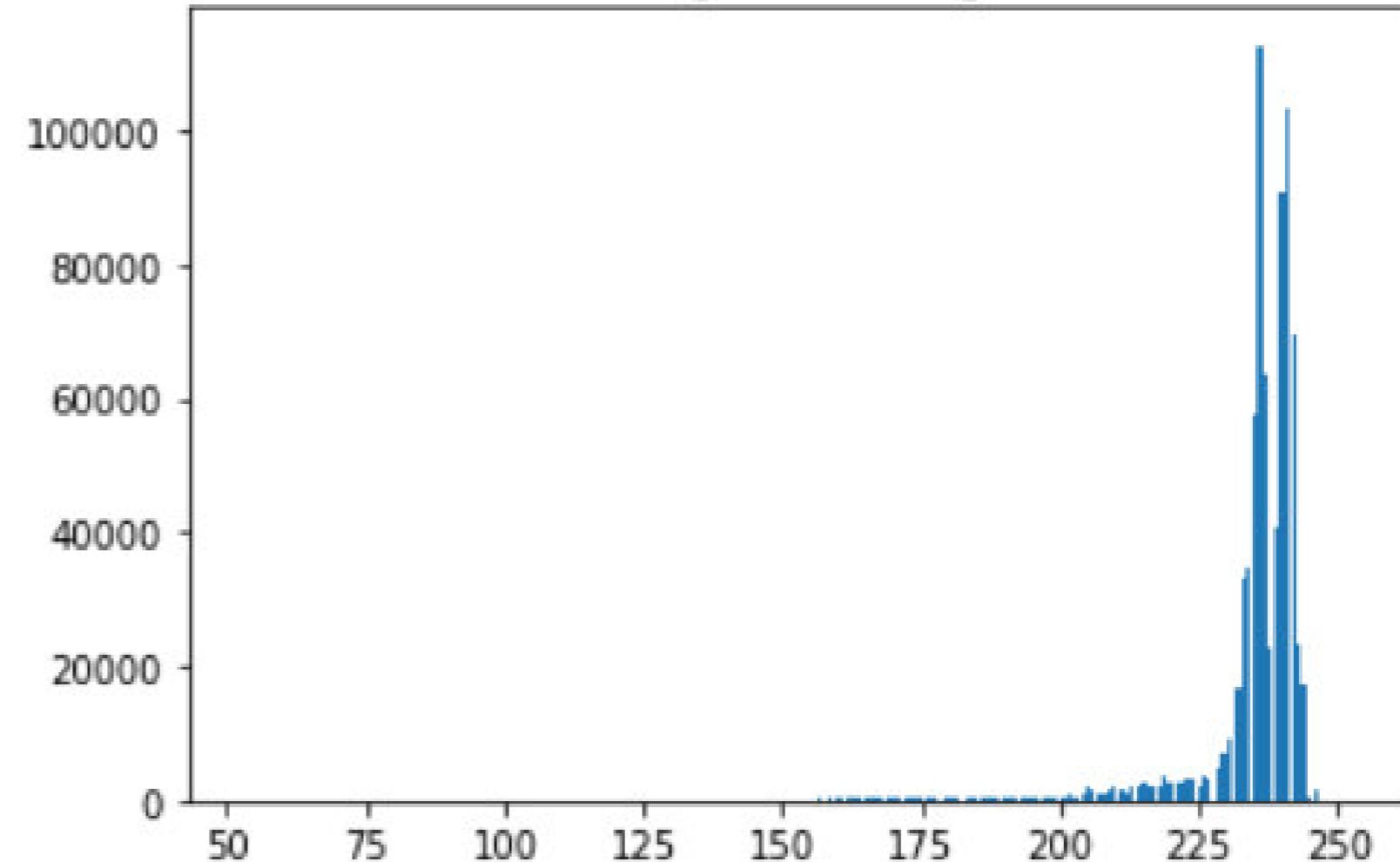


Contrast

Low contrast image - light



Histogram of image



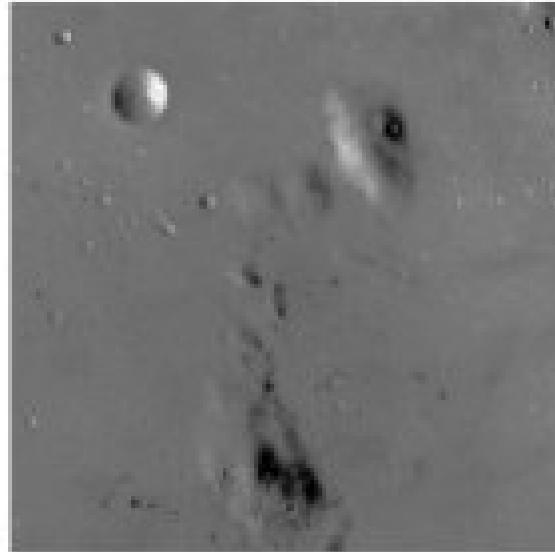
Enhance contrast

- 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)

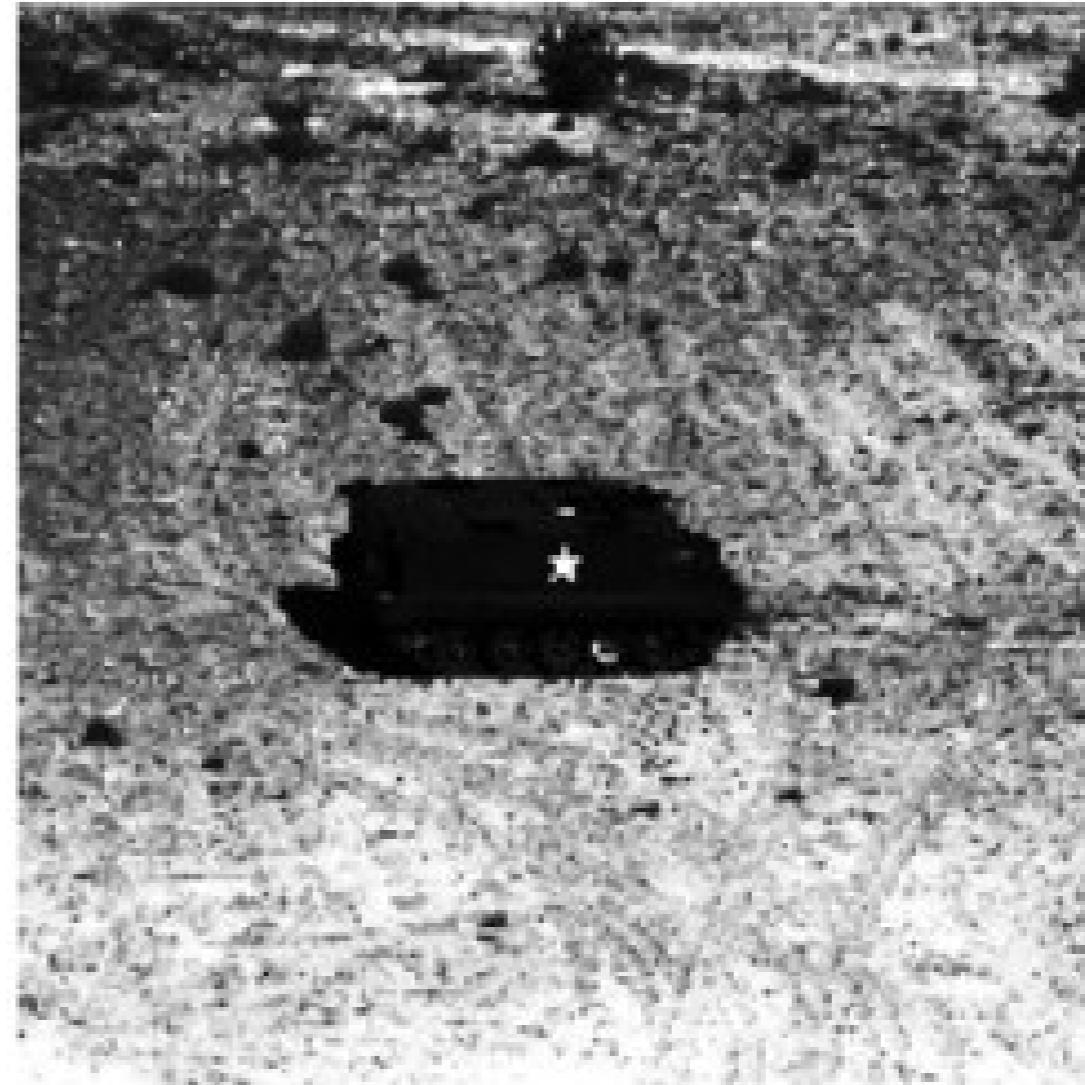
# Show original and result
show_image(image, 'Original')
show_image(image_eq, 'Histogram equalized')
```

Histogram equalization

Original



Histogram Equalization



Adaptive Equalization

- Contrast Limited Adaptive Histogram Equalization

Original



Adaptive Equalization



Contrastive Limited Adaptive Equalization

Original



Histogram Equalization Adaptive Equalization



CLAHE in scikit-image

```
from skimage import exposure

# Apply adaptive Equalization
image_adapteq = exposure.equalize_adapthist(image, clip_limit=0.03)

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

CLAHE in scikit-image

Original



Adaptive Equalization

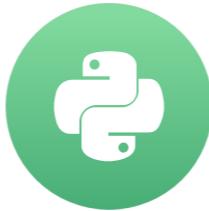


Let's practice!

IMAGE PROCESSING IN PYTHON

Transformations

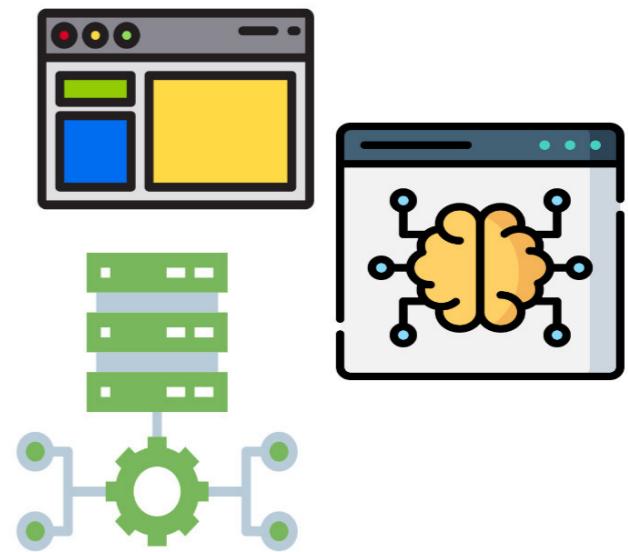
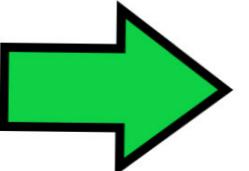
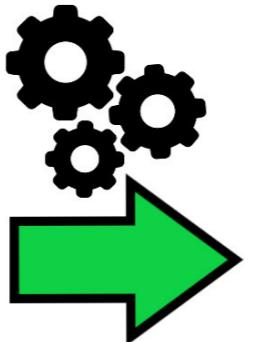
IMAGE PROCESSING IN PYTHON



Rebeca Gonzalez
Data Engineer

Why transform images?

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

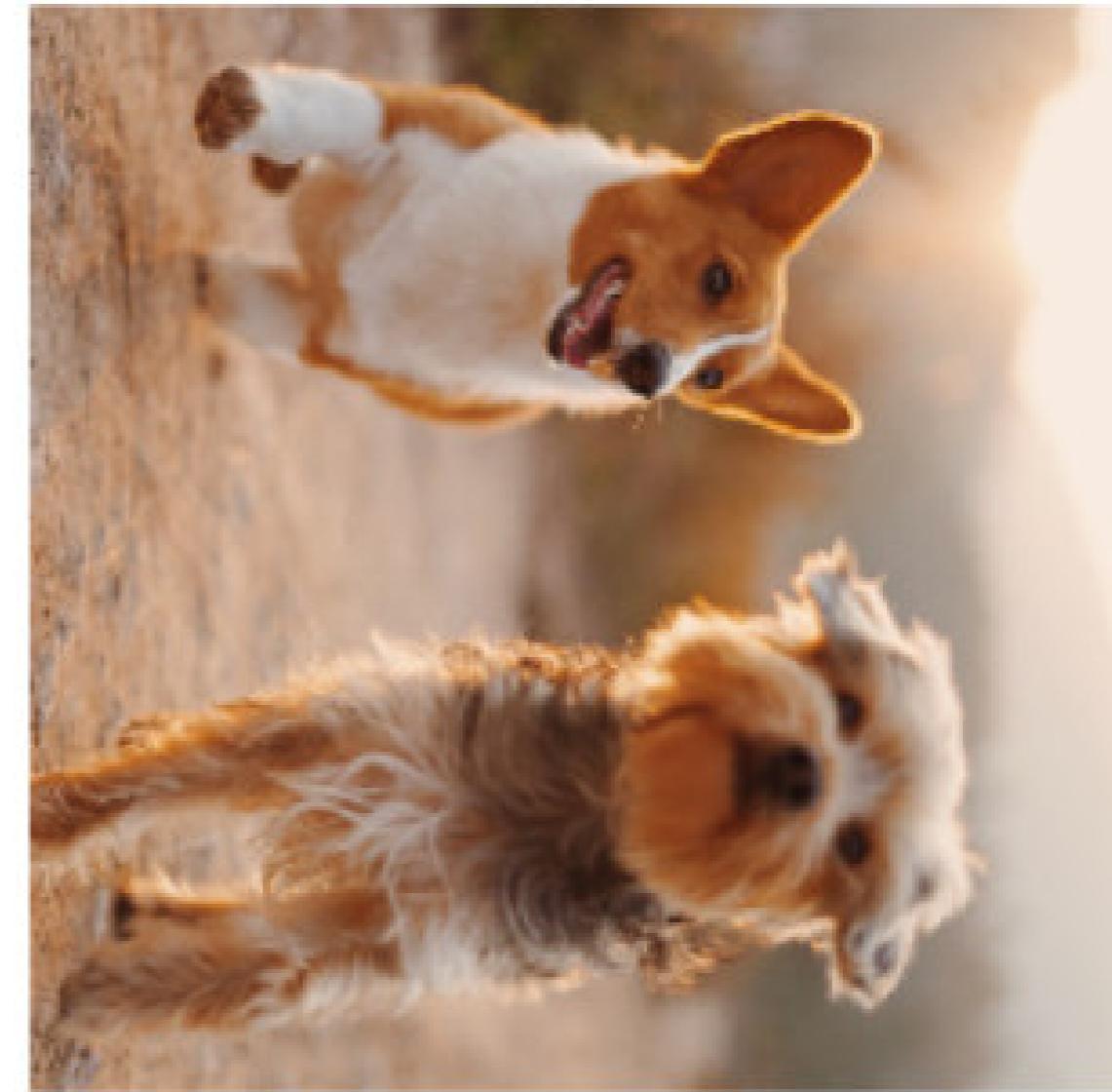


Rotating

Original

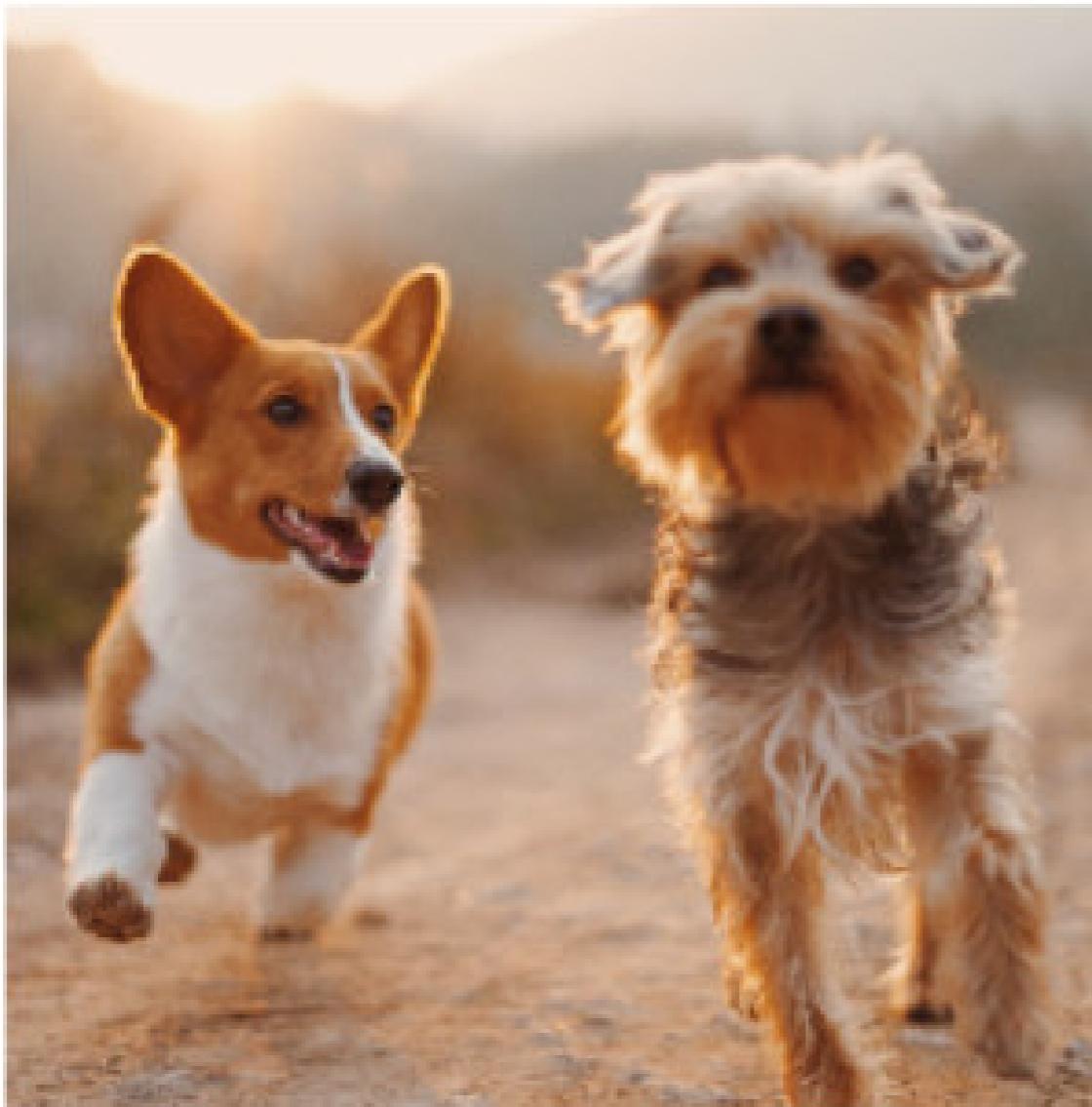


Rotated 90 degrees clockwise

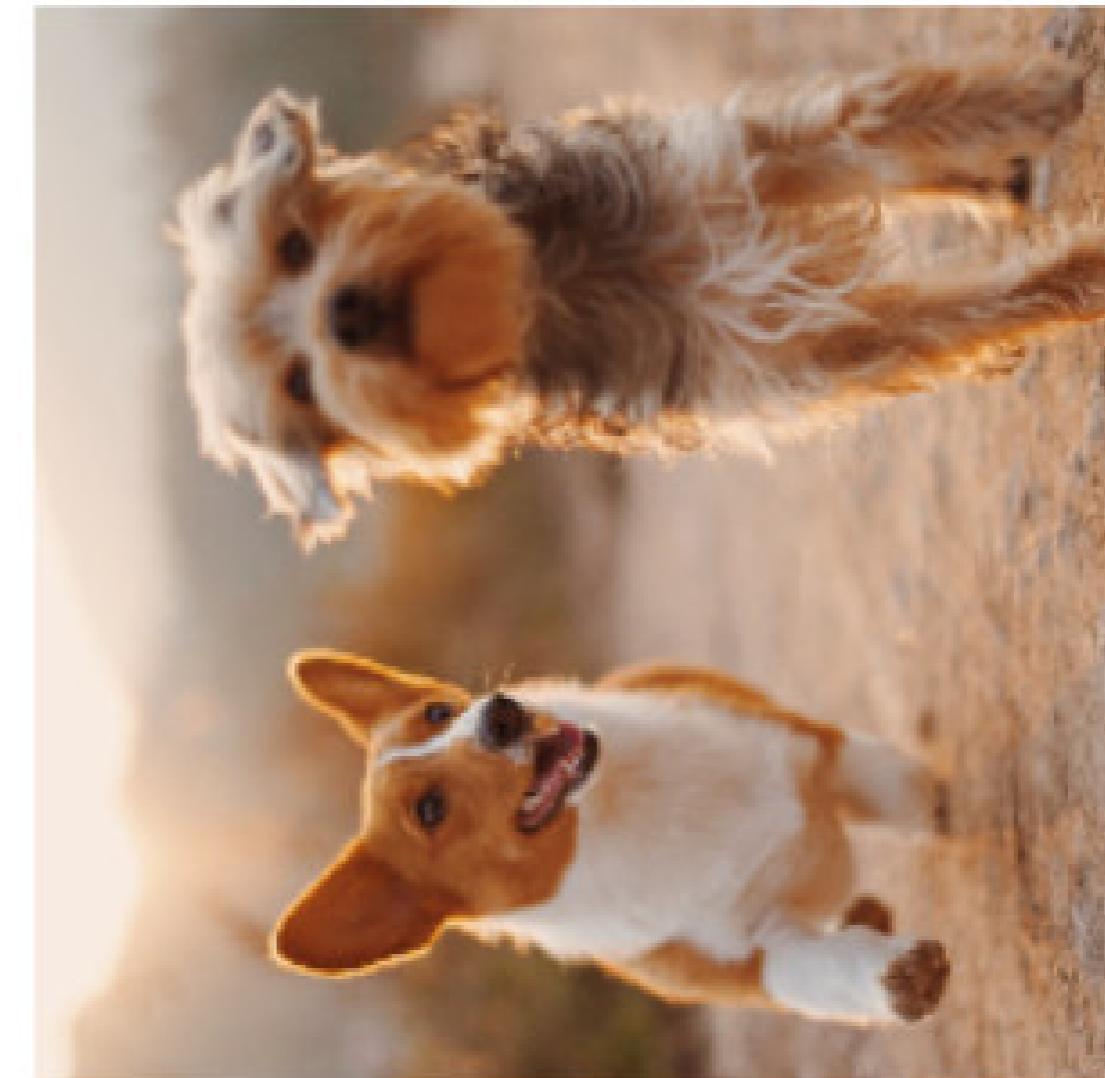


Rotating

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')
```

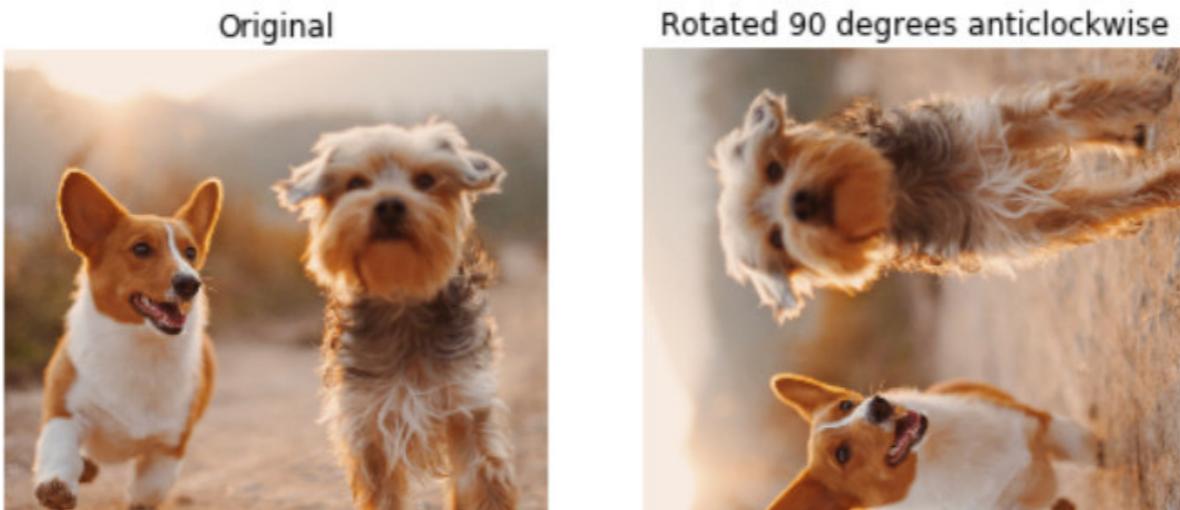


Rotating anticlockwise

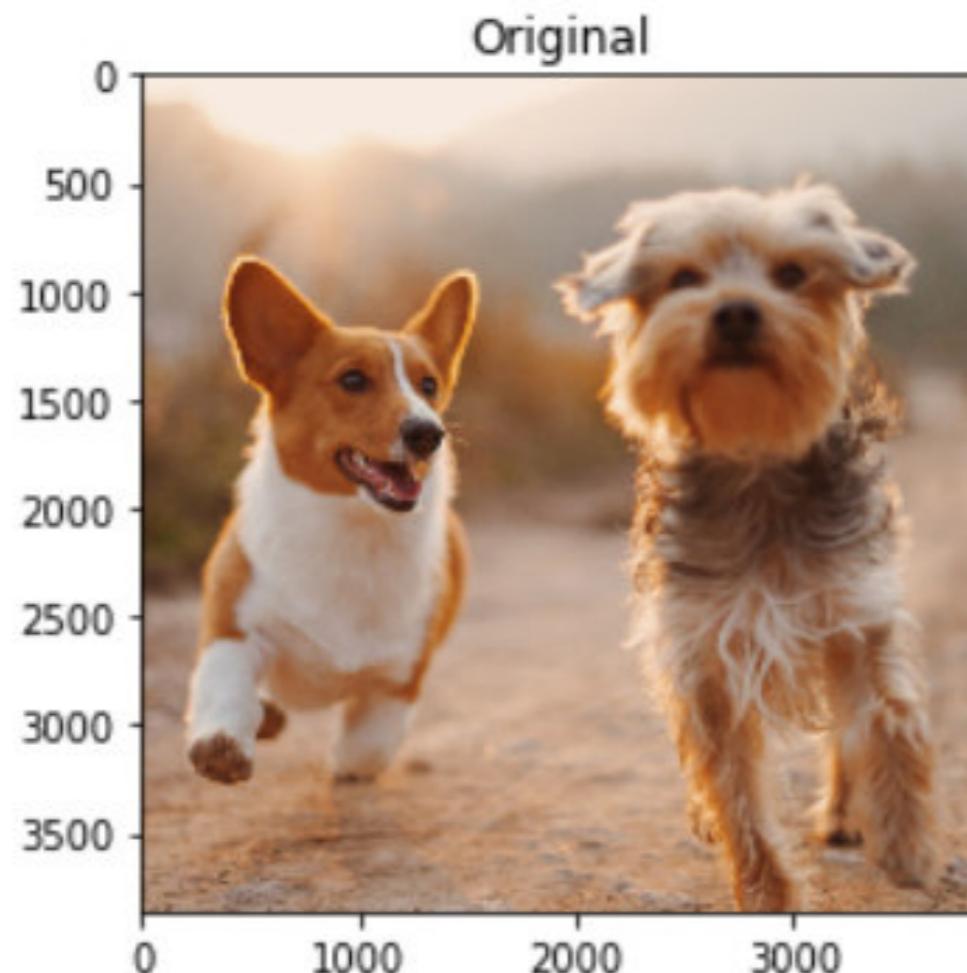
```
from skimage.transform import rotate

