Unit III Image Data Analytics

Image Acquisition and Representation, Image Resizing and Rescaling, Image Rotation, Image Intensity, Image Cropping, Edge Extraction using Sobel Filter, Edge Extraction using Prewitt Filter

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

Manipulating exposure and color channels

RGB to grayscale

This example converts an image with RGB channels into an image with a single grayscale channel.

The value of each grayscale pixel is calculated as the weighted sum of the corresponding red, green and blue pixels as:

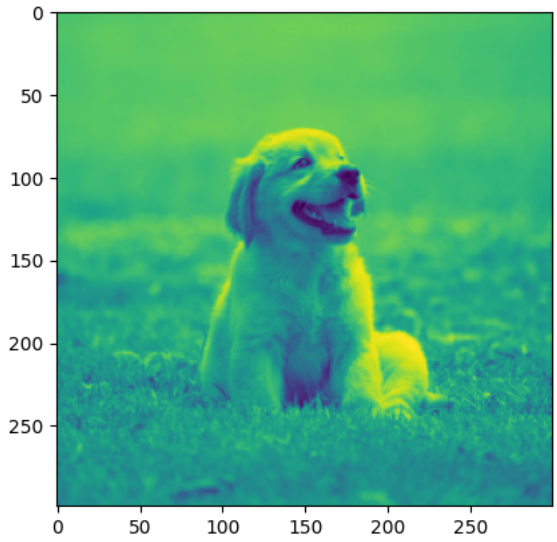
Y = 0.2125 R + 0.7154 G + 0.0721 B

```
import matplotlib.pyplot as plt
from skimage import data
from skimage.color import rgb2gray
original = data.astronaut()
grayscale = rgb2gray(original)
fig, axes = plt.subplots(1, 2, figsize=(8, 4))
ax = axes.ravel()
ax[0].imshow(original)
ax[0].set_title("Original")
ax[1].imshow(grayscale, cmap=plt.cm.gray)
ax[1].set_title("Grayscale")
fig.tight_layout()
plt.show()
```

Q.1 Take a cat image from data and it original to Grayscale


```
from skimage import io
from matplotlib import pyplot as plt
img =io.imread("images/test_image.jpg",as_gray=True)
from skimage.transform import rescale, resize, downscale_local_mean

rescaled_img =rescale(img,1.0/4.0,anti_aliasing=True)
resized_img =resize(img,(200,200))
downscaled_img =downscale_local_mean(img,(4,3))
plt.imshow(downscaled_img)
```

```
<matplotlib.image.AxesImage at 0x7fc090146b90>
```



Edge detection using Prewitt, Scharr and Sobel Operator

The discontinuity in the image brightness is called an edge. Edge detection is the technique used to identify the regions in the image where the brightness of the image changes sharply. This sharp change in the intensity value is observed at the local minima or local maxima in the image histogram, using the first-order derivative.

The techniques used here in this article are first-order derivative techniques namely:

Prewitt Operator

Scharr Operator

Sobel Operator

Prewitt Operator

The Prewitt operator was developed by Judith M. S. Prewitt. Prewitt operator is used for edge detection in an image. Prewitt operator detects both types of edges, these are:

Horizontal edges or along the x-axis,

Vertical Edges or along the y-axis.

Wherever there is a sudden change in pixel intensities, an edge is detected by the mask. Since the edge is defined as the change in pixel intensities, it can be calculated by using differentiation. Prewitt mask is a first-order derivative mask. In the graph representation of Prewitt-mask's result, the edge is represented by the local maxima or local minima

Both the first and second derivative masks follow these three properties:

More weight means more edge detection.

The opposite sign should be present in the mask. (+ and -)

The Sum of the mask values must be equal to zero.

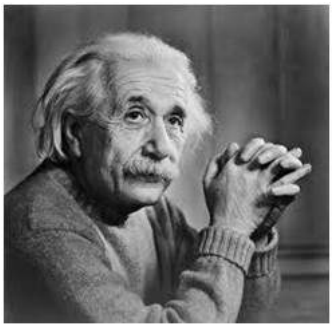
Prewitt operator provides us two masks one for detecting edges in the horizontal direction and another for detecting edges in a vertical direction.

