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

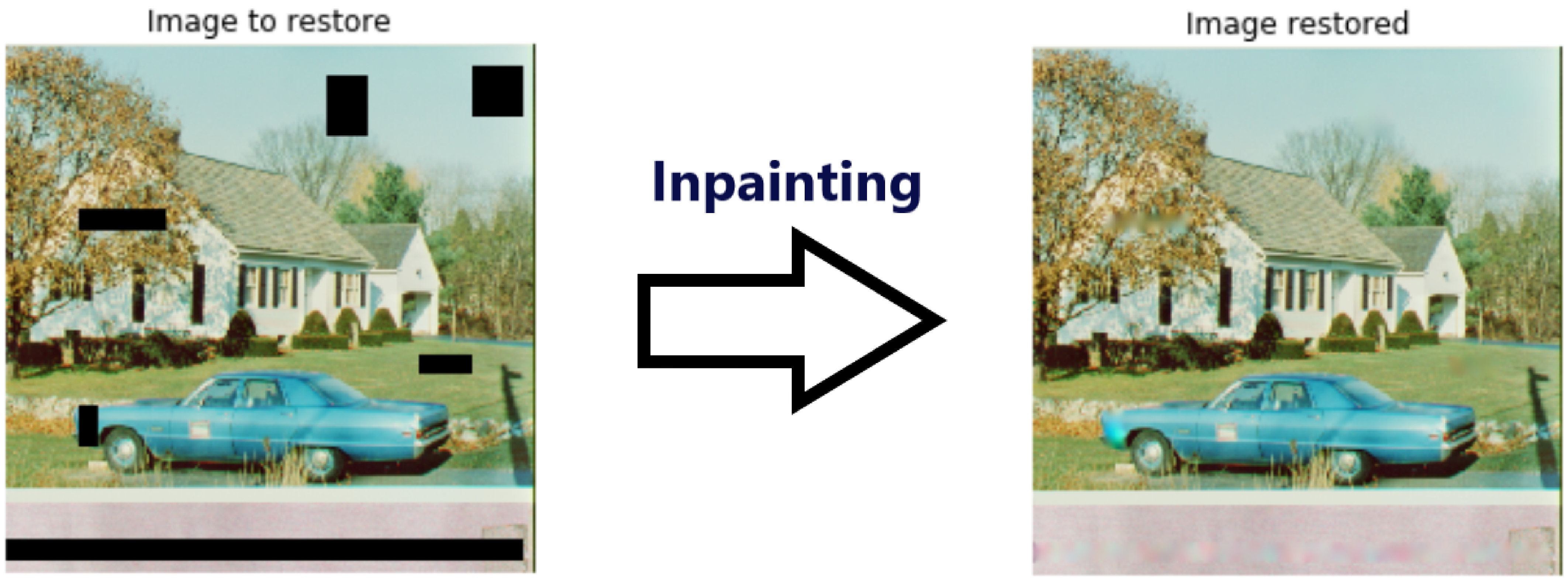


Image reconstruction

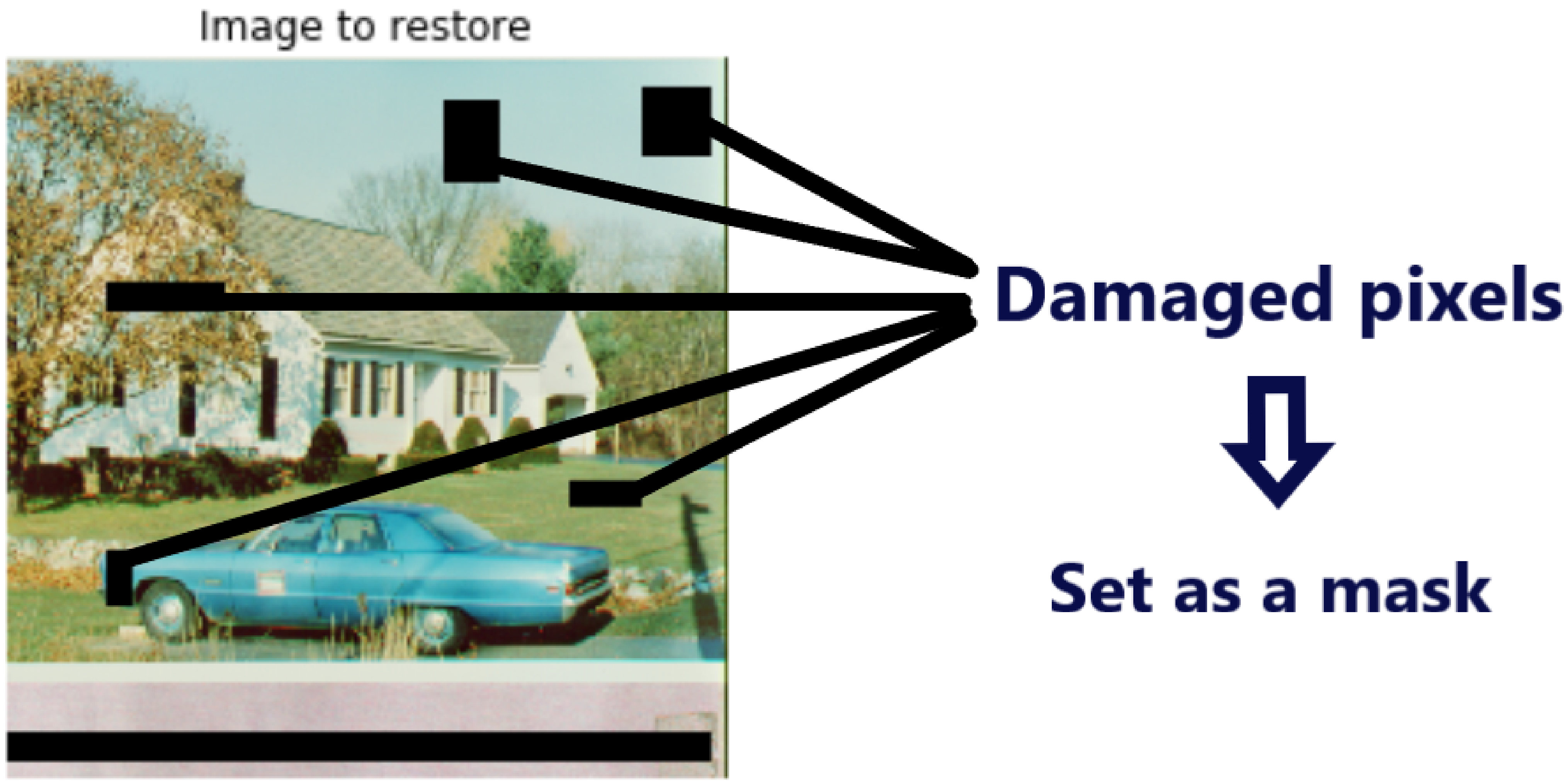


Image reconstruction in scikit-image

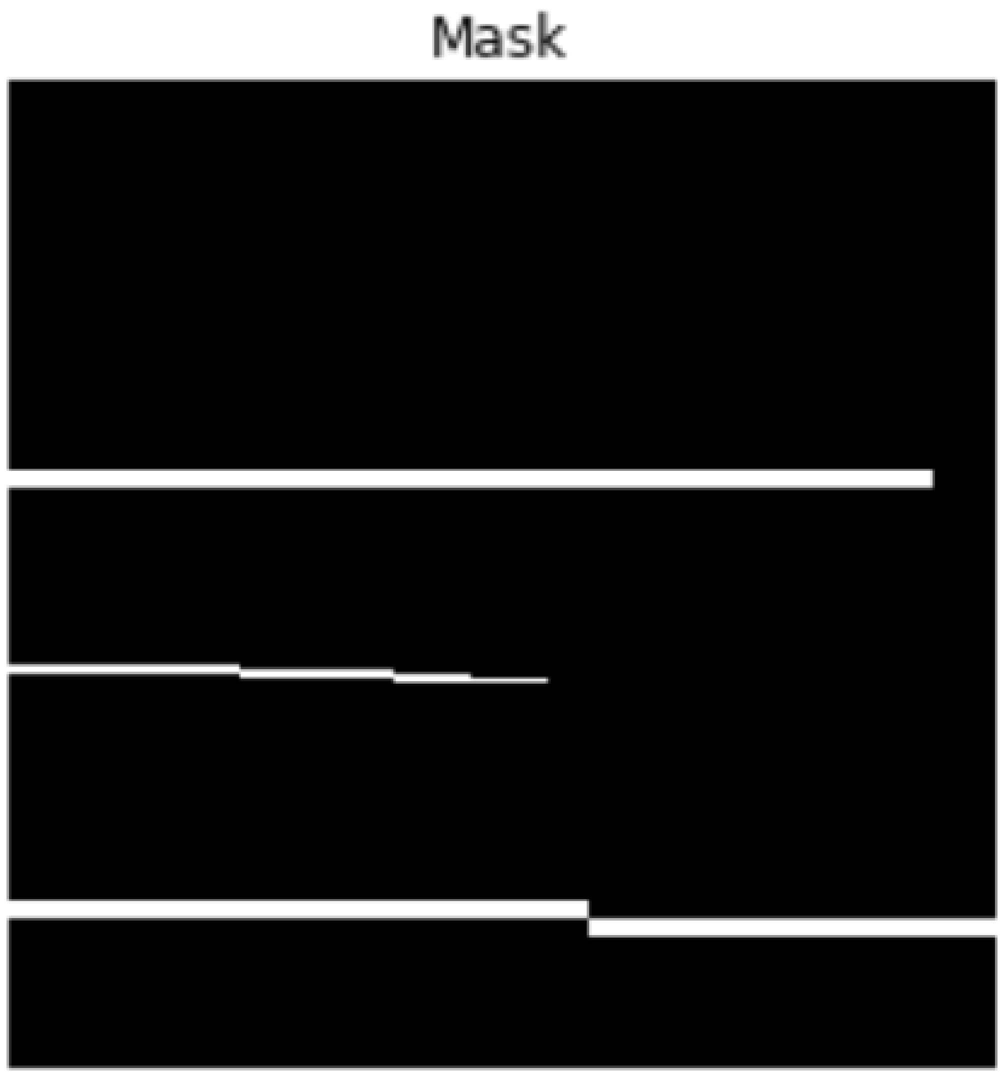
```python
from skimage.restoration import inpaint

# 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')
```