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

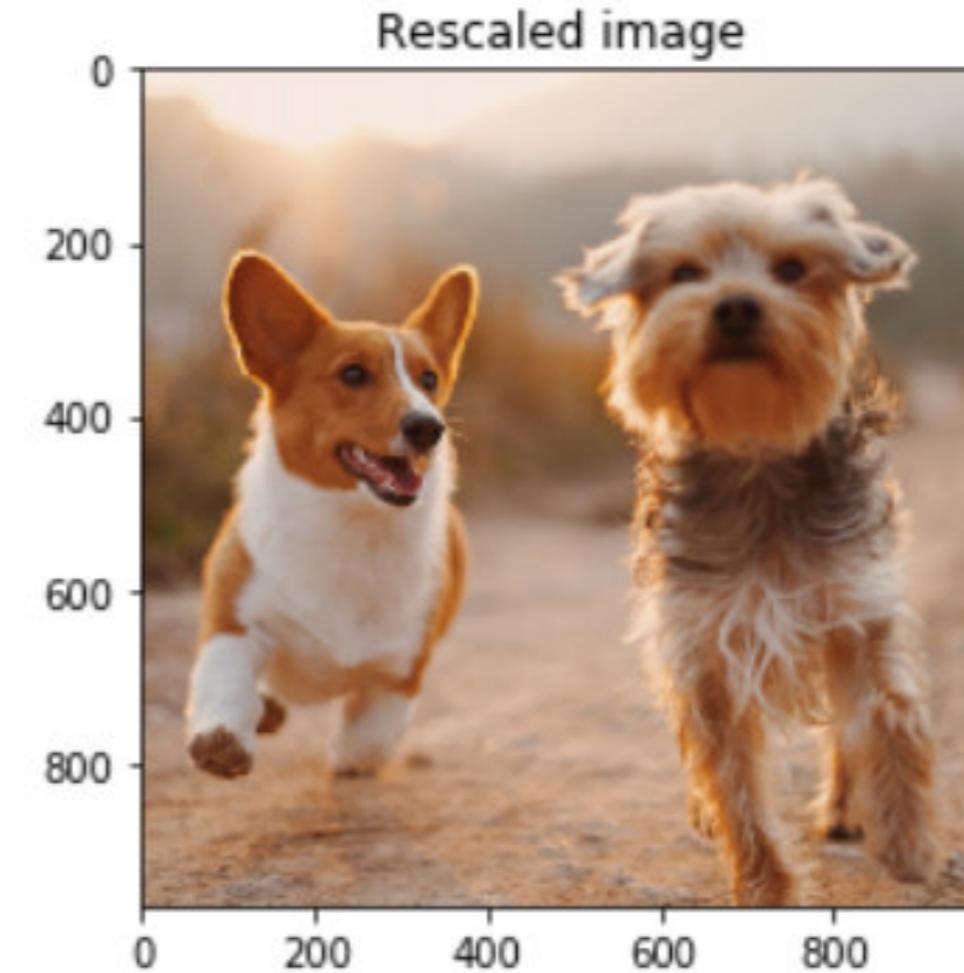
show_image(image, 'Original')
show_image(image_rotated, 'Rotated 90 degrees anticlockwise')
```



Rescaling



Original size



Rescaled size

Rescaling

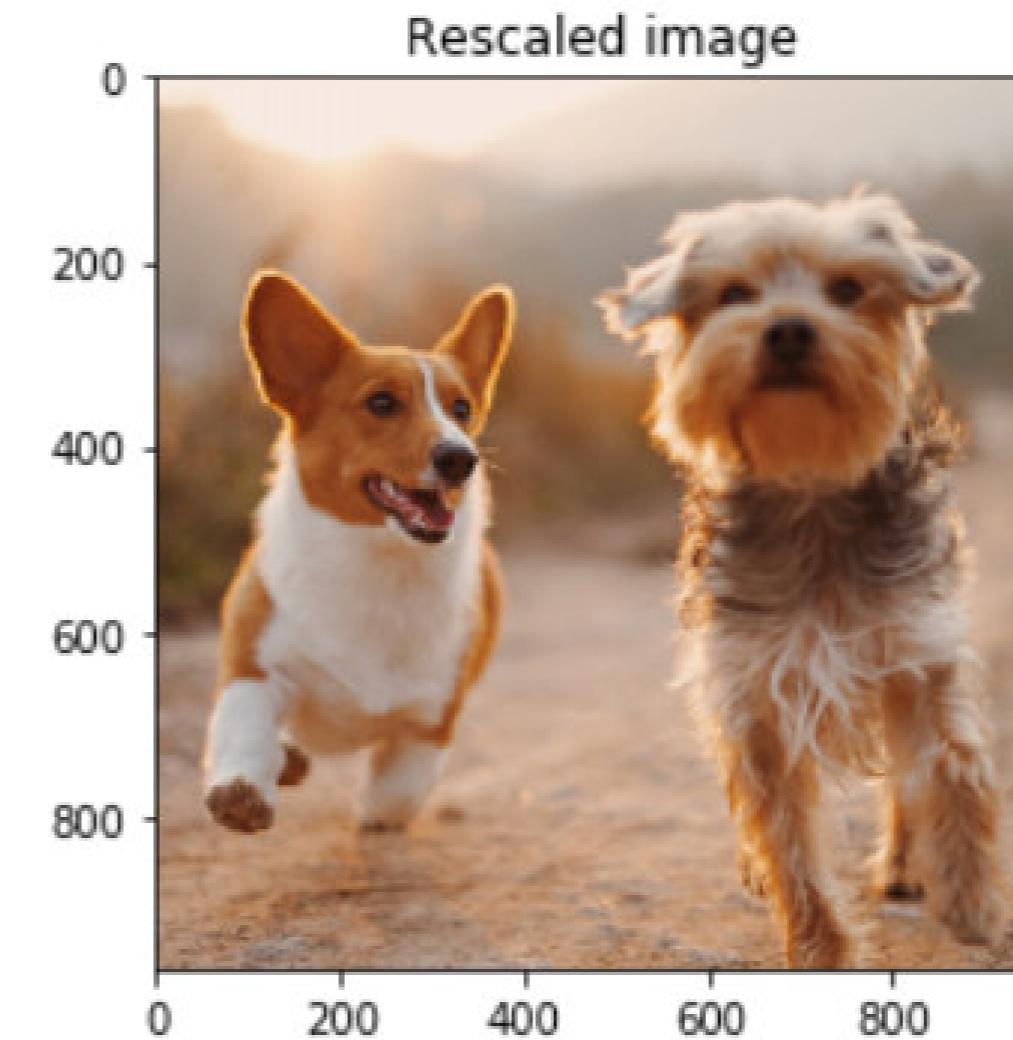
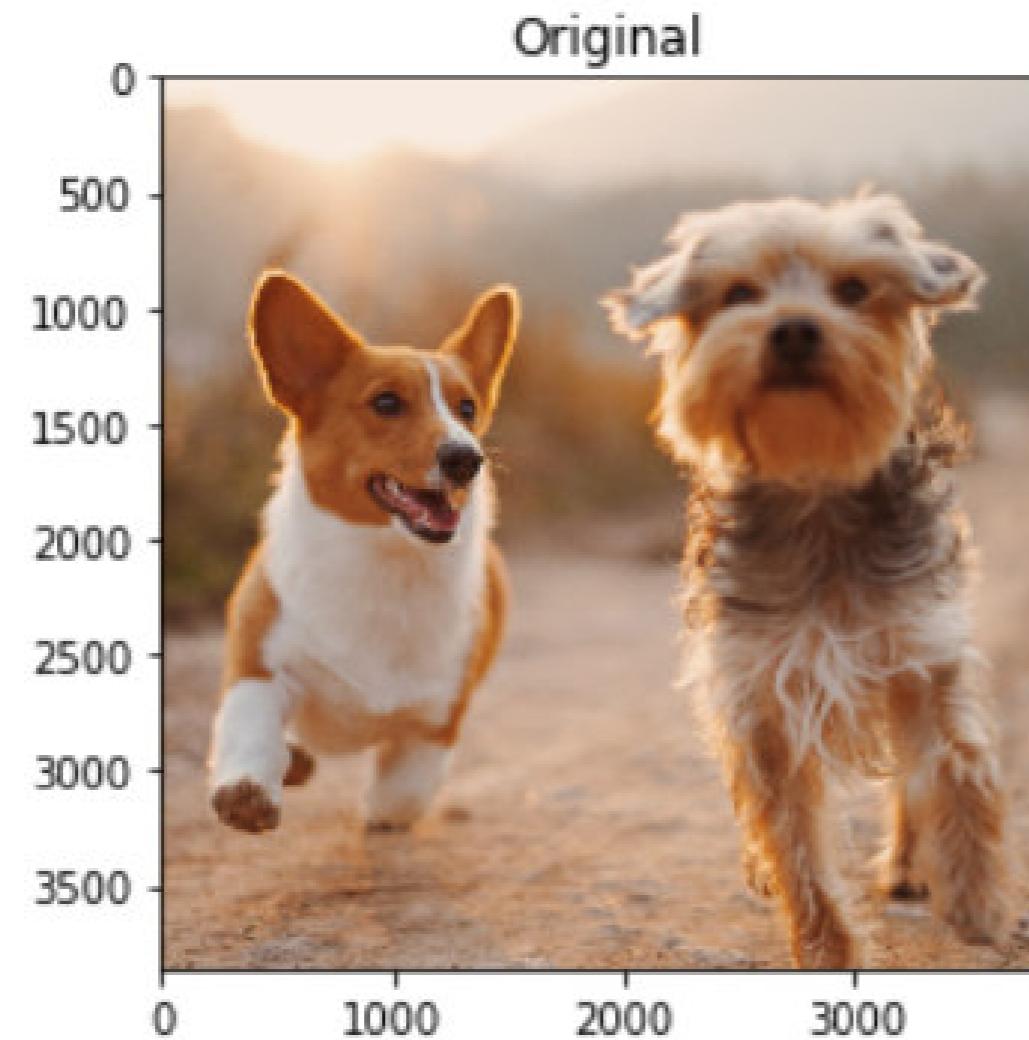
Downgrading

```
from skimage.transform import rescale

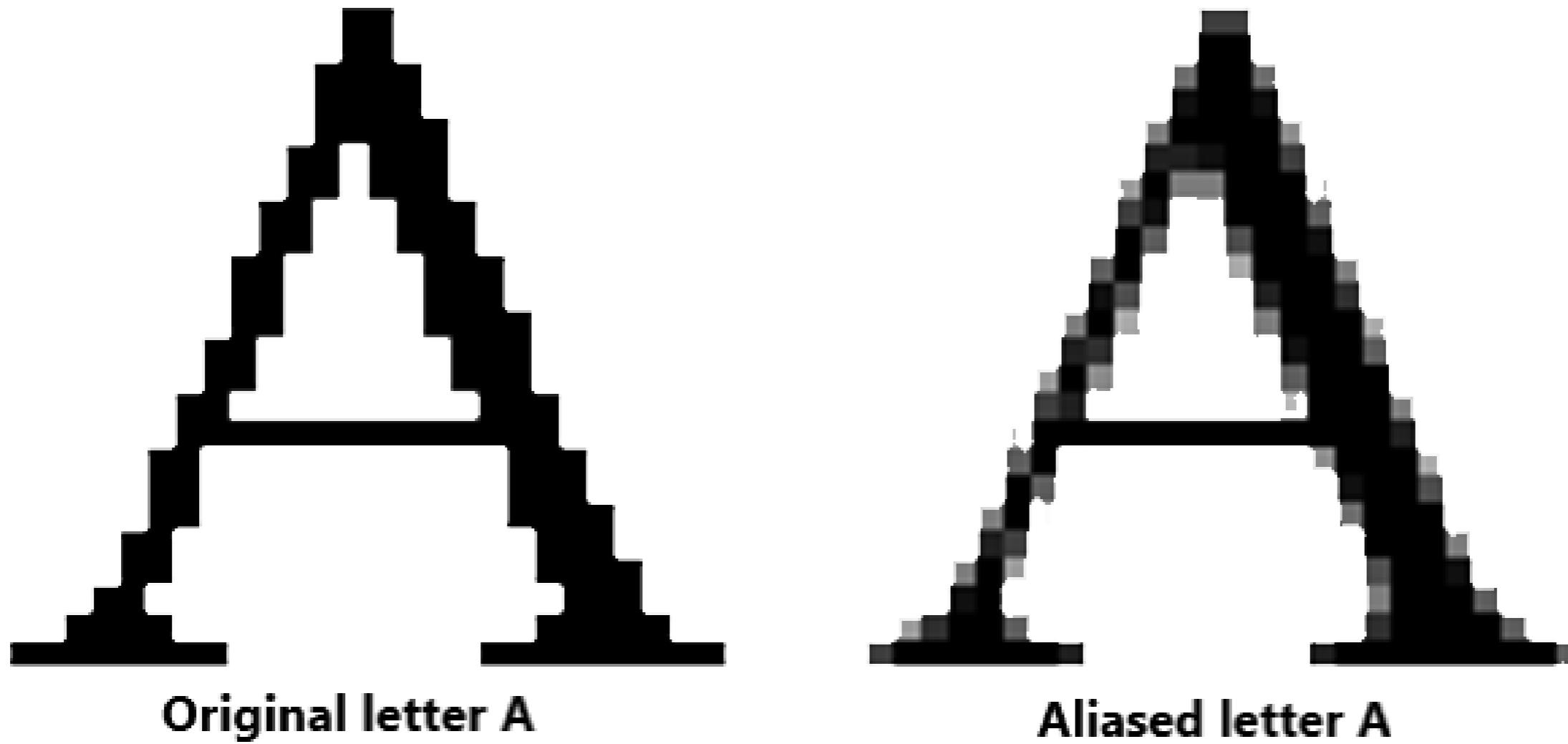
# Rescale the image to be 4 times smaller
image_rescaled = rescale(image, 1/4, anti_aliasing=True, multichannel=True)

show_image(image, 'Original image')
show_image(image_rescaled, 'Rescaled image')
```

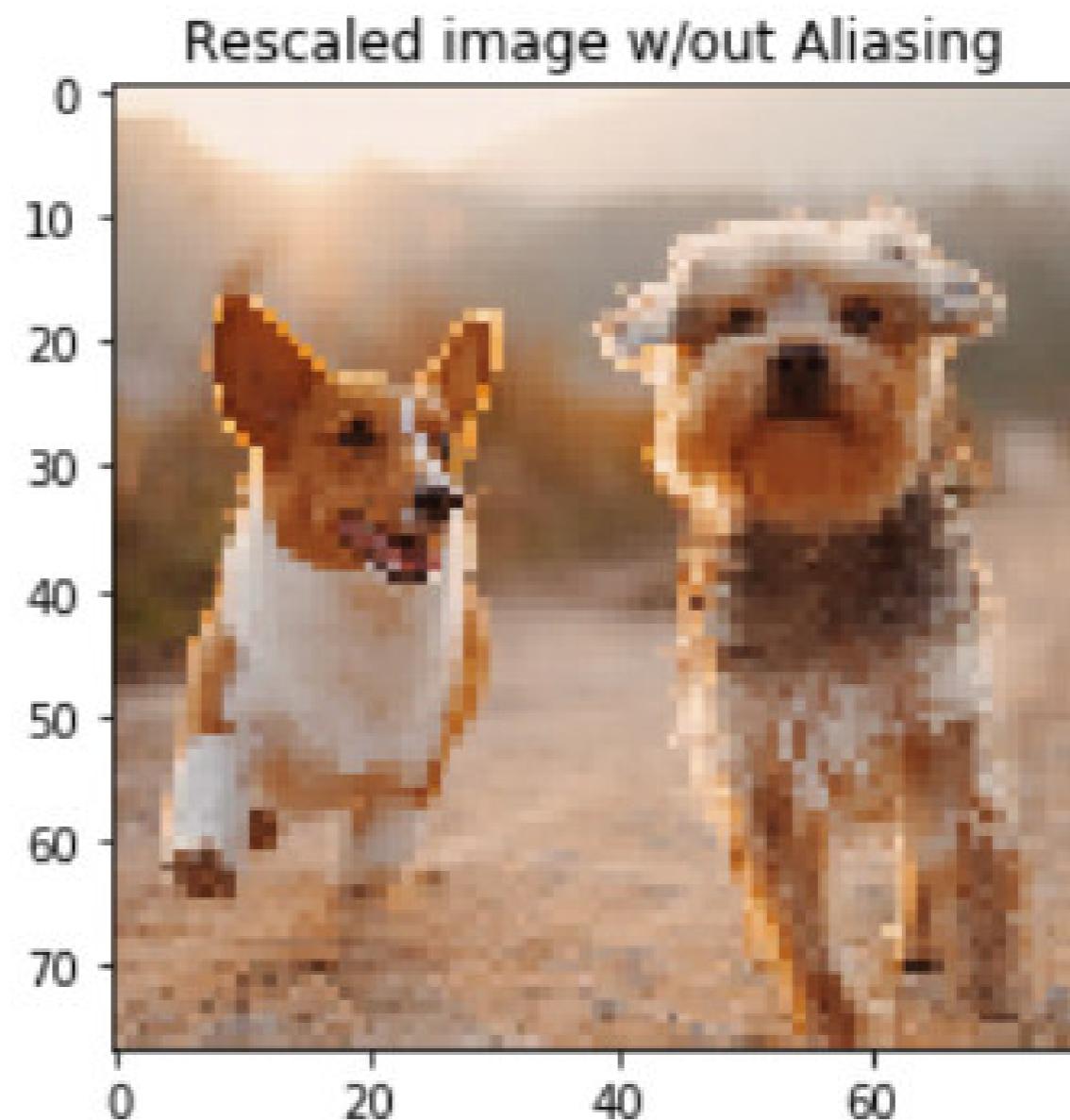
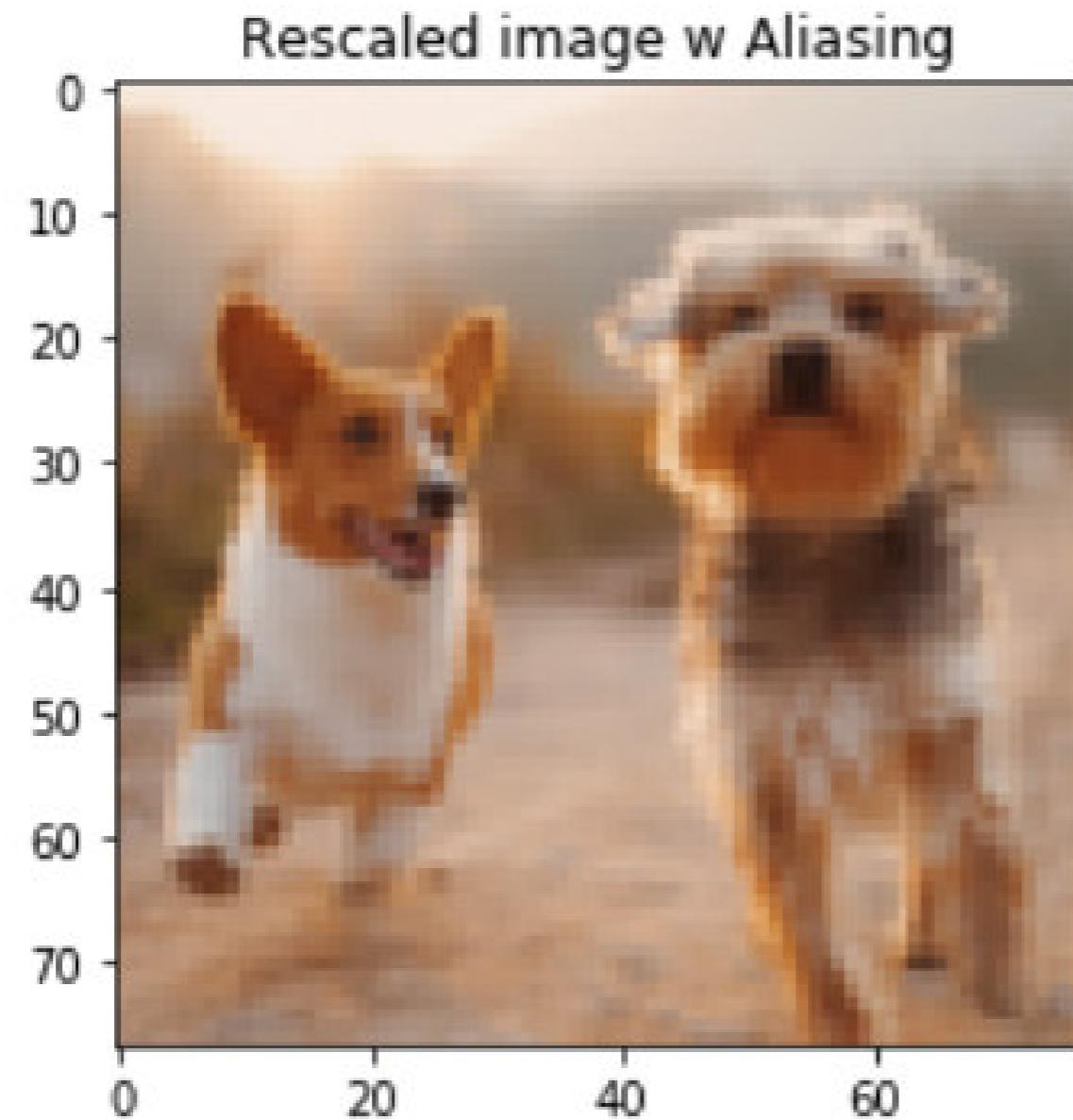
Rescaling



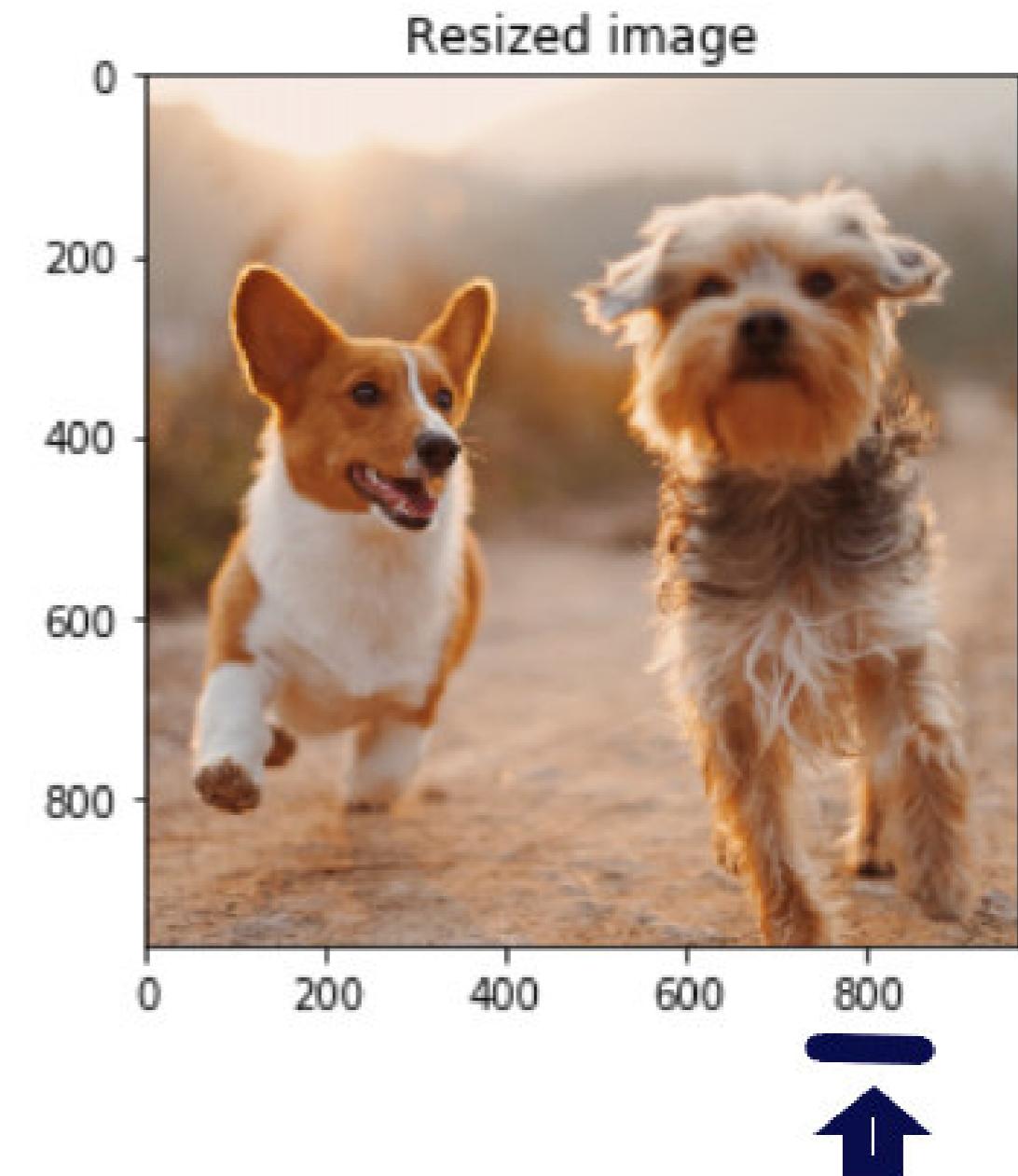
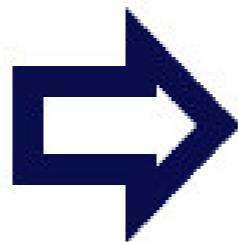
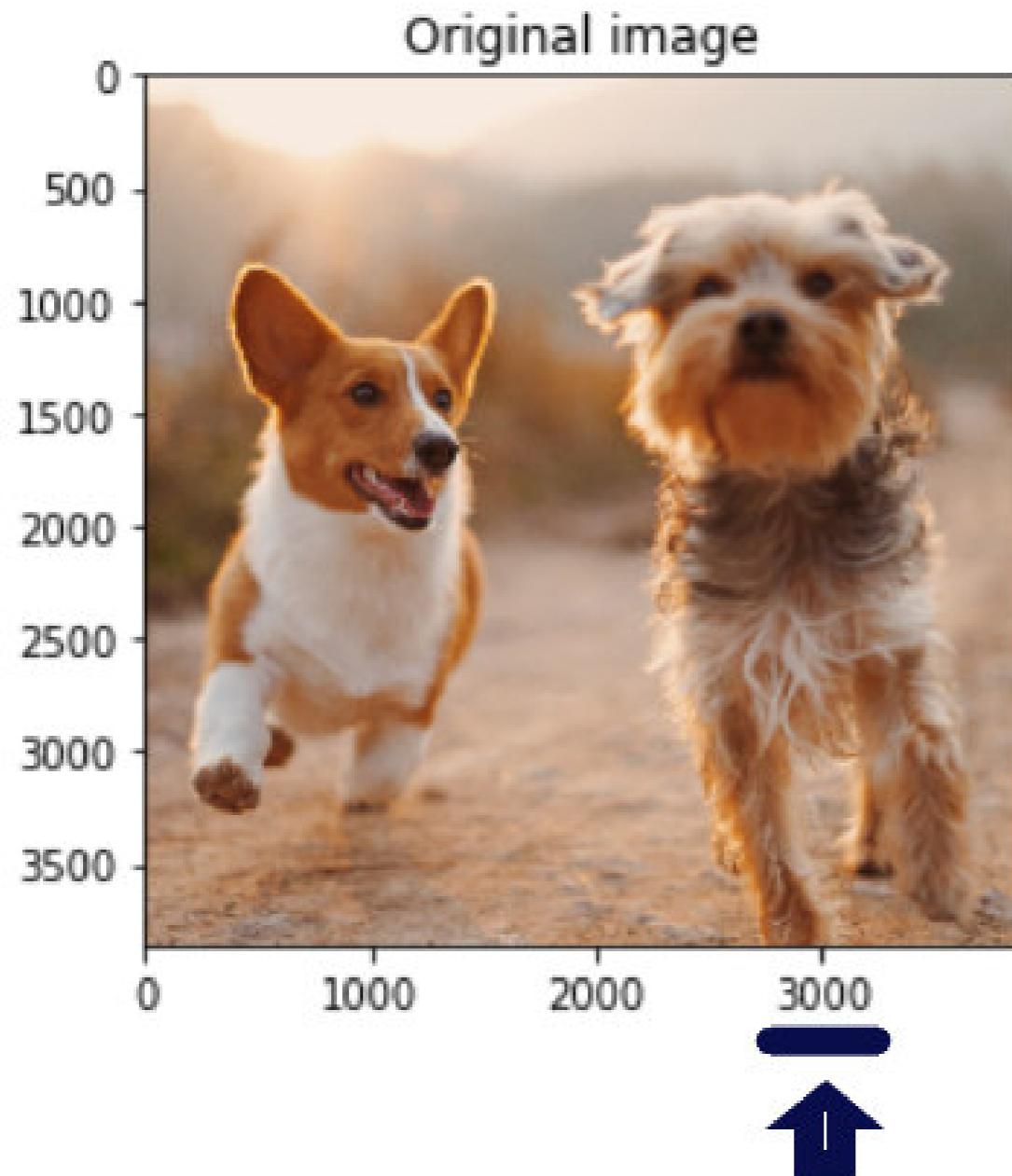
Aliasing in digital images



Aliasing in digital images



Resizing



Resizing

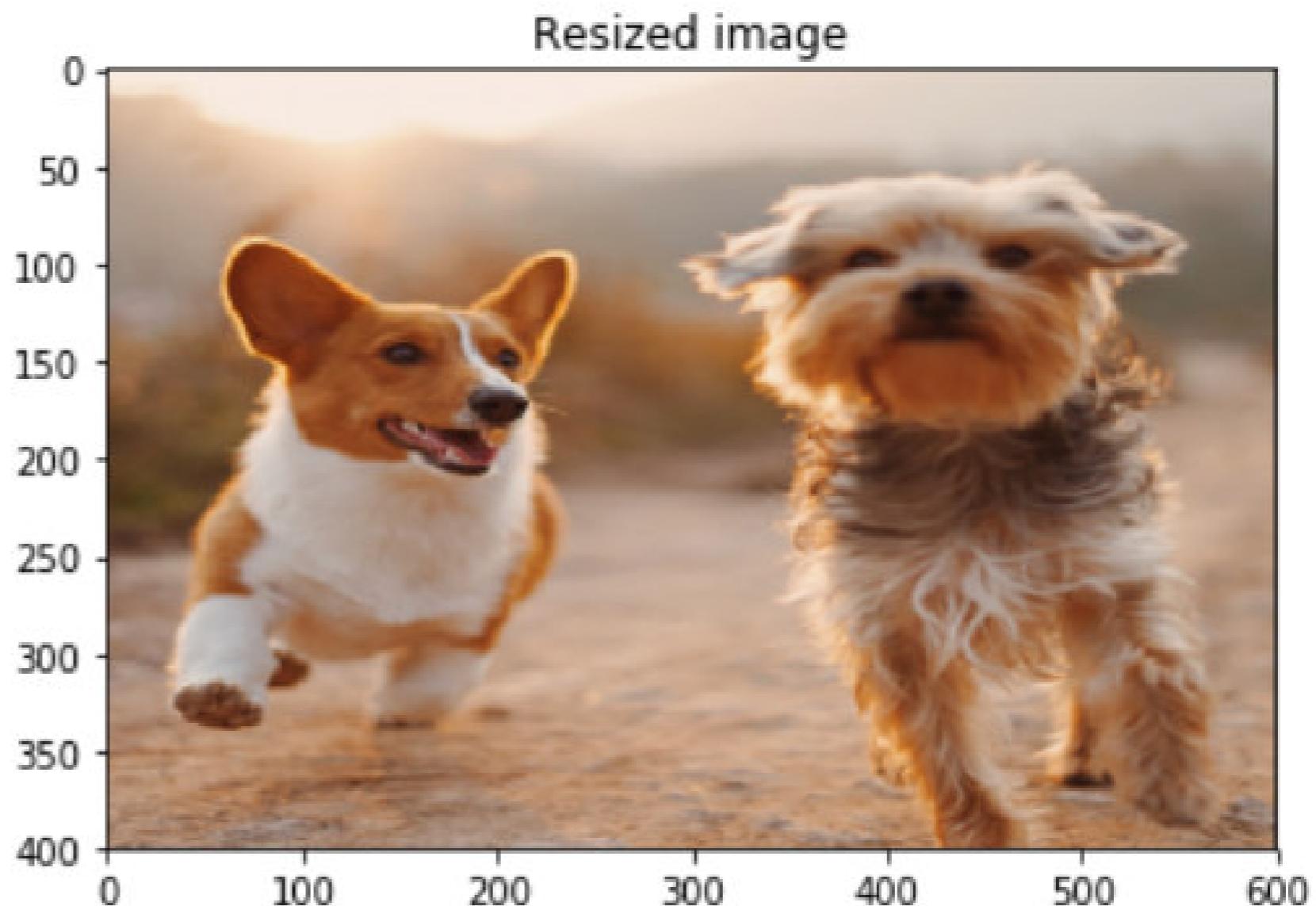
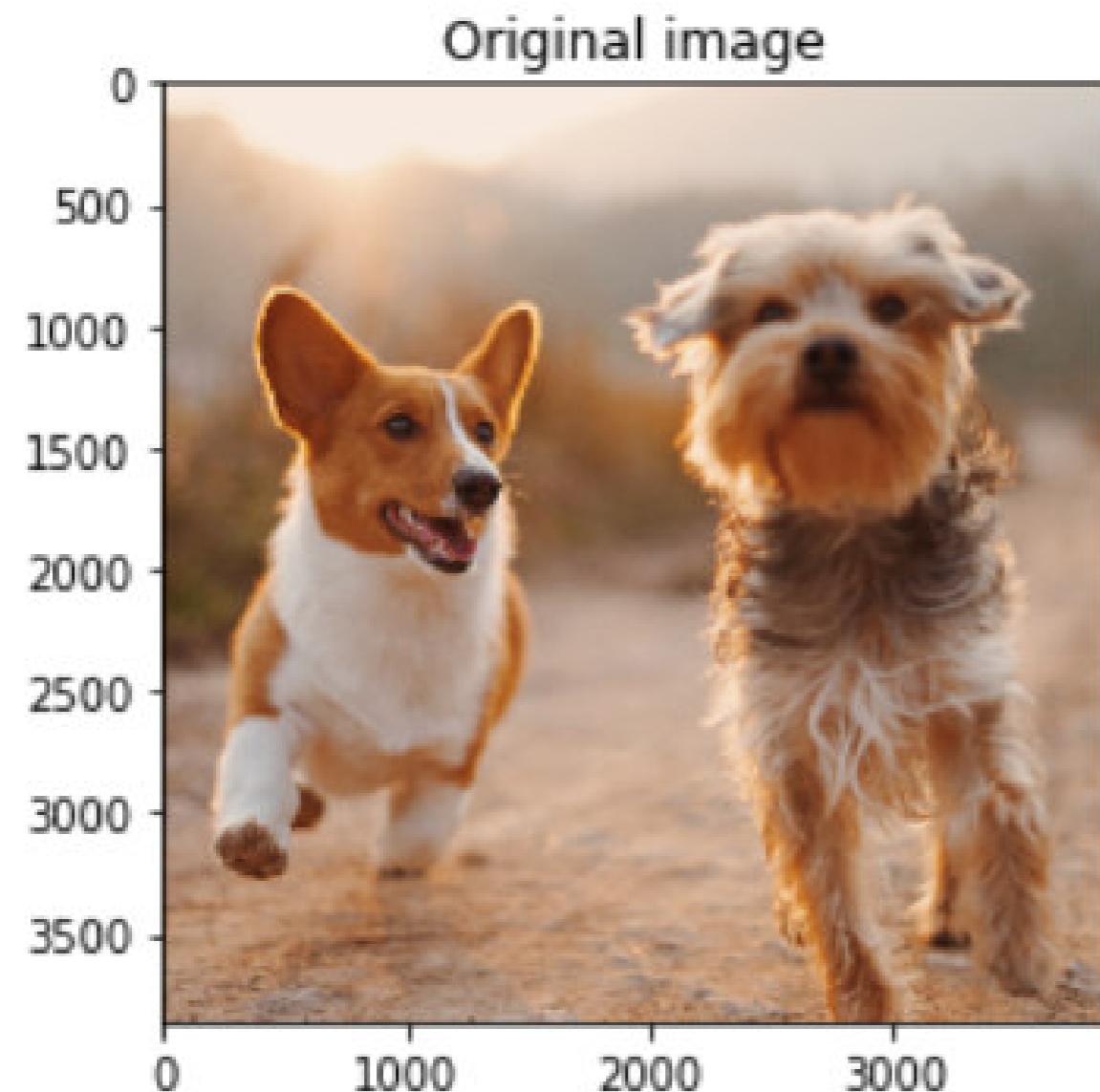
```
from skimage.transform import resize