Prewitt Operator [X-axis] = [-1 0 1; -1 0 1; -1 0 1]

Prewitt Operator [Y-axis] = [-1 -1 -1; 0 0 0; 1 1 1]

Sample Image

Following is a sample picture on which we will apply above two masks one at time.



After applying Vertical Mask

After applying vertical mask on the above sample image, following image will be obtained. This image contains vertical edges. You can judge it more correctly by comparing with horizontal edges picture.

After applying Horizontal Mask

After applying horizontal mask on the above sample image, following image will be obtained.

Comparison

As you can see that in the first picture on which we apply vertical mask, all the vertical edges are more visible than the original image. Similarly in the second picture we have applied the horizontal mask and in result all the horizontal edges are visible. So in this way you can see that we can detect both horizontal and vertical edges from an image.

Sobel Operator

The sobel operator is very similar to Prewitt operator. It is also a derivate mask and is used for edge detection. Like Prewitt operator sobel operator is also used to detect two kinds of edges in an image:

Vertical direction

Horizontal direction

Difference with Prewitt Operator

The major difference is that in sobel operator the coefficients of masks are not fixed and they can be adjusted according to our requirement unless they do not violate any property of derivative masks.

Following is the vertical Mask of Sobel Operator:

Following is the vertical Mask of Sobel Operator:

-1	0	1
-2	0	2
-1	0	1

This mask works exactly same as the Prewitt operator vertical mask. There is only one difference that is it has "2" and "-2" values in center of first and third column. When applied on an image this mask will highlight the vertical edges.

How it works

When we apply this mask on the image it prominent vertical edges. It simply works like as first order derivate and calculates the difference of pixel intensities in a edge region.

As the center column is of zero so it does not include the original values of an image but rather it calculates the difference of right and left pixel values around that edge. Also the center values of both the first and third column is 2 and -2 respectively.

This give more weight age to the pixel values around the edge region. This increase the edge intensity and it become enhanced comparatively to the original image.

Following is the horizontal Mask of Sobel Operator

-1	-2	-1
0	0	0
1	2	1

Above mask will find edges in horizontal direction and it is because that zeros column is in horizontal direction. When you will convolve this mask onto an image it would prominent horizontal edges in the image. The only difference between it is that it have 2 and -2 as a center element of first and third row.

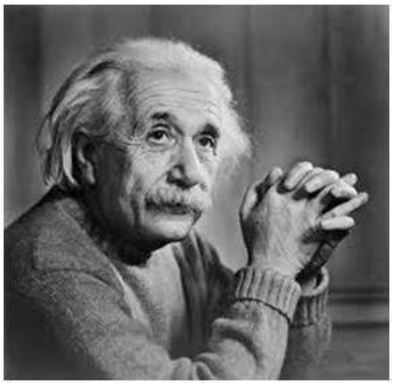
How it works

This mask will prominent the horizontal edges in an image. It also works on the principle of above mask and calculates difference among the pixel intensities of a particular edge. As the center row of mask is consist of zeros so it does not include the original values of edge in the image but rather it calculate the difference of above and below pixel intensities of the particular edge. Thus increasing the sudden change of intensities and making the edge more visible.

Now it’s time to see these masks in action:

Sample Image

Following is a sample picture on which we will apply above two masks one at time.



After applying Vertical Mask

After applying vertical mask on the above sample image, following image will be obtained.