Masks

The mask is a black and white image with patches that have the position of the image bits that have been corrupted.



Masks

```python
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!

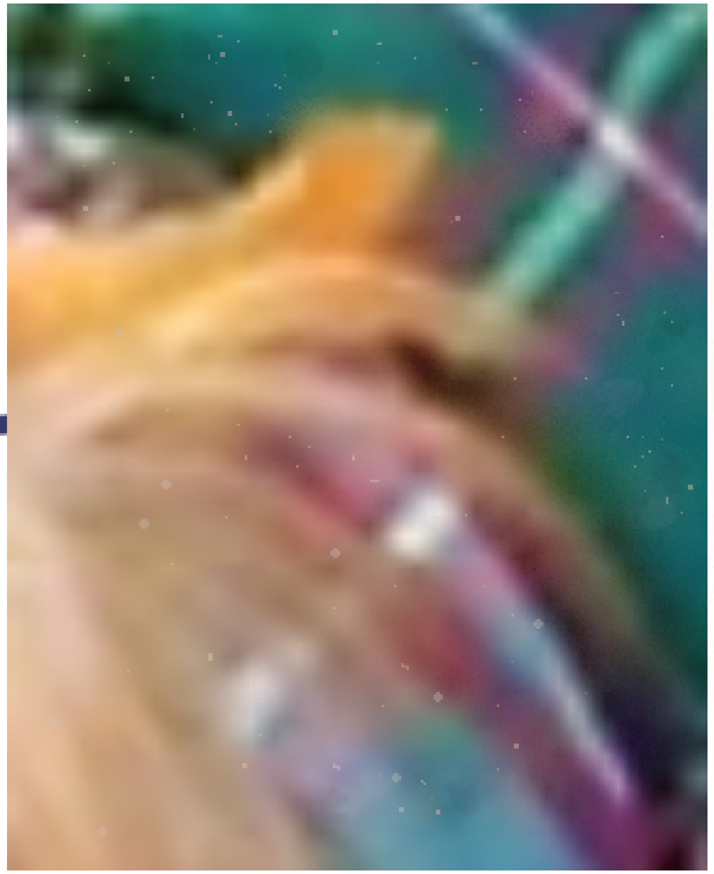
IMAGE PROCESSING IN PYTHON

Noise

IMAGE PROCESSING IN PYTHON

Rebeca Gonzalez
Data Engineer

Noise



Noise

Noise is the result of errors in the image acquisition process that result in pixel values that do not reflect the true intensities of the real scene.

Apply noise in scikit-image

```python
# 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)

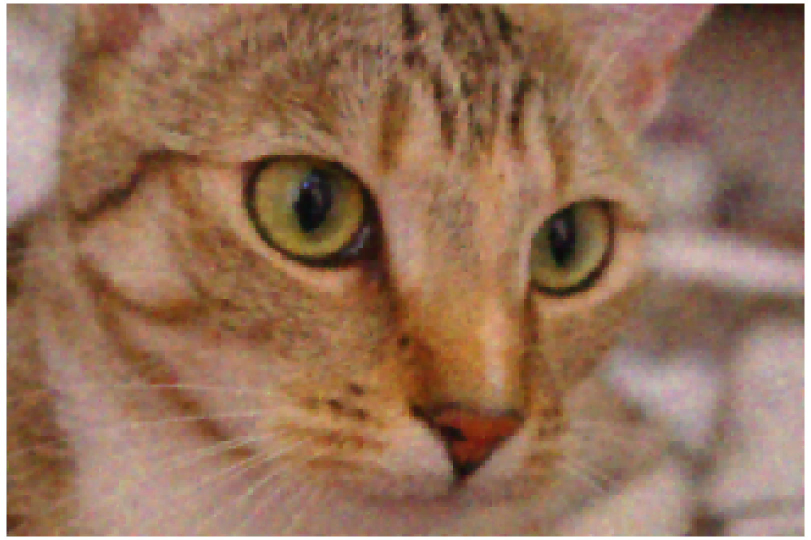
This filter tries to minimize the total variation of the image. It tends to produce “cartoon-like” images, that is, piecewise-constant images.

- Bilateral

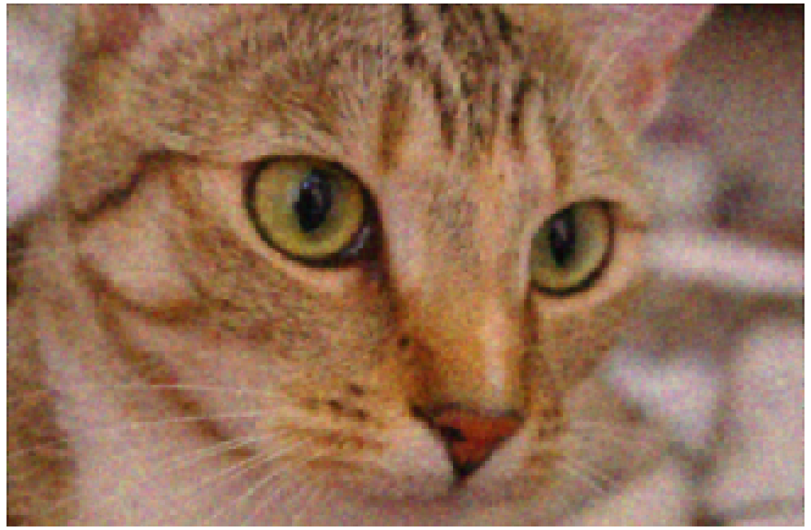
Bilateral filtering smooths images while preserving edges. It replaces the intensity of each pixel with a weighted average of intensity values from nearby pixels.

