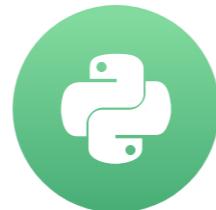


Finding the edges with Canny

IMAGE PROCESSING IN PYTHON



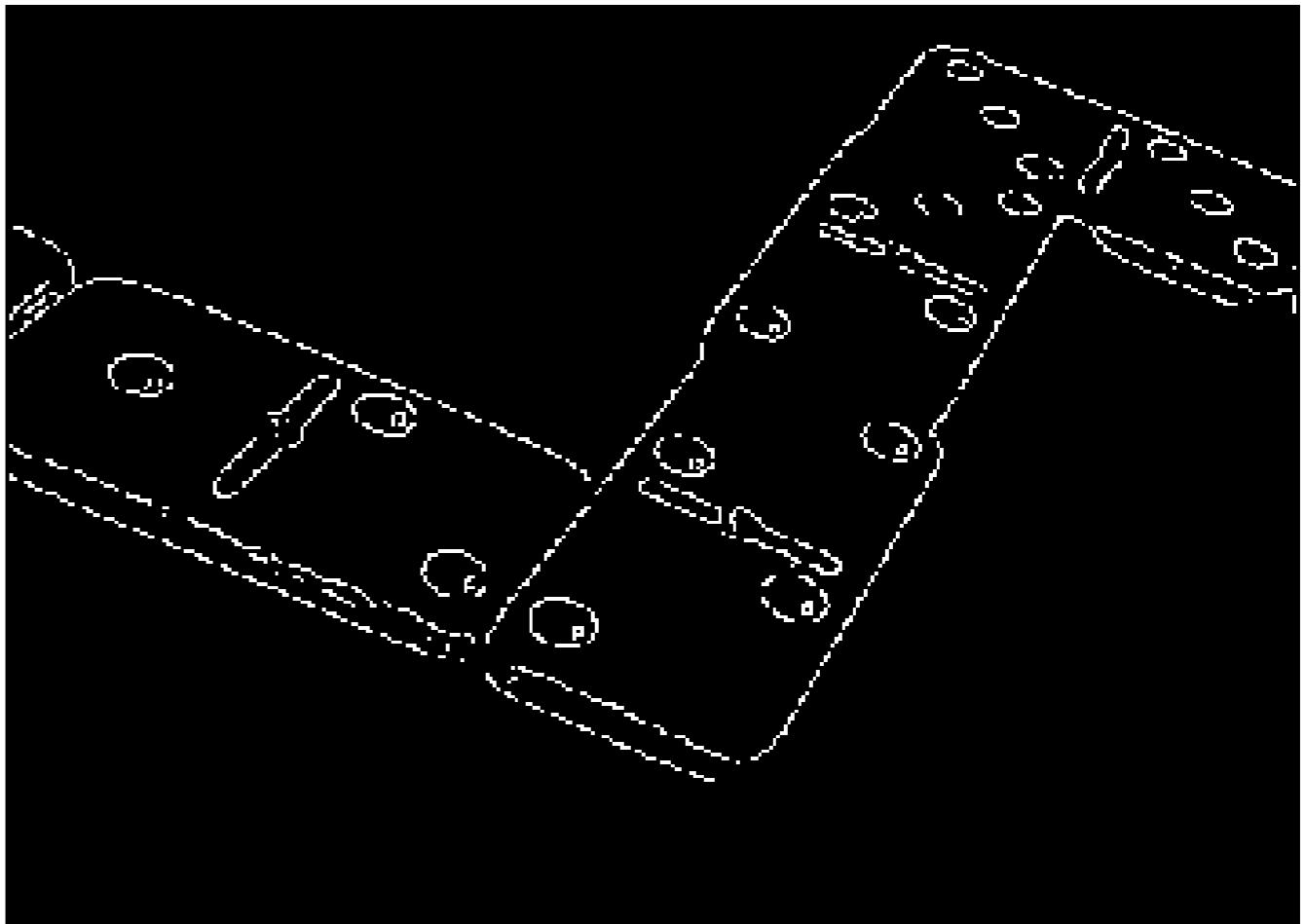
Rebeca Gonzalez
Data Engineer

Detecting edges

Original

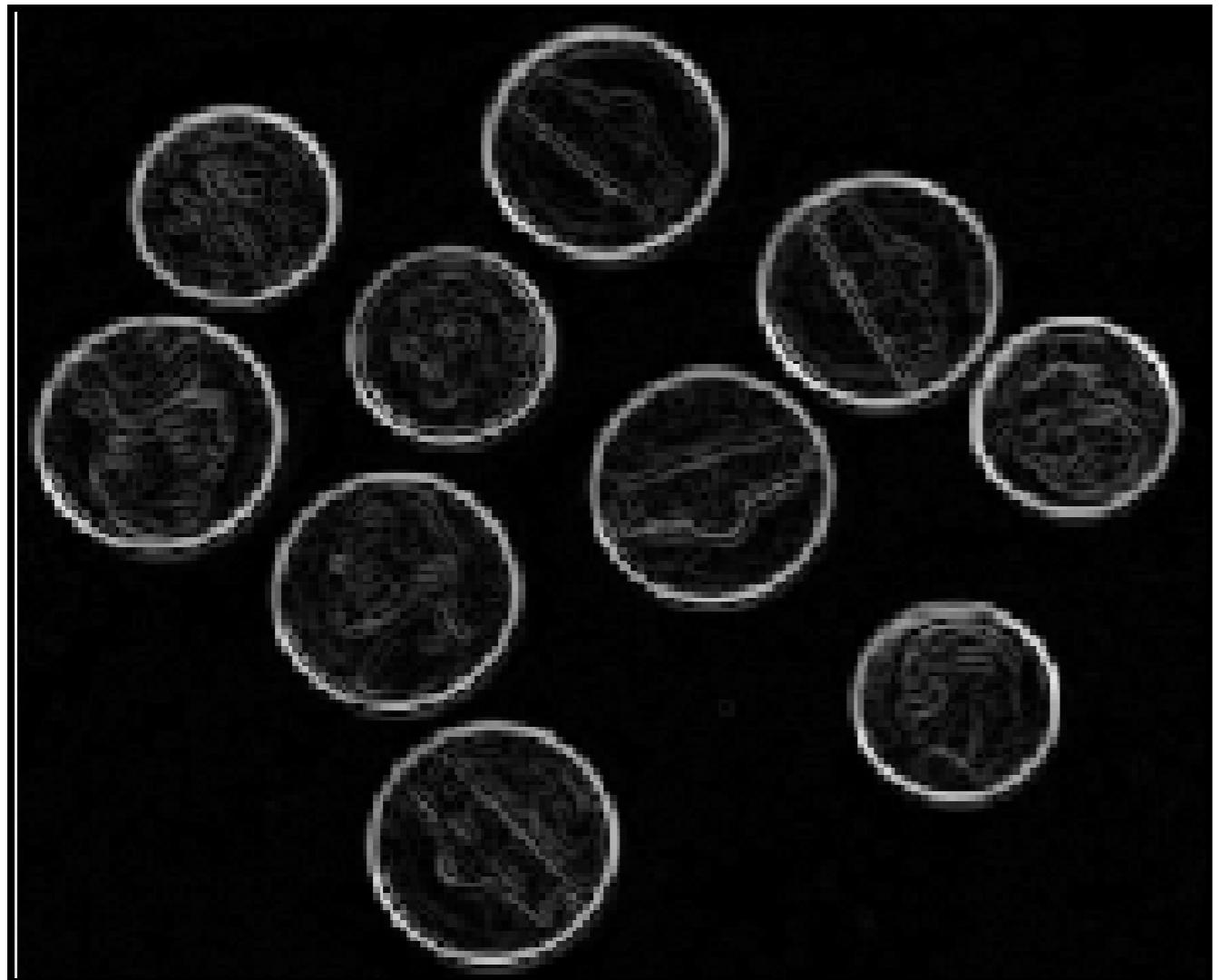


Edges with Canny

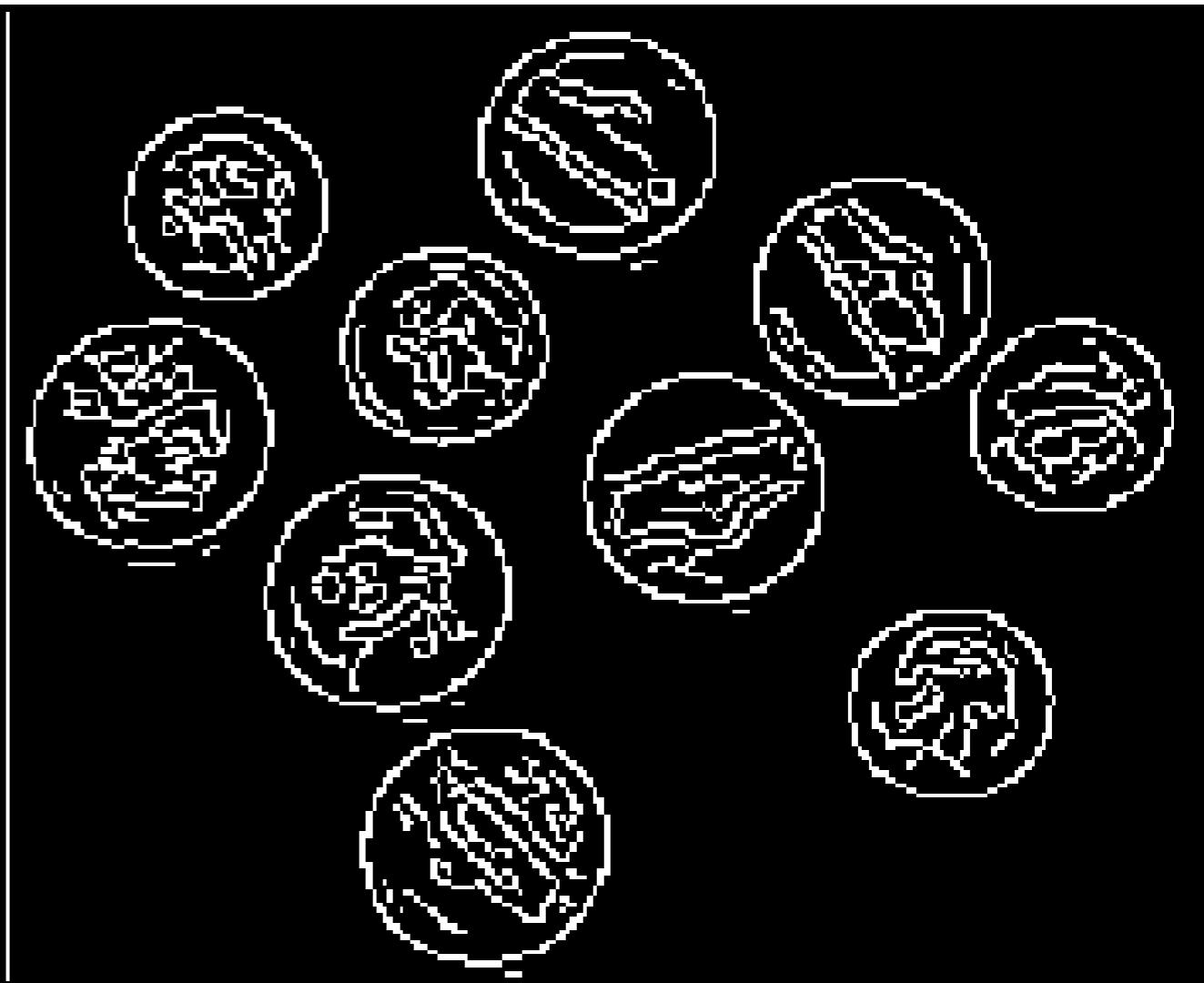


Edge detection

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)