After applying Horizontal Mask

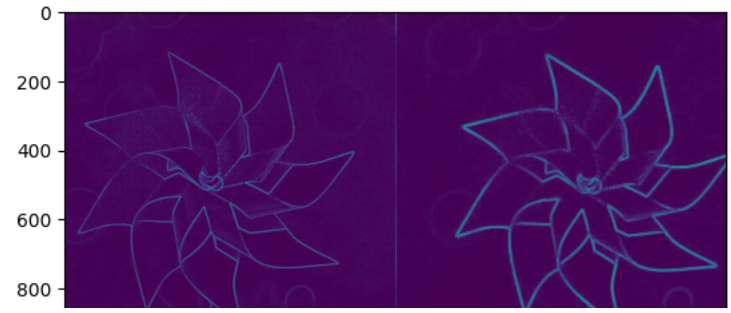
After applying horizontal mask on the above sample image, following image will be obtained

- Comparison

As we can see that in the first picture on which we apply vertical mask, all the vertical edges are more visible than the original image. Similarly in the second picture we have applied the horizontal mask and in result all the horizontal edges are visible.

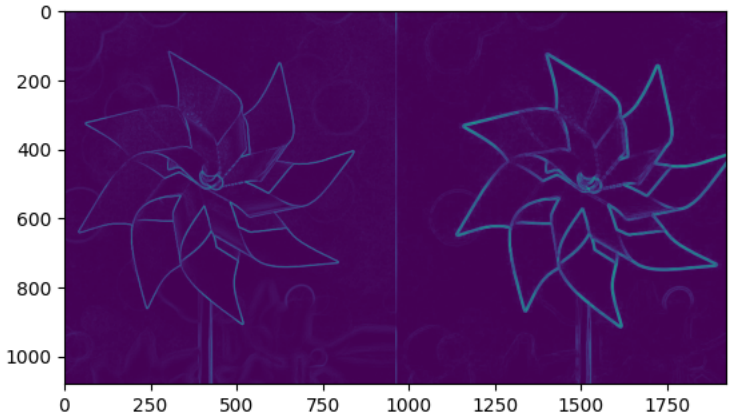
```
from skimage import io
from matplotlib import pyplot as plt


img =io.imread("edge.png",as_gray=True)
from skimage.filters import roberts,sobel,scharr,prewitt
edge_roberts=roberts(img)
plt.imshow(edge_roberts)
```

```
<matplotlib.image.AxesImage at 0x7b81b5d2f9d0>
```



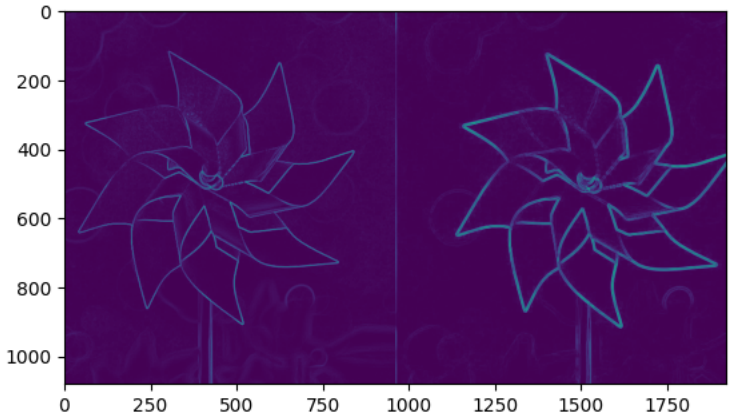

```
img =io.imread("edge.png",as_gray=True)
from skimage.filters import roberts,sobel,scharr,prewitt
edge_roberts=sobel(img)
plt.imshow(edge_roberts)
```

```
<matplotlib.image.AxesImage at 0x7b81b4c49db0>
```




```
img =io.imread("edge.png",as_gray=True)
from skimage.filters import roberts,sobel,scharr,prewitt
edge_roberts=scharr(img)
plt.imshow(edge_roberts)
```

```
<matplotlib.image.AxesImage at 0x7b81b4d10af0>
```