- Wavelet denoising
- Non-local means denoising

TV denoising



Bilateral 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')
```

The greater the weight, the more denoising but it could also make the image smoother.

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!

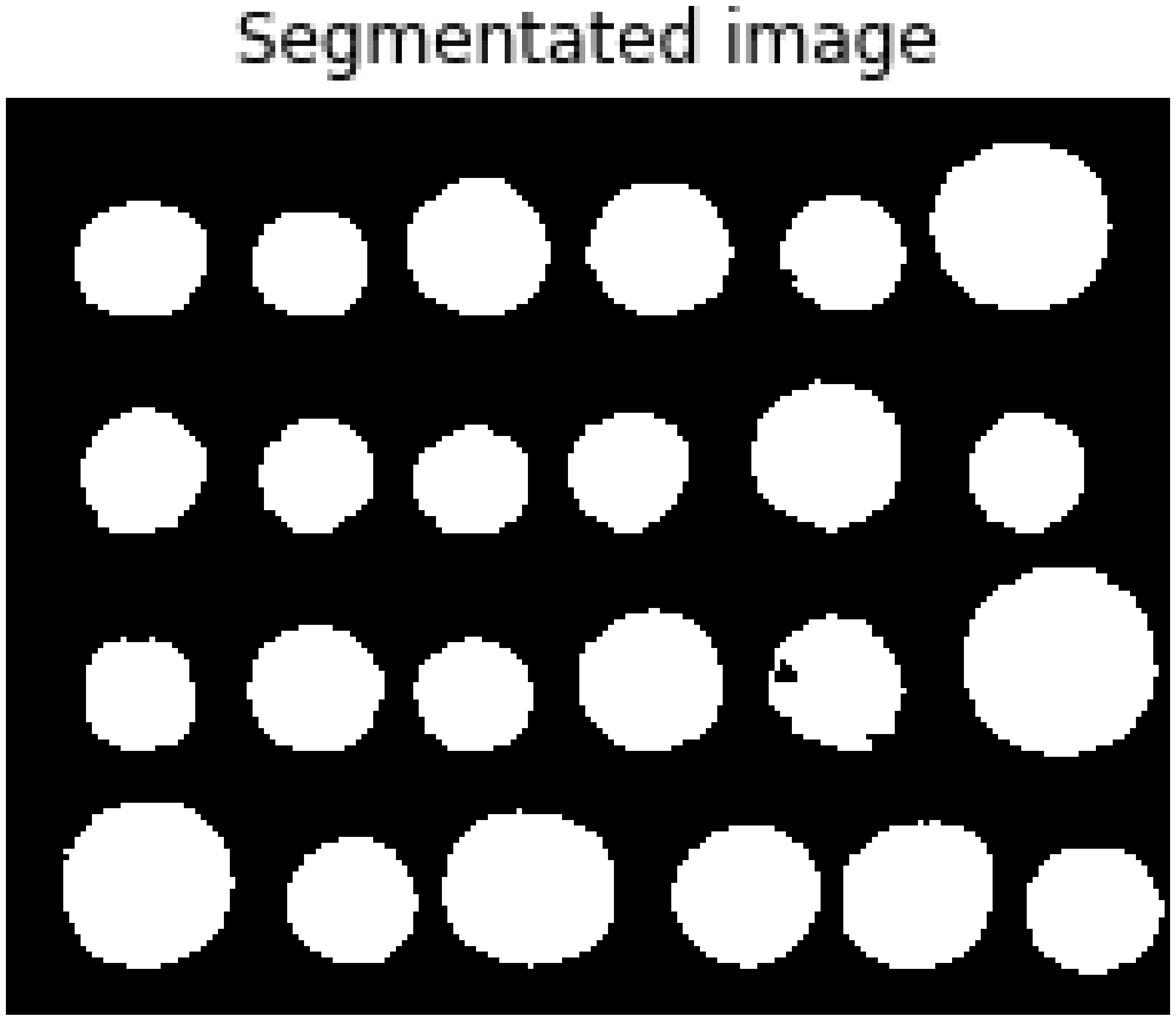
IMAGE PROCESSING IN PYTHON

Superpixels & segmentation

IMAGE PROCESSING IN PYTHON

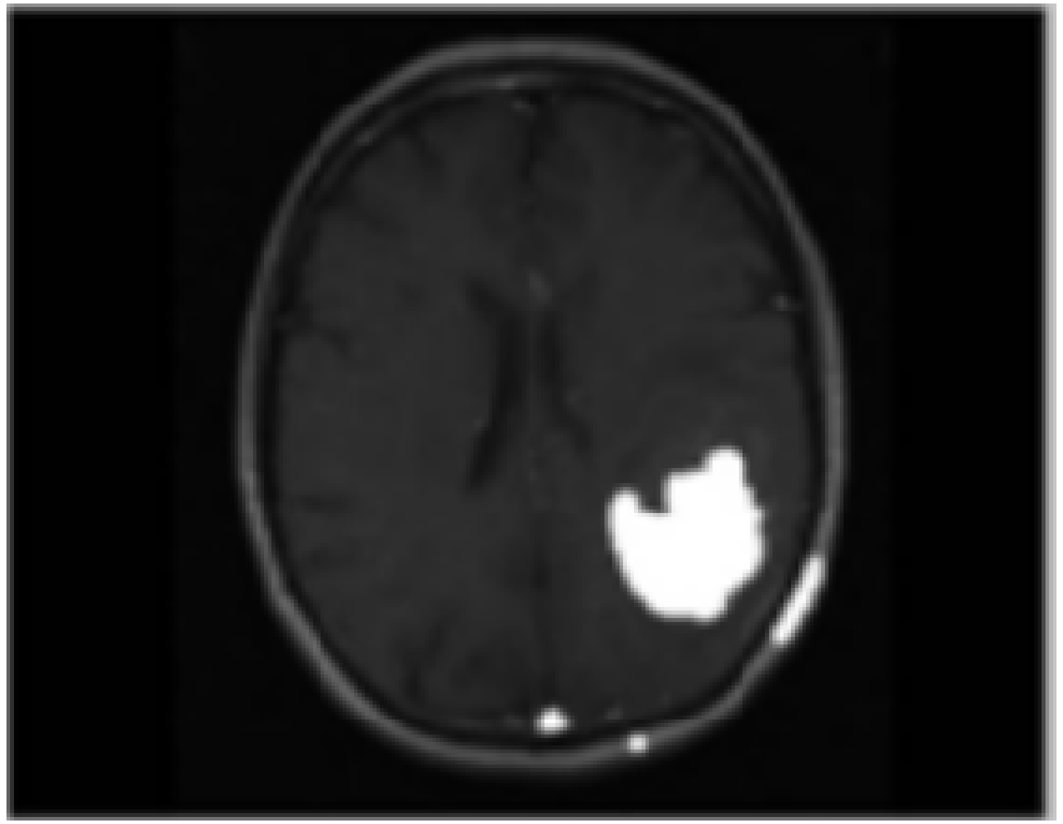
Rebeca Gonzalez
Data Engineer

Segmentation



Segmentation

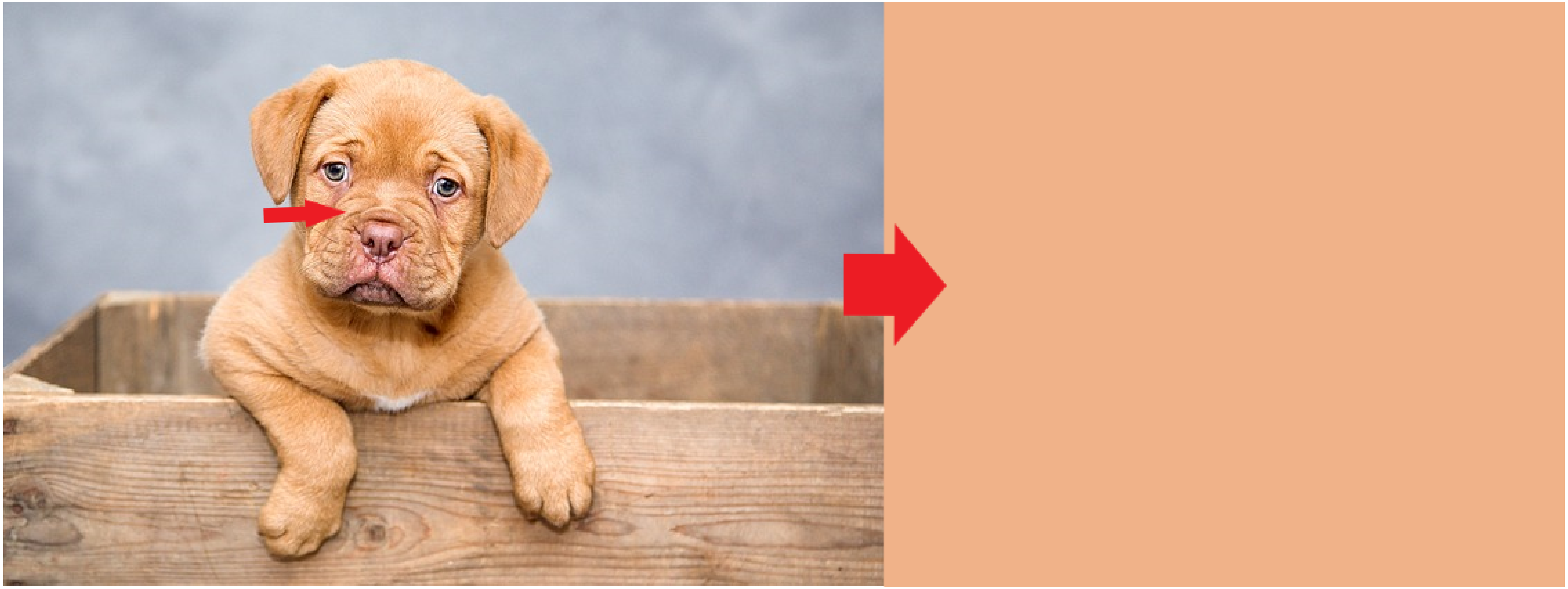
Segmented



Original

Segmented

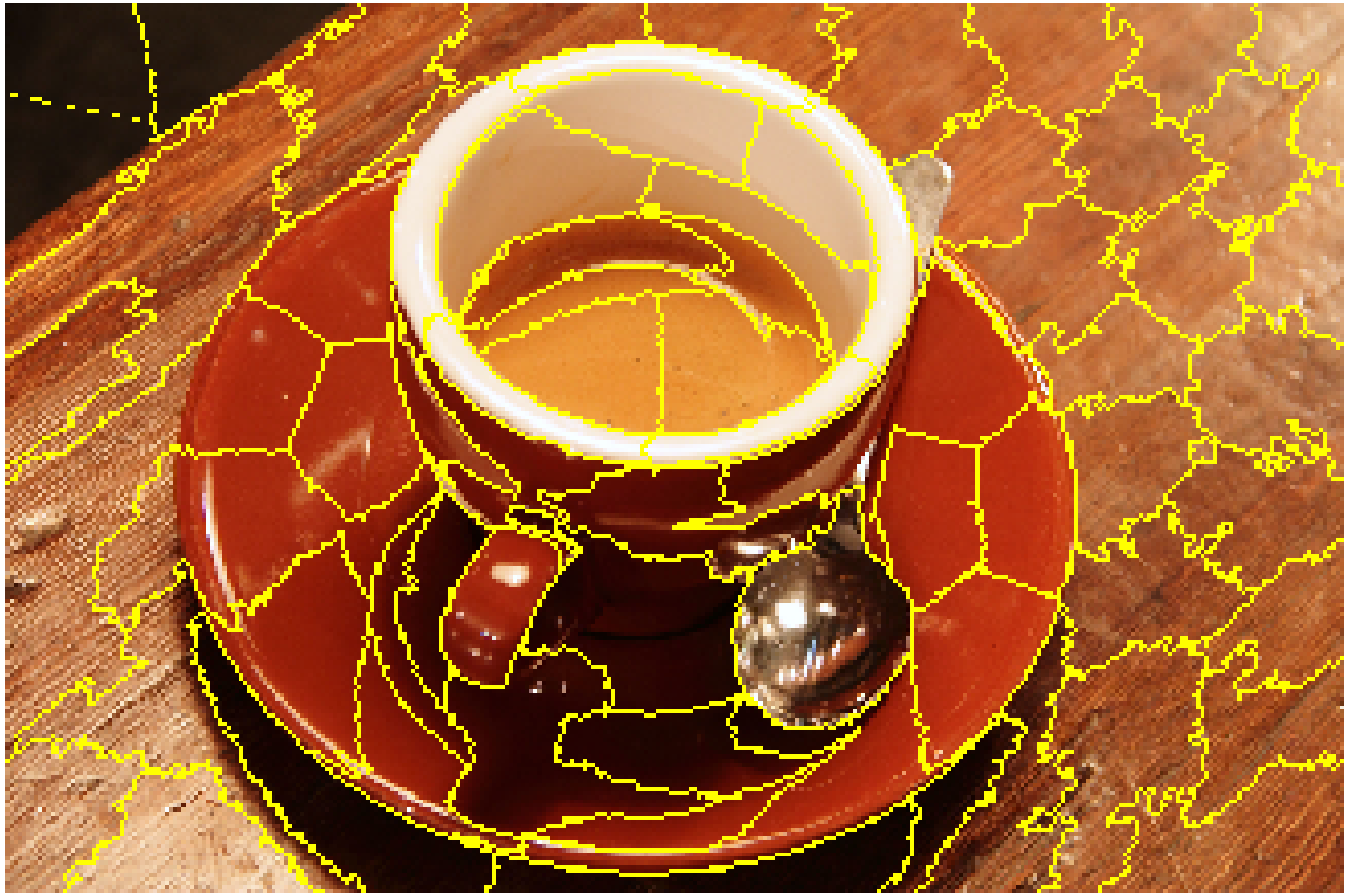
Image representation