# 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

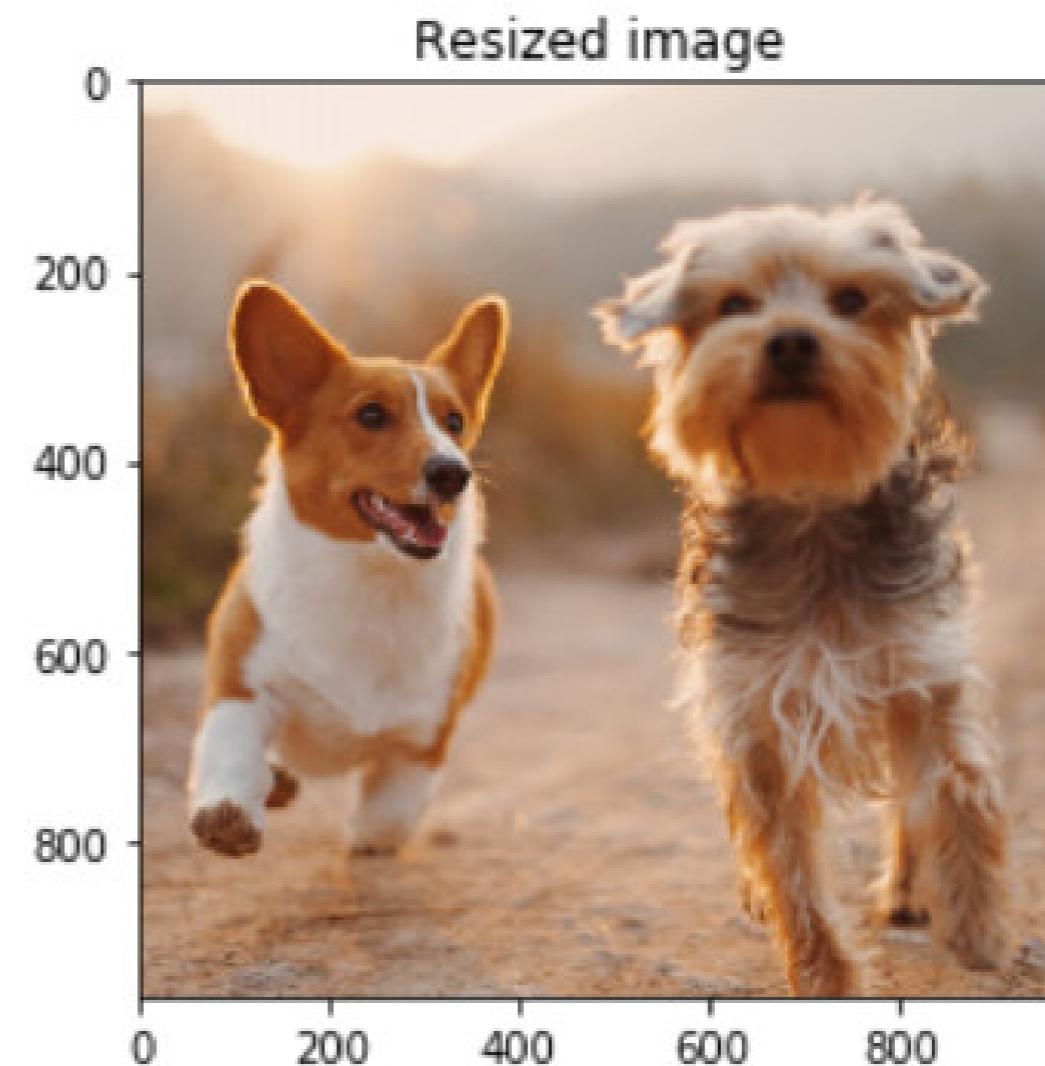
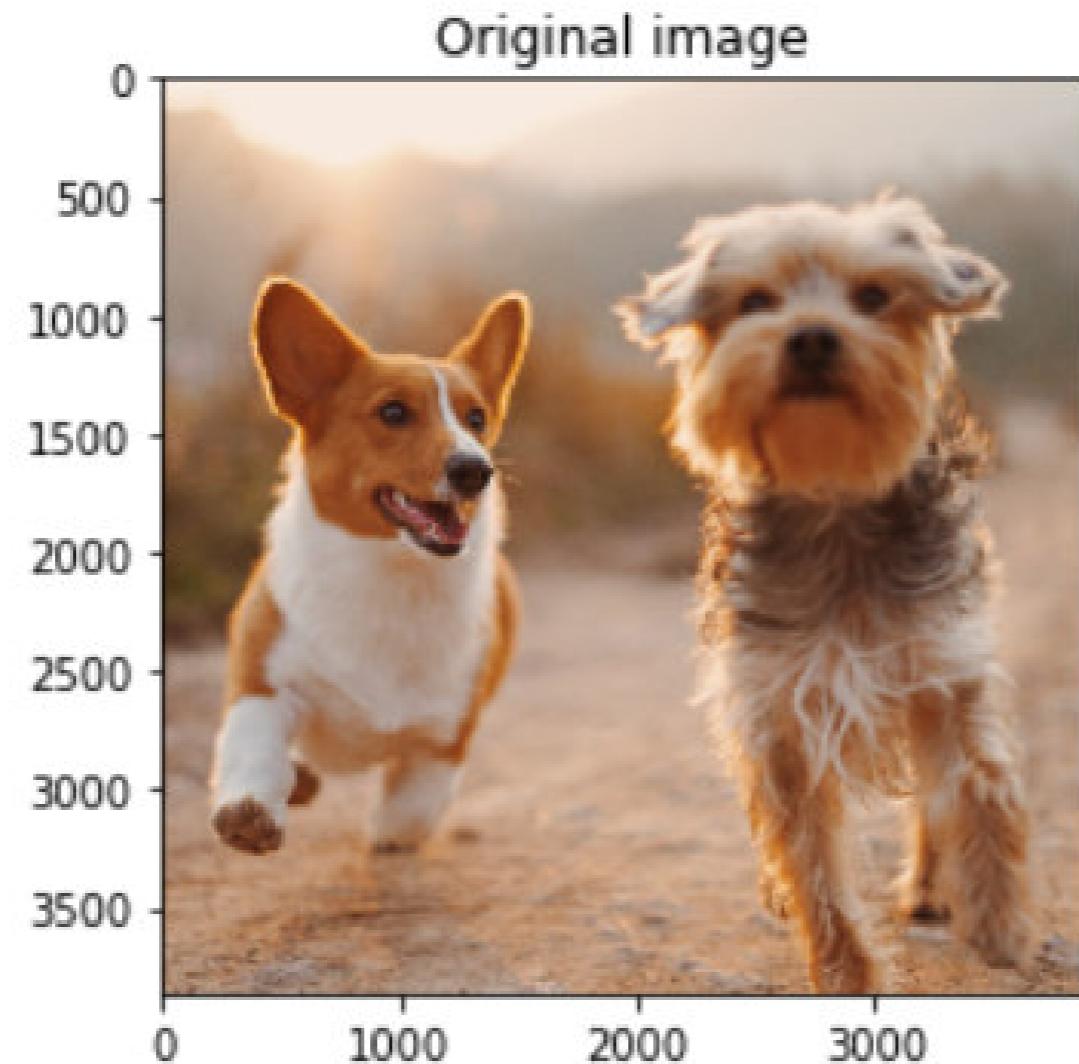
```
from skimage.transform import resize

# 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



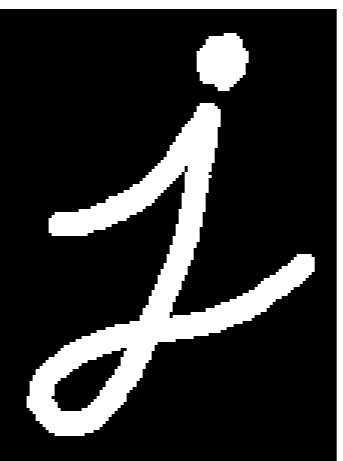
Grayscale



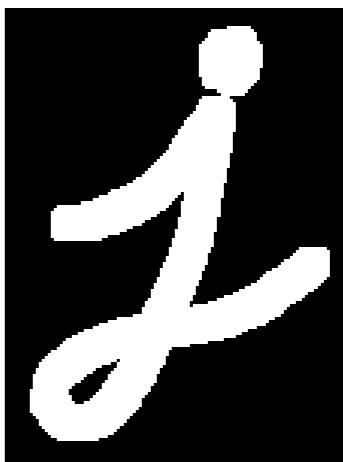
Morphological operations

- Dilation
- Erosion

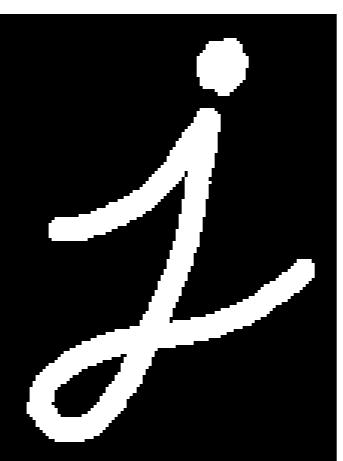
Original



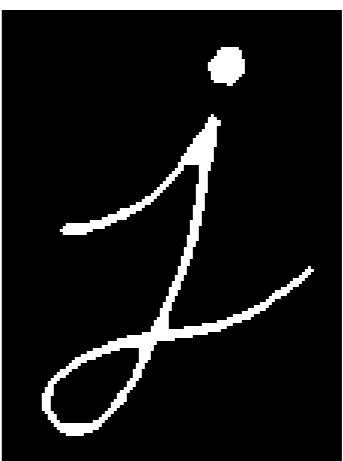
Dilated



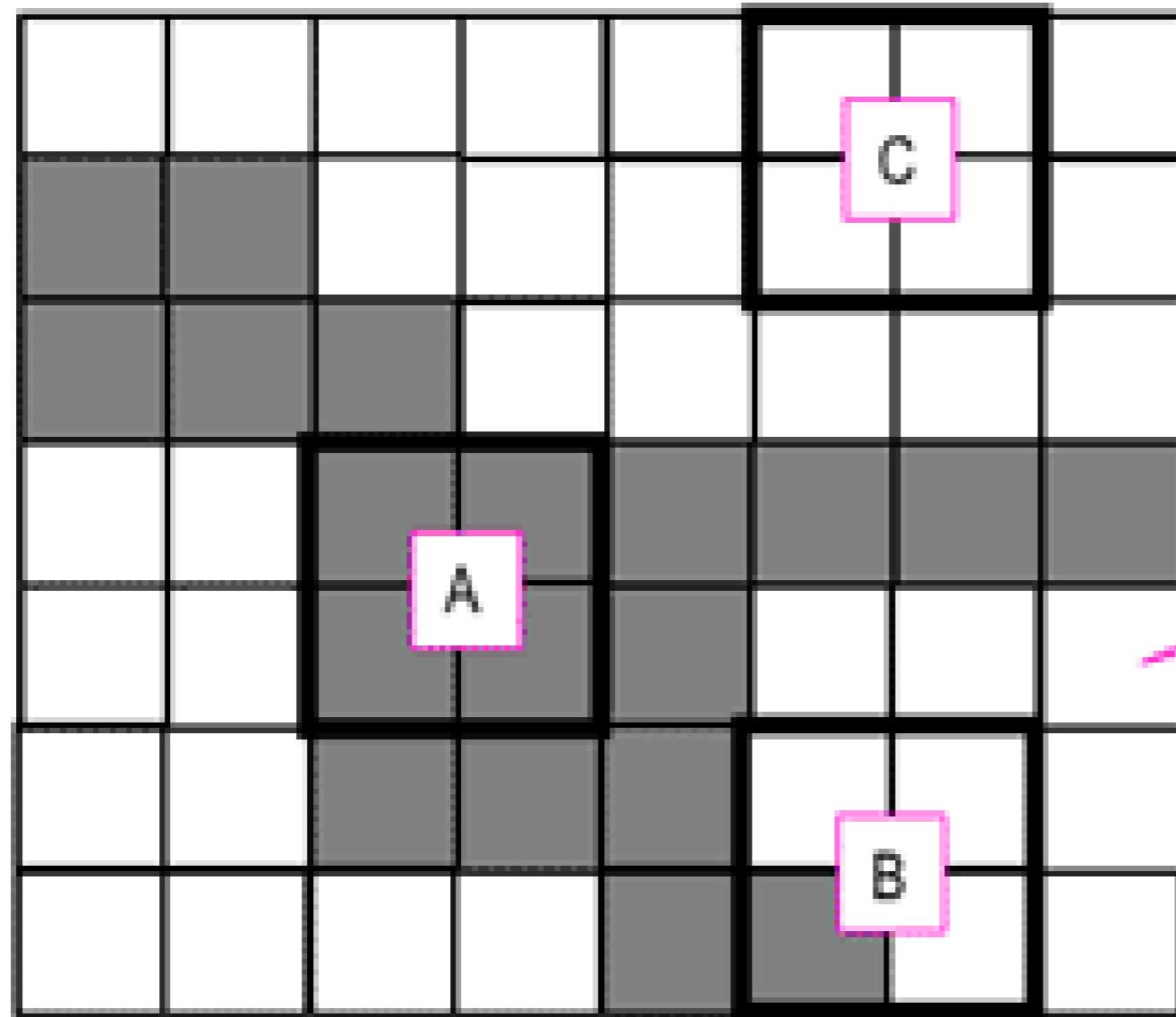
Original



Eroded

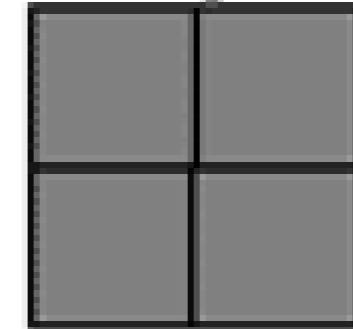


Structuring element



- 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)
```

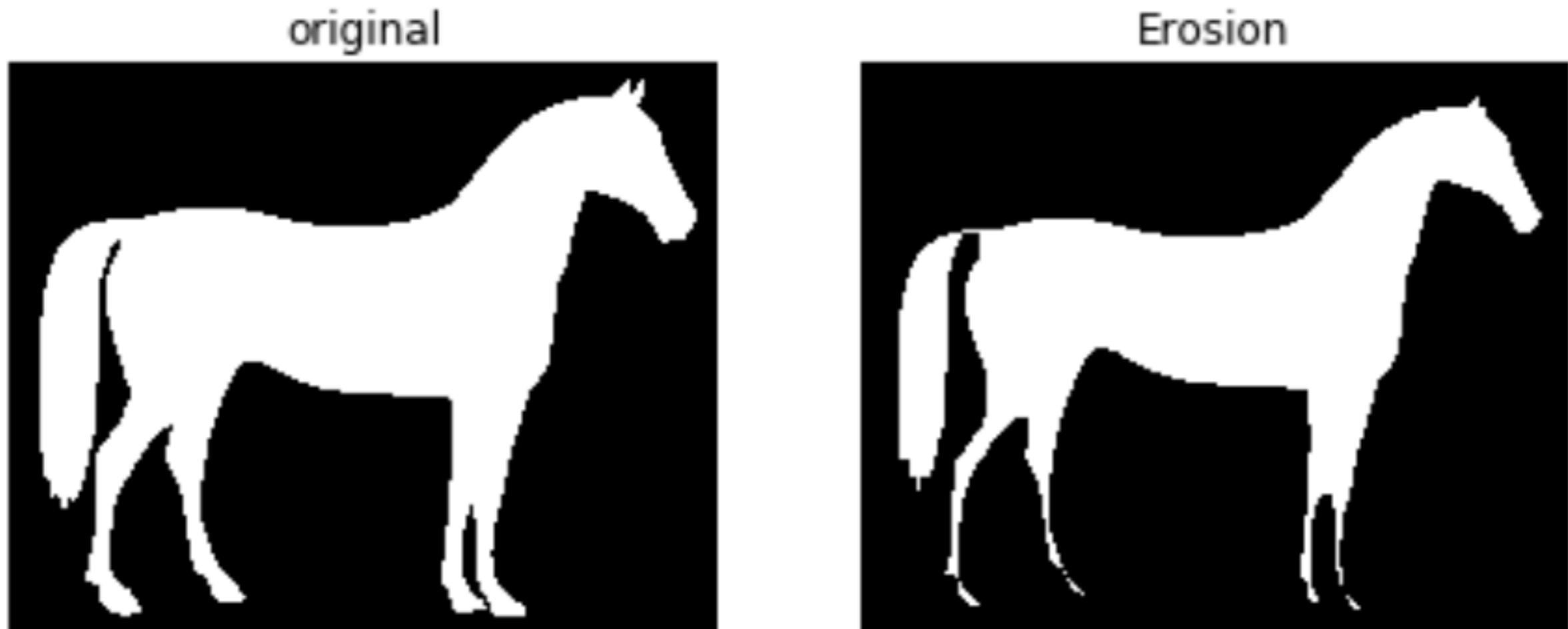
```
[[1 1]
 [1 1]
 [1 1]
 [1 1]]
```

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')
```



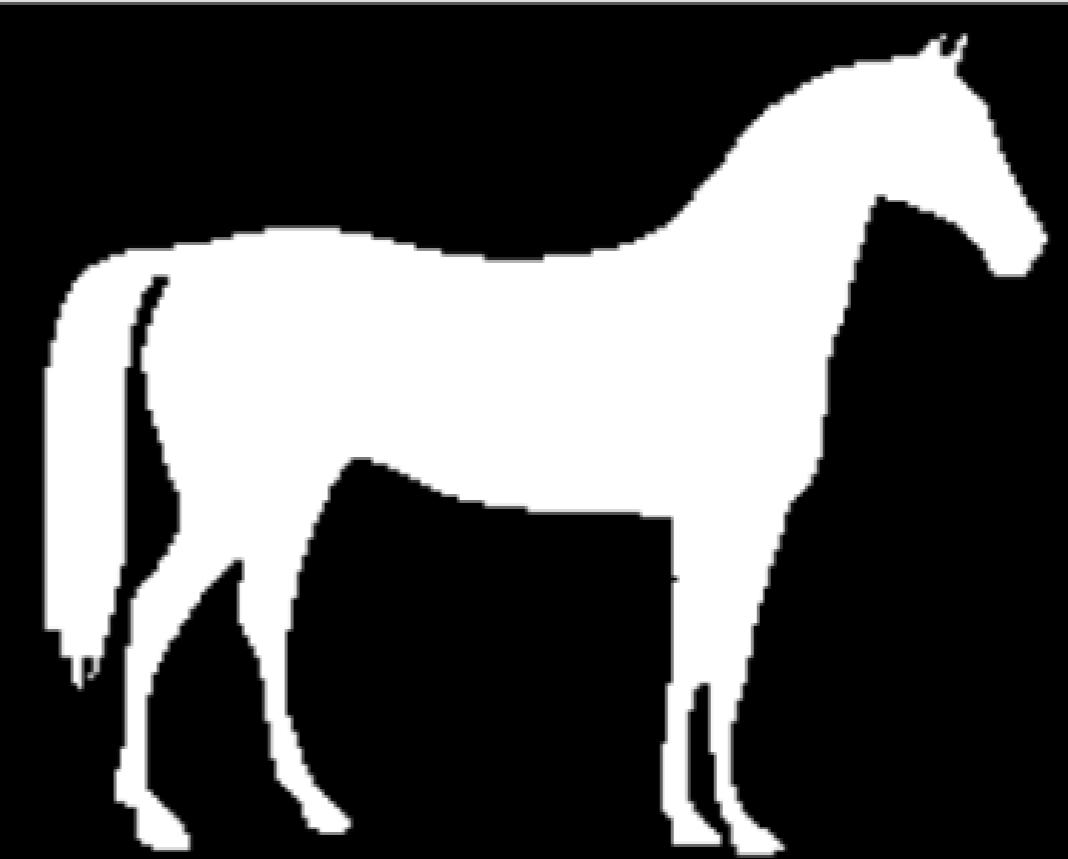
Binary erosion with default selem

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

original



Erosion

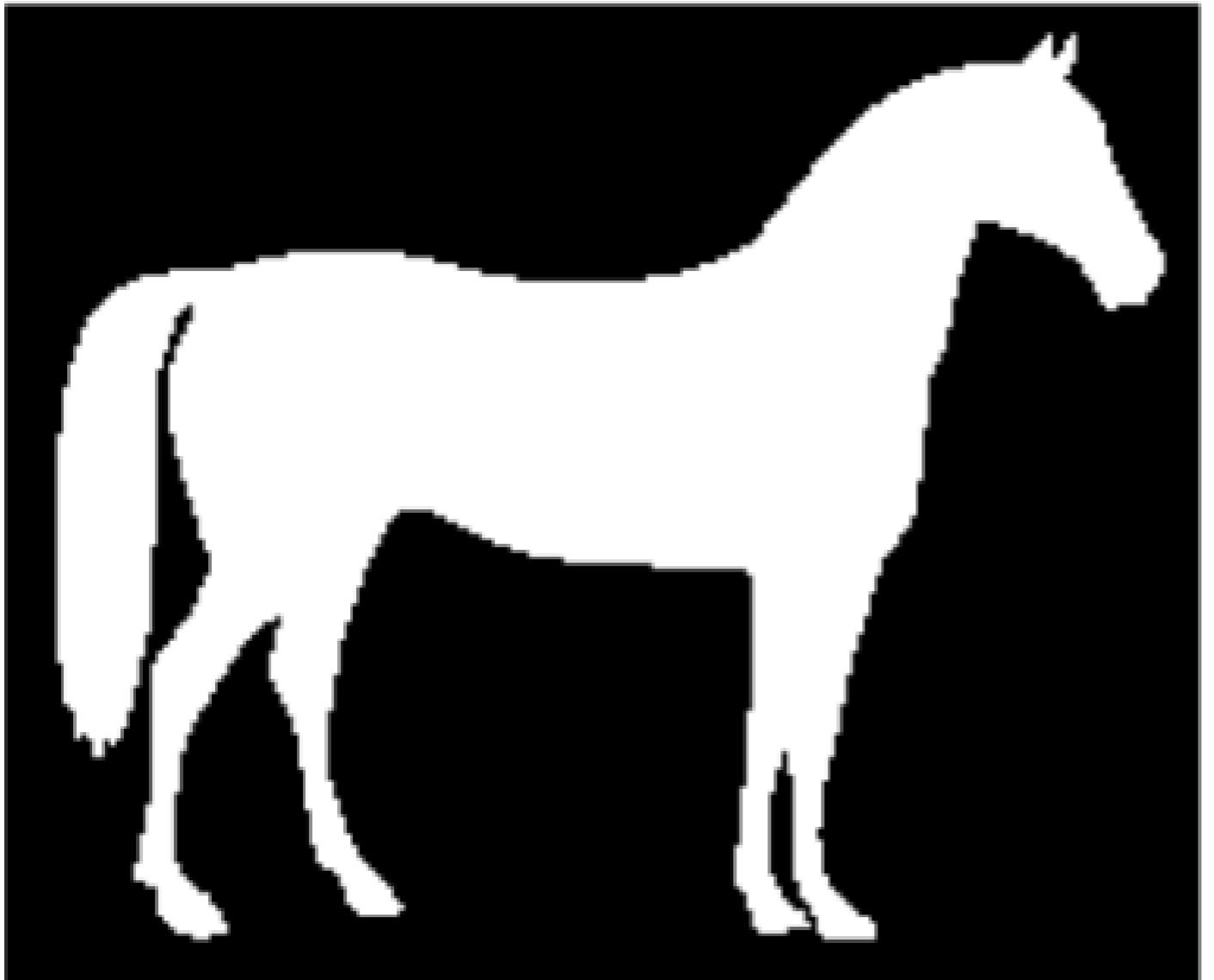


Dilation in scikit-image