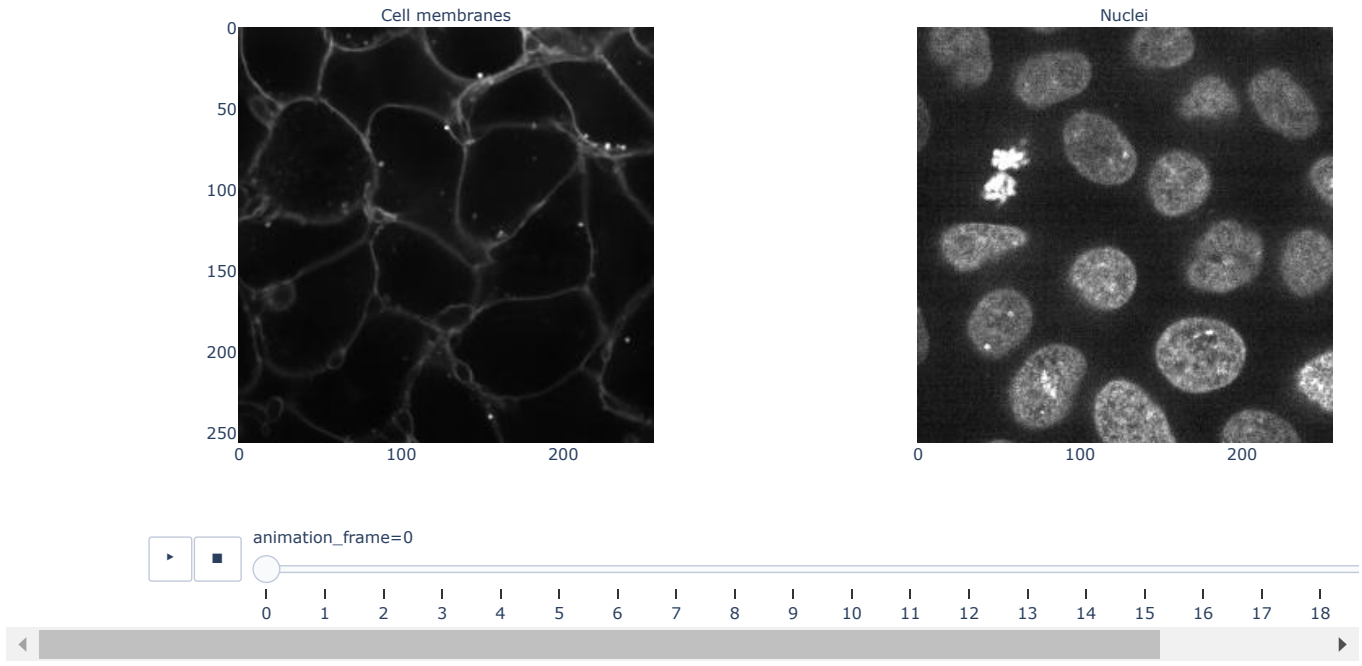
Datasets with 3 or more spatial dimensions

Most scikit-image functions are compatible with 3D datasets, i.e., images with 3 spatial dimensions (to be distinguished from 2D multichannel images, which are also arrays with three axes). skimage.data.cells3d() returns a 3D fluorescence microscopy image of

cells. The returned dataset is a 3D multichannel image with dimensions provided in (z, c, y, x) order. Channel 0 contains cell membranes, while channel 1 contains nuclei.

Double-click (or enter) to edit

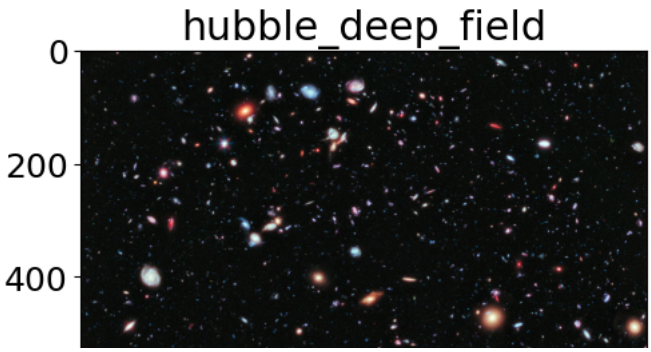
```
from skimage import data
import plotly
import plotly.express as px
import numpy as np

img = data.cells3d()[20:]

# omit some slices that are partially empty
img = img[5:26]

upper_limit = 1.5 * np.percentile(img, q=99)
img = np.clip(img, 0, upper_limit)

fig = px.imshow(
    img,
    facet_col=1,
    animation_frame=0,
    binary_string=True,
    binary_format="jpg",
)
fig.layout.annotations[0]["text"] = "Cell membranes"
fig.layout.annotations[1]["text"] = "Nuclei"
plotly.io.show(fig)
```