Superpixels

A superpixel is a group of connected pixels with similar colors or gray levels.

Superpixel segmentation, 100 segments



Benefits of superpixels

- More meaningful regions
- Computational efficiency

Segmentation

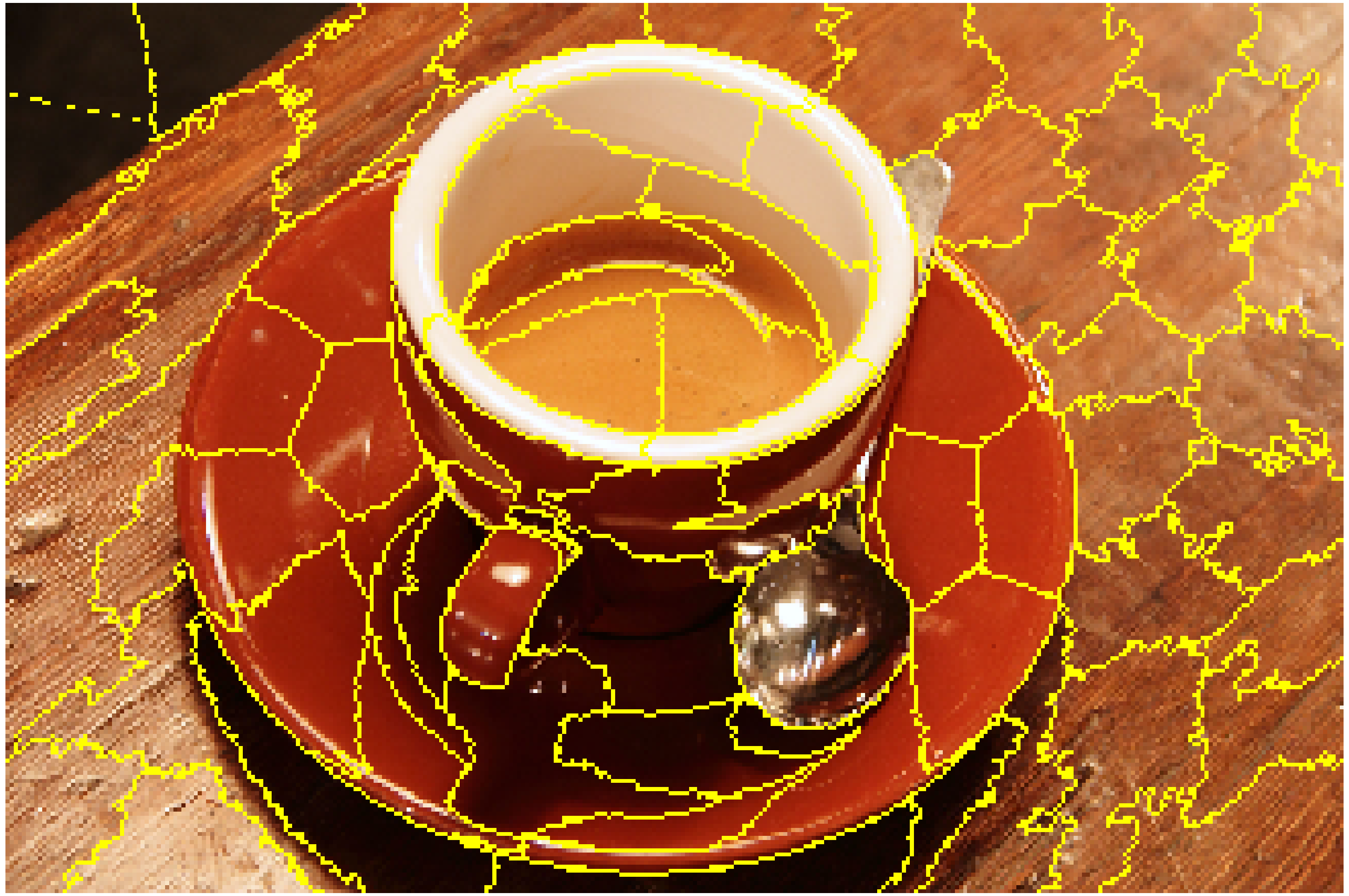
- Supervised
- Unsupervised

Unsupervised segmentation

Simple Linear Iterative Clustering (SLIC)

It segments the image using a machine learning algorithm called K-Means clustering. It takes in all the pixel values of the image and tries to separate them into a predefined number of sub-regions.

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")
```