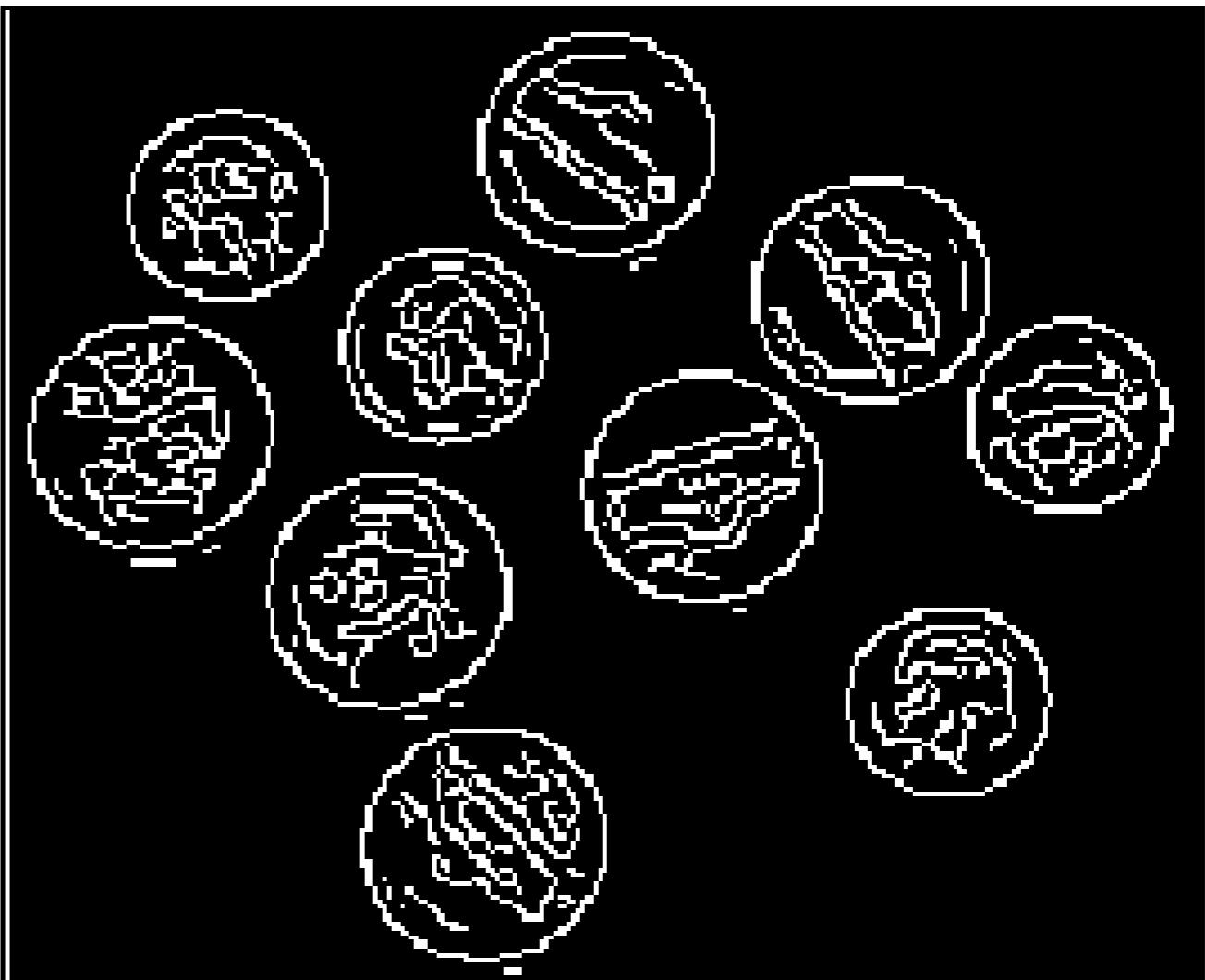
# Show resulted image with edges
show_image(canny_edges, "Edges with Canny")
```

Edge detection

Original



Edges with Canny



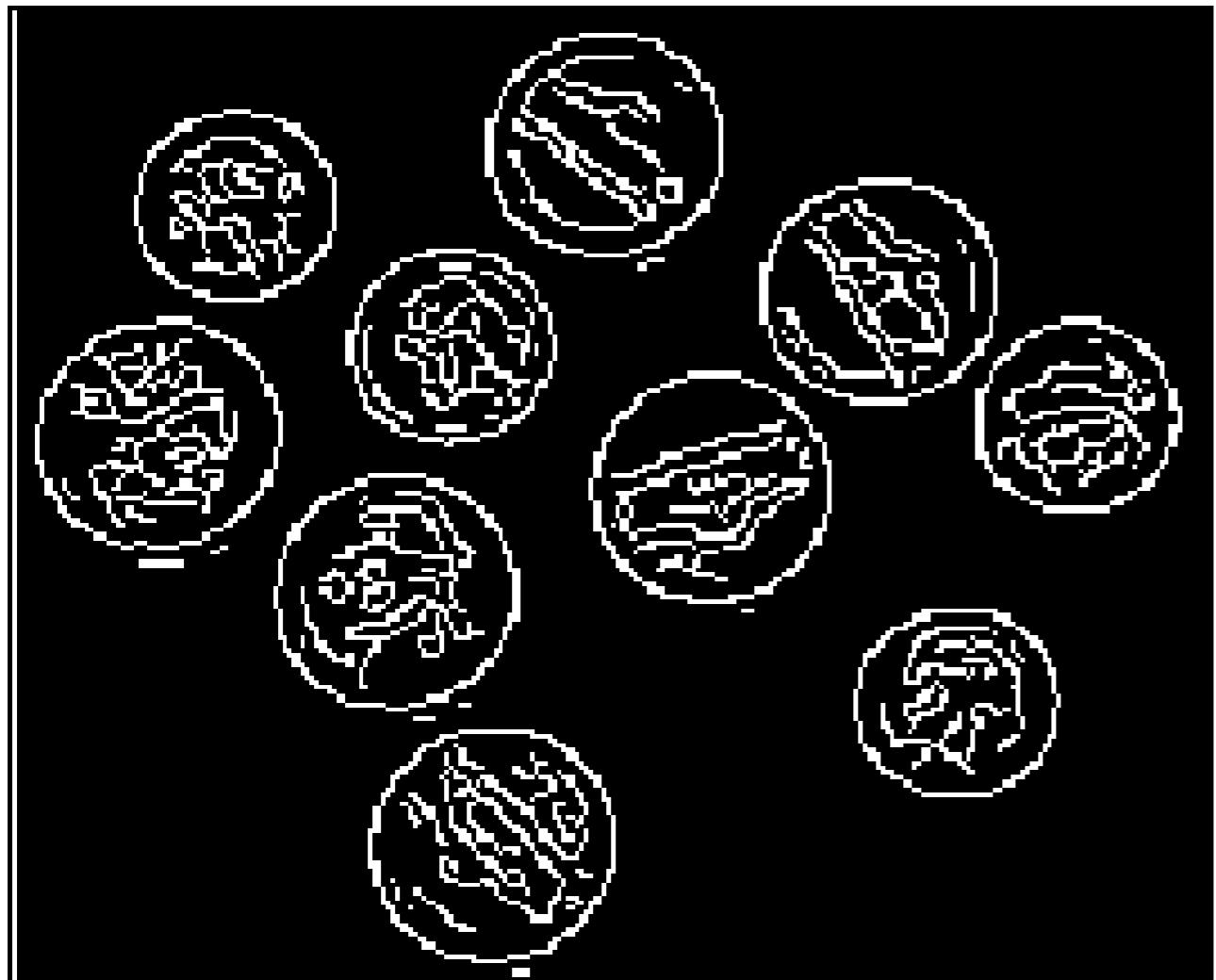
Canny edge detector

```
# Apply Canny detector with a sigma of 0.5
canny_edges_0_5 = canny(coins, sigma=0.5)