Scientific images

The title of each image indicates the name of the function

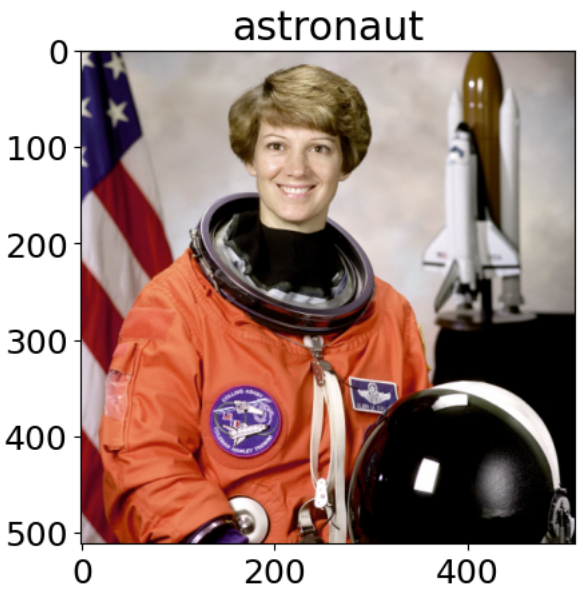
```
import matplotlib.pyplot as plt
import matplotlib
import numpy as np

from skimage import data

matplotlib.rcParams['font.size'] = 18

images = ('hubble_deep_field',
          'immunohistochemistry'

          )


for name in images:
    caller = getattr(data, name)
    image = caller()
    plt.figure()
    plt.title(name)
    if image.ndim == 2:
        plt.imshow(image, cmap=plt.cm.gray)
    else:
        plt.imshow(image)

plt.show()
```