```
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

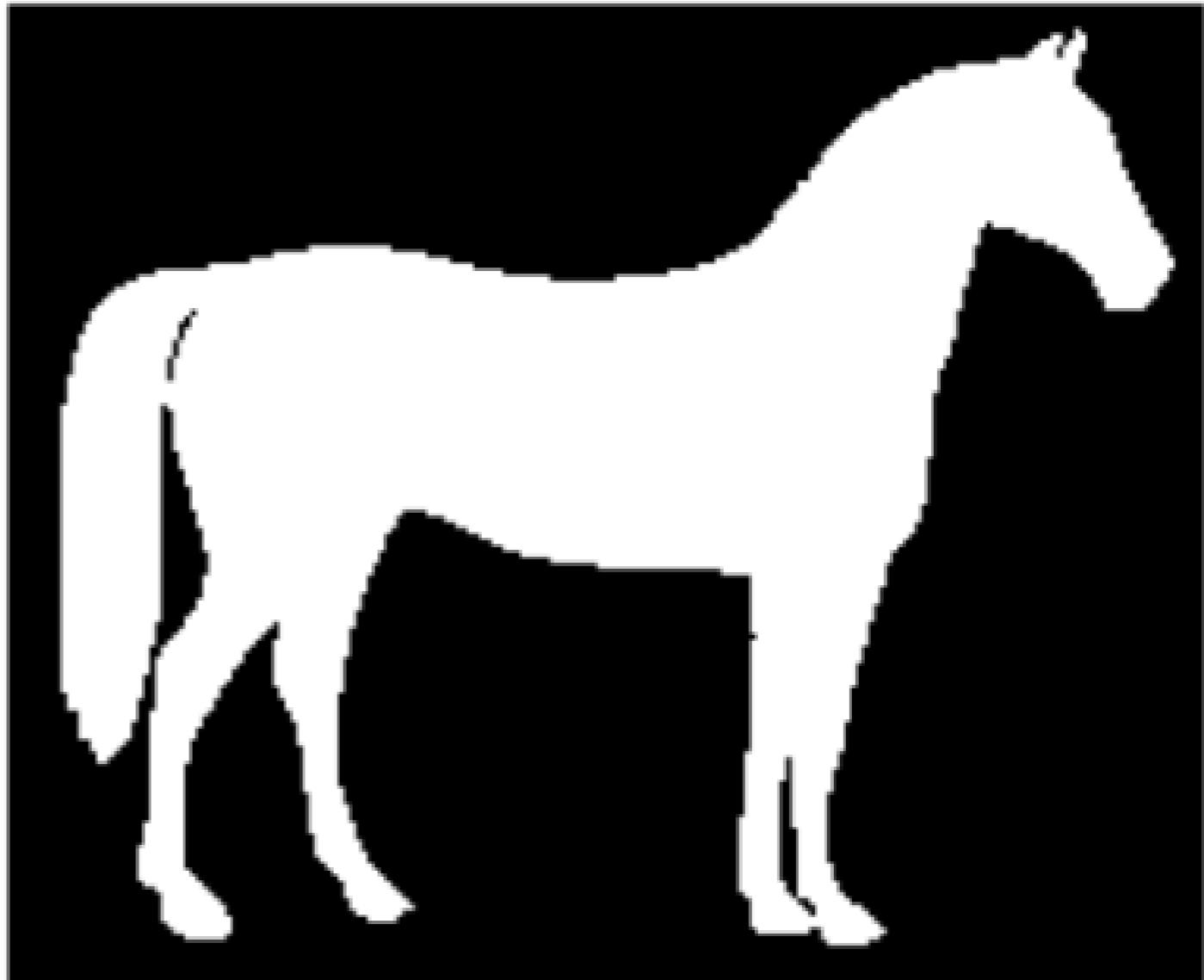


Image restoration

IMAGE PROCESSING IN PYTHON



Rebeca Gonzalez
Data Engineer

Restore an image

Image to restore



Image restored



Image reconstruction

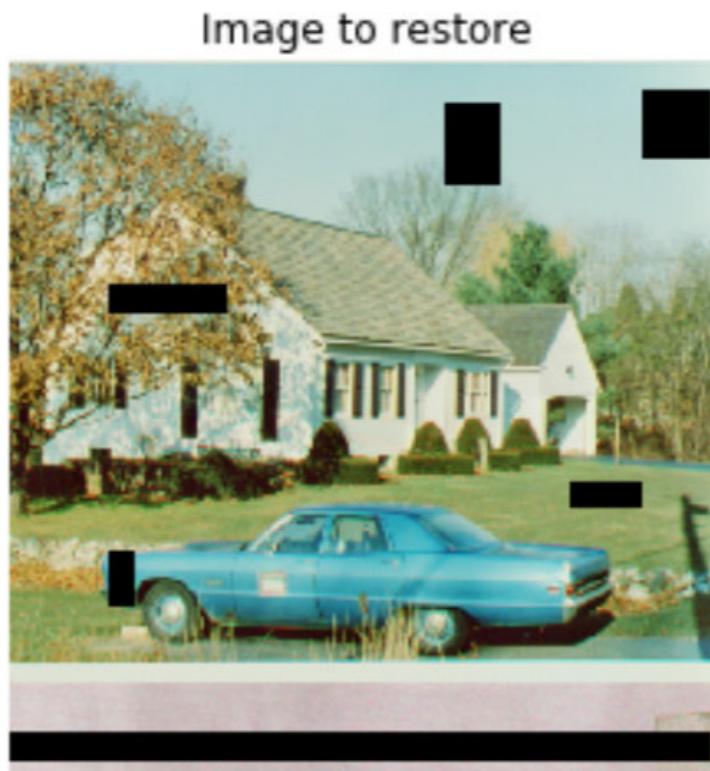
- Fixing damaged images
- Text removing
- Logo removing
- Object removing



Image reconstruction

Inpainting

- Reconstructing lost parts of images
- Looking at the non-damaged regions



Inpainting

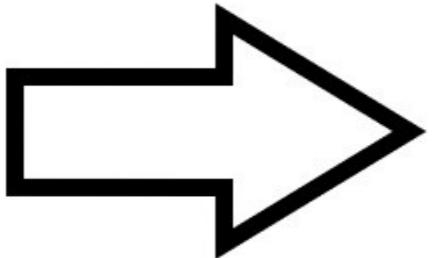


Image reconstruction

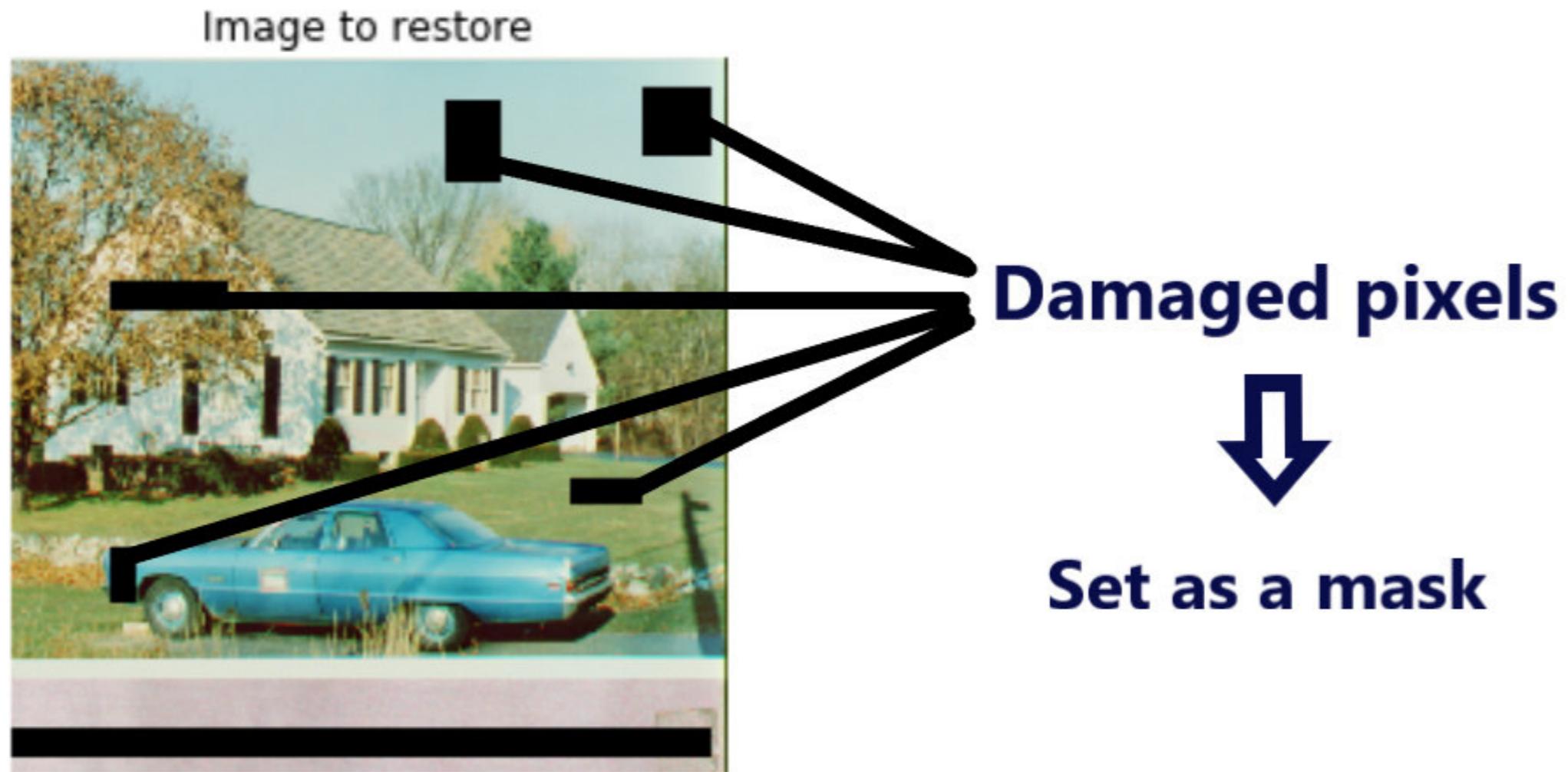


Image reconstruction in scikit-image

```
from skimage.restoration import inpaint

# Obtain the mask
mask = get_mask(defect_image)

# Apply inpainting to the damaged image using the mask
restored_image = inpaint.inpaint_biharmonic(defect_image,
                                             mask,
                                             multichannel=True)

# Show the resulting image
show_image(restored_image)
```

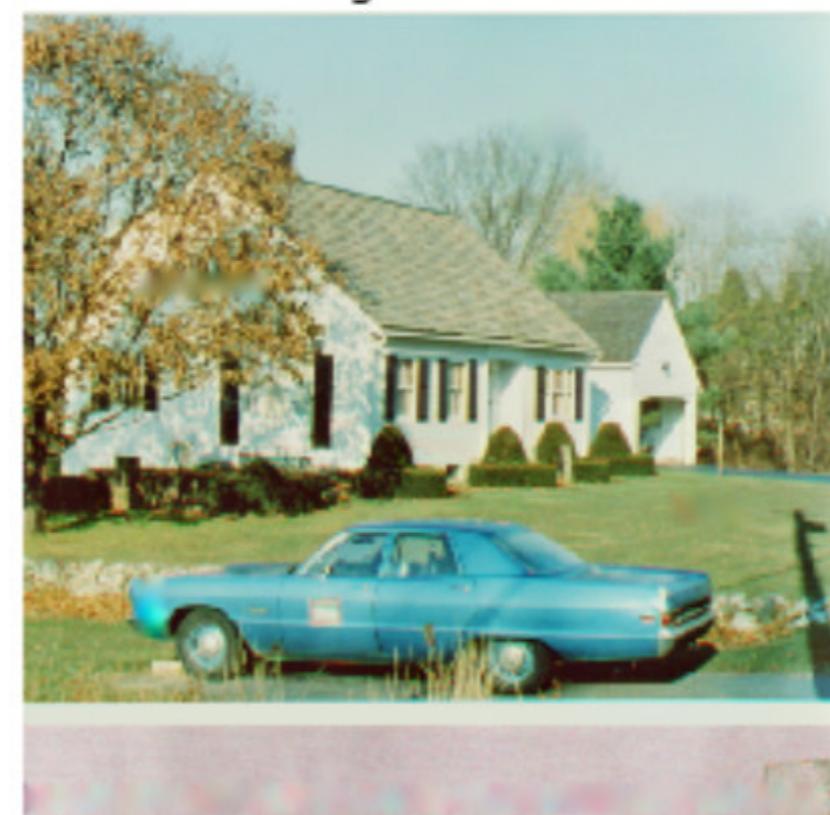
Image reconstruction in scikit-image

```
# Show the defect and resulting images  
show_image(defect_image, 'Image to restore')  
show_image(restored_image, 'Image restored')
```

Image to restore



Image restored



Masks

Image to restore



Mask



Masks

```
def get_mask(image):
    ''' Creates mask with three defect regions '''
    mask = np.zeros(image.shape[:-1])

    mask[101:106, 0:240] = 1

    mask[152:154, 0:60] = 1
    mask[153:155, 60:100] = 1
    mask[154:156, 100:120] = 1
    mask[155:156, 120:140] = 1

    mask[212:217, 0:150] = 1
    mask[217:222, 150:256] = 1

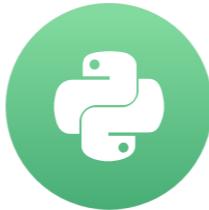
    return mask
```

Let's practice!

IMAGE PROCESSING IN PYTHON

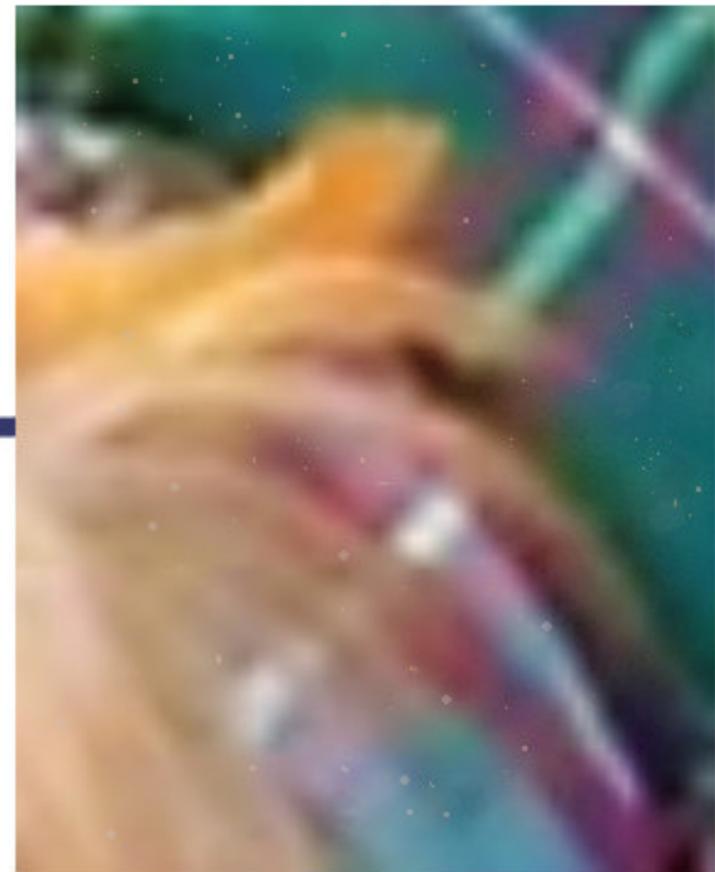
Noise

IMAGE PROCESSING IN PYTHON



Rebeca Gonzalez
Data Engineer

Noise



Noise



Apply noise in scikit-image

```
# Import the module and function
from skimage.util import random_noise

# Add noise to the image
noisy_image = random_noise(dog_image)

# Show original and resulting image
show_image(dog_image)
show_image(noisy_image, 'Noisy image')
```

Apply noise in scikit-image

Original



Noisy image



Reducing noise

Noisy image



Denoised



Denoising types

- Total variation (TV)
- Bilateral
- Wavelet denoising
- Non-local means denoising



Denoising

Using total variation filter denoising

```
from skimage.restoration import denoise_tv_chambolle

# Apply total variation filter denoising
denoised_image = denoise_tv_chambolle(noisy_image,
                                         weight=0.1,
                                         multichannel=True)

# Show denoised image
show_image(noisy_image, 'Noisy image')
show_image(denoised_image, 'Denoised image')
```

Denoising

Total variation filter

Noisy image



Denoised image



Denoising

Bilateral filter

```
from skimage.restoration import denoise_bilateral

# Apply bilateral filter denoising
denoised_image = denoise_bilateral(noisy_image, multichannel=True)

# Show original and resulting images
show_image(noisy_image, 'Noisy image')
show_image(denoised_image, 'Denoised image')
```

Denoising

Bilateral filter

Noisy image



Denoised image

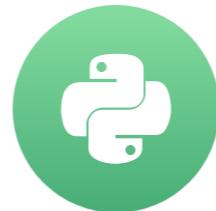


Let's practice!

IMAGE PROCESSING IN PYTHON

Superpixels & segmentation

IMAGE PROCESSING IN PYTHON



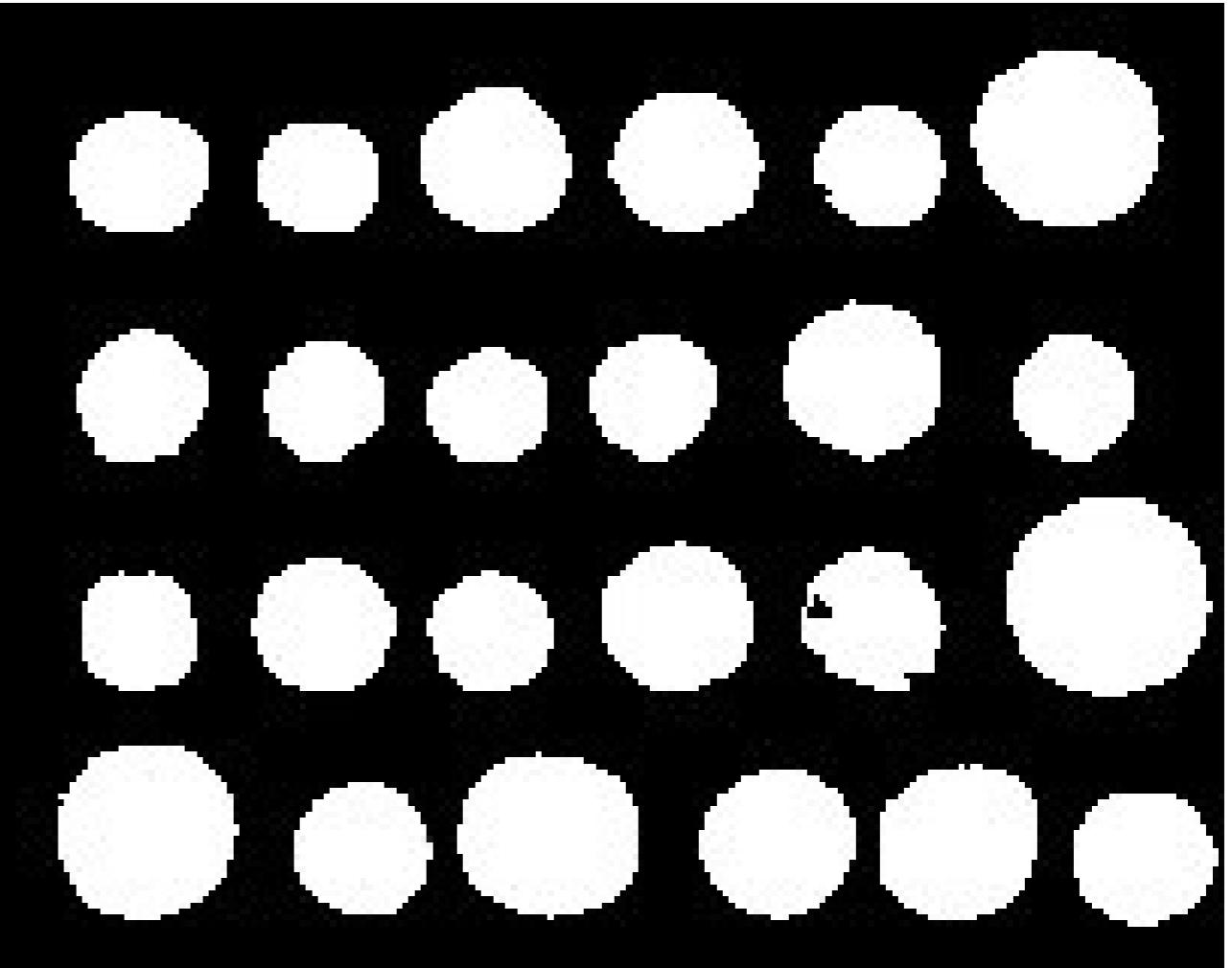
Rebeca Gonzalez
Data Engineer

Segmentation

Original

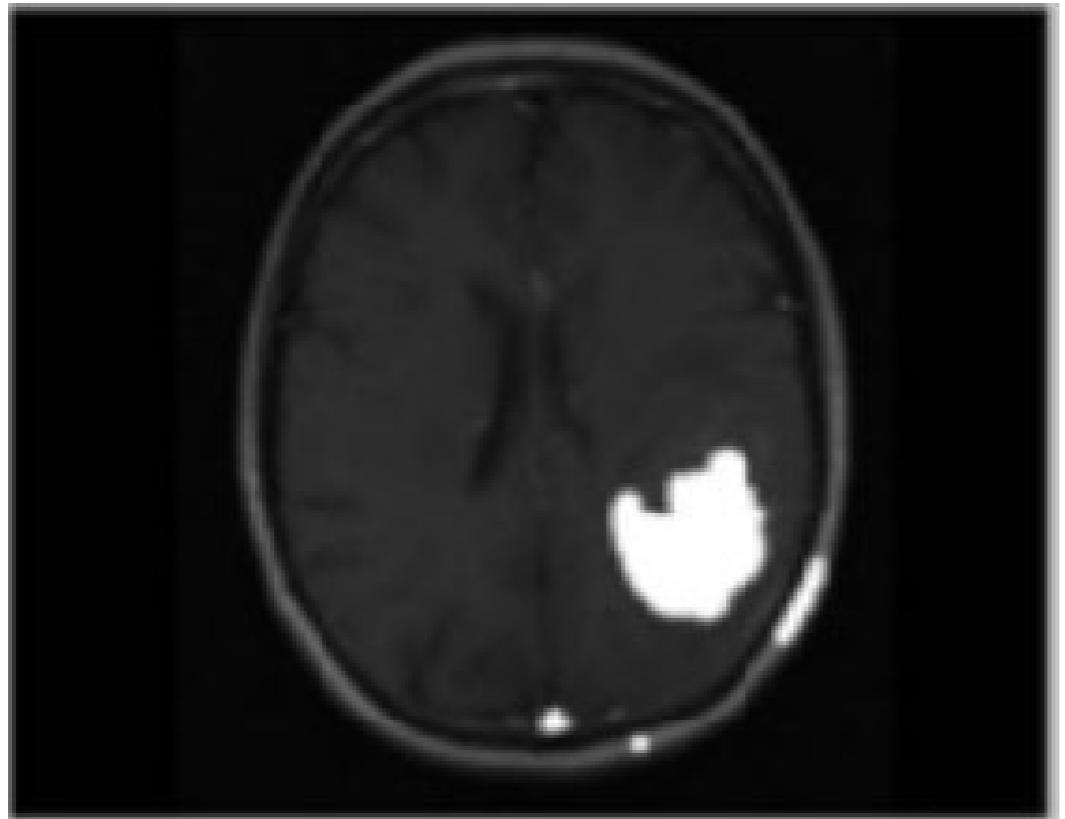


Segmented image



Segmentation

Segmented



Original



Segmented

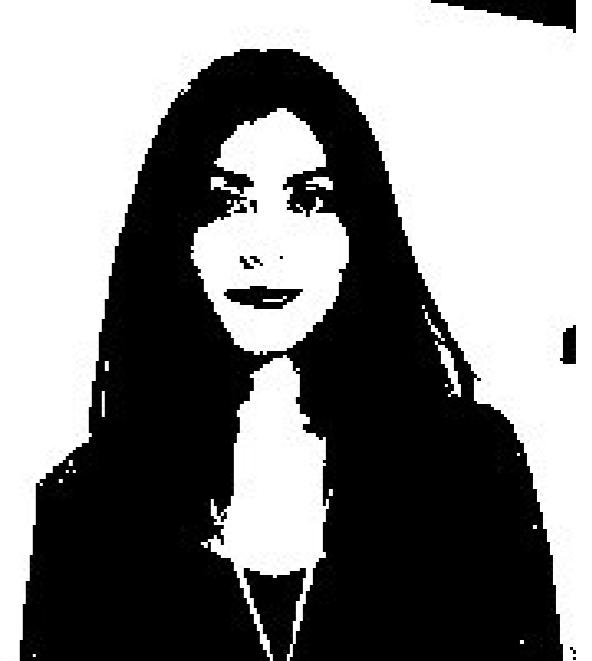
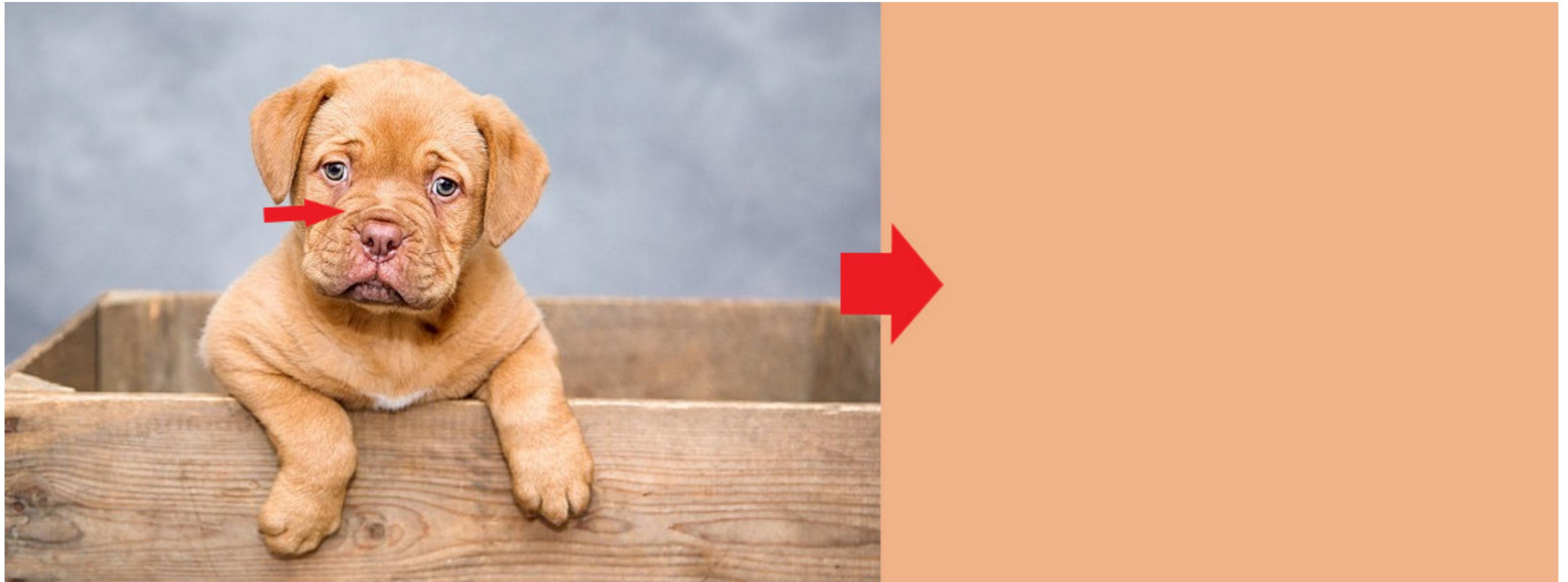
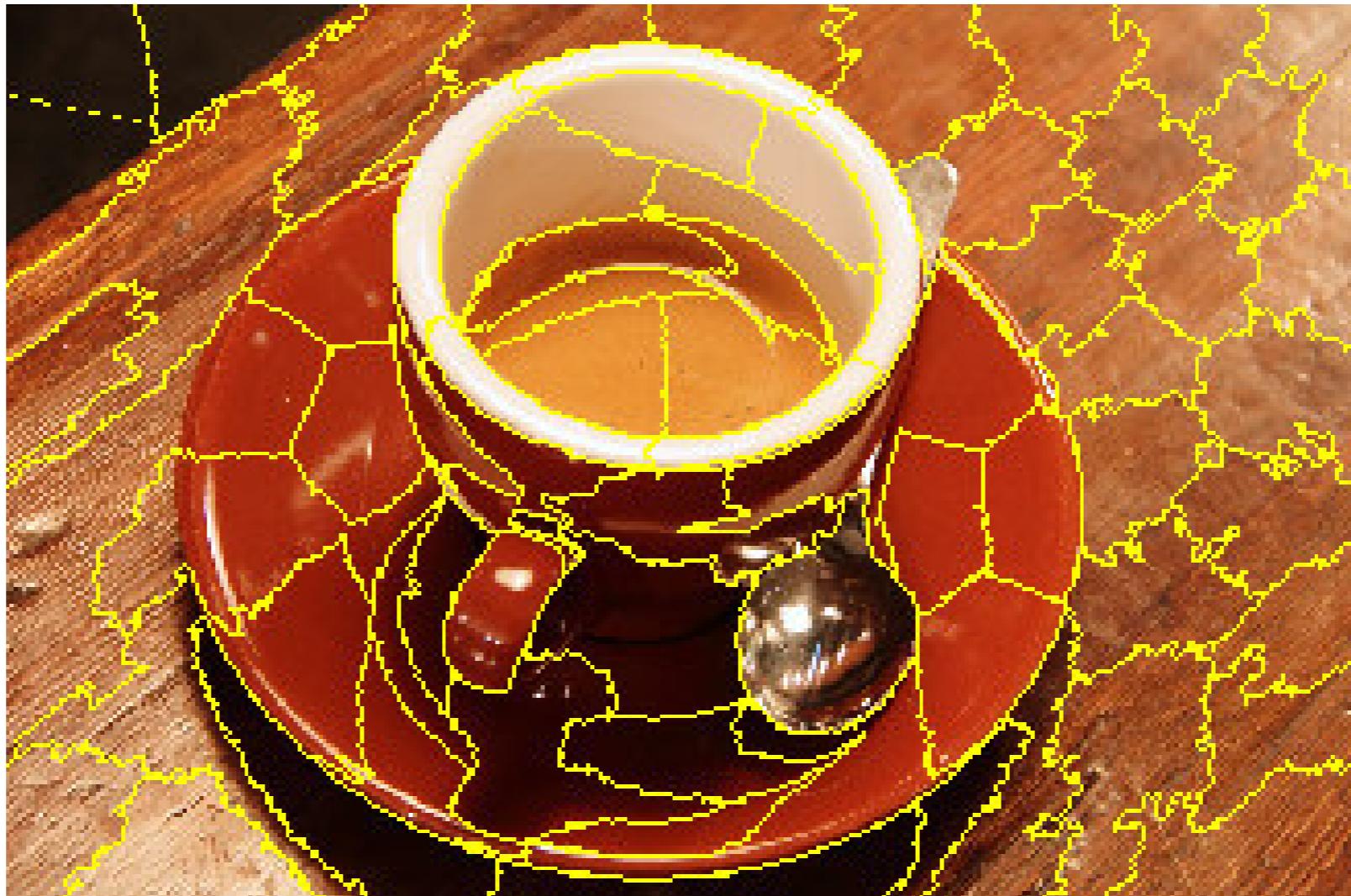