We'll use the label2rgb method from the color module to return an image where the segments obtained from the slic method will be highlighted, either with random colors or with the average color of the superpixel segment.

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!

Finding contours

IMAGE PROCESSING IN PYTHON

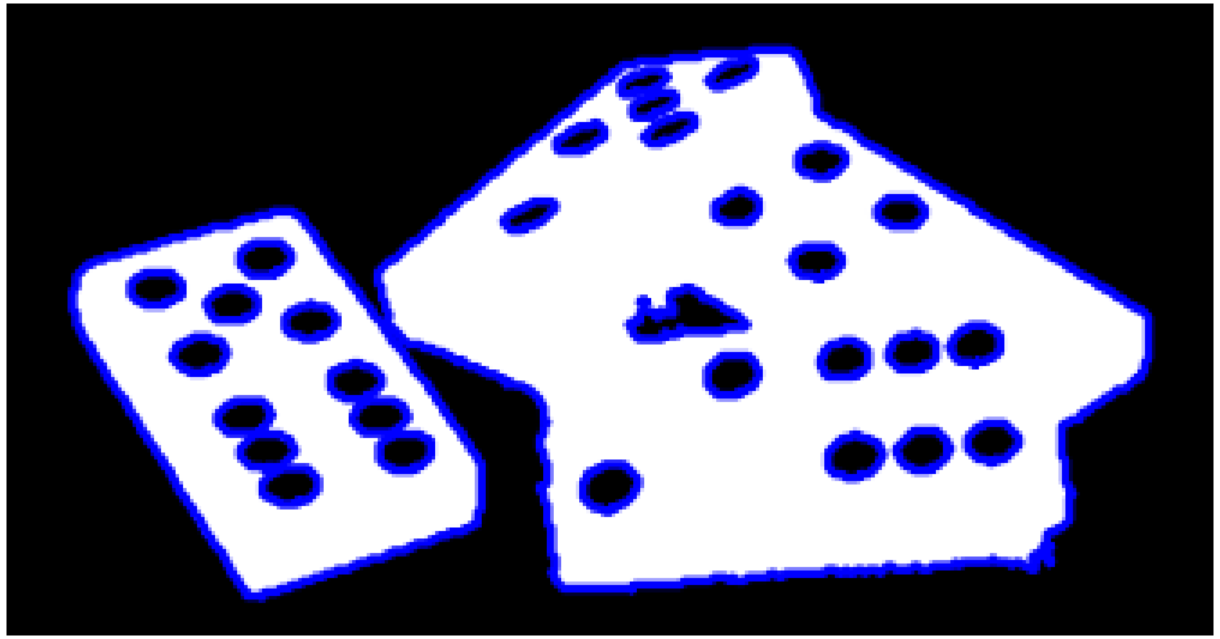
Rebeca Gonzalez
Data Engineer

Finding contours

A contour is a closed shape of points or line segments, representing the boundaries of these objects.

Original image

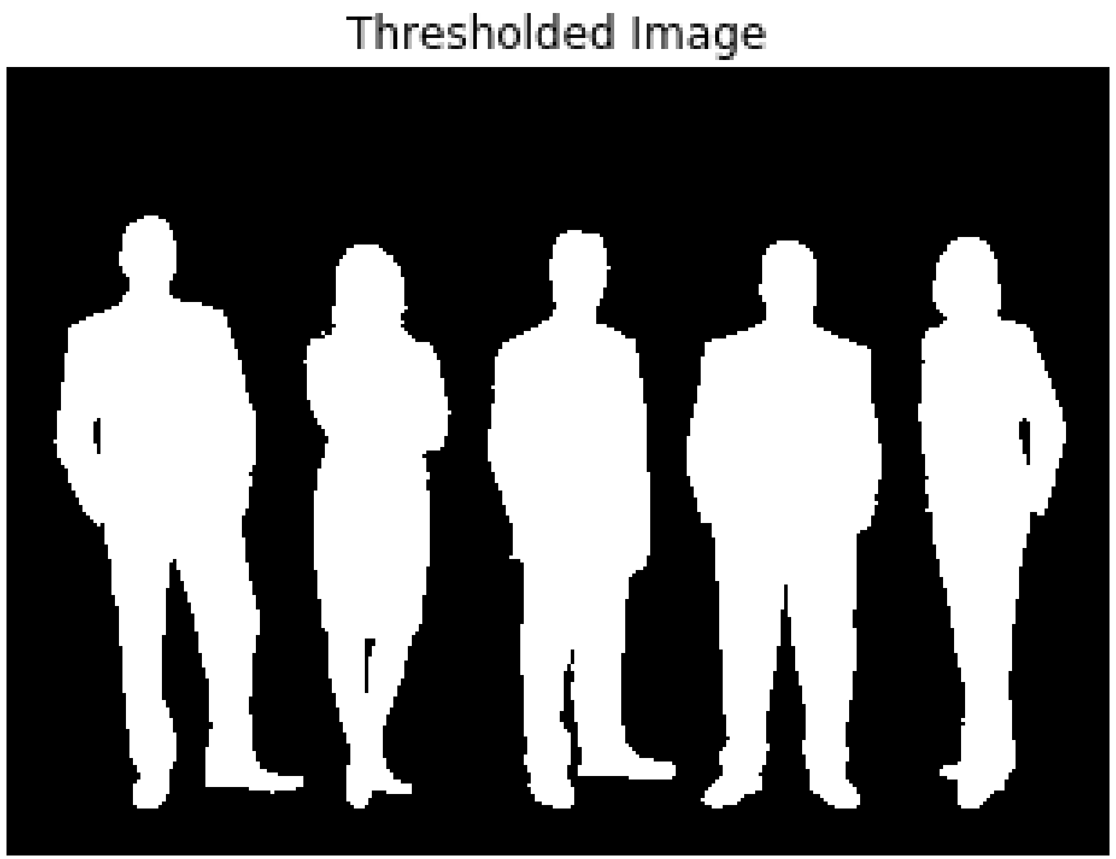
Contours

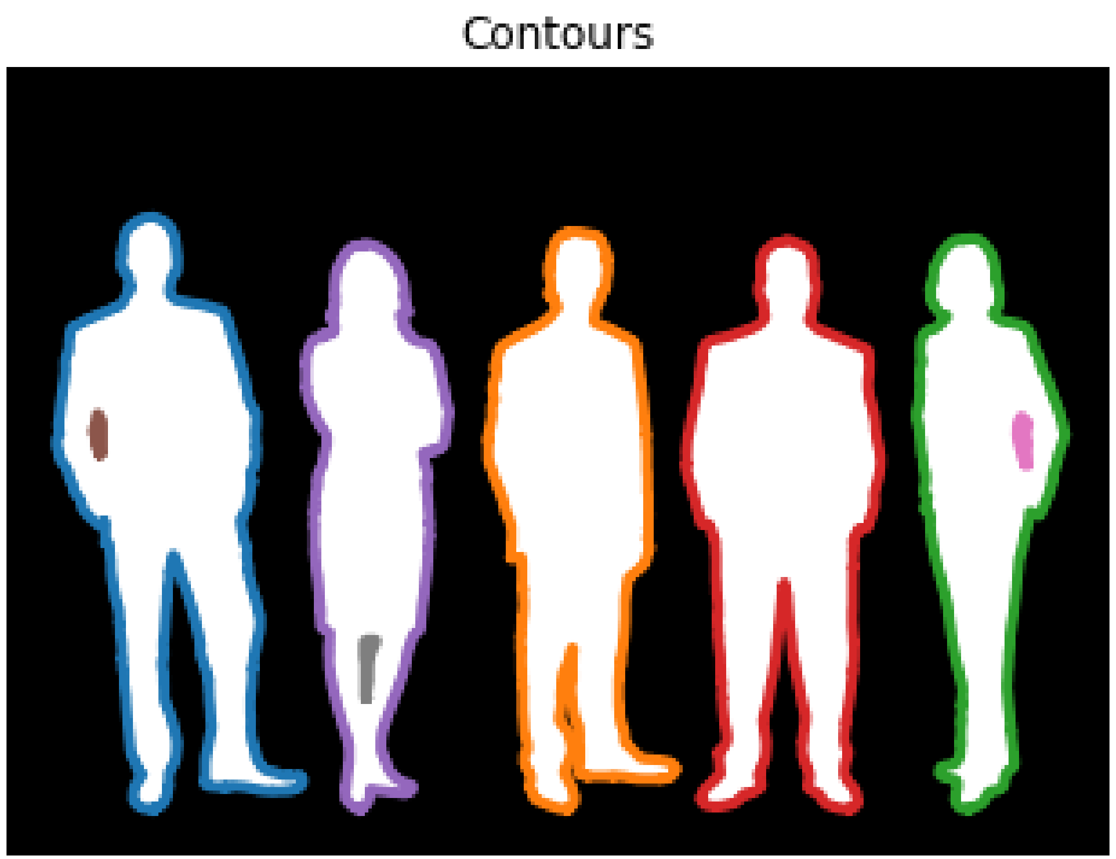


- Measure size
- Classify shapes
- Determine the number of objects

Total points in domino tokens: 35.