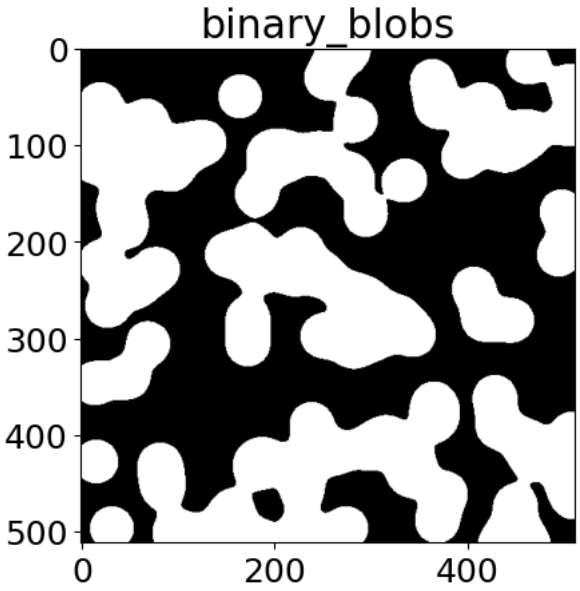


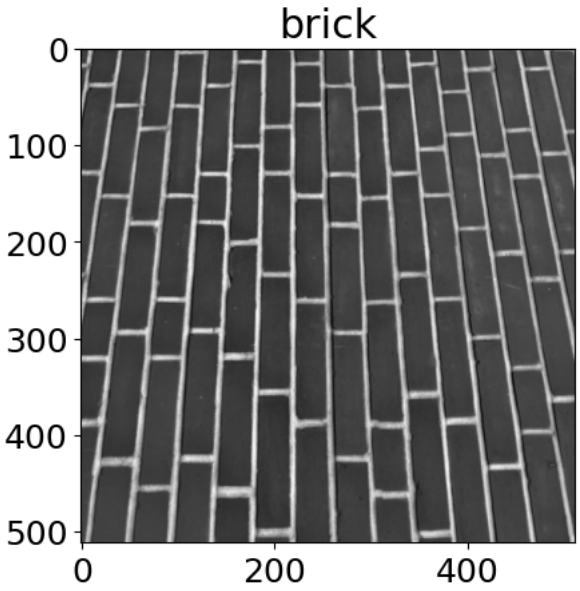
Double-click (or enter) to edit

General-purpose images

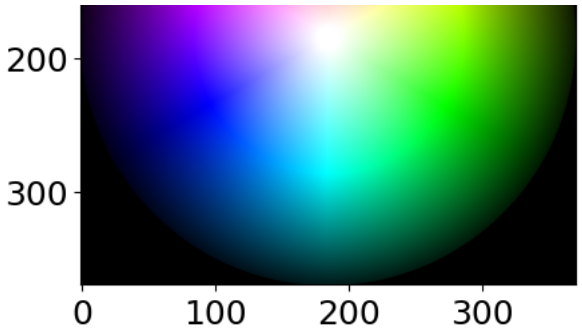
```
import matplotlib.pyplot as plt
import matplotlib
import numpy as np

from skimage import data

matplotlib.rcParams['font.size'] = 18

images = ('astronaut',

          'rocket',
          )


for name in images:
    caller = getattr(data, name)
    image = caller()
    plt.figure()
    plt.title(name)
    if image.ndim == 2:
        plt.imshow(image, cmap=plt.cm.gray)
    else:
        plt.imshow(image)

plt.show()
```