Image representation



Superpixels

Superpixel segmentation, 100 segments



Benefits of superpixels

- More meaningful regions
- Computational efficiency

Segmentation

- Supervised
- Unsupervised

Supervised thresholding



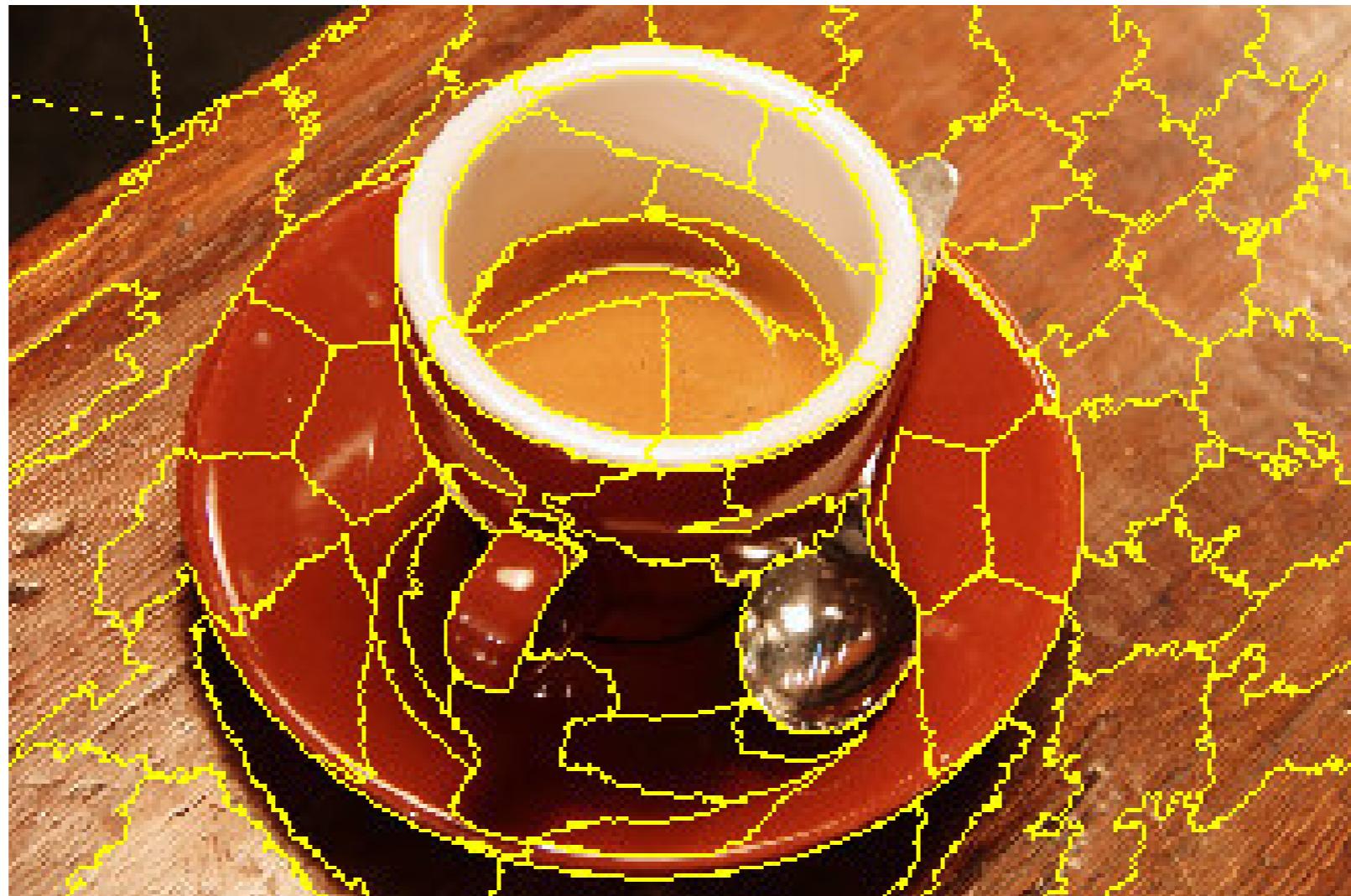
Unsupervised thresholding



Unsupervised segmentation

Simple Linear Iterative Clustering (SLIC)

Superpixel segmentation, 100 segments



Unsupervised segmentation (SLIC)

```
# Import the modules
from skimage.segmentation import slic
from skimage.color import label2rgb

# Obtain the segments
segments = segmentation.slic(image)

# Put segments on top of original image to compare
segmented_image = label2rgb(segments, image, kind='avg')

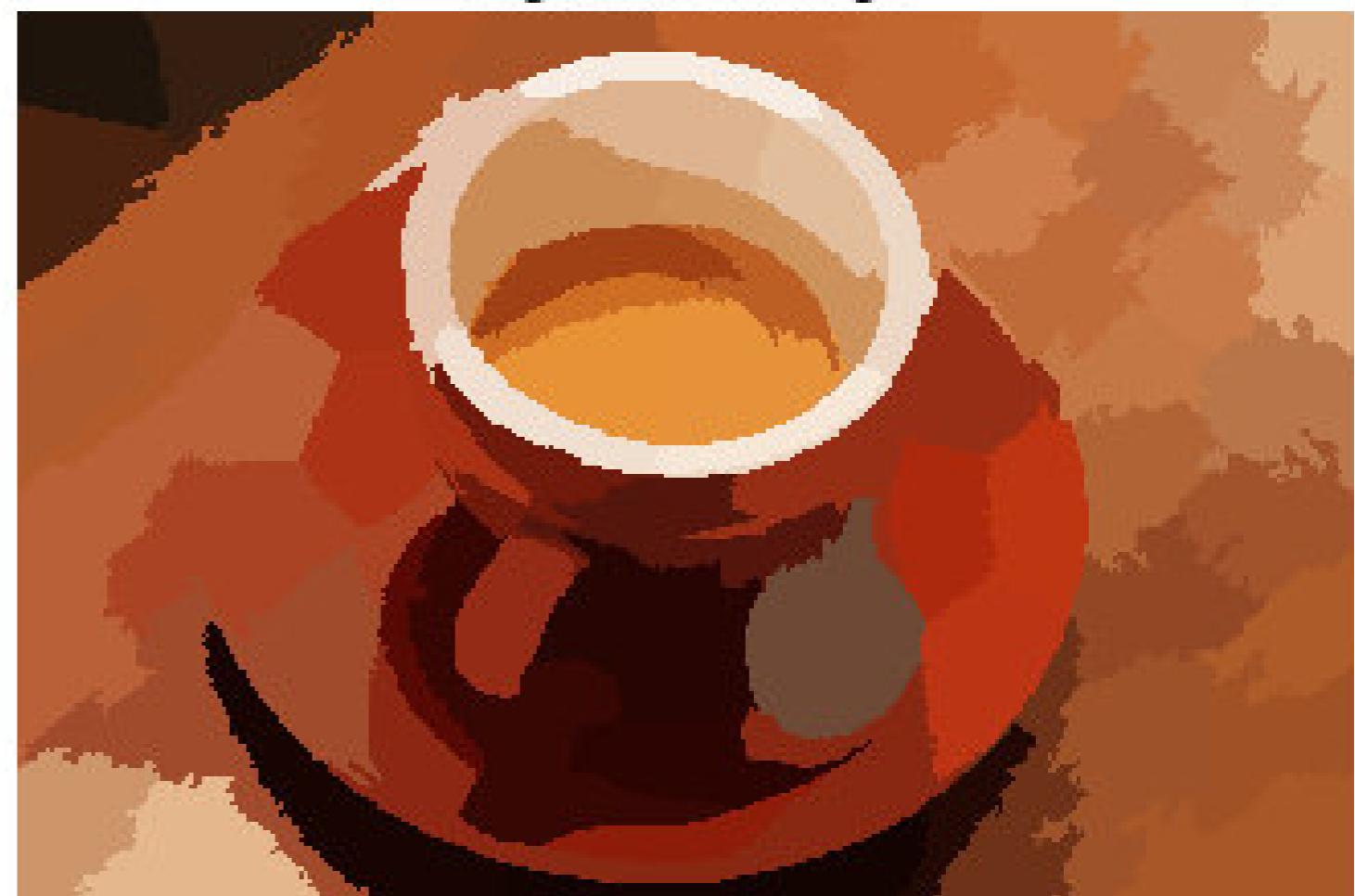
show_image(image)
show_image(segmented_image, "Segmented image")
```

Unsupervised segmentation (SLIC)

Original



Segmented image



More segments

```
# Import the modules
from skimage.segmentation import slic
from skimage.color import label2rgb

# Obtain the segmentation with 300 regions
segments = slic(image, n_segments= 300)

# Put segments on top of original image to compare
segmented_image = label2rgb(segments, image, kind='avg')

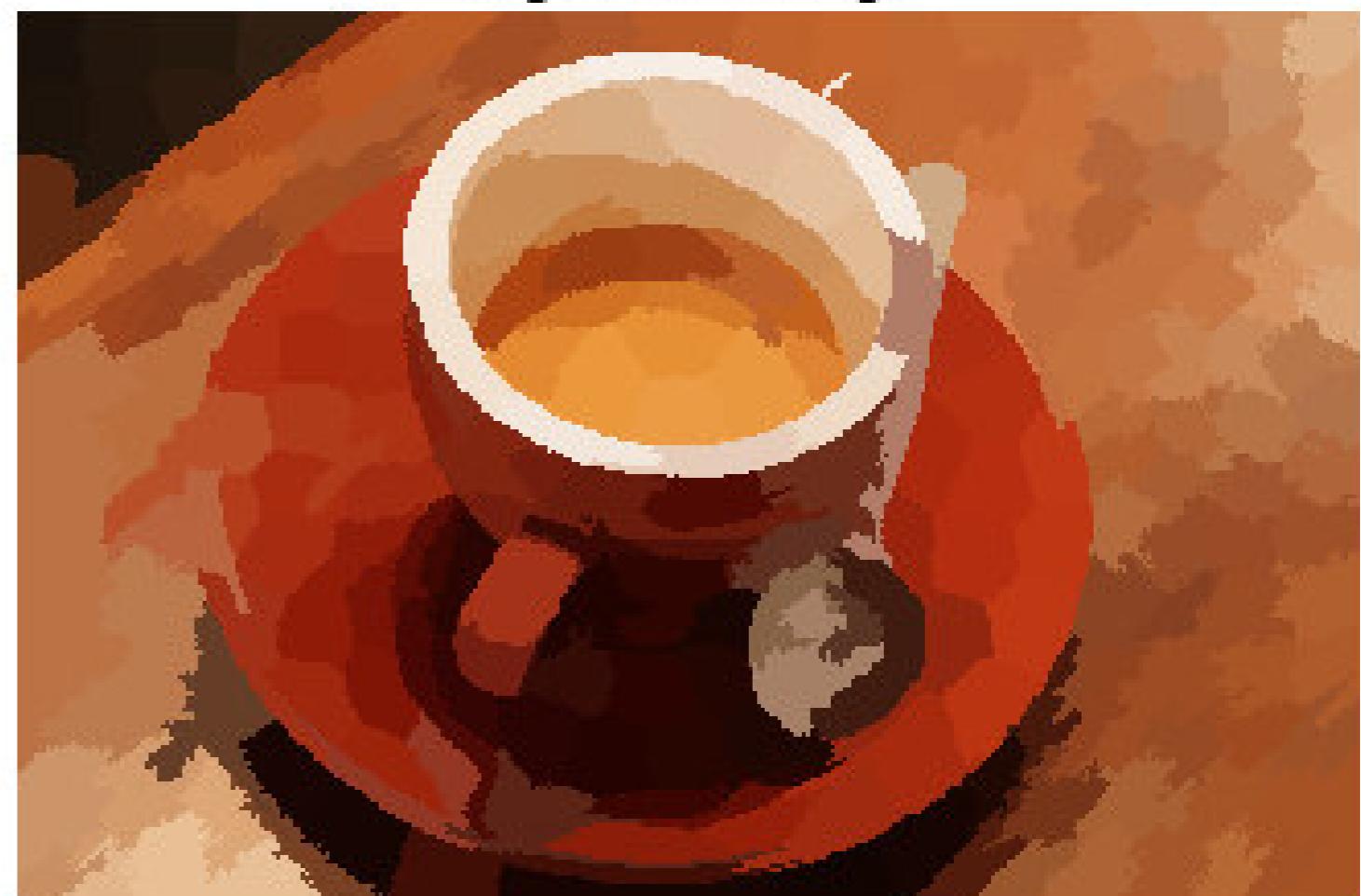
show_image(segmented_image)
```

More segments

Original



Segmented image

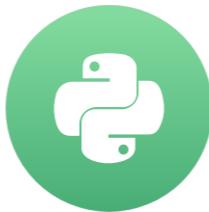


Let's practice!

IMAGE PROCESSING IN PYTHON

Finding contours

IMAGE PROCESSING IN PYTHON



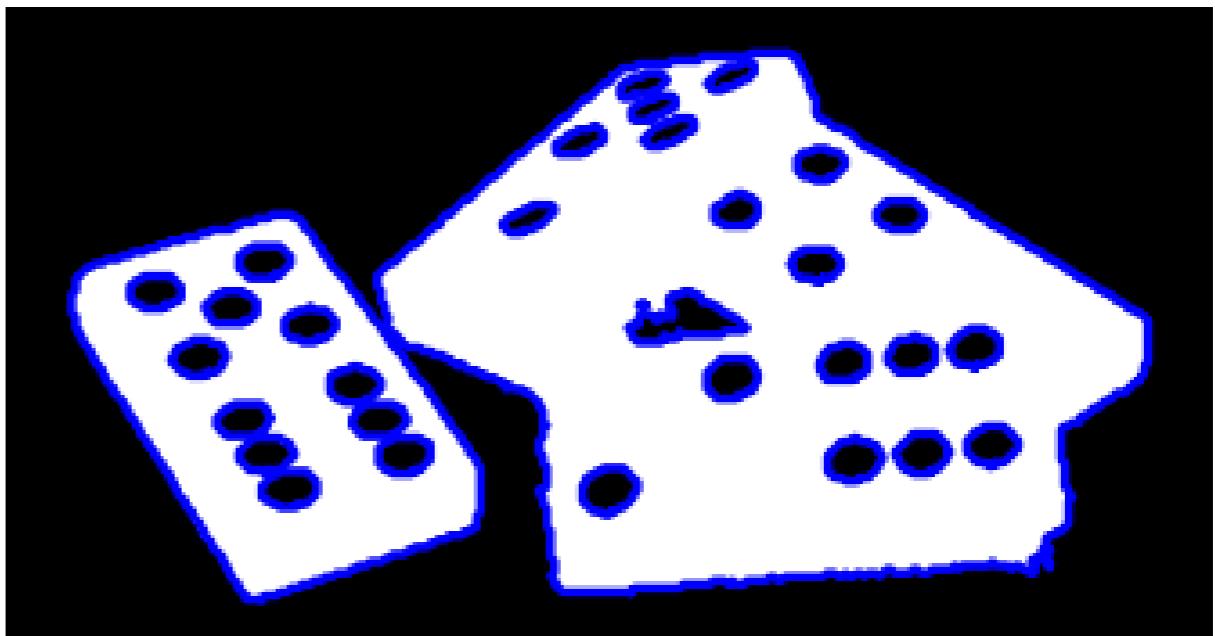
Rebeca Gonzalez
Data Engineer

Finding contours

Original image



Contours

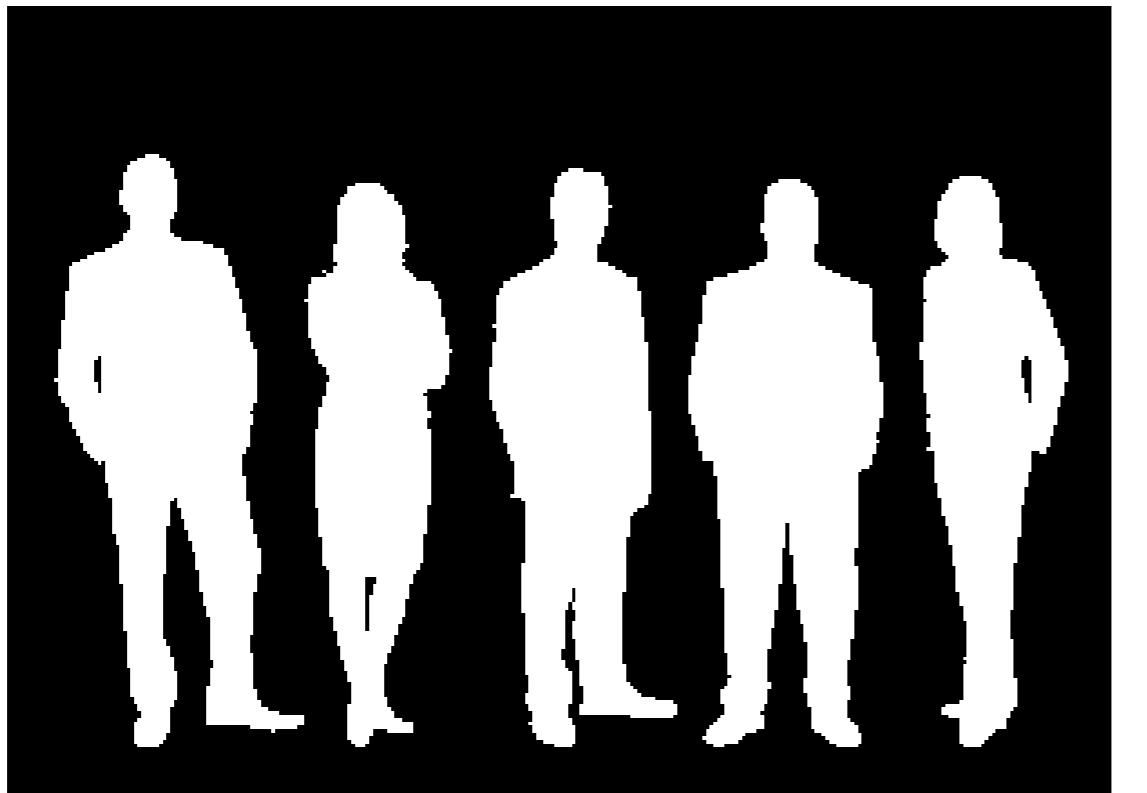


- Measure size
- Classify shapes
- Determine the number of objects

Total points in domino tokens: 35.

Binary images

Thresholded Image



Contours



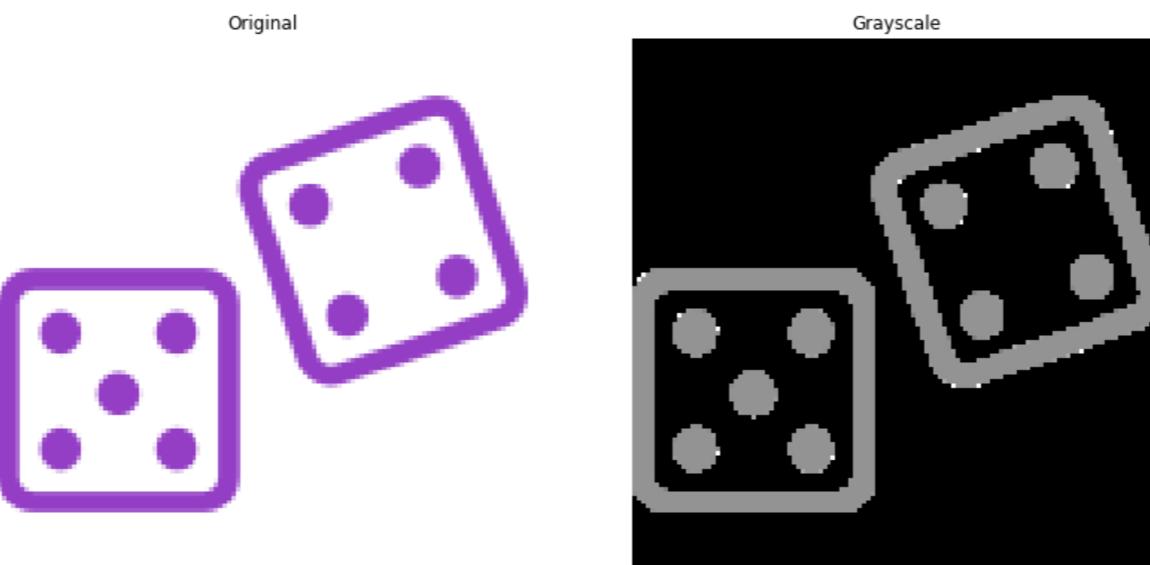
We can obtain a binary image applying
thresholding or using edge detection

Find contours using scikit-image

PREPARING THE IMAGE

Transform the image to 2D grayscale.

```
# Make the image grayscale  
image = color.rgb2gray(image)
```

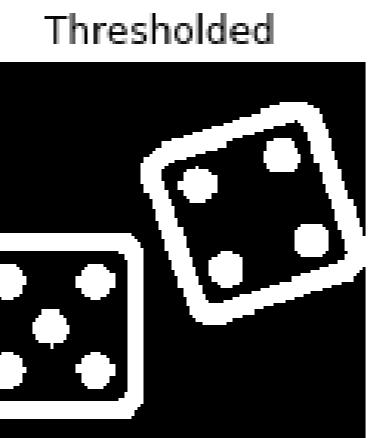


Find contours using scikit-image

PREPARING THE IMAGE

Binarize the image

```
# Obtain the thresh value  
thresh = threshold_otsu(image)  
  
# Apply thresholding  
thresholded_image = image > thresh
```

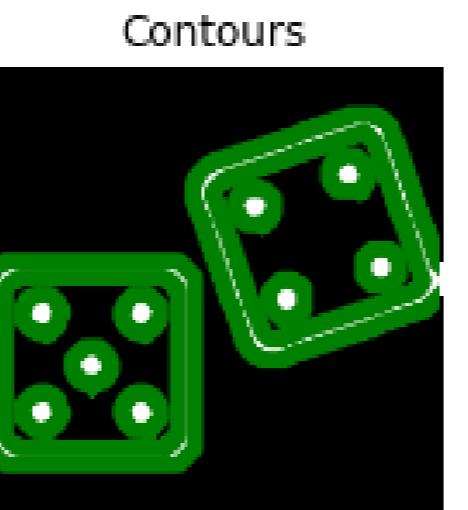


Find contours using scikit-image

And then use `findContours()`.

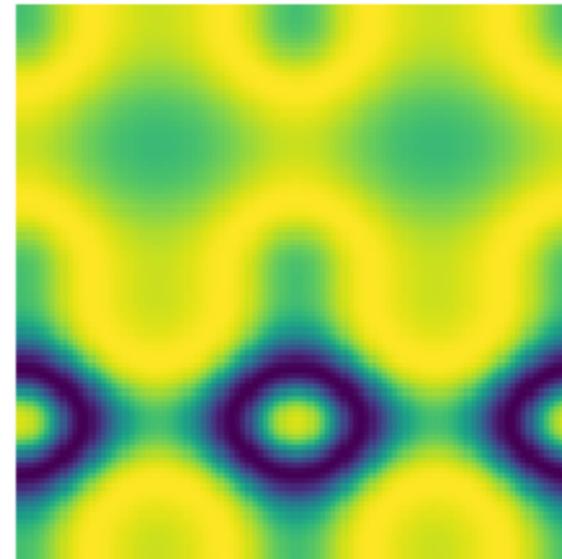
```
# Import the measure module
from skimage import measure

# Find contours at a constant value of 0.8
contours = measure.find_contours(thresholded_image, 0.8)
```

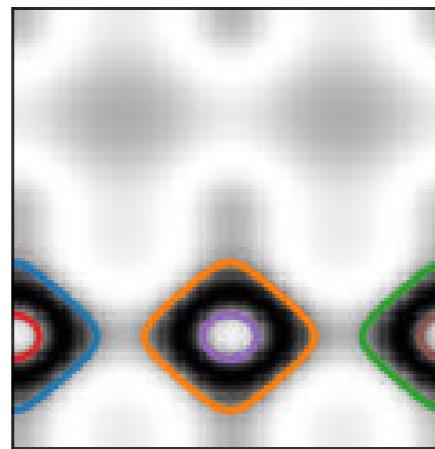


Constant level value

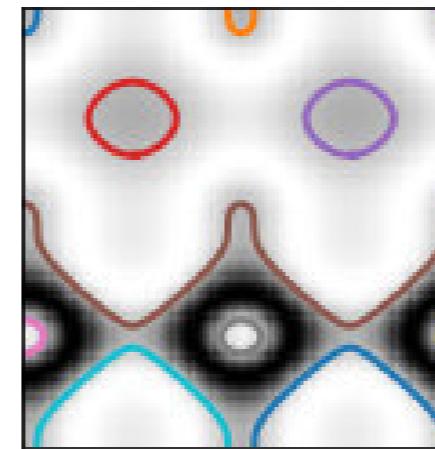
Original



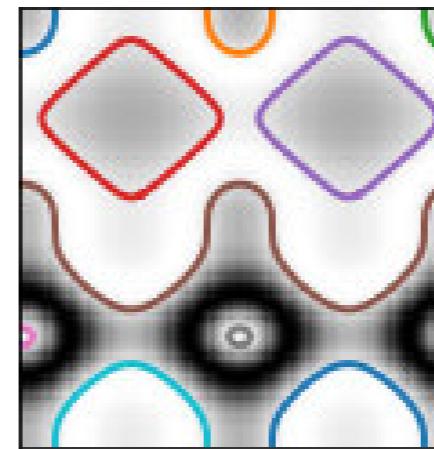
Level value of 0.1



Level value of 0.5



Level value of 0.8



The steps to spotting contours

```
from skimage import measure
from skimage.filters import threshold_otsu

# Make the image grayscale
image = color.rgb2gray(image)

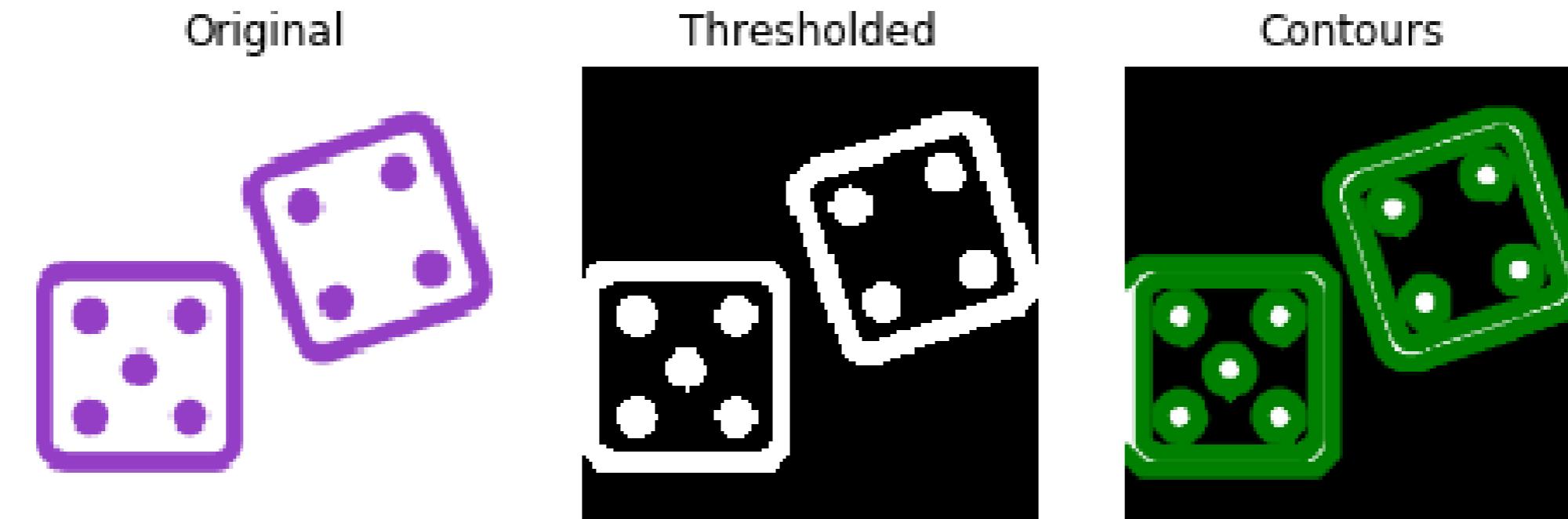
# Obtain the optimal thresh value of the image
thresh = threshold_otsu(image)

# Apply thresholding and obtain binary image
thresholded_image = image > thresh

# Find contours at a constant value of 0.8
contours = measure.find_contours(thresholded_image, 0.8)
```