Binary images





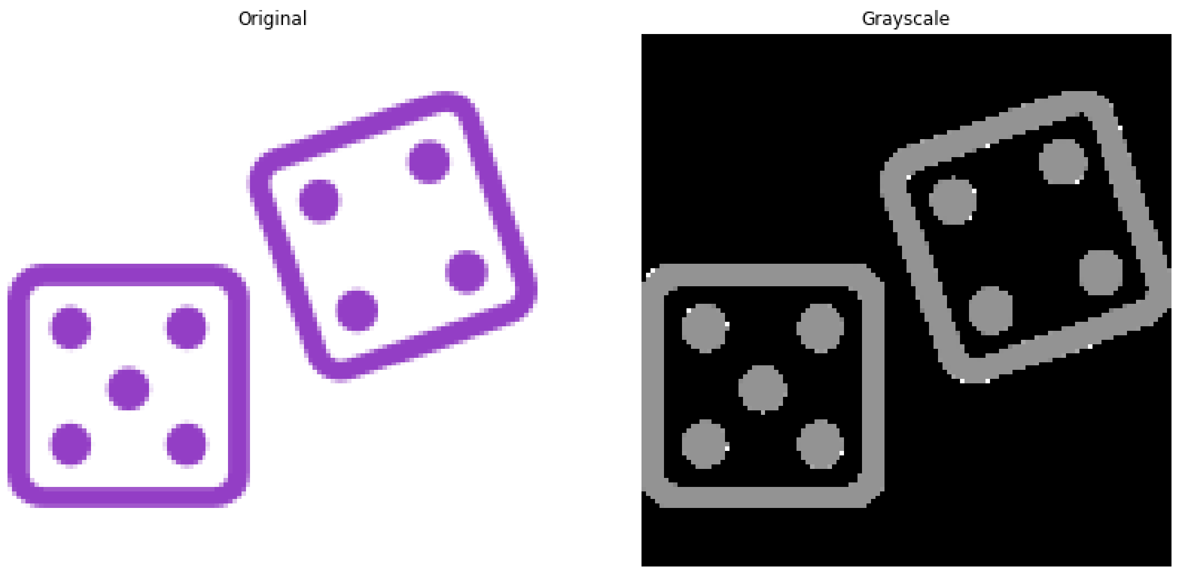
We can obtain a binary image applying **thresholding** or using **edge detection**

Find contours using scikit-image

PREPARING THE IMAGE

Transform the image to 2D grayscale.

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

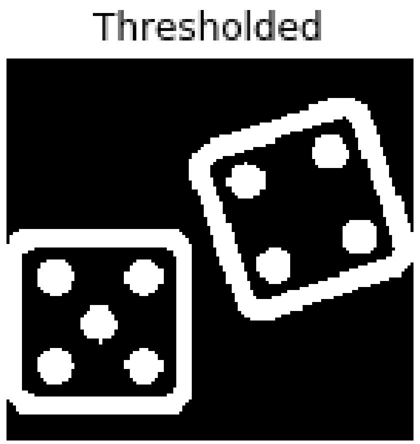


Find contours using scikit-image

PREPARING THE IMAGE

Binarize the image

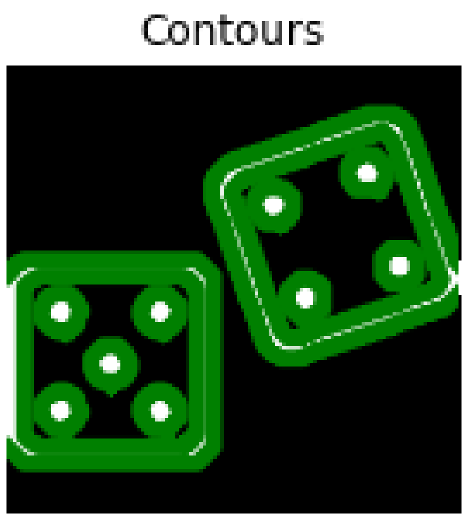
```
# Obtain the thresh value
thresh = threshold_otsu(image)

# Apply thresholding
thresholded_image = image > thresh
```



Find contours using scikit-image

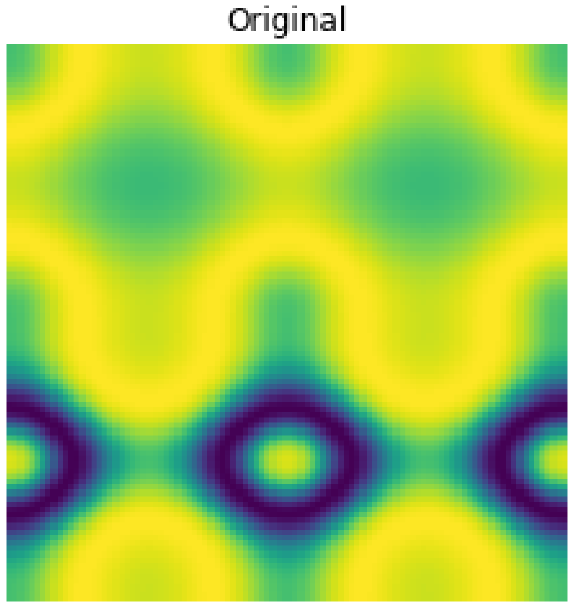
And then use `find_contours()` .

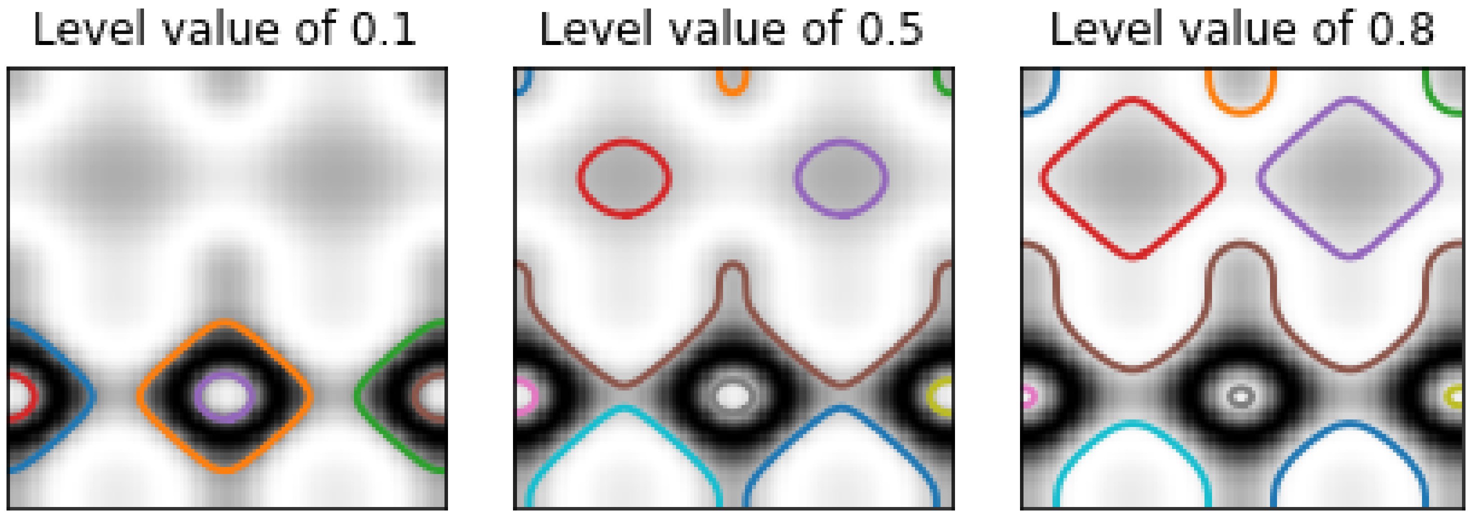
```
# Import the measure module
from skimage import measure

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



Constant level value

The level value varies between 0 and 1, the closer to 1 the more sensitive the method is to detect contours, so more complex contours will be detected.



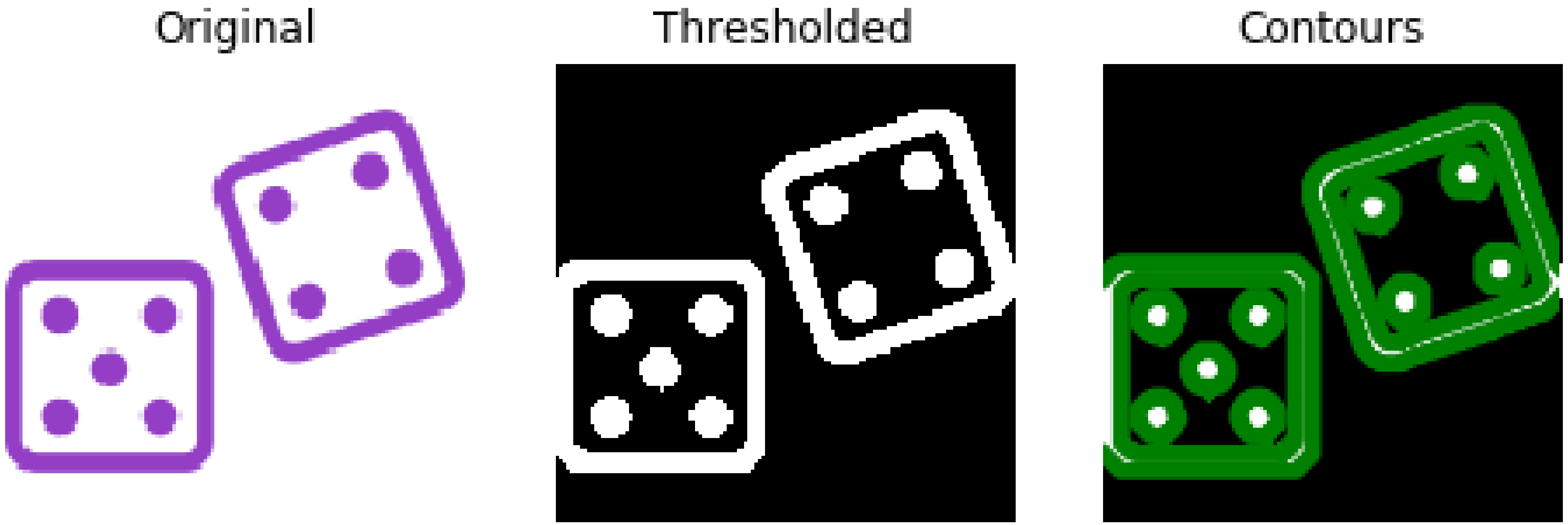


The steps to spotting contours