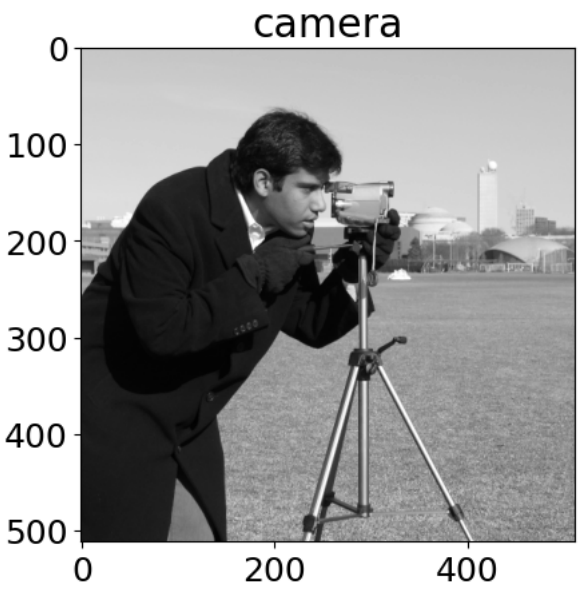

astronaut
0
100
200
300
400
500
0
200
400

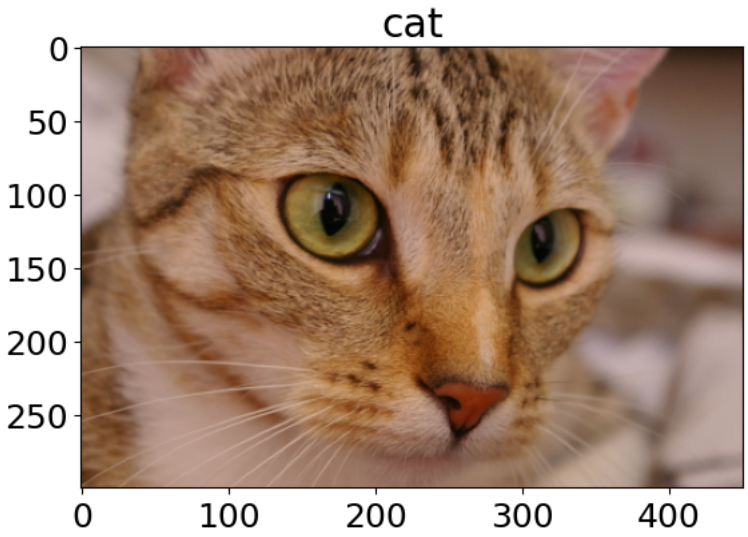

binary_blobs
0
100
200
300
400
500
0
200
400

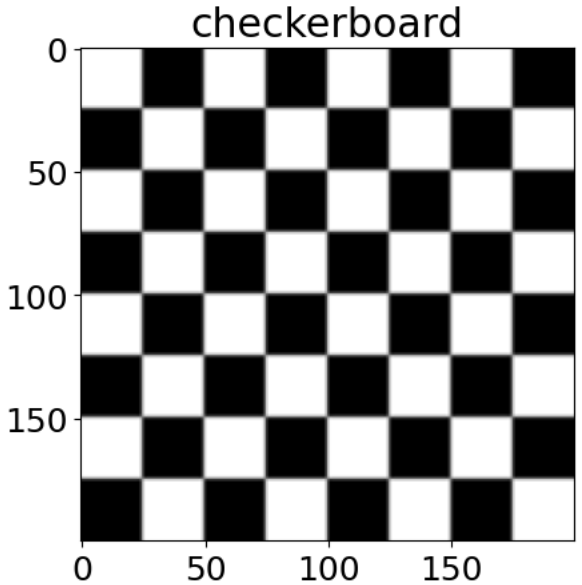

brick
0
100
200
300
400
500
0
200
400

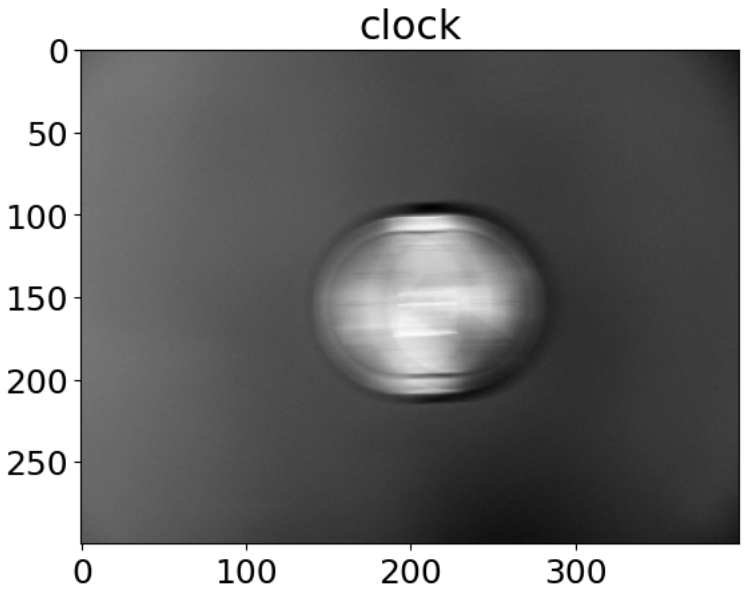

colorwheel
0
100

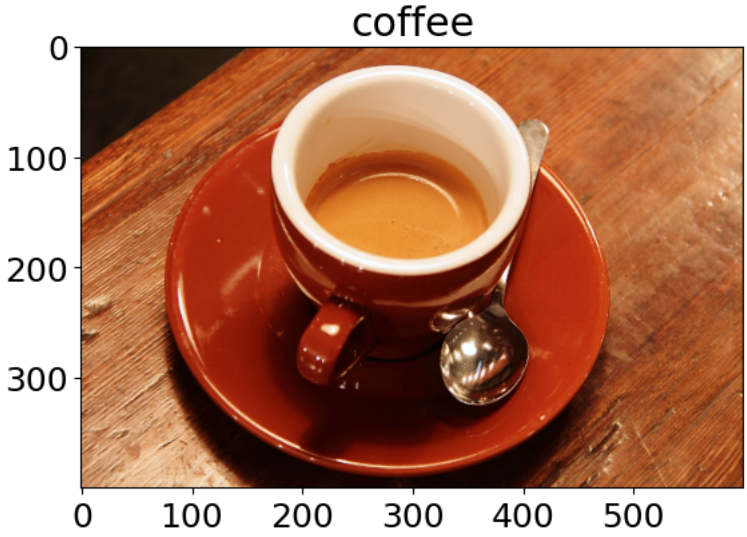