The steps to spotting contours

Resulting in

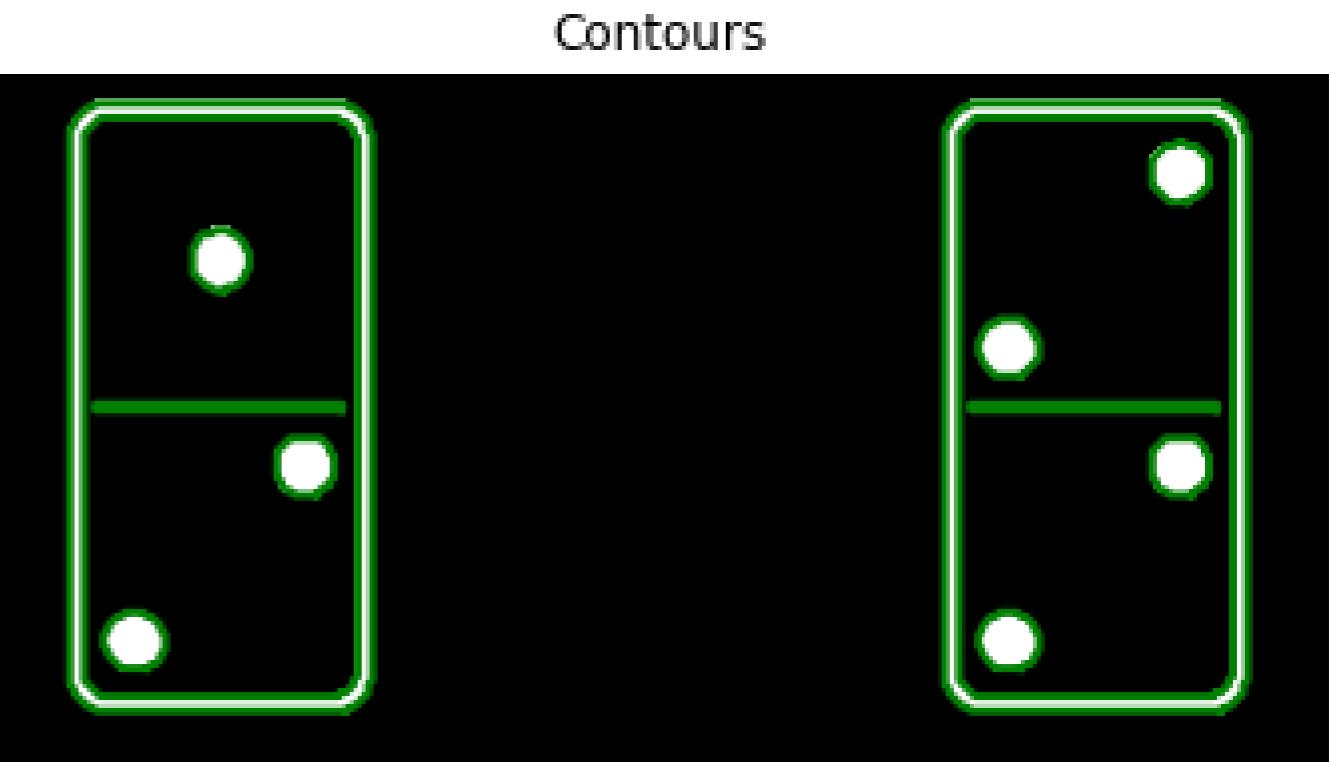


A contour's shape

Contours: list of (n,2) - ndarrays.

```
for contour in contours:  
    print(contour.shape)
```

```
(433, 2)  
(433, 2)  
(401, 2)  
(401, 2)  
(123, 2)  
(123, 2)  
(59, 2)  
(59, 2)  
(59, 2)  
(57, 2)  
(57, 2)
```

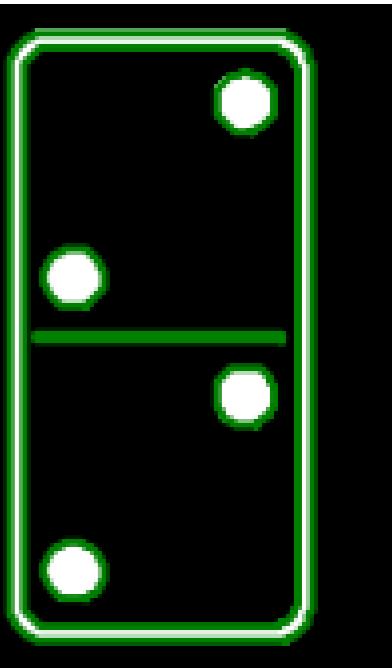
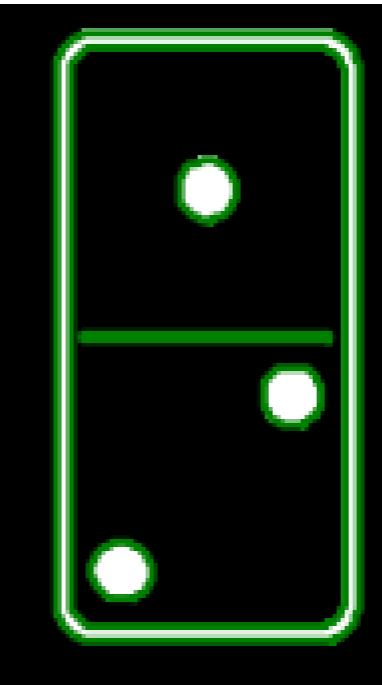


A contour's shape

```
for contour in contours:  
    print(contour.shape)
```

```
(433, 2)  
(433, 2) --> Outer border  
(401, 2)  
(401, 2)  
(123, 2)  
(123, 2)  
( 59, 2)  
( 59, 2)  
( 59, 2)  
( 57, 2)  
( 57, 2)  
( 59, 2)  
( 59, 2)
```

Contours

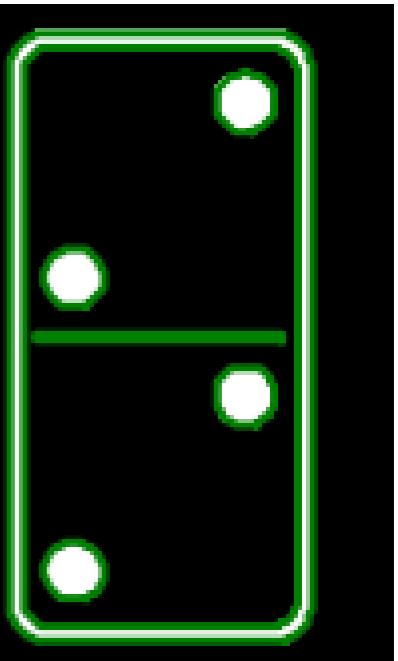
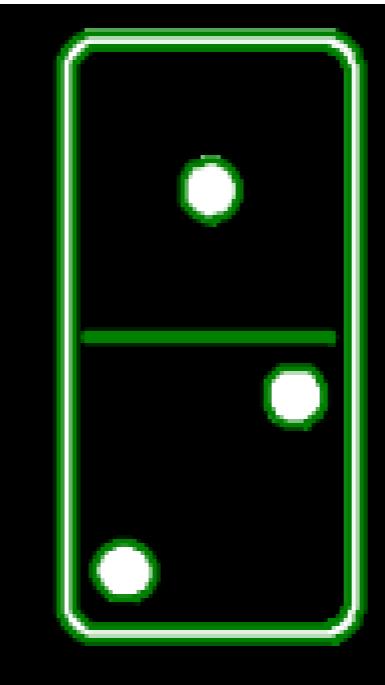


A contour's shape

```
for contour in contours:  
    print(contour.shape)
```

```
(433, 2)  
(433, 2) --> Outer border  
(401, 2)  
(401, 2) --> Inner border  
(123, 2)  
(123, 2)  
(59, 2)  
(59, 2)  
(59, 2)  
(57, 2)  
(57, 2)  
(59, 2)  
(59, 2)
```

Contours

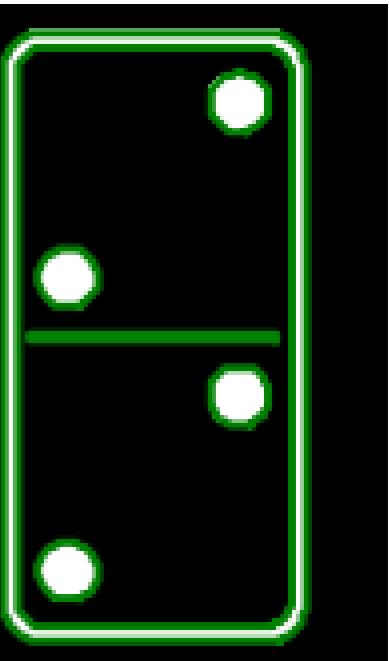
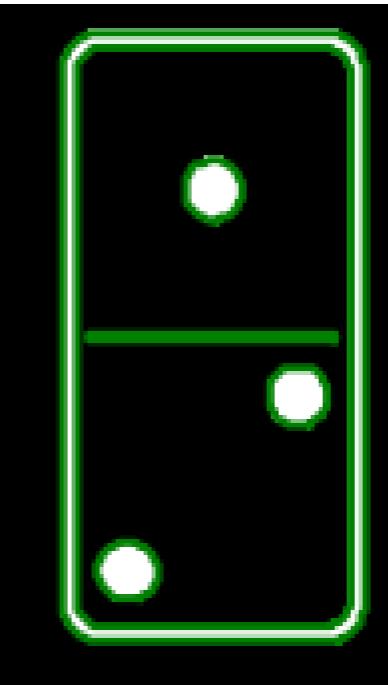


A contour's shape

```
for contour in contours:  
    print(contour.shape)
```

```
(433, 2)  
(433, 2) --> Outer border  
(401, 2)  
(401, 2) --> Inner border  
(123, 2)  
(123, 2) --> Divisory line of tokens  
( 59, 2)  
( 59, 2)  
( 59, 2)  
( 57, 2)  
( 57, 2)  
( 59, 2)  
( 59, 2)
```

Contours

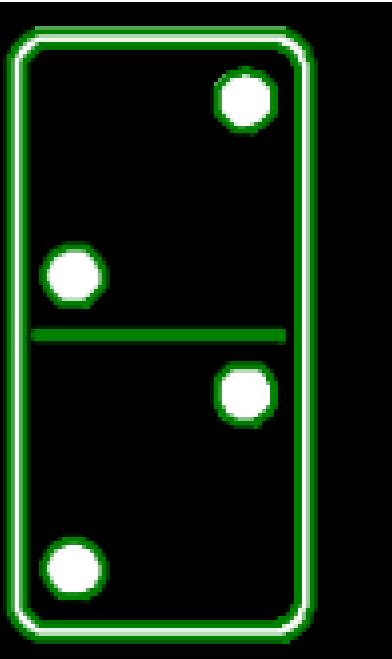
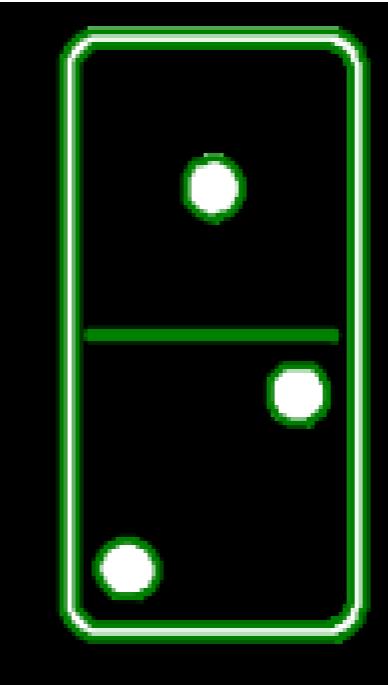


A contour's shape

```
for contour in contours:  
    print(contour.shape)
```

```
(433, 2)  
(433, 2) --> Outer border  
(401, 2)  
(401, 2) --> Inner border  
(123, 2)  
(123, 2) --> Divisory line of tokens  
(59, 2)  
(59, 2)  
(59, 2)  
(57, 2)  
(57, 2)  
(59, 2)  
(59, 2) --> Dots
```

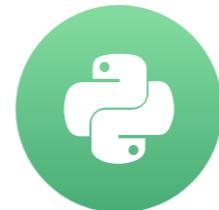
Contours



Number of dots: 7.

Finding the edges with Canny

IMAGE PROCESSING IN PYTHON



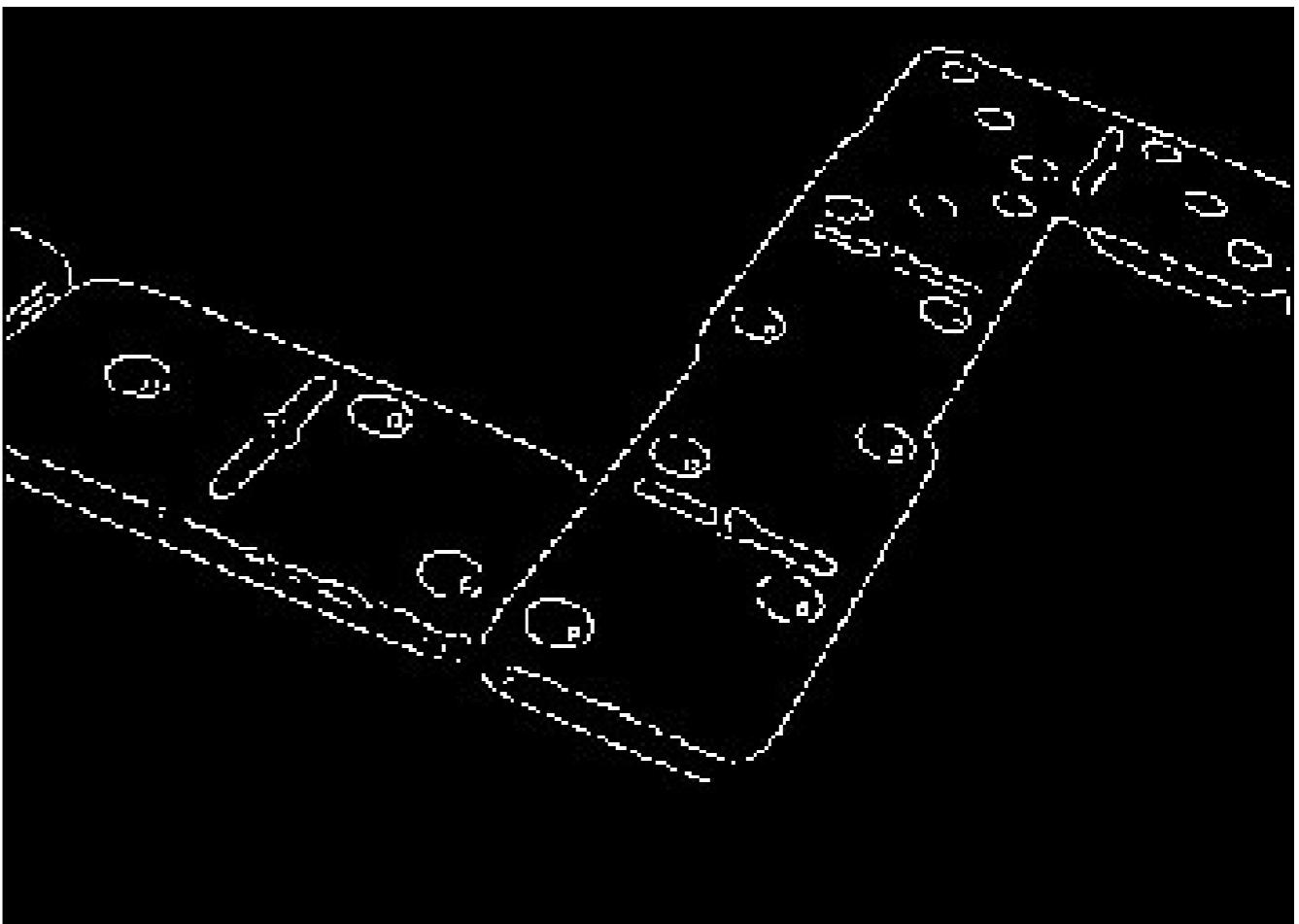
Rebeca Gonzalez
Data Engineer

Detecting edges

Original

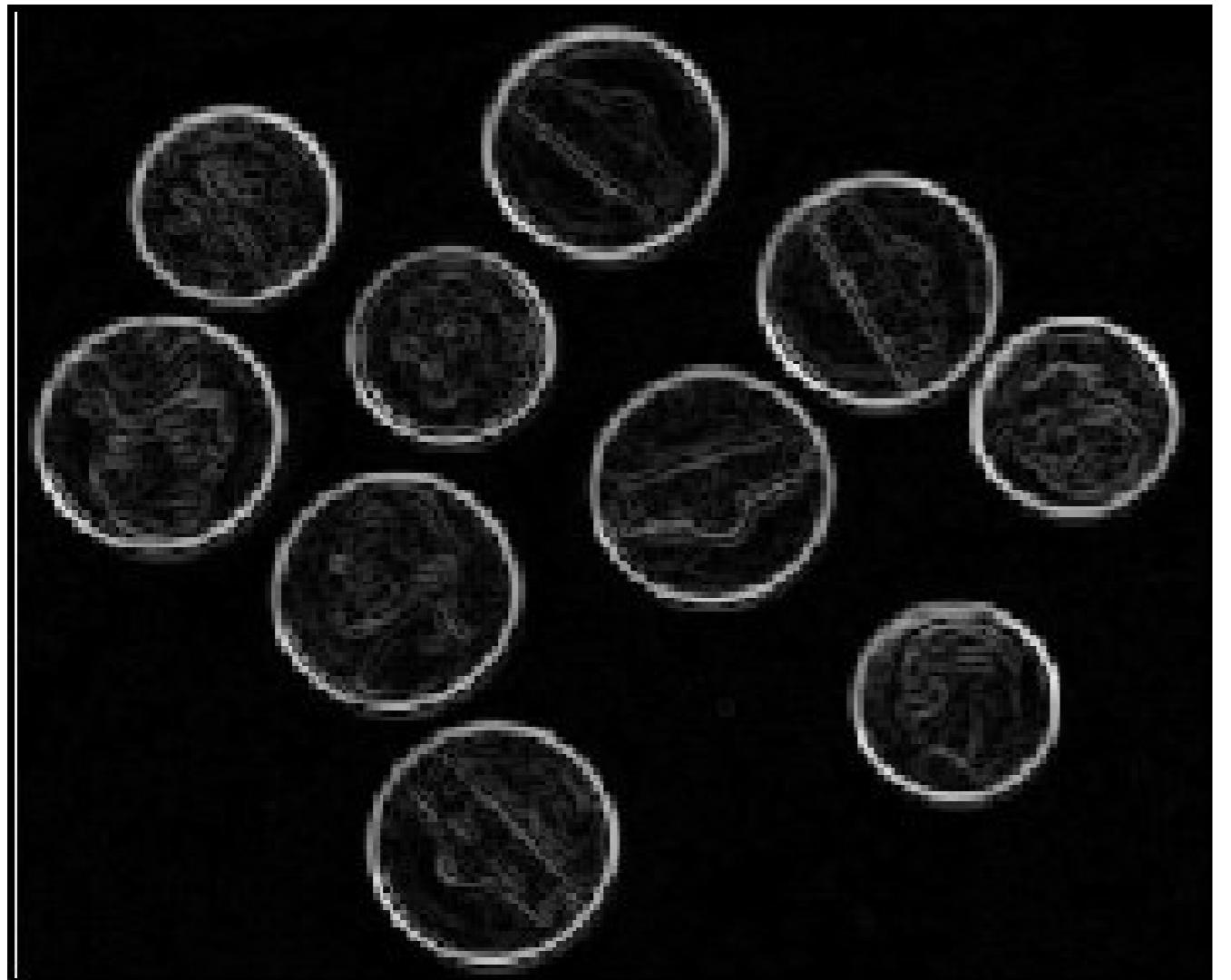


Edges with Canny

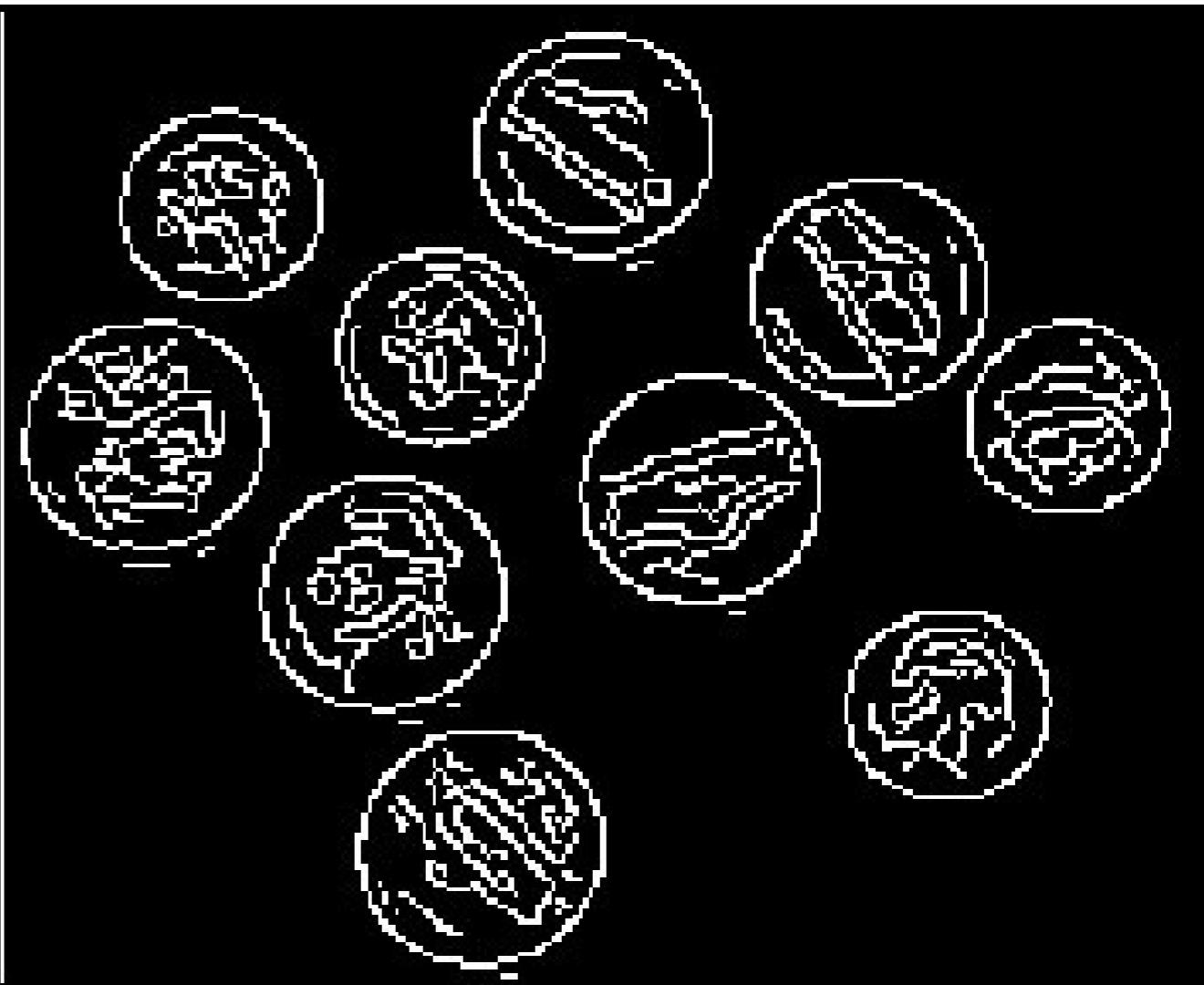


Edge detection

Edges with Sobel



Edges with Canny



Edge detection

```
from skimage.feature import canny

# Convert image to grayscale
coins = color.rgb2gray(coins)

# Apply Canny detector
canny_edges = canny(coins)

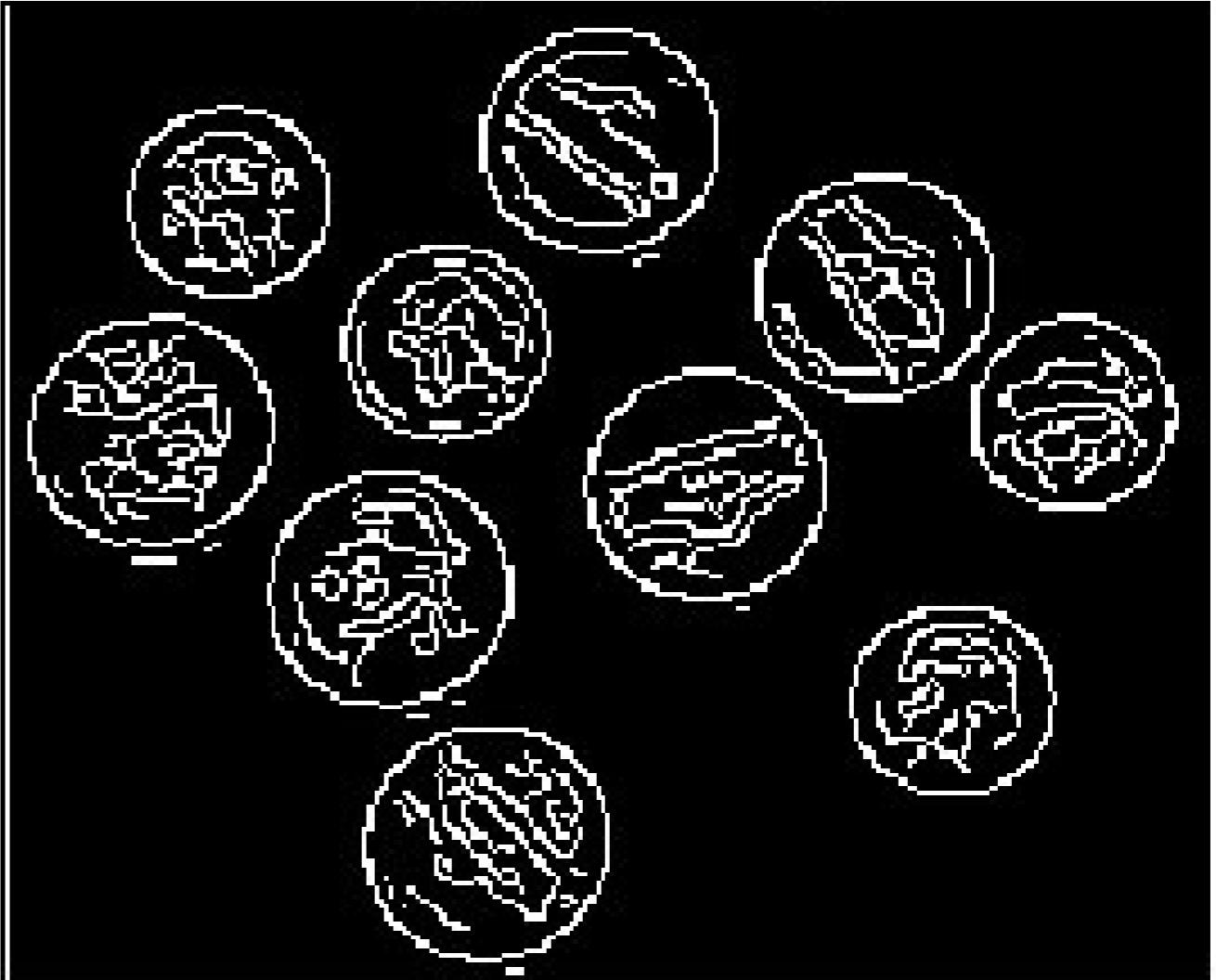
# Show resulted image with edges
show_image(canny_edges, "Edges with Canny")
```