```
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



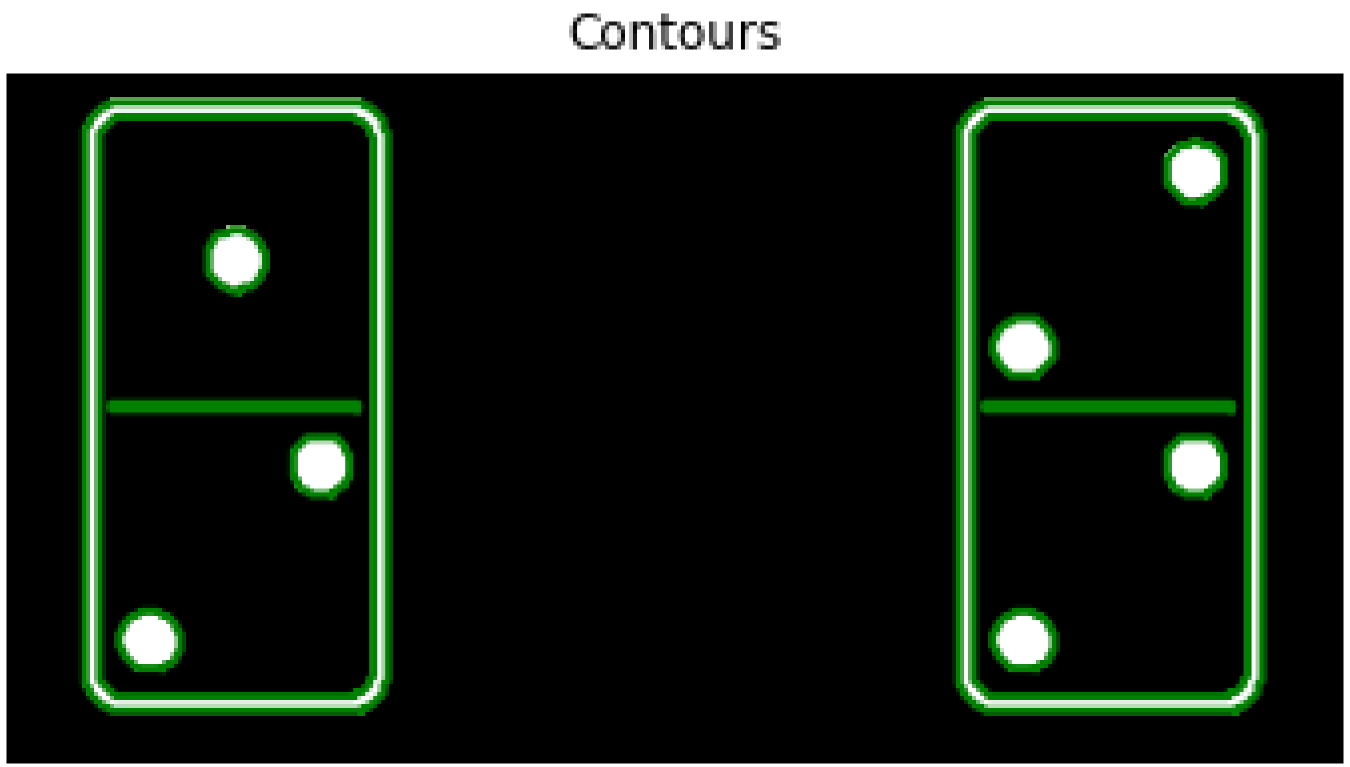
A contour's shape

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

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

Each contour is an ndarray of shape (n,2), consisting of n row and column coordinates along the contour.

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



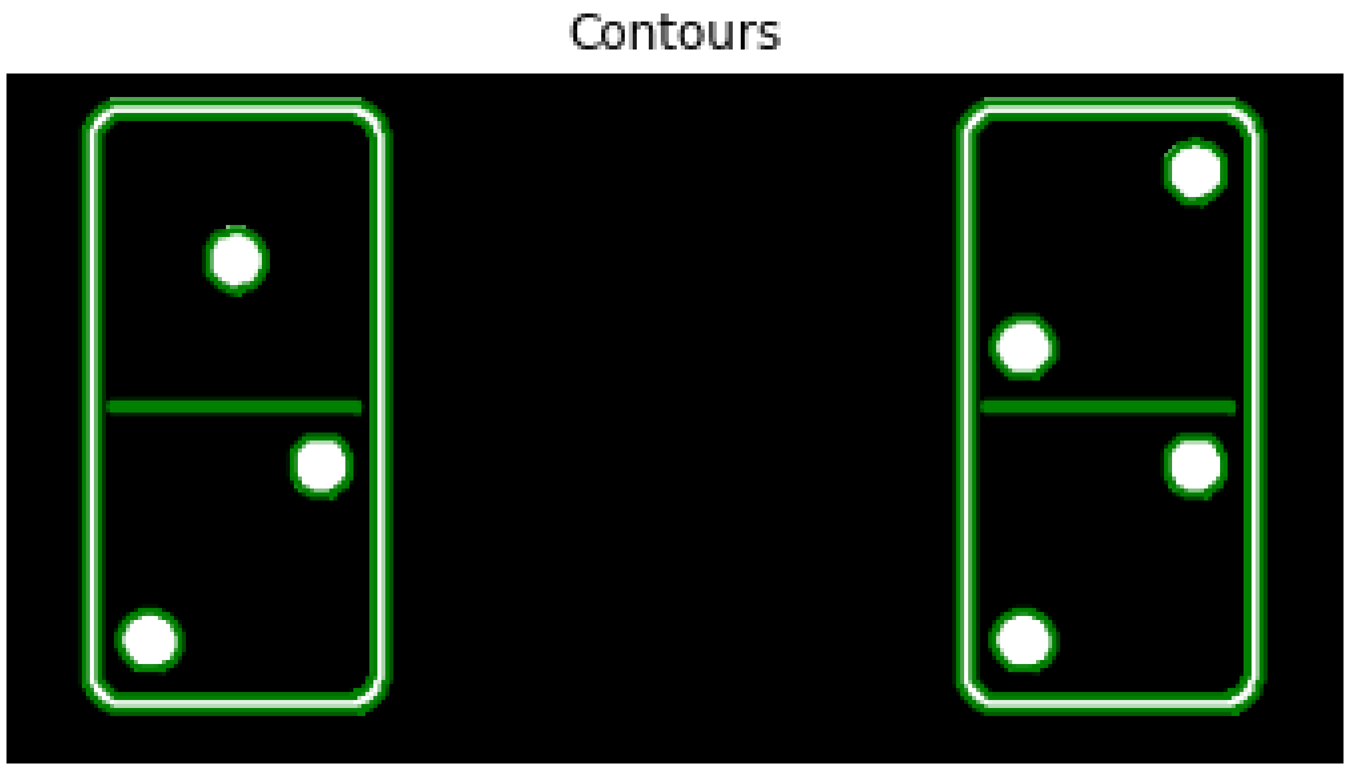
In this way, a contour is like an outline formed by multiple points joined together. The bigger the contour, the more points joined together and the wider the perimeter formed.