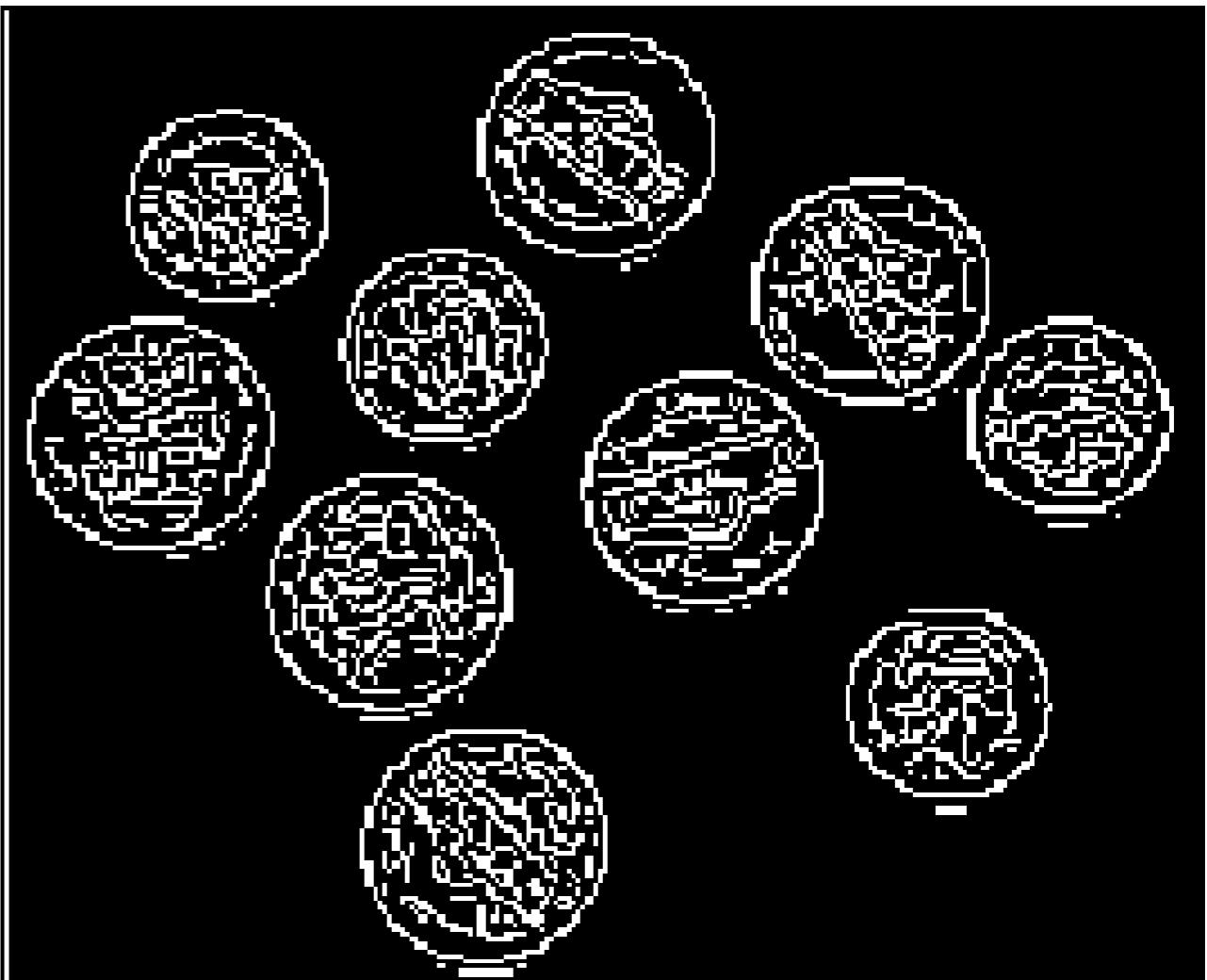
# Show resulted images with edges
show_image(canny_edges, "Sigma of 1")
show_image(canny_edges_0_5, "Sigma of 0.5")
```

Canny edge detector

Sigma of 1



Sigma of 0.5

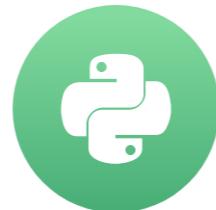


Let's practice!

IMAGE PROCESSING IN PYTHON