Edge detection

Original



Edges with Canny

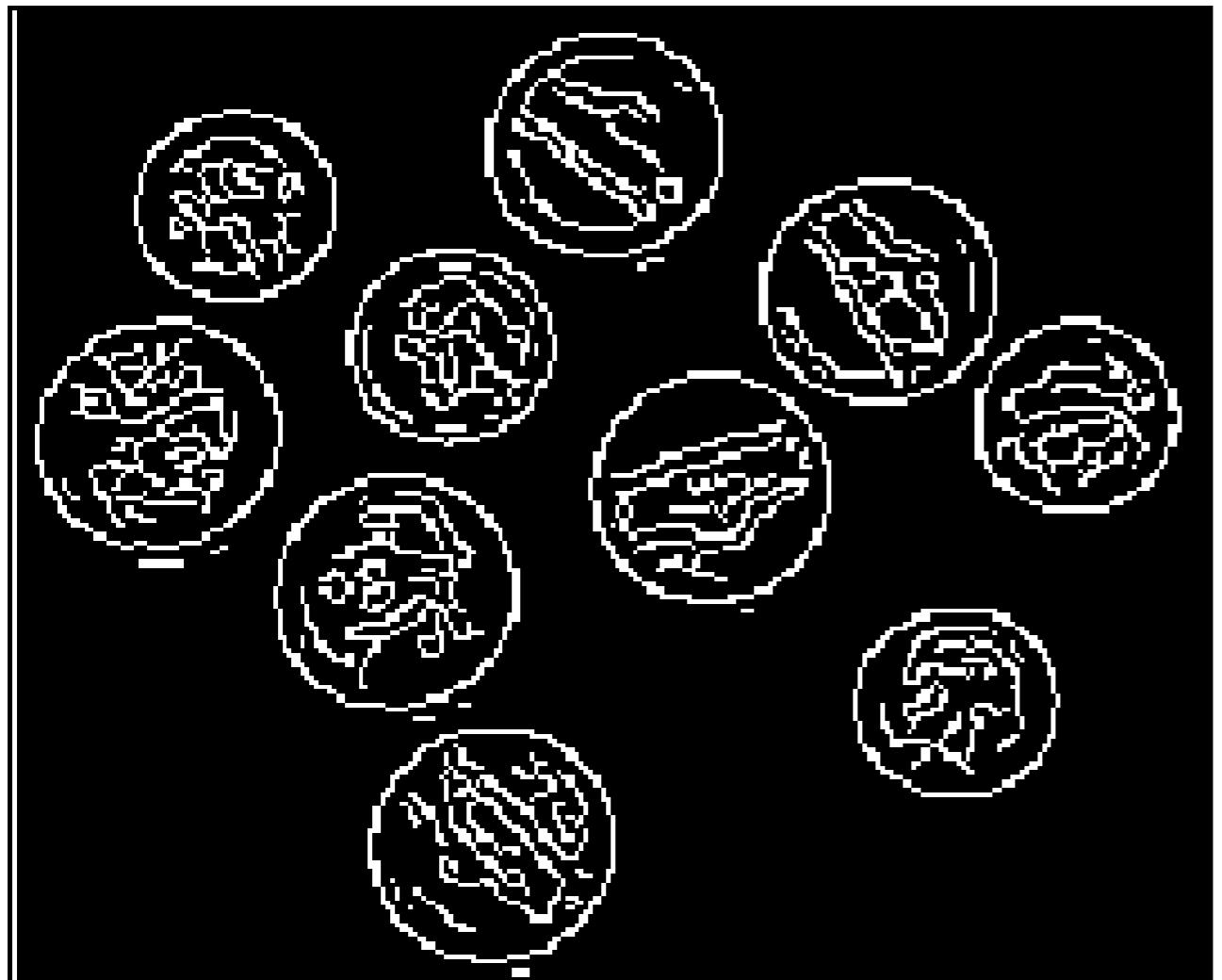


Canny edge detector

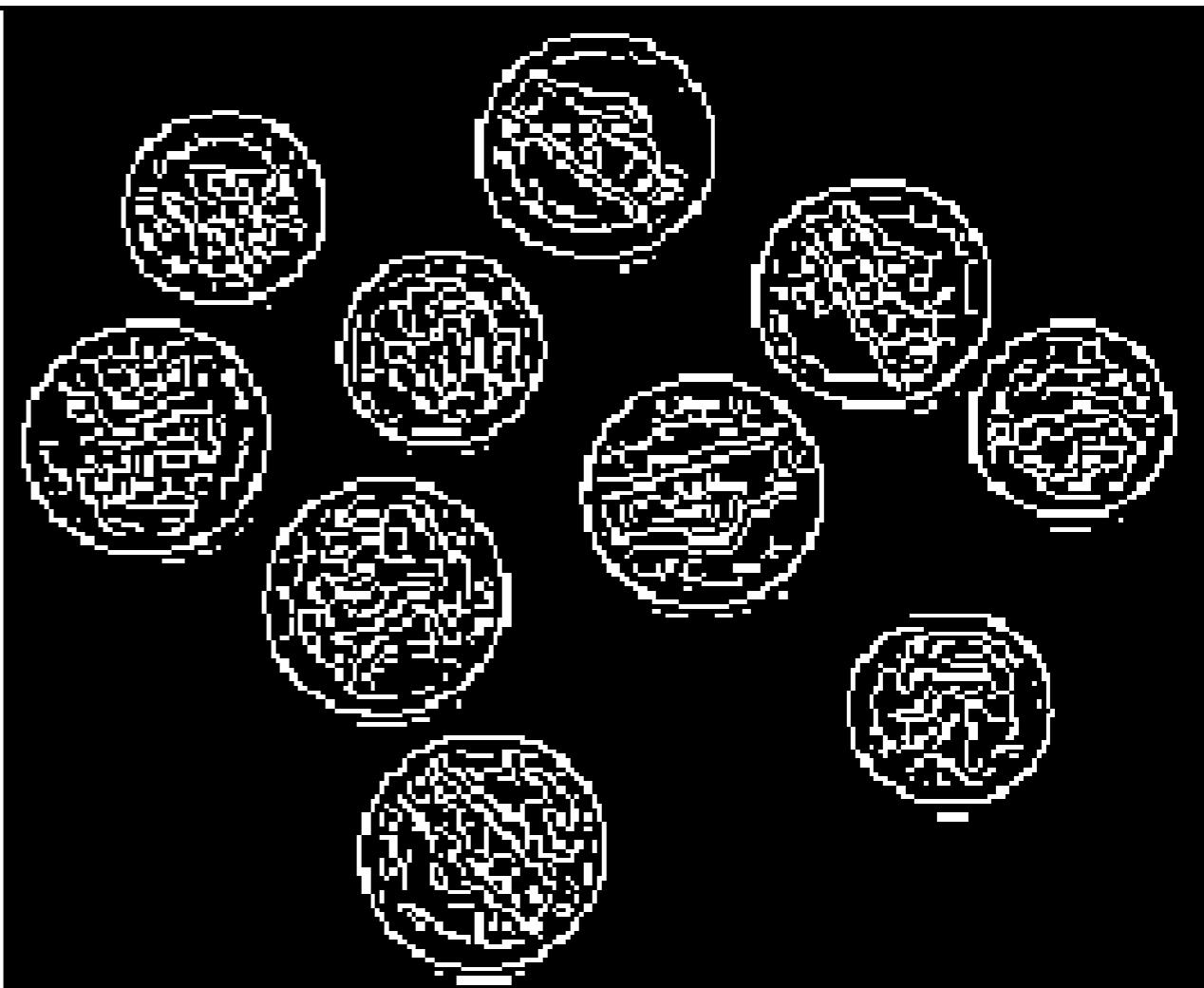
```
# Apply Canny detector with a sigma of 0.5  
canny_edges_0_5 = canny(coins, sigma=0.5)  
  
# Show resulted images with edges  
show_image(canny_edges, "Sigma of 1")  
show_image(canny_edges_0_5, "Sigma of 0.5")
```

Canny edge detector

Sigma of 1



Sigma of 0.5



Let's practice!

IMAGE PROCESSING IN PYTHON

Right around the corner

IMAGE PROCESSING IN PYTHON



Rebeca Gonzalez
Data Engineer

Corner detection

Original image

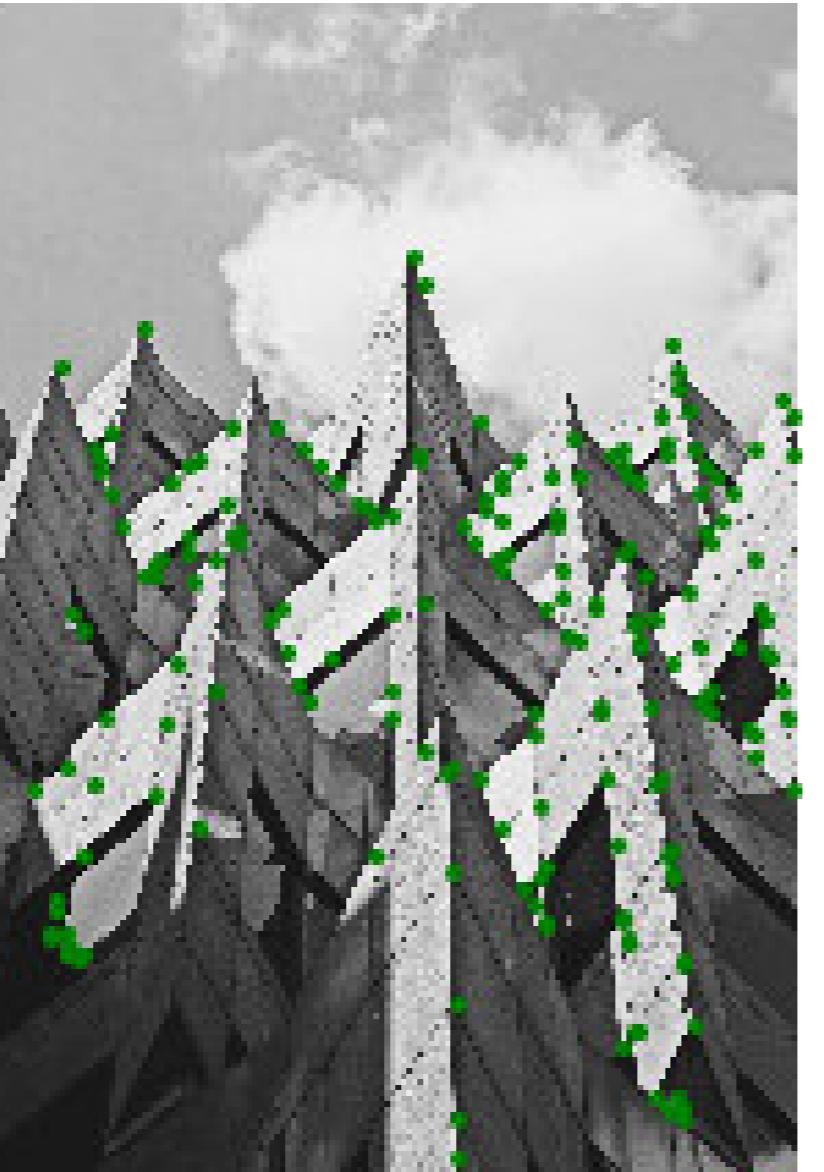


Corners detected



Points of interest

Points of interest

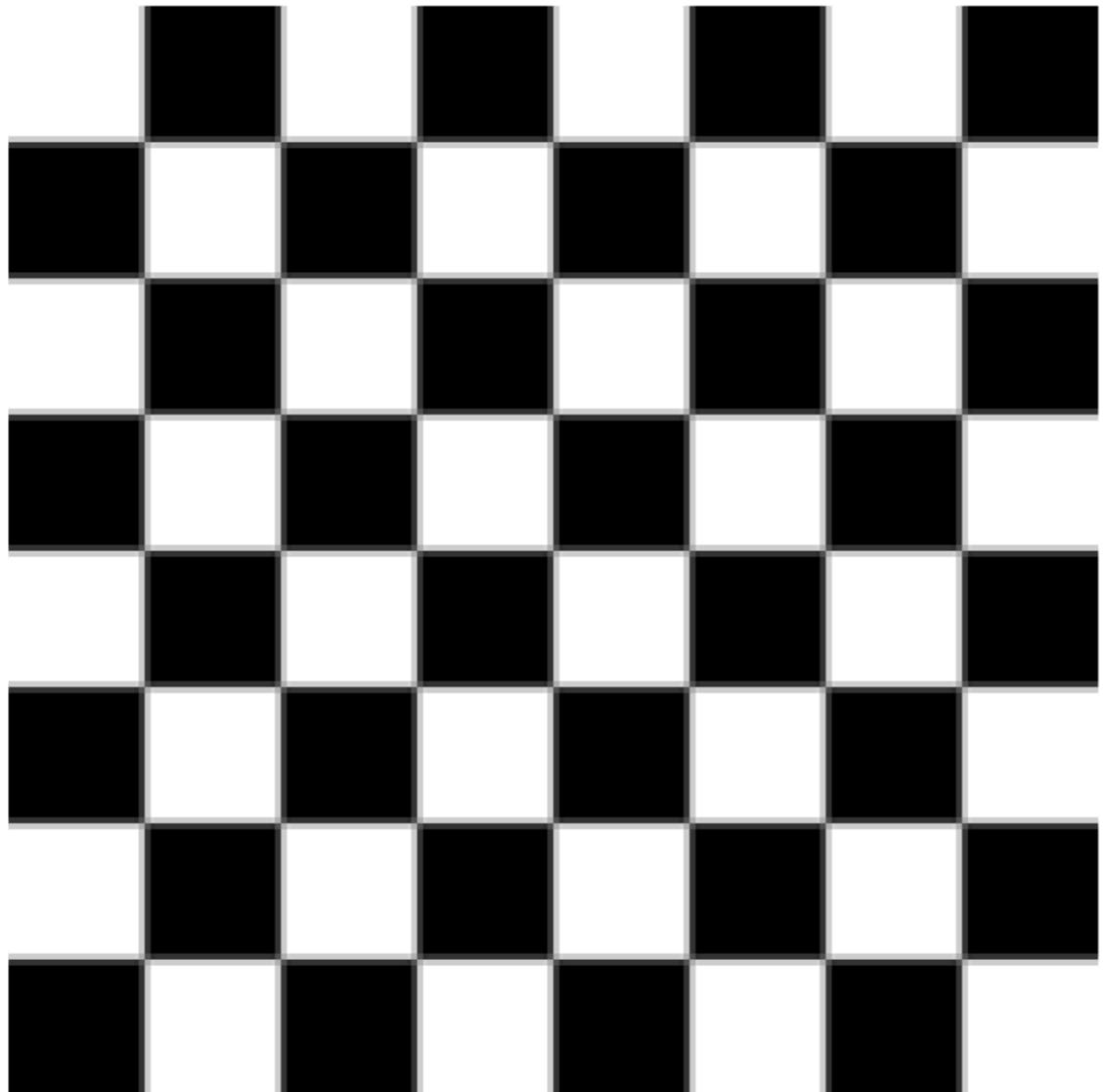


Edges with Sobel

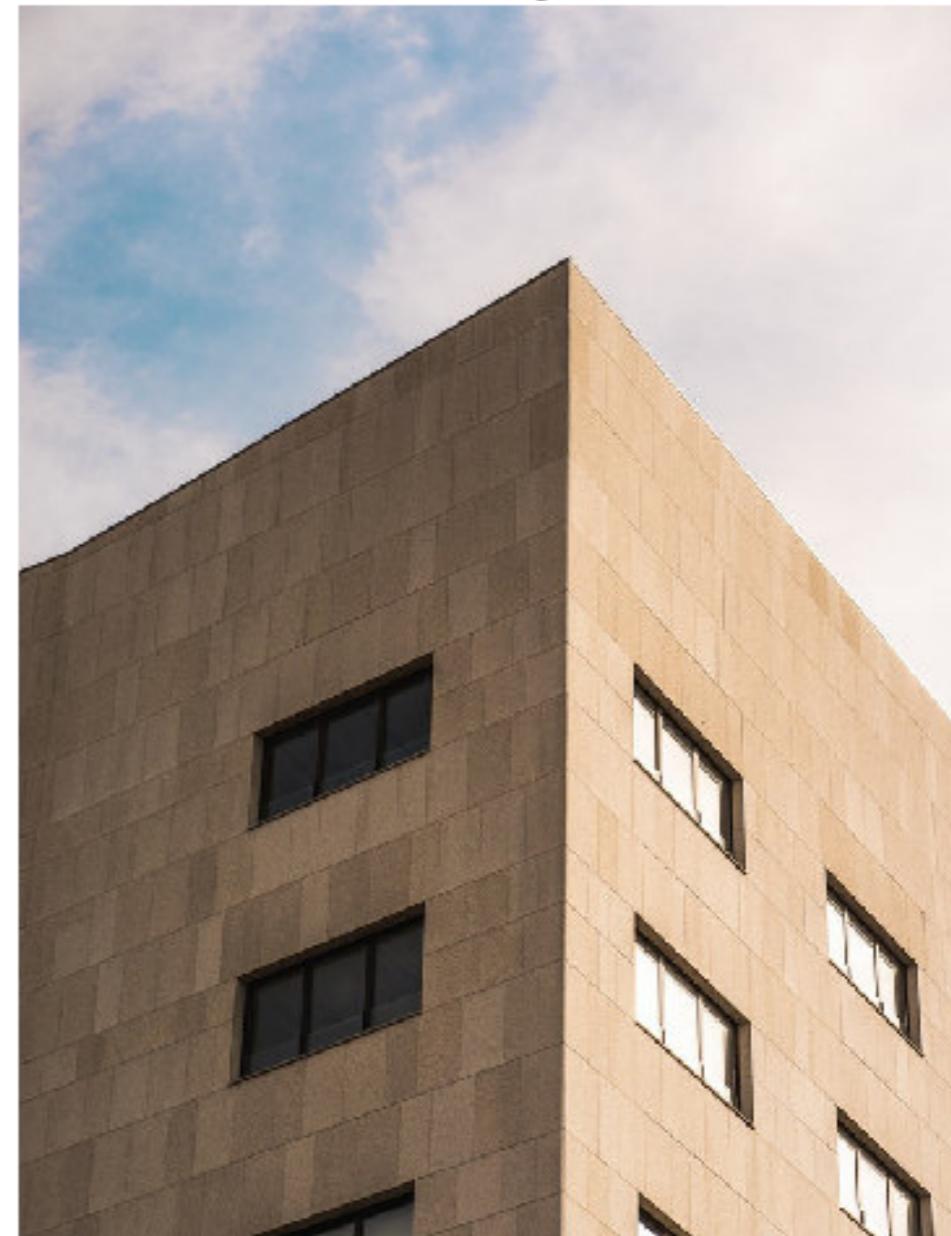


Corners

Checkerboard

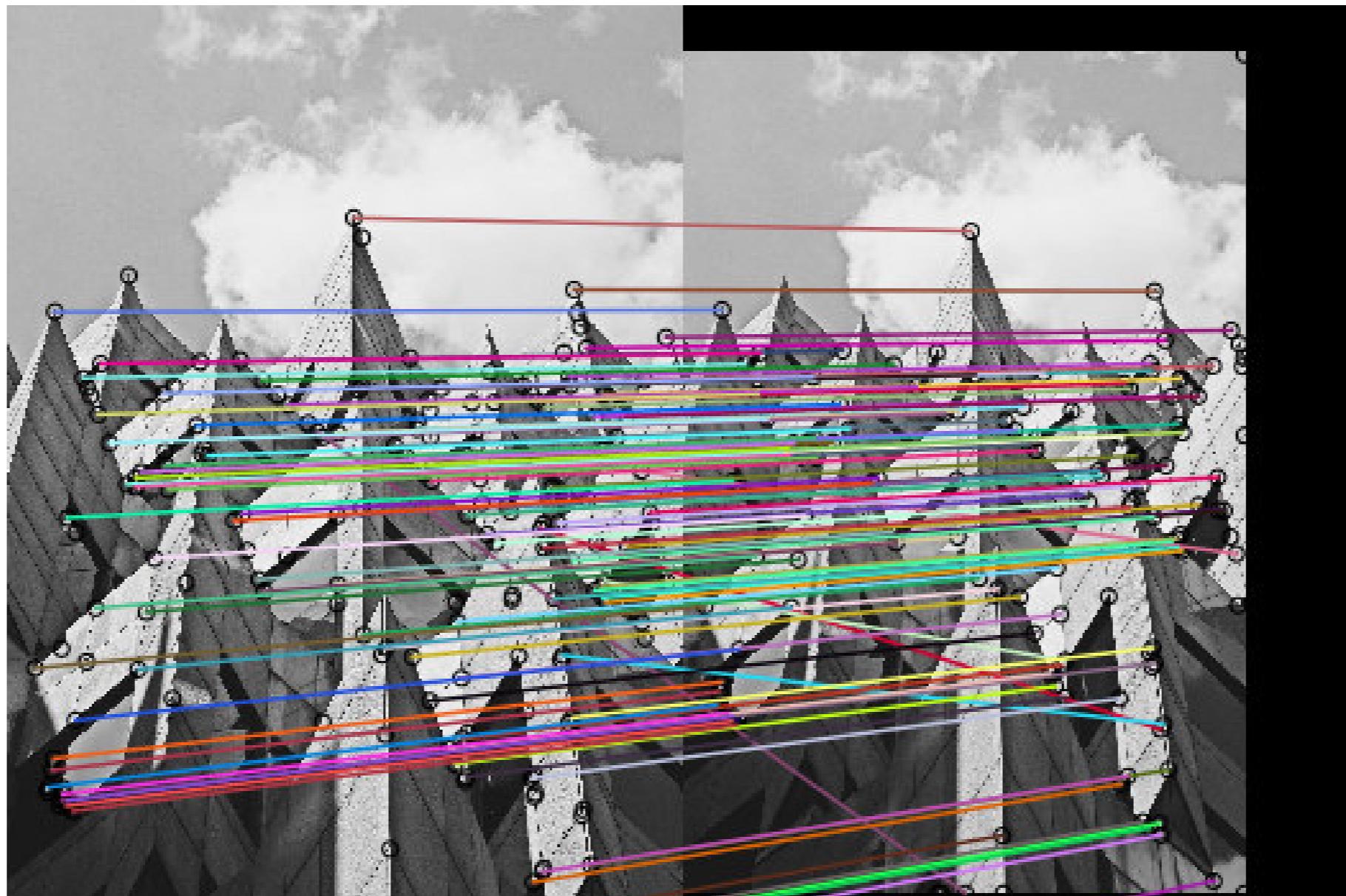


Building



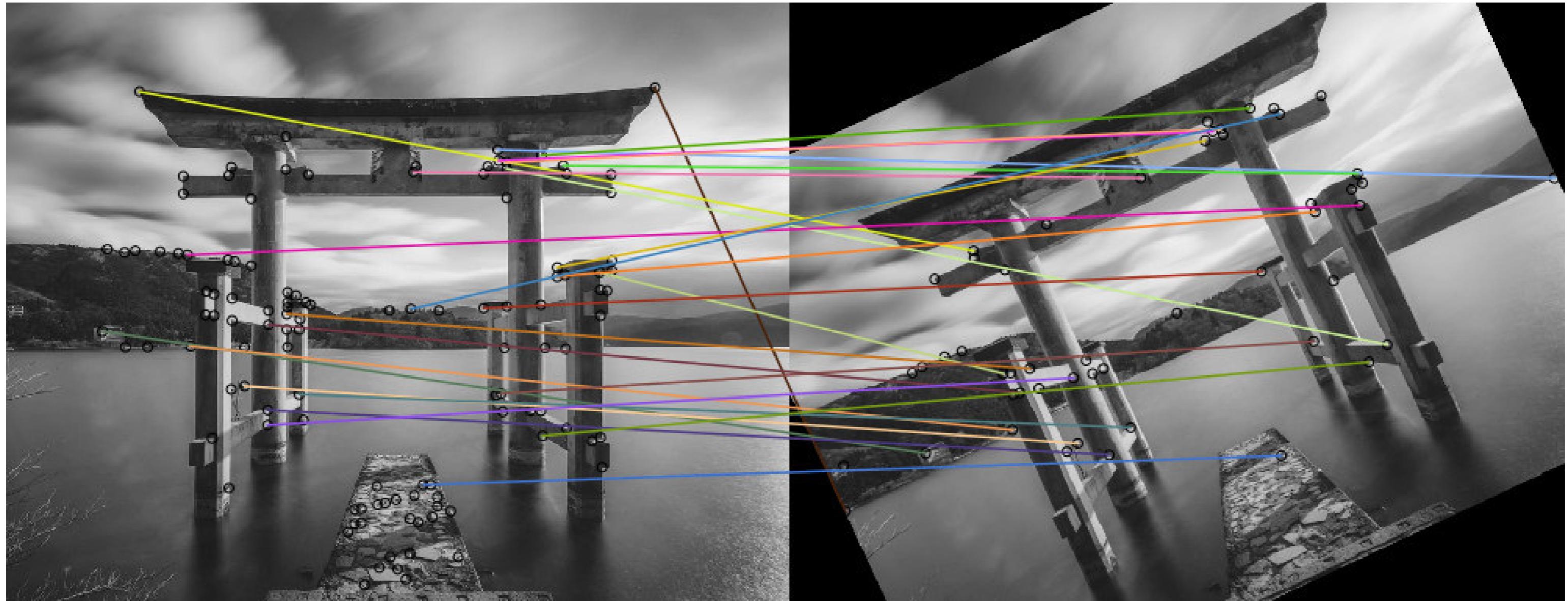
Matching corners

Original Image vs. Transformed Image



Matching corners

Original Image vs. Transformed Image



Harris corner detector

Original image



Corners detected



Harris corner detector



Harris corner detector

```
from skimage.feature import corner_harris

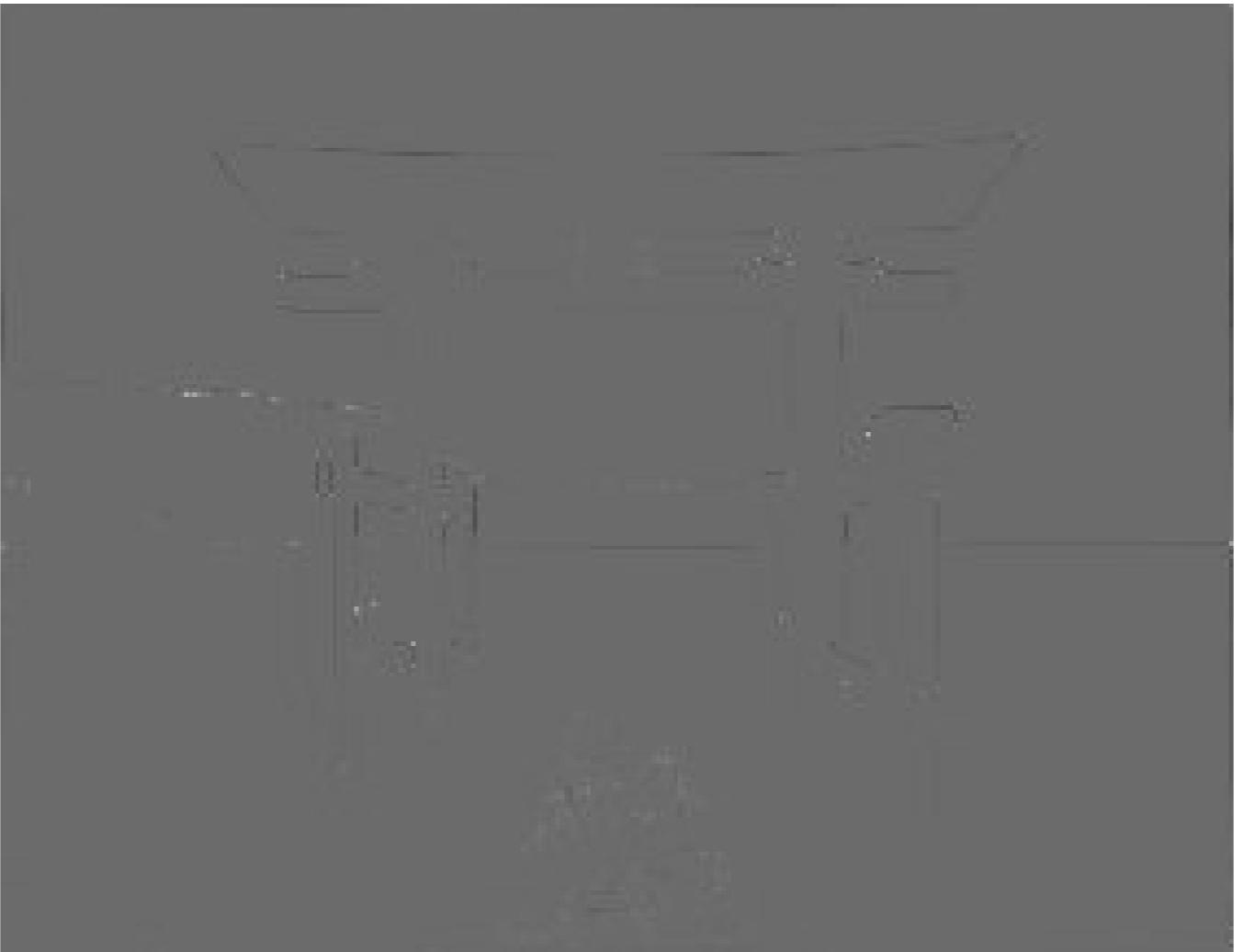
# Convert image to grayscale
image = rgb2gray(image)

# Apply the Harris corner detector on the image
measure_image = corner_harris(image)

# Show the Harris response image
show_image(measure_image)
```