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)
```

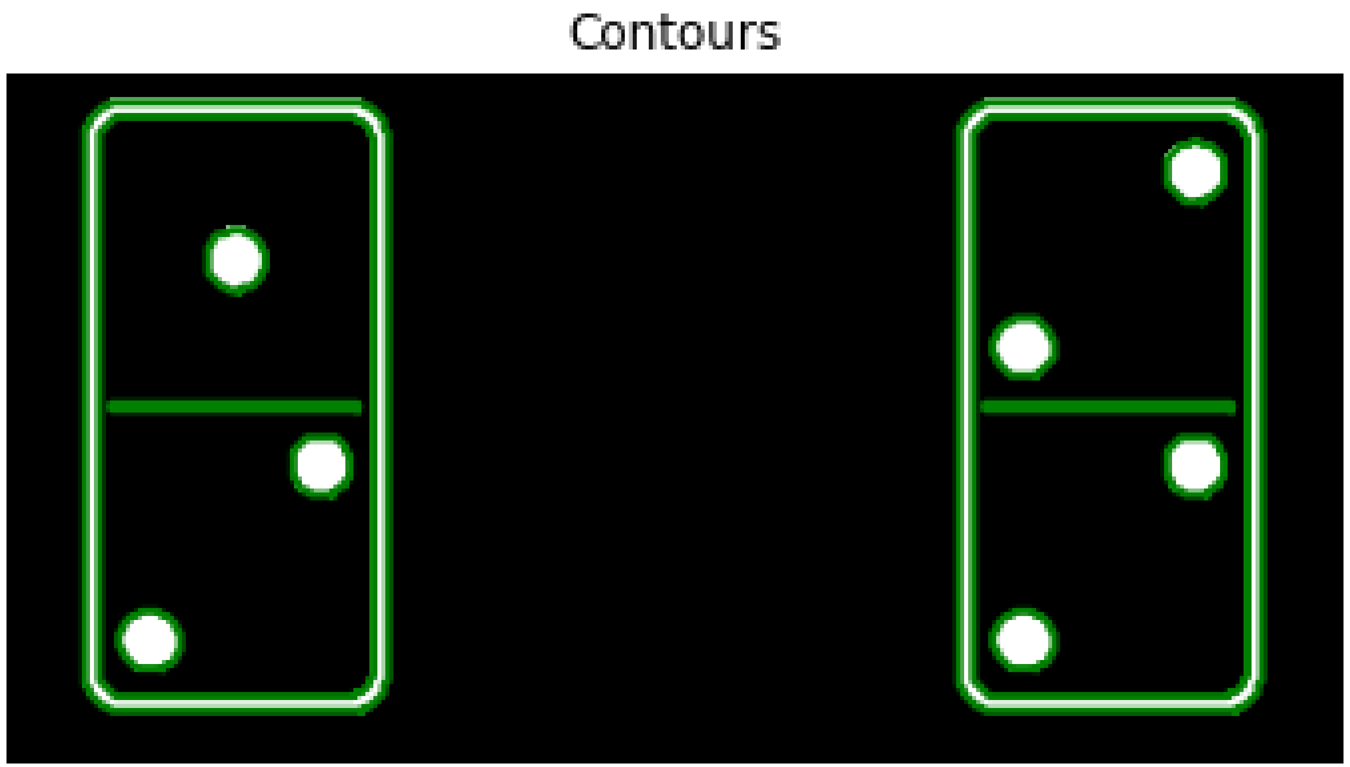
We deduct they are the outer border contour of the tokens because they are the longest.



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)
```



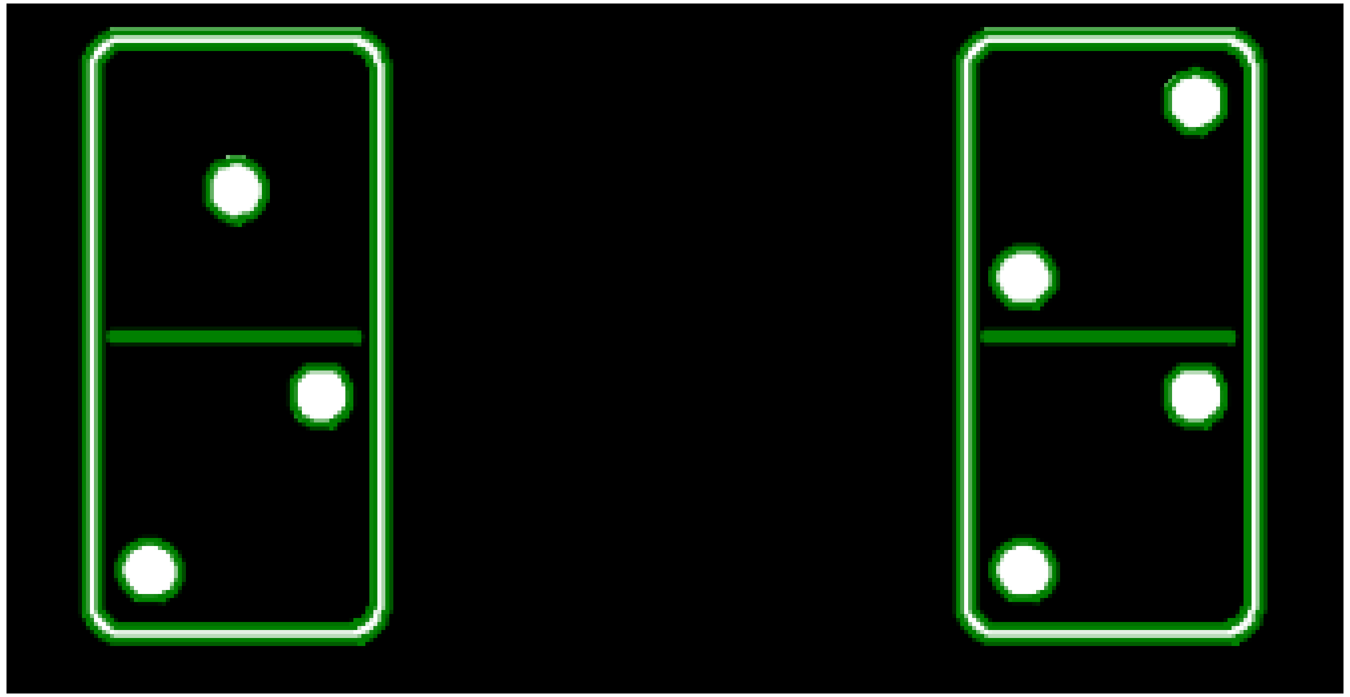
A contour's shape

```
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)
```

The (123, 2) is the dividing line in the middle of the tokens.

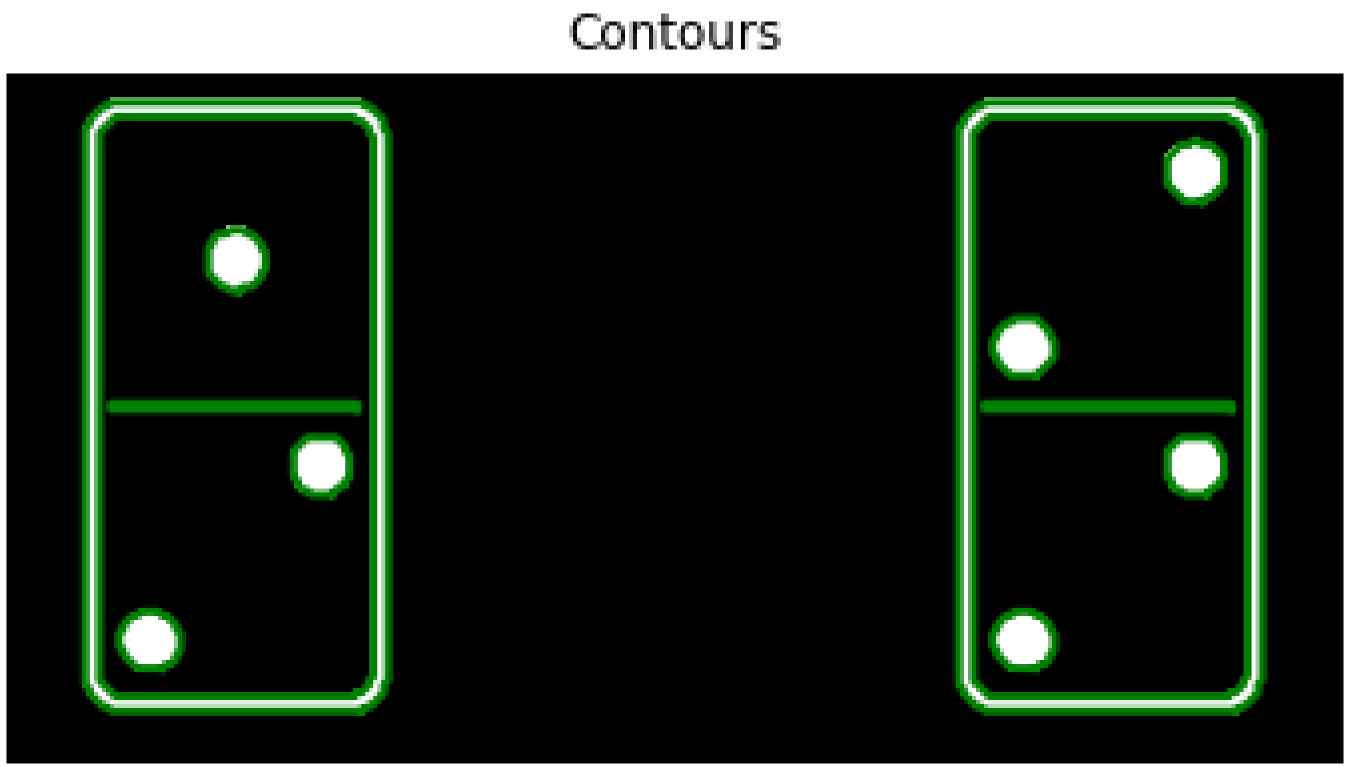
Contours



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
```



Number of dots: 7.

Let's practice!

IMAGE PROCESSING IN PYTHON