Right around the corner

IMAGE PROCESSING IN PYTHON



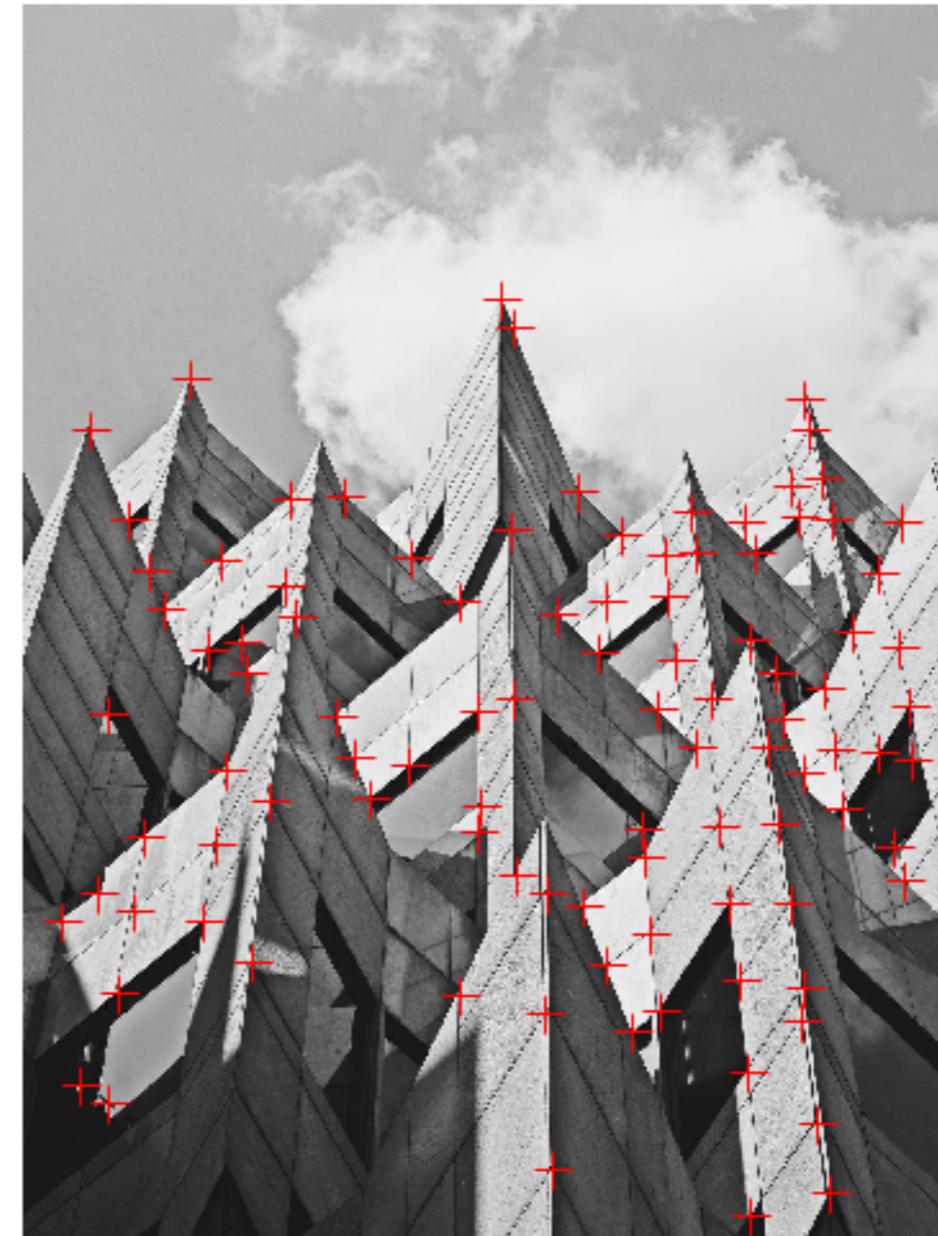
Rebeca Gonzalez
Data Engineer

Corner detection

Original image

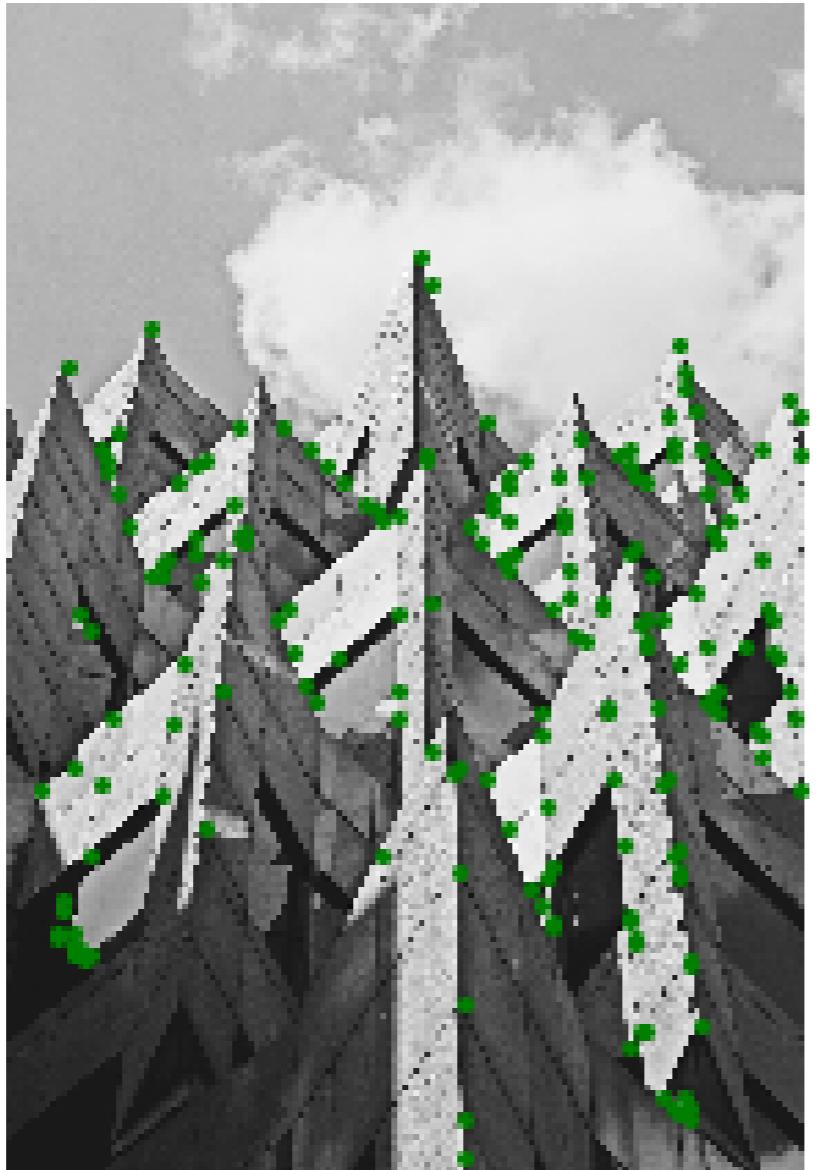


Corners detected



Points of interest

Points of interest

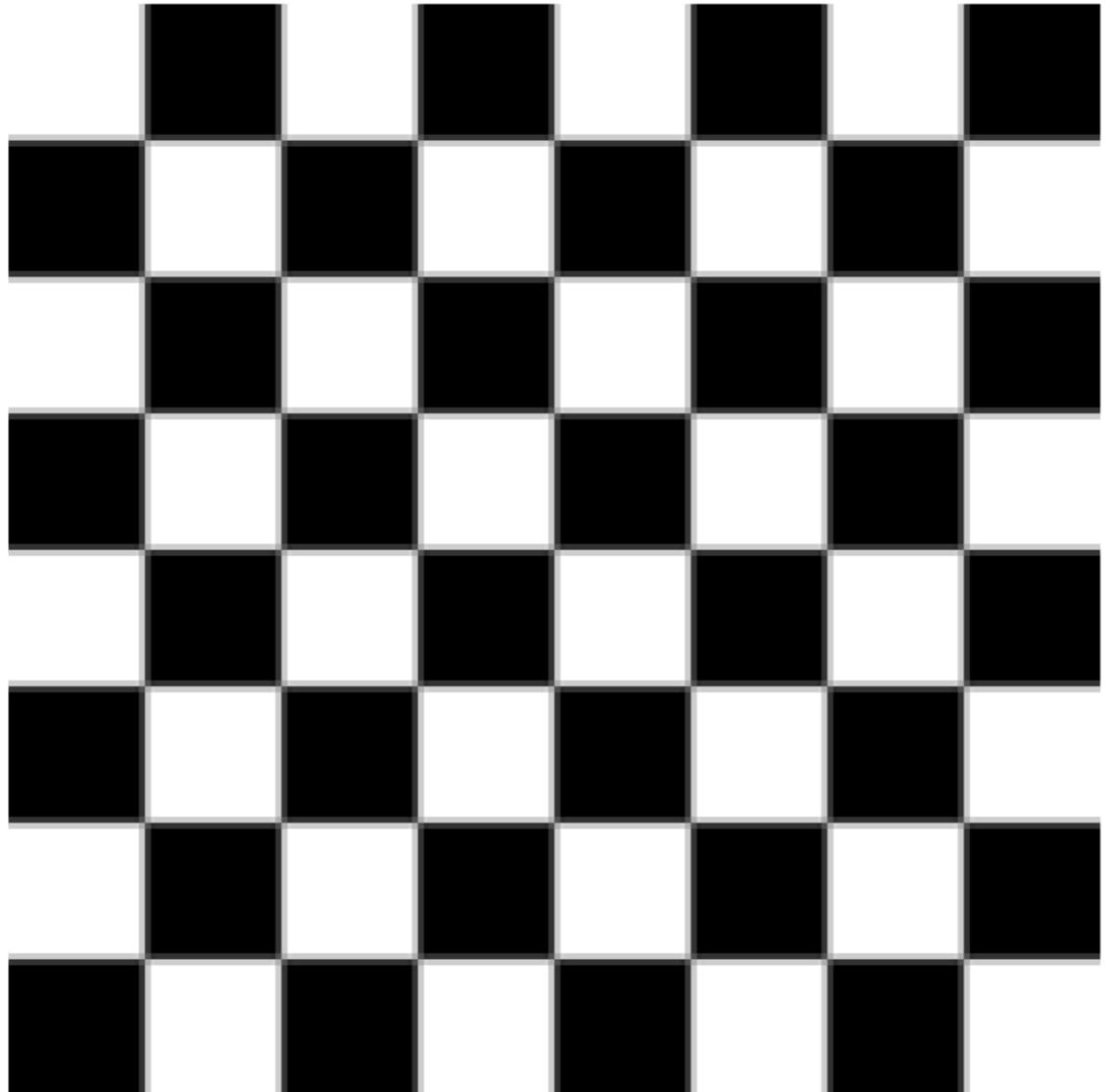