Harris corner detector

Harris response image



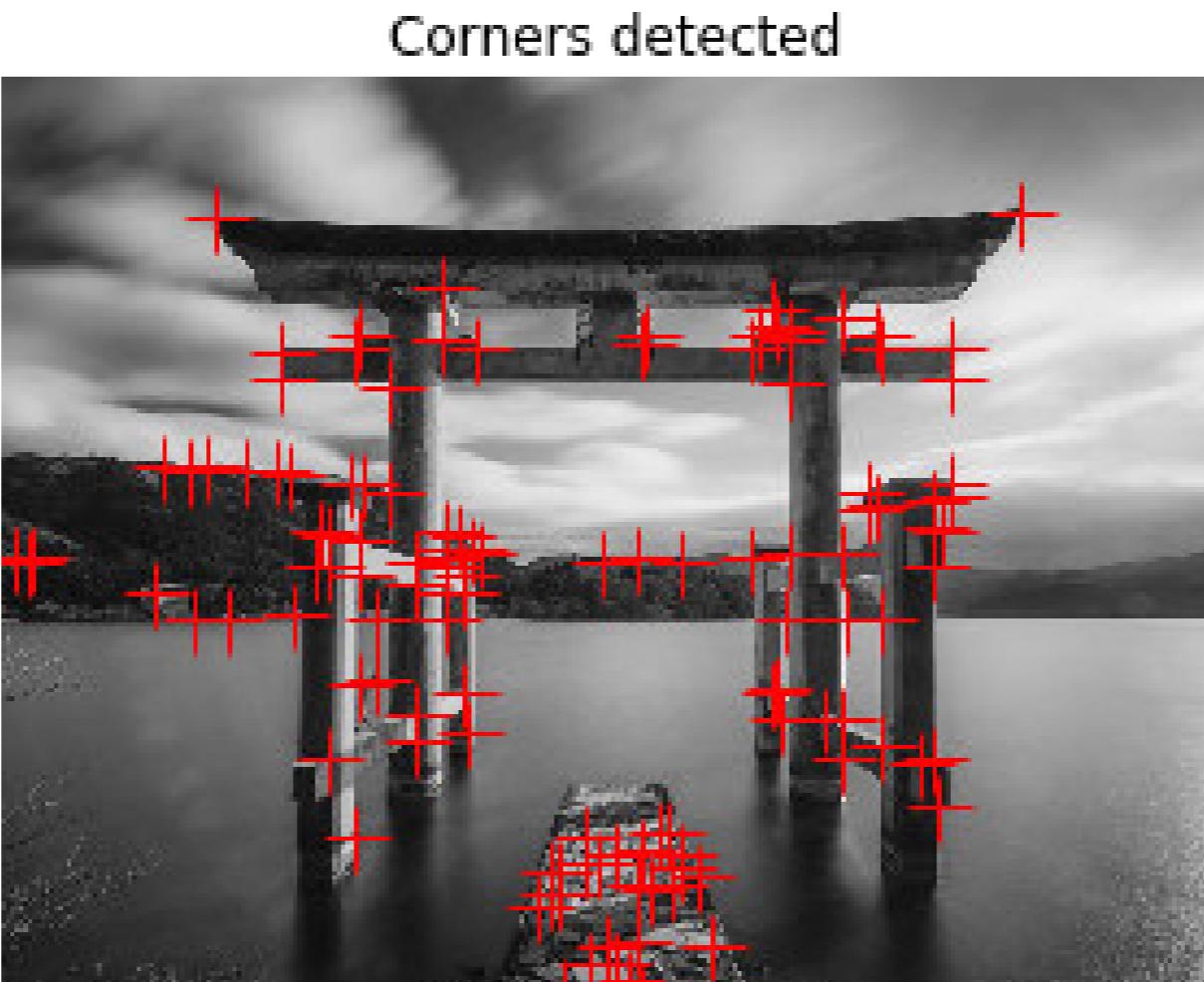
Harris corner detector

```
# Finds the coordinates of the corners  
coords = corner_peaks(corner_harris(image), min_distance=5)  
  
print("A total of", len(coords), "corners were detected.")
```

A total of 122 corners were found from measure response image.

Corners detected

```
# Show image with marks in detected corners  
show_image_with_detected_corners(image, coords)
```



Show image with contours

```
def show_image_with_corners(image, coords, title="Corners detected"):  
    plt.imshow(image, interpolation='nearest', cmap='gray')  
    plt.title(title)  
    plt.plot(coords[:, 1], coords[:, 0], '+r', markersize=15)  
    plt.axis('off')  
    plt.show()
```

Let's practice!

IMAGE PROCESSING IN PYTHON

Face detection

IMAGE PROCESSING IN PYTHON



Rebeca Gonzalez
Data Engineer

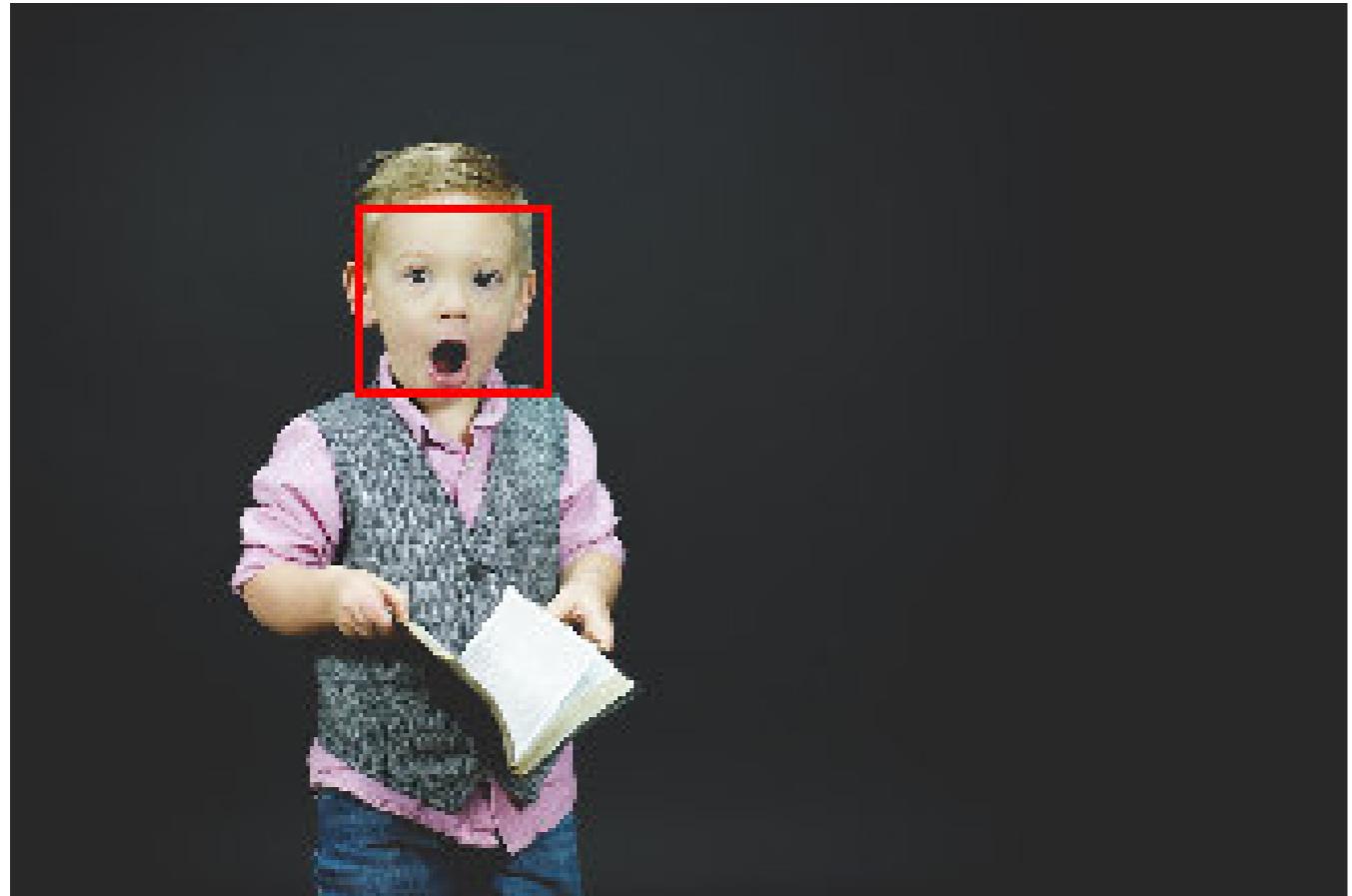
Face detection use cases

- Filters
- Auto focus
- Recommendations
- Blur for privacy protection
- To recognize emotions later on

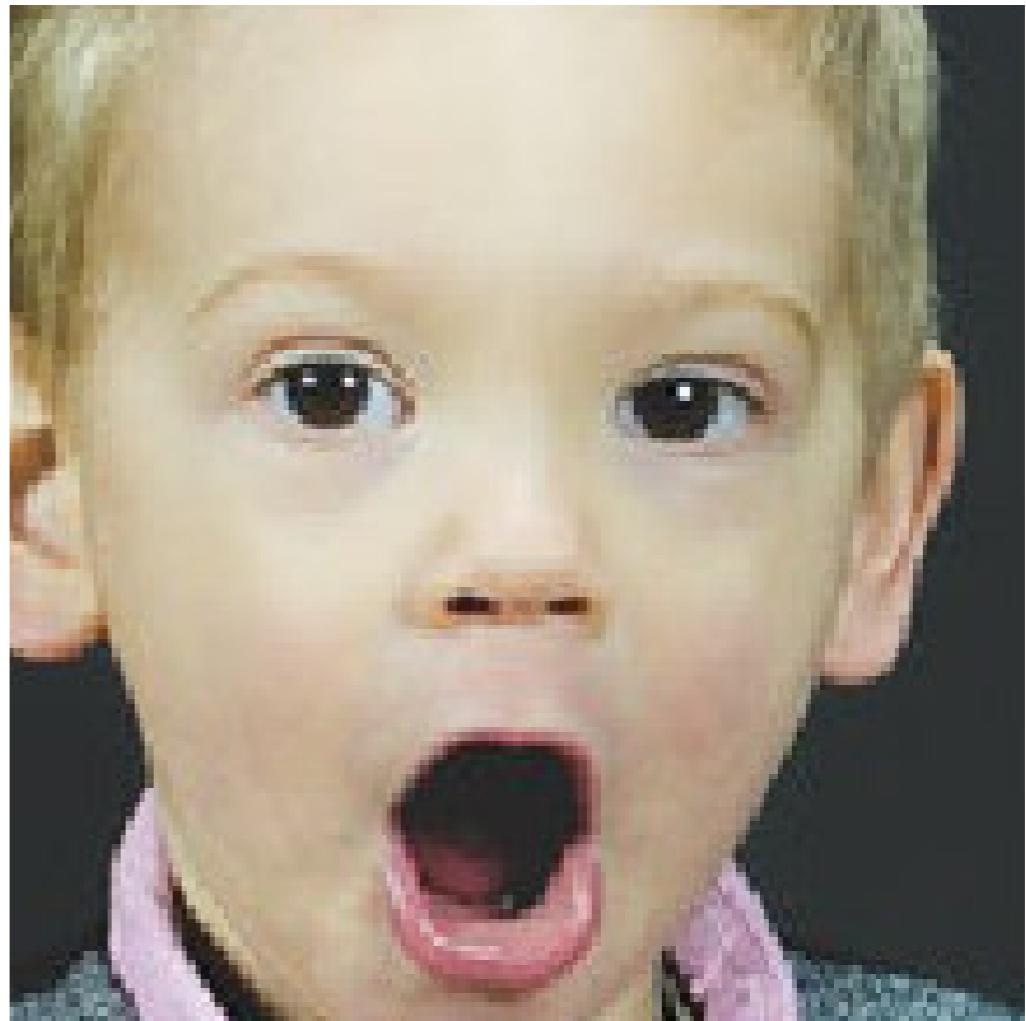


Detecting faces with scikit-image

Face image



Face detected



Detecting faces with scikit-image

```
# Import the classifier class
from skimage.feature import Cascade

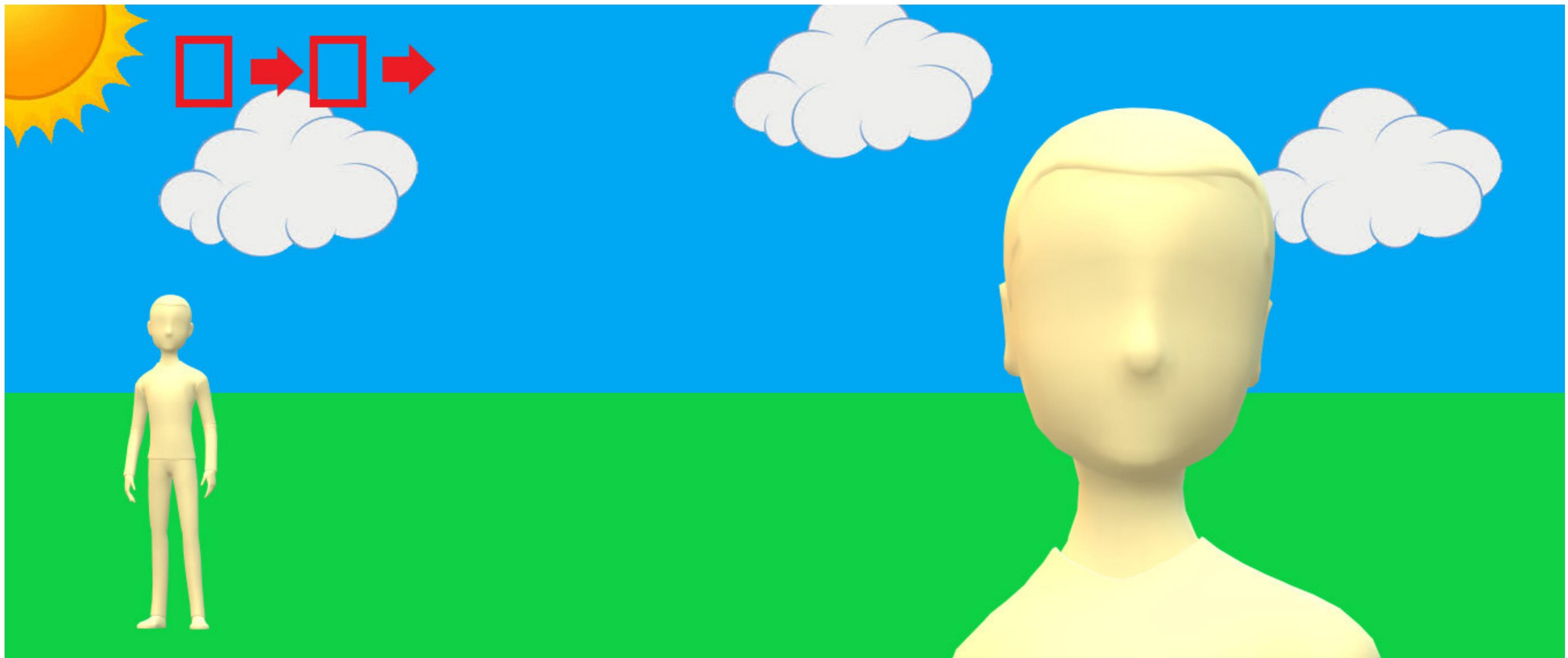
# Load the trained file from the module root.
trained_file = data.lbp_frontal_face_cascade_filename()

# Initialize the detector cascade.
detector = Cascade(trained_file)
```

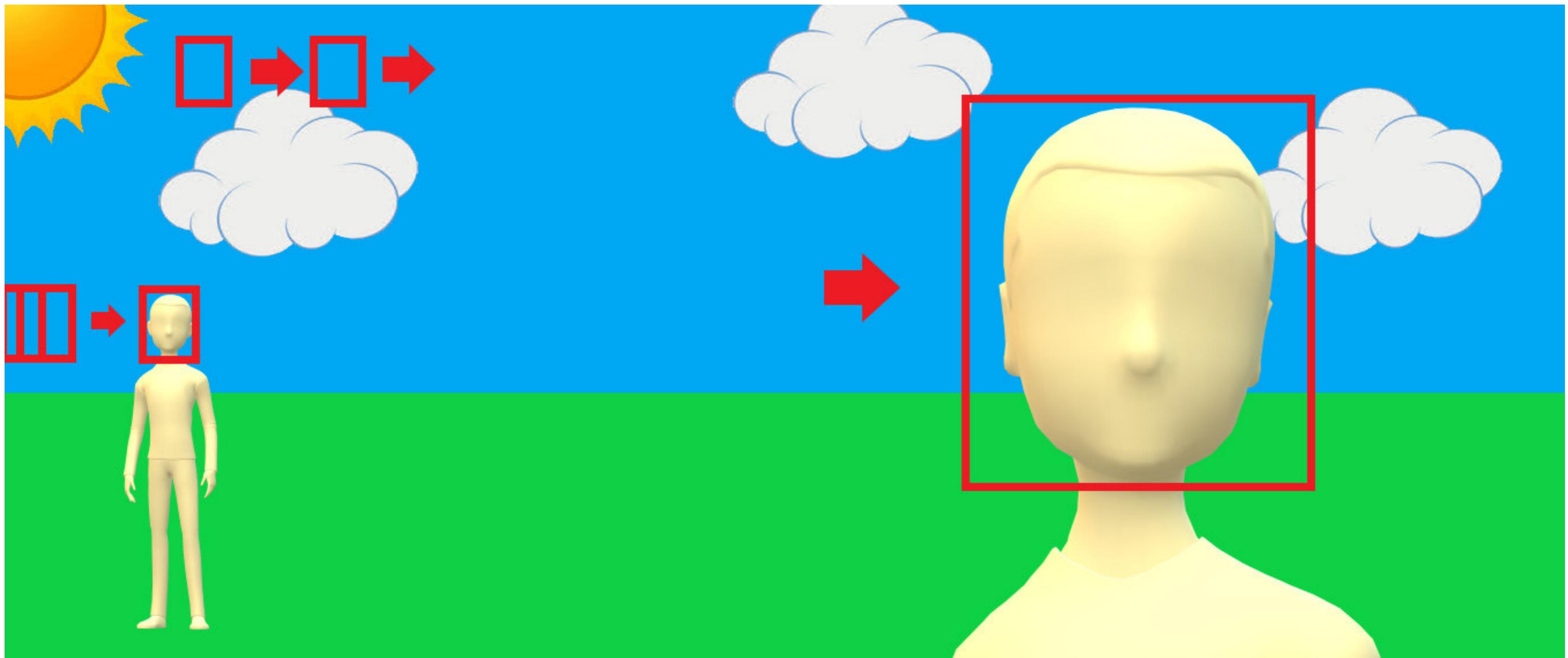
Let's try it



Detecting faces



Detecting faces



Detecting faces

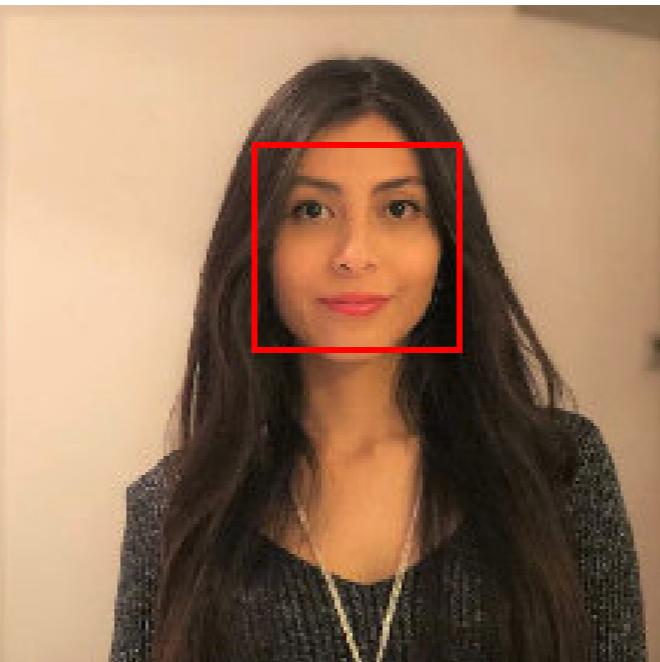
```
# Apply detector on the image
detected = detector.detect_multi_scale(img=image,
                                         scale_factor=1.2,
                                         step_ratio=1,
                                         min_size=(10, 10),
                                         max_size=(200, 200))
```

Detected faces

```
print(detected)  
# Show image with detected face marked  
show_detected_face(image, detected)
```

```
Detected face: [ {'r': 115, 'c': 210, 'width': 167, 'height': 167}]
```

Face image



Show detected faces

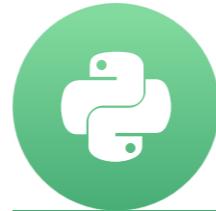
```
def show_detected_face(result, detected, title="Face image"):  
    plt.imshow(result)  
    img_desc = plt.gca()  
    plt.set_cmap('gray')  
    plt.title(title)  
    plt.axis('off')  
  
    for patch in detected:  
        img_desc.add_patch(  
            patches.Rectangle(  
                (patch['c'], patch['r']),  
                patch['width'],  
                patch['height'],  
                fill=False, color='r', linewidth=2)  
        )  
    plt.show()
```

Let's practice!

IMAGE PROCESSING IN PYTHON

Real-world applications

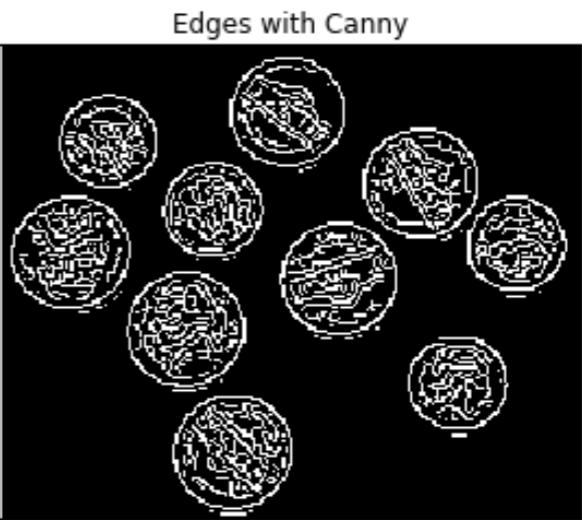
IMAGE PROCESSING IN PYTHON



Rebeca Gonzalez
Data engineer

Applications

- Turning to grayscale before detecting edges/corners
- Reducing noise and restoring images
- Blurring faces detected
- Approximation of objects' sizes



Privacy protection



Privacy protection

```
# Import Cascade of classifiers and gaussian filter
from skimage.feature import Cascade
from skimage.filters import gaussian
```

Privacy protection

```
# Detect the faces
detected = detector.detect_multi_scale(img=image,
                                         scale_factor=1.2, step_ratio=1,
                                         min_size=(50, 50), max_size=(100, 100))

# For each detected face
for d in detected:
    # Obtain the face cropped from detected coordinates
    face = getFace(d)
```

Privacy protection

```
def getFace(d):
    ''' Extracts the face rectangle from the image using the
    coordinates of the detected.'''
    # X and Y starting points of the face rectangle
    x, y = d['r'], d['c']

    # The width and height of the face rectangle
    width, height = d['r'] + d['width'], d['c'] + d['height']

    # Extract the detected face
    face= image[x:width, y:height]
    return face
```

Privacy protection

```
# Detect the faces
detected = detector.detect_multi_scale(img=image,
                                         scale_factor=1.2, step_ratio=1,
                                         min_size=(50, 50), max_size=(100, 100))

# For each detected face
for d in detected:
    # Obtain the face cropped from detected coordinates
    face = getFace(d)

    # Apply gaussian filter to extracted face
    gaussian_face = gaussian(face, multichannel=True, sigma = 10)

    # Merge this blurry face to our final image and show it
    resulting_image = mergeBlurryFace(image, gaussian_face)
```

Privacy protection

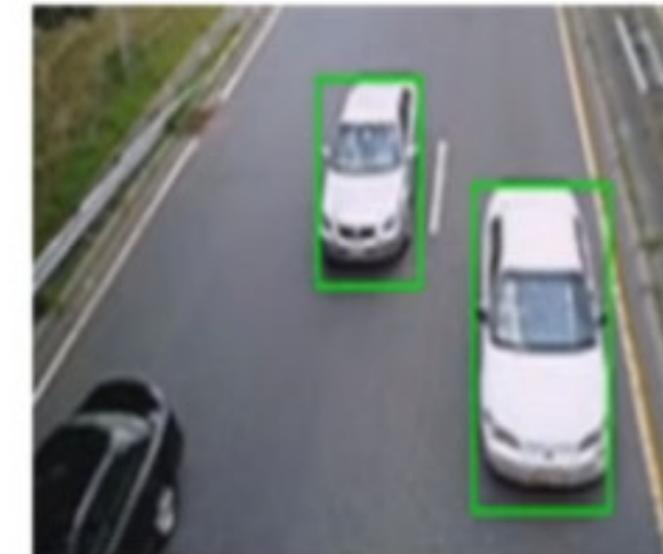
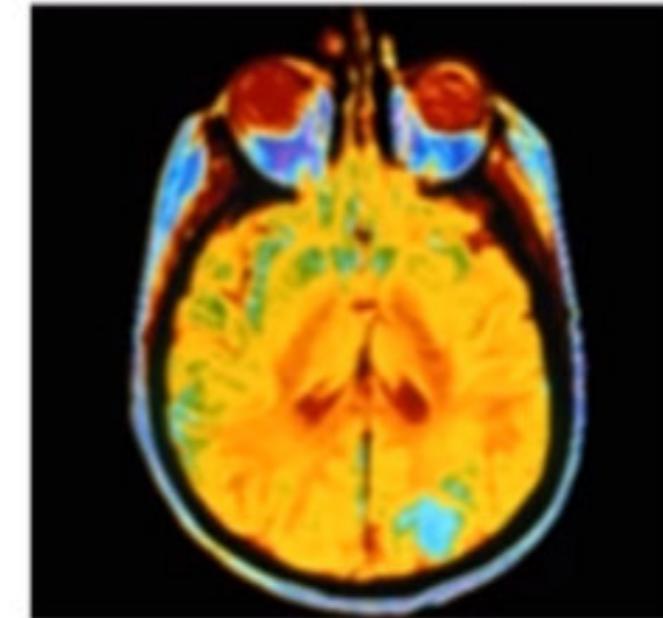
```
def mergeBlurryFace(original, gaussian_image):  
    # X and Y starting points of the face rectangle  
    x, y = d['r'], d['c']  
    # The width and height of the face rectangle  
    width, height = d['r'] + d['width'], d['c'] + d['height']  
  
    original[ x:width, y:height] = gaussian_image  
    return original
```

Privacy protection

Blurred faces



More cases



Let's practice!

IMAGE PROCESSING IN PYTHON

Amazing work!

IMAGE PROCESSING IN PYTHON



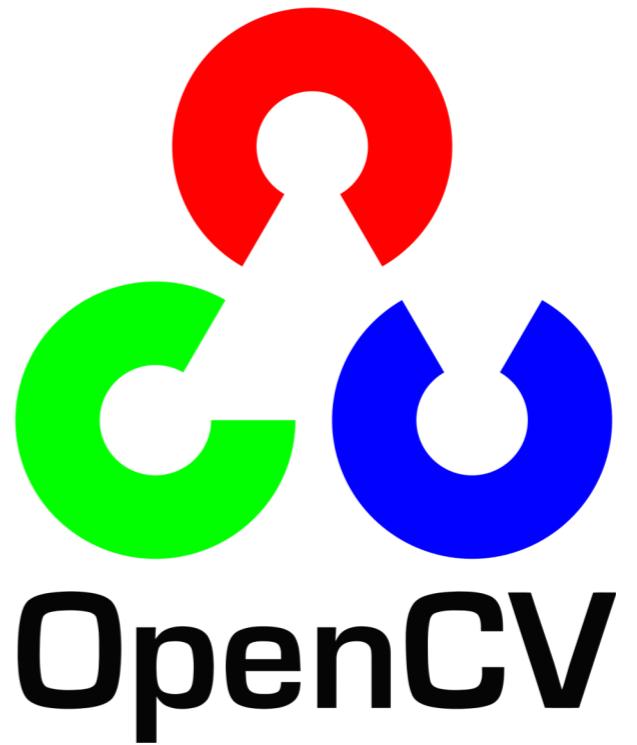
Rebeca Gonzalez
Data Engineer

Recap: What you have learned

- Improved contrast
- Restored images
- Applied filters
- Rotated, flipped and resized!
- Segmented: supervised and unsupervised
- Applied morphological operators
- Created and reduced noise
- Detected edges, corners and faces
- And mixed them up to solve problems!

What's next?

- Tinting gray scale images
- Matching
- Approximation
- Many others!



Congrats!

IMAGE PROCESSING IN PYTHON