Edges with Sobel

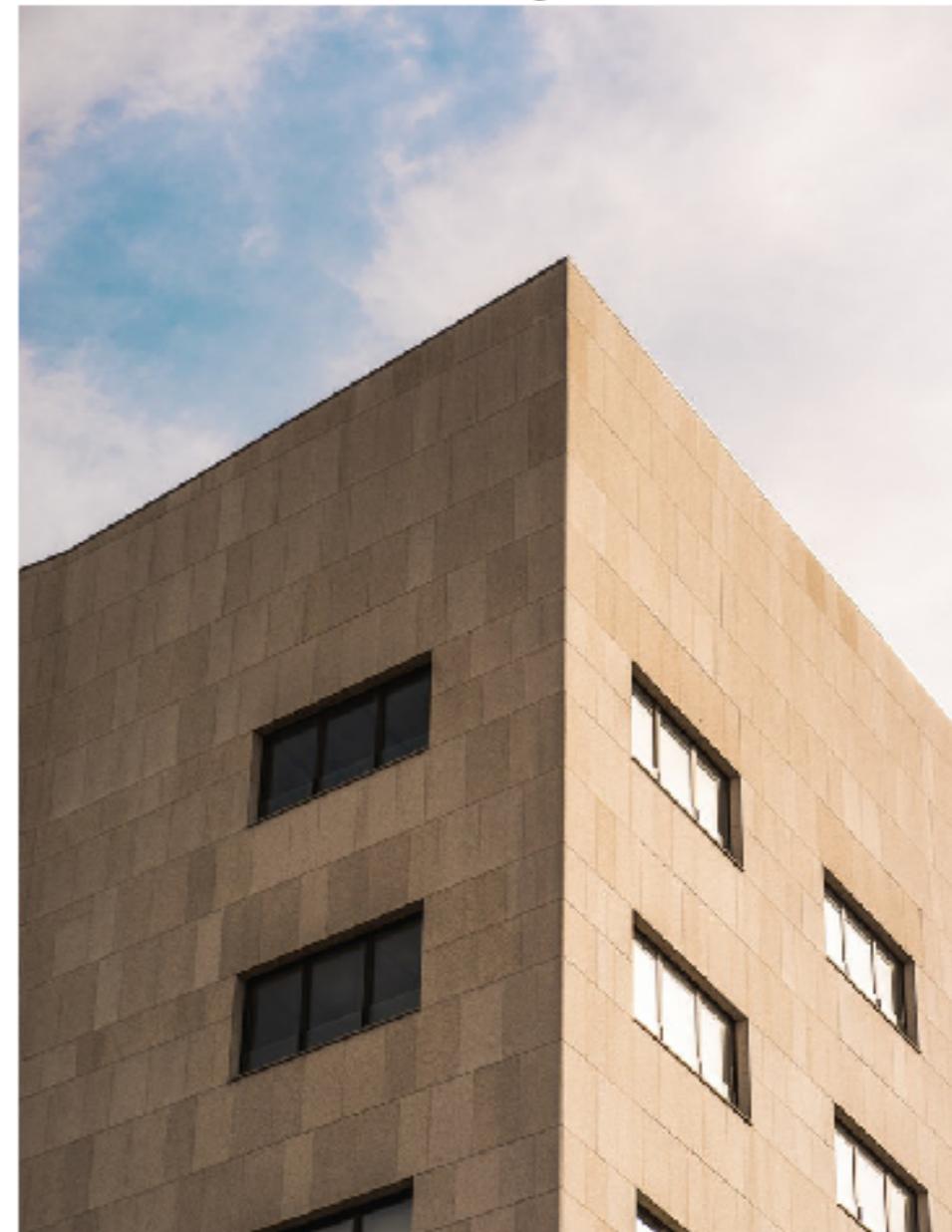


Corners

Checkerboard

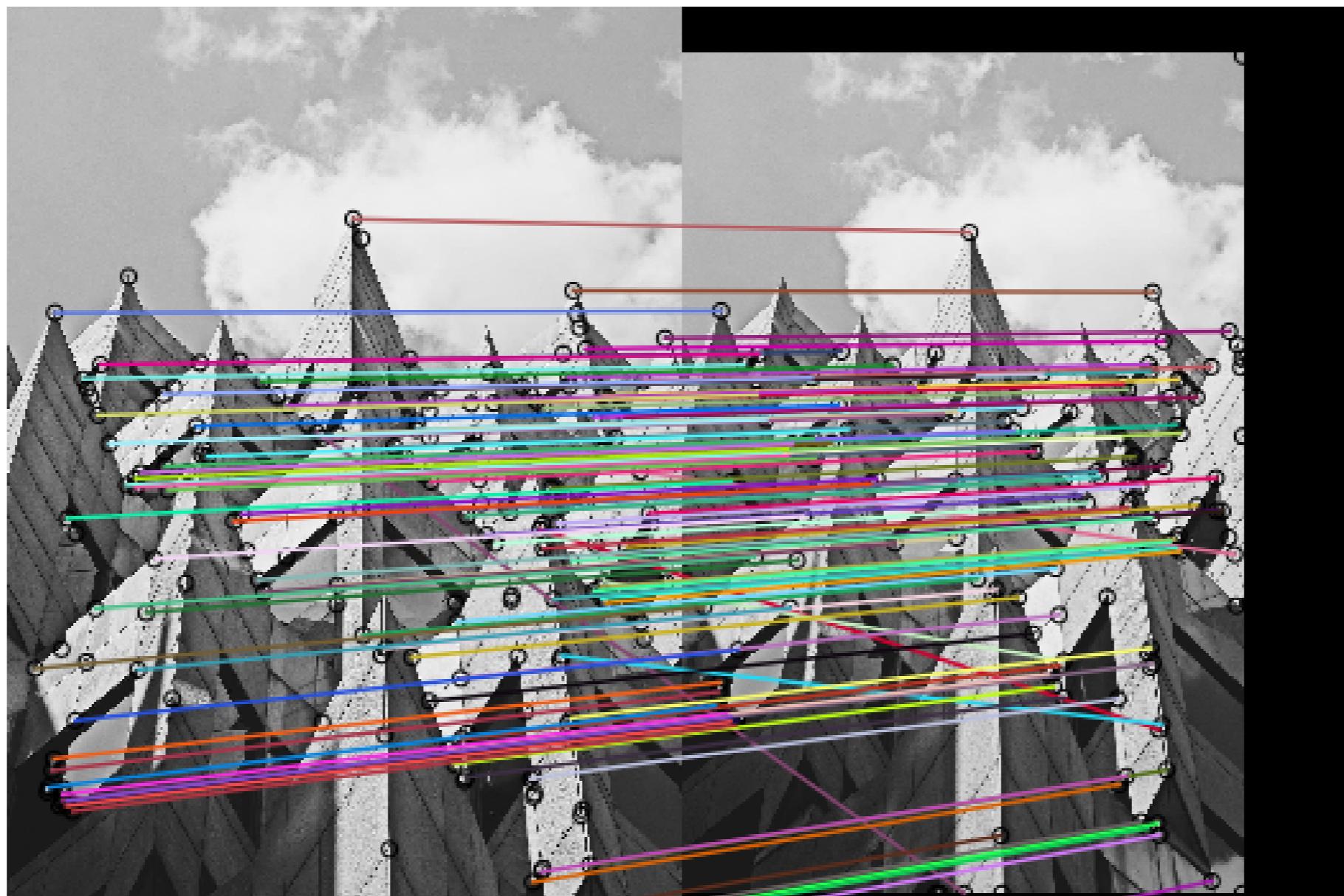


Building



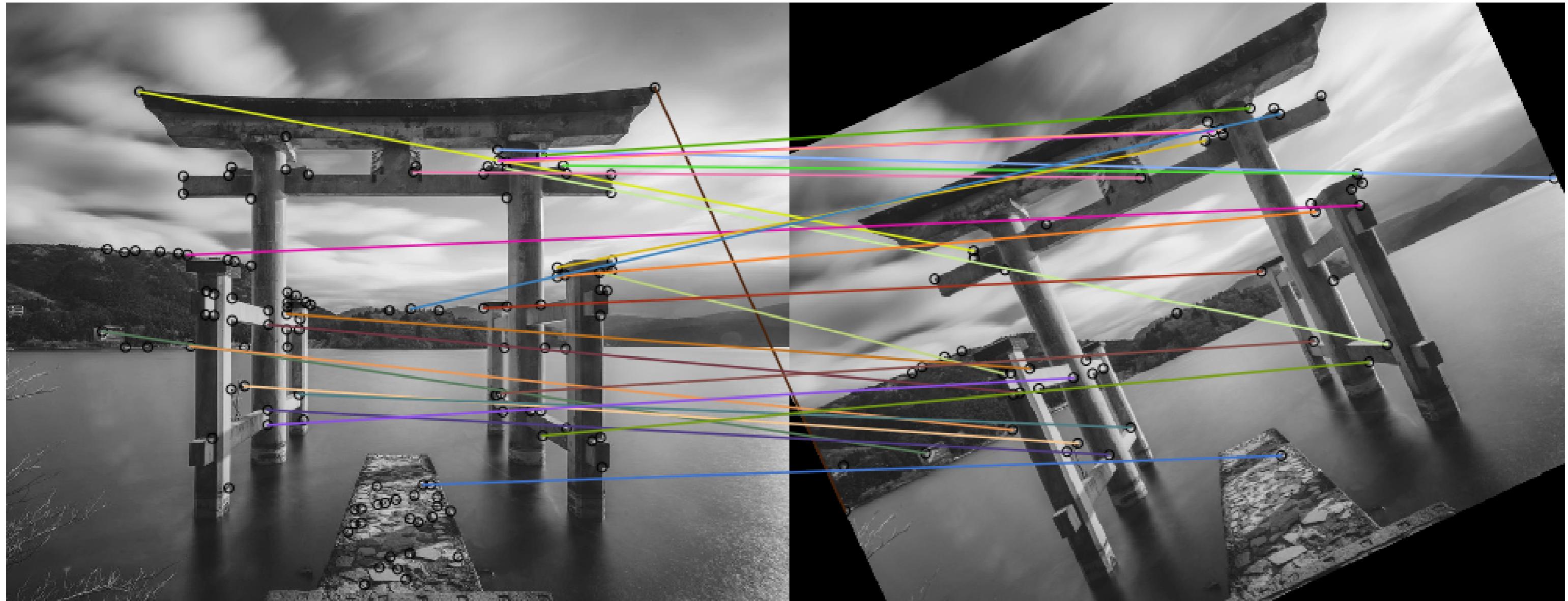
Matching corners

Original Image vs. Transformed Image



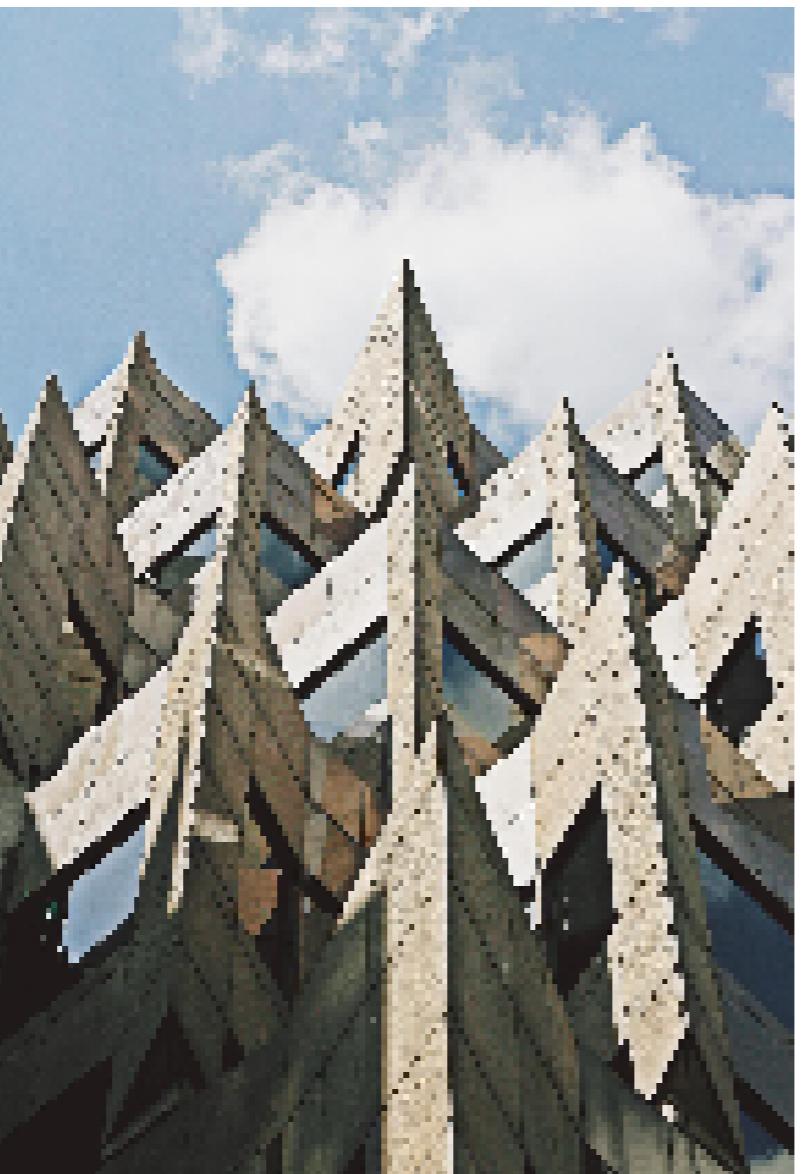
Matching corners

Original Image vs. Transformed Image

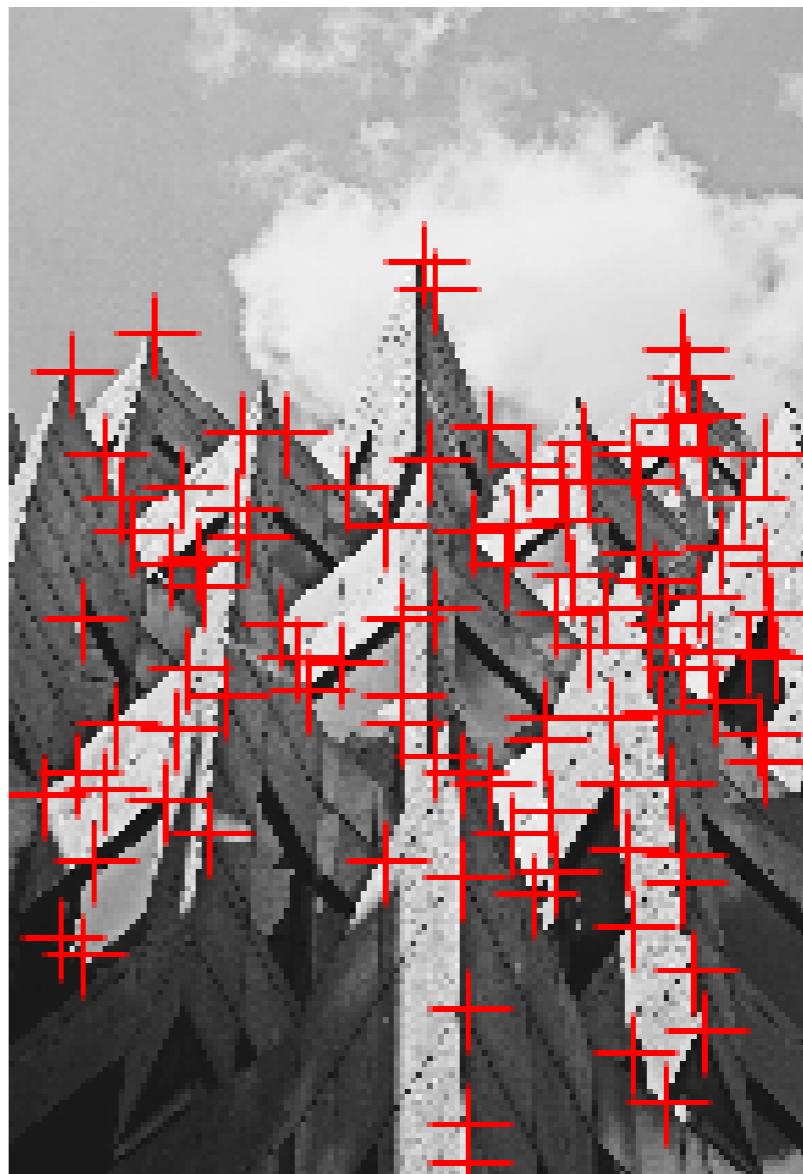


Harris corner detector

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

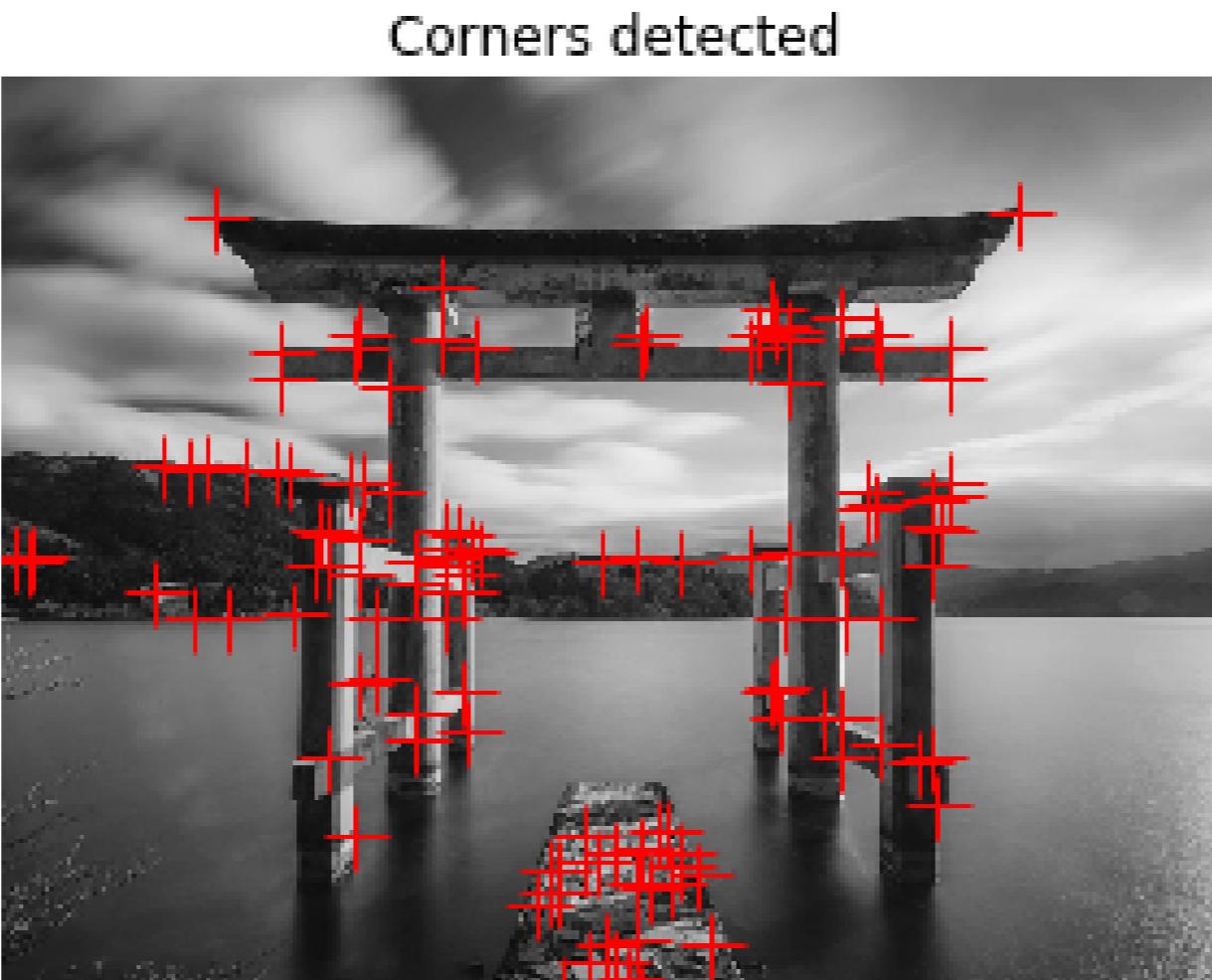
```
# Finds the coordinates of the corners
coords = corner_peaks(corner_harris(image), min_distance=5)

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

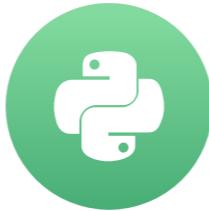
```
def show_image_with_corners(image, coords, title="Corners detected"):  
    plt.imshow(image, interpolation='nearest', cmap='gray')  
    plt.title(title)  
    plt.plot(coords[:, 1], coords[:, 0], '+r', markersize=15)  
    plt.axis('off')  
    plt.show()
```

Let's practice!

IMAGE PROCESSING IN PYTHON

Face detection

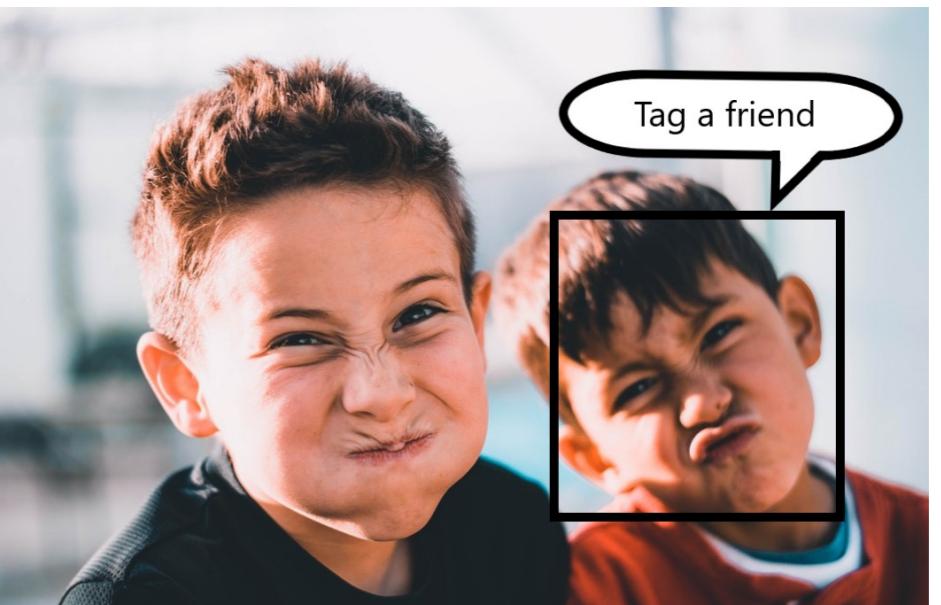
IMAGE PROCESSING IN PYTHON



Rebeca Gonzalez
Data Engineer

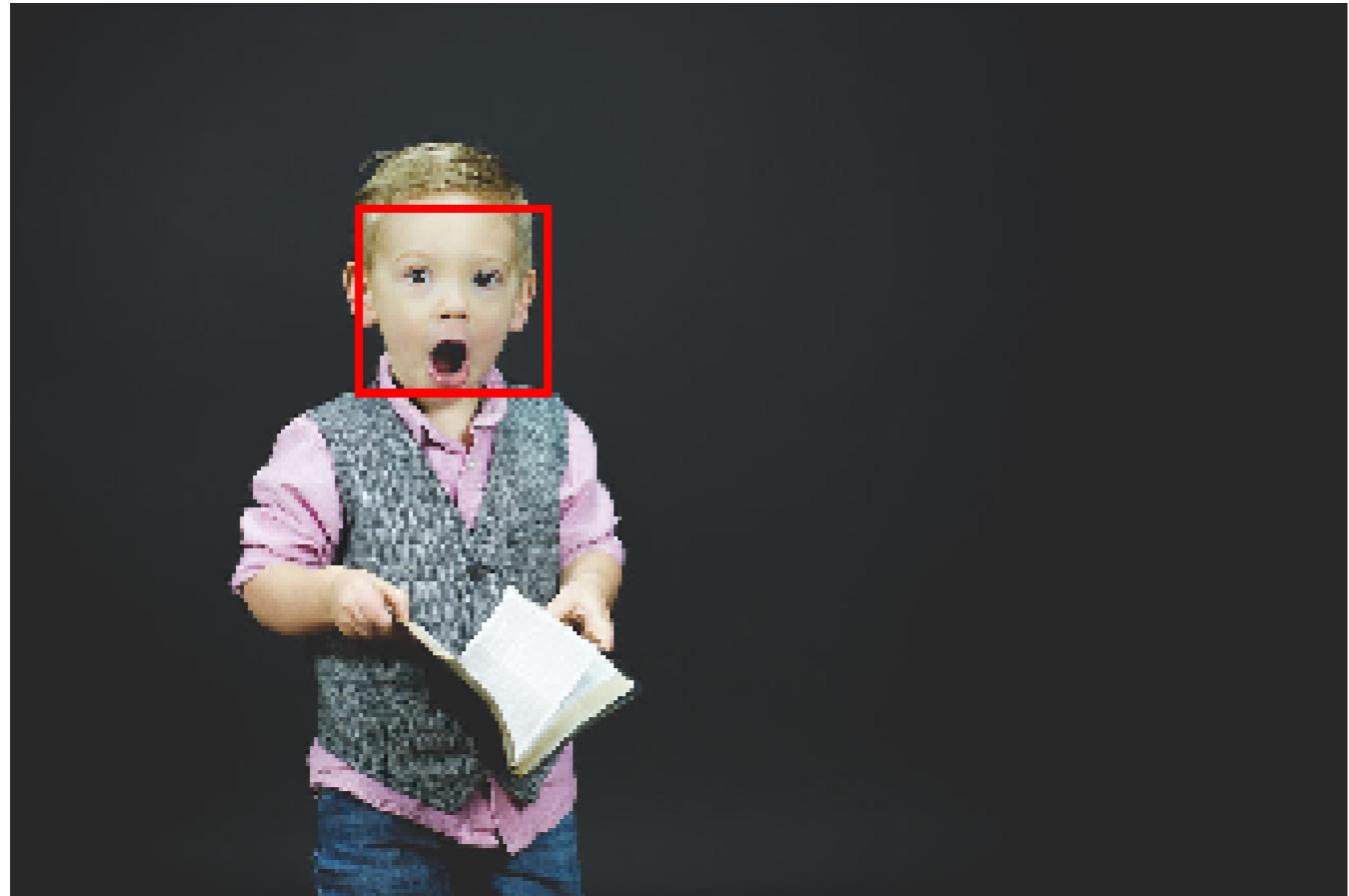
Face detection use cases

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

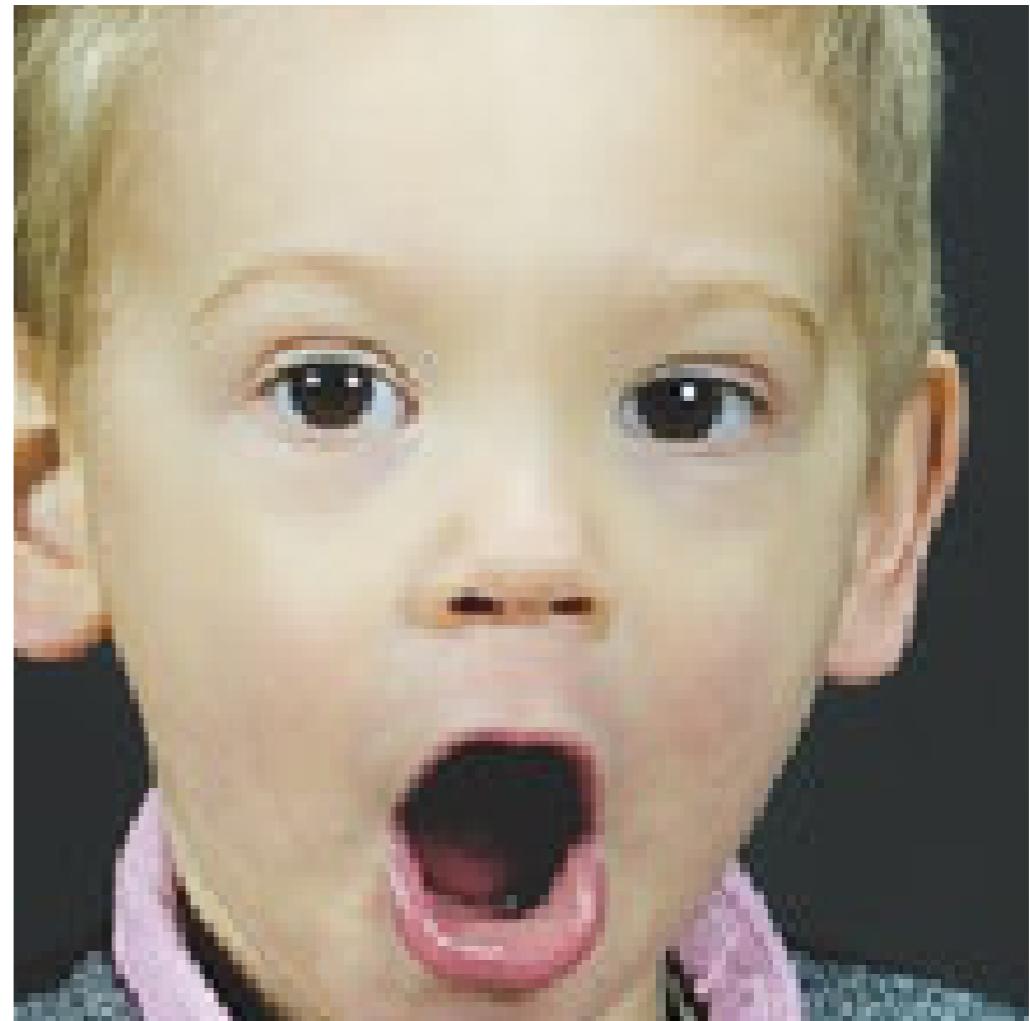


Detecting faces with scikit-image

Face image



Face detected



Detecting faces with scikit-image

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

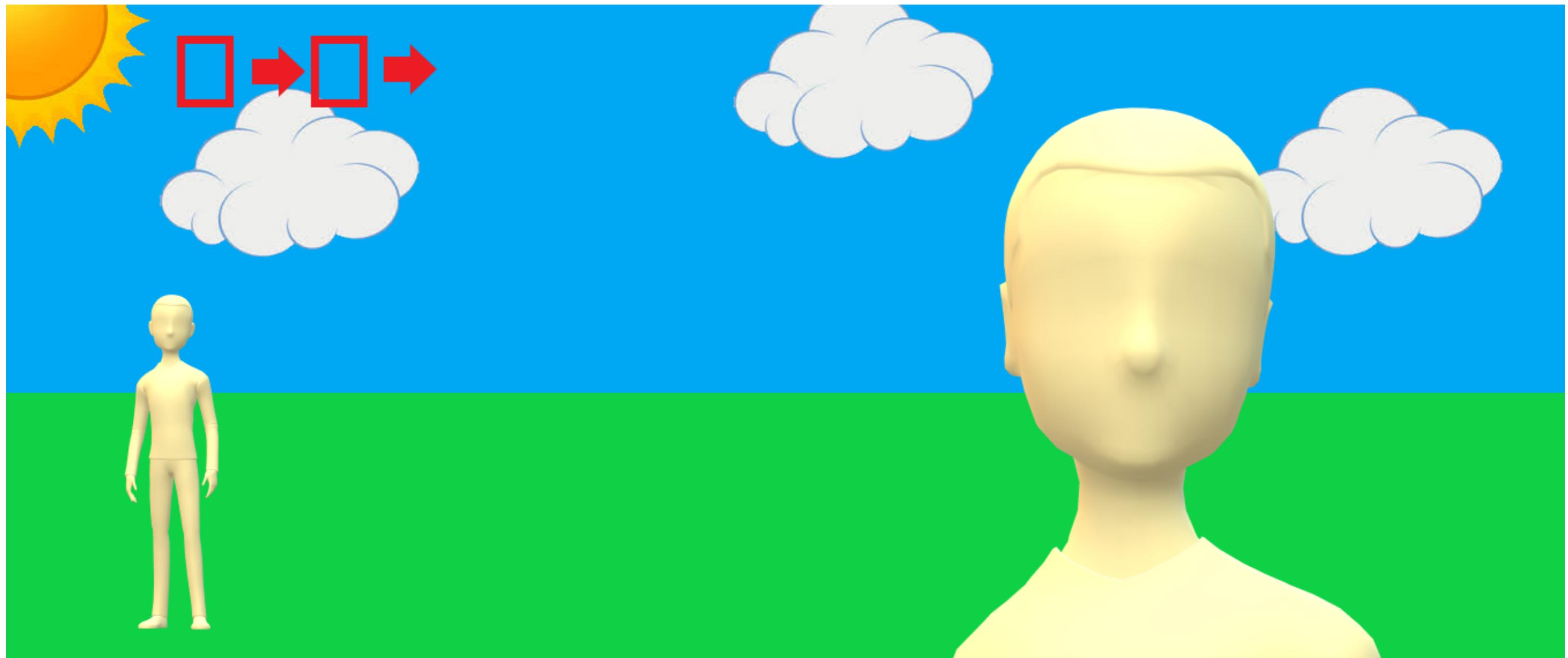
# Load the trained file from the module root.
trained_file = data.lbp_frontal_face_cascade_filename()

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

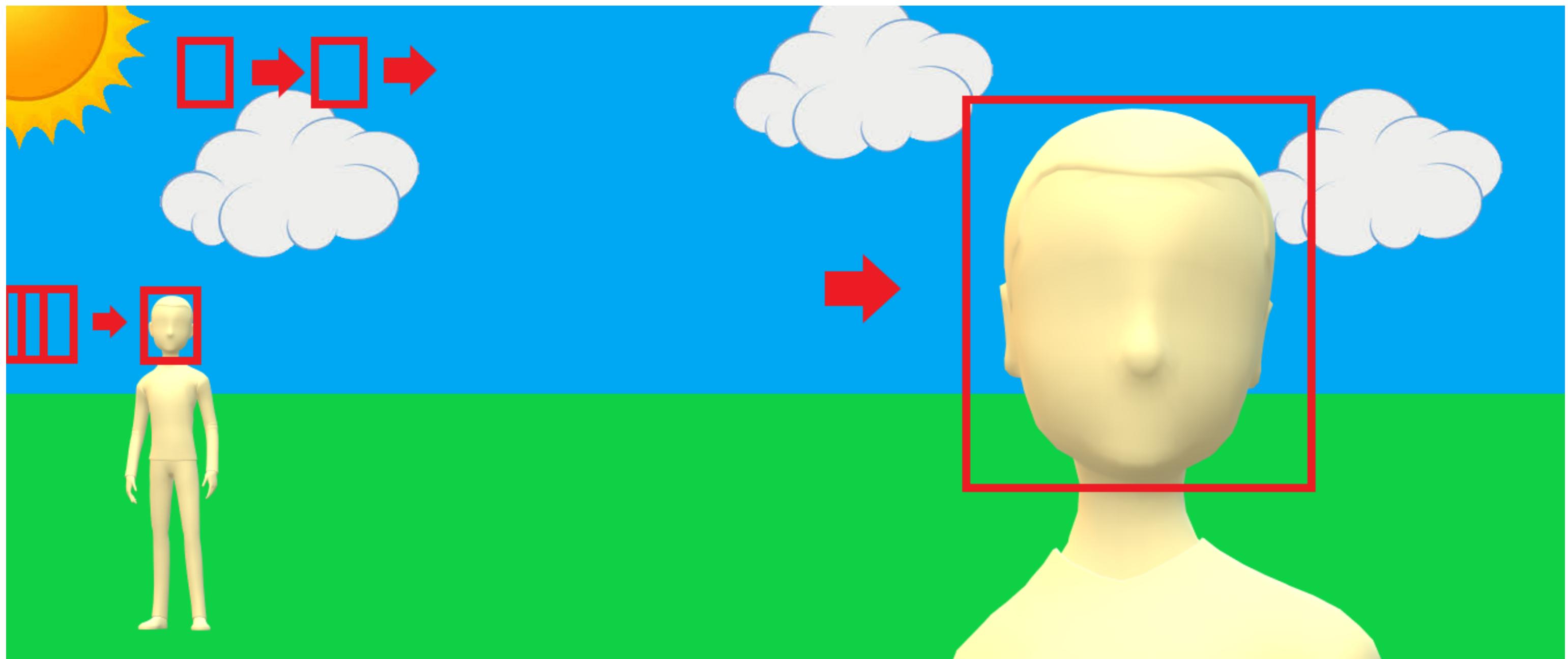
Let's try it



Detecting faces



Detecting faces



Detecting faces

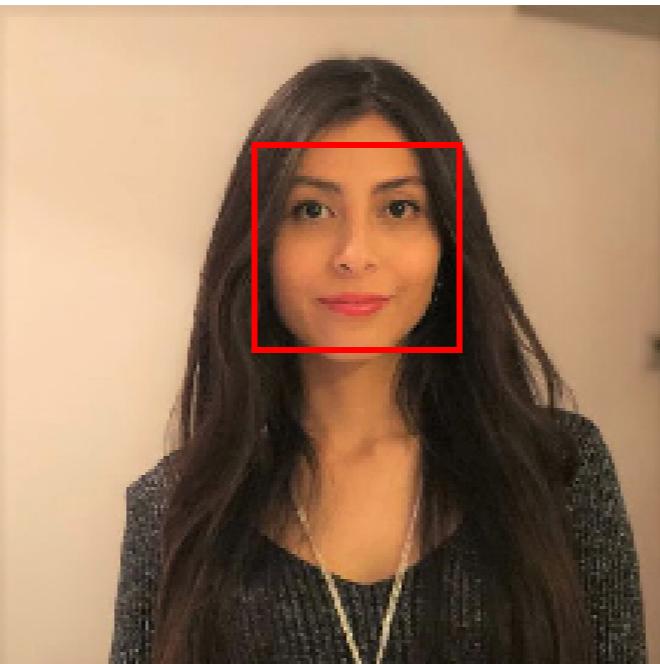
```
# Apply detector on the image
detected = detector.detect_multi_scale(img=image,
                                         scale_factor=1.2,
                                         step_ratio=1,
                                         min_size=(10, 10),
                                         max_size=(200,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

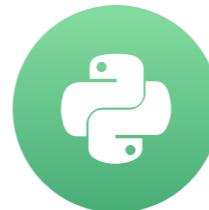
```
def show_detected_face(result, detected, title="Face image"):  
    plt.imshow(result)  
    img_desc = plt.gca()  
    plt.set_cmap('gray')  
    plt.title(title)  
    plt.axis('off')  
  
    for patch in detected:  
        img_desc.add_patch(  
            patches.Rectangle(  
                (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

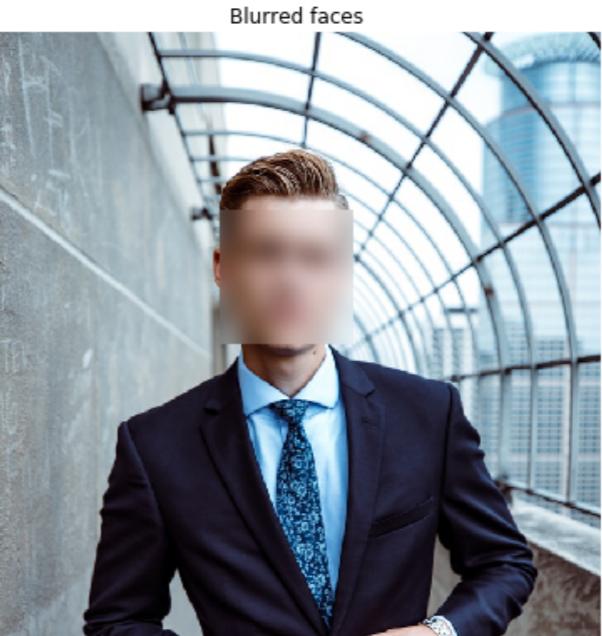
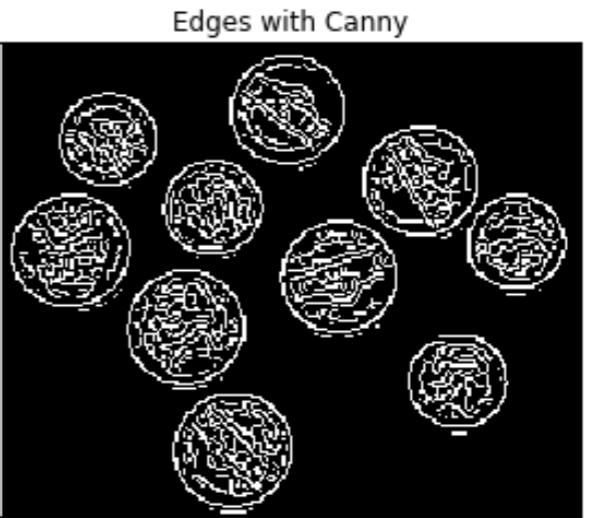
IMAGE PROCESSING IN PYTHON



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

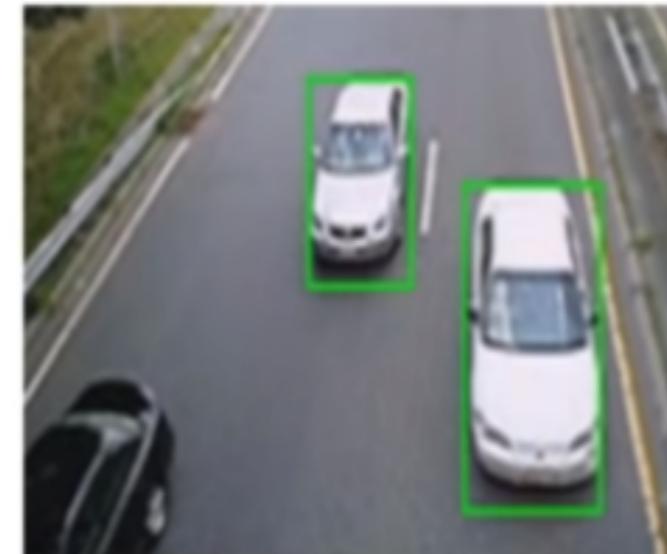
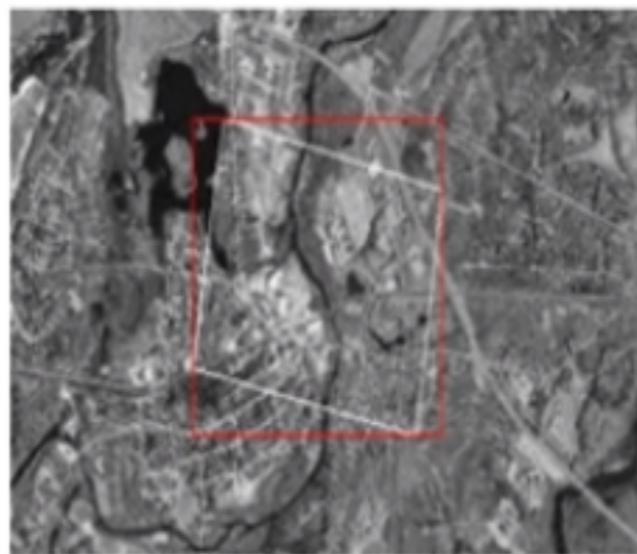
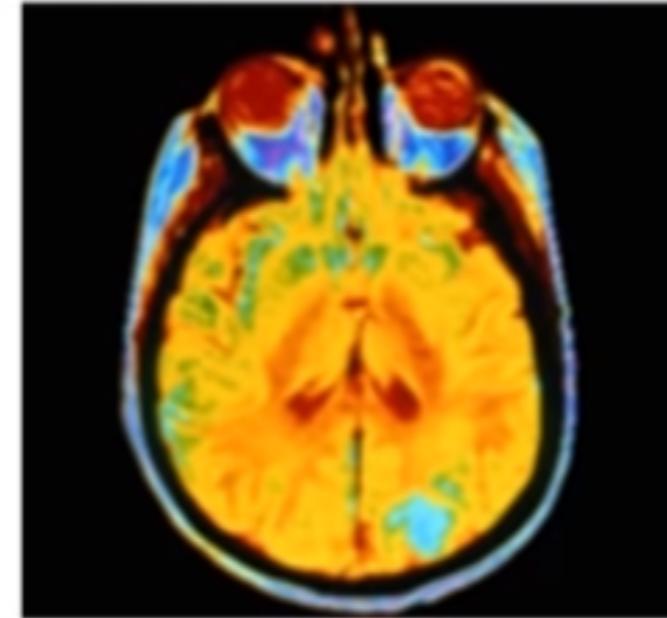
```
def mergeBlurryFace(original, gaussian_image):  
    # X and Y starting points of the face rectangle  
    x, y = d['r'], d['c']  
    # The width and height of the face rectangle  
    width, height = d['r'] + d['width'], d['c'] + d['height']  
  
    original[ x:width, y:height] = gaussian_image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

Privacy protection

Blurred faces



More cases



Let's practice!

IMAGE PROCESSING IN PYTHON

Amazing work!

IMAGE PROCESSING IN PYTHON



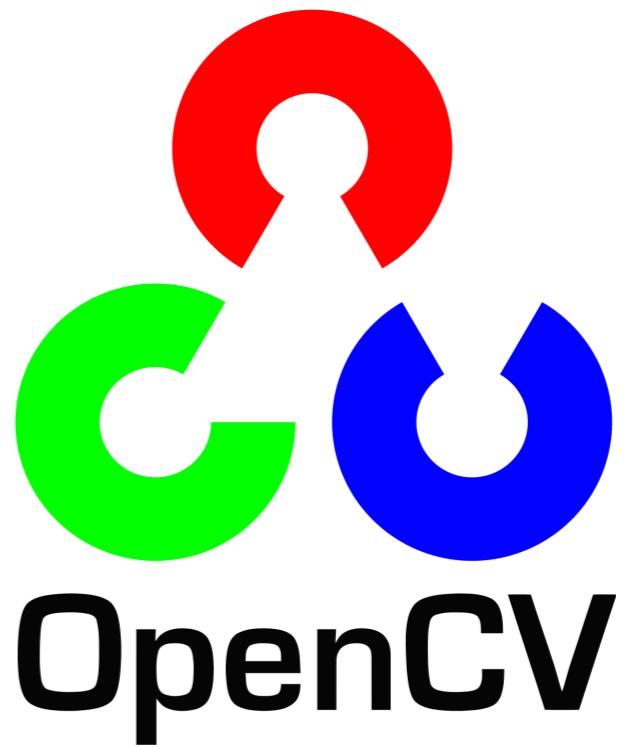
Rebeca Gonzalez
Data Engineer

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