200
300
0
100
200
300

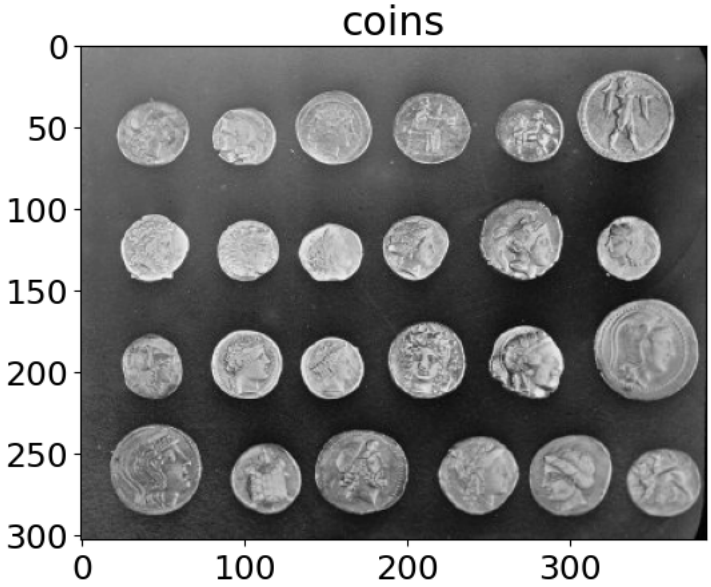

camera
0
100
200
300
400
500
0
200
400

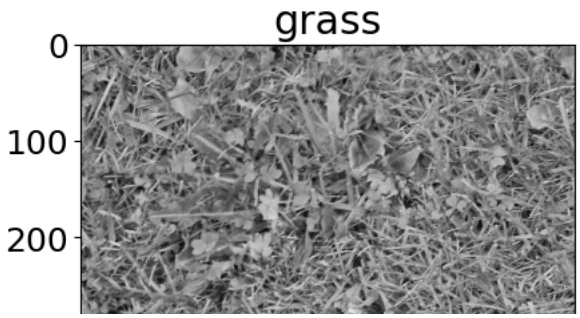

cat
0
50
100
150
200
250
0
100
200
300
400

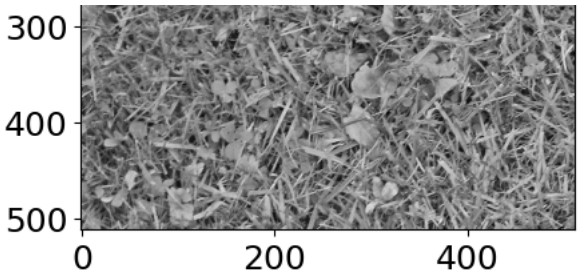

checkerboard
0
50
100
150
0
50
100
150

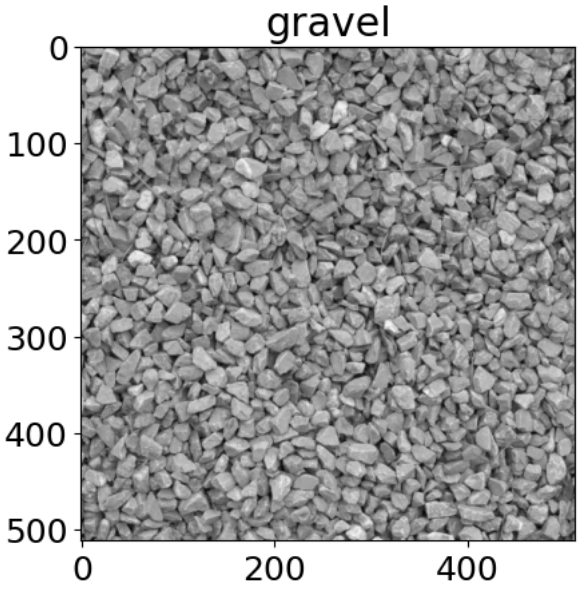

clock
0
50
100
150
200
250
0
100
200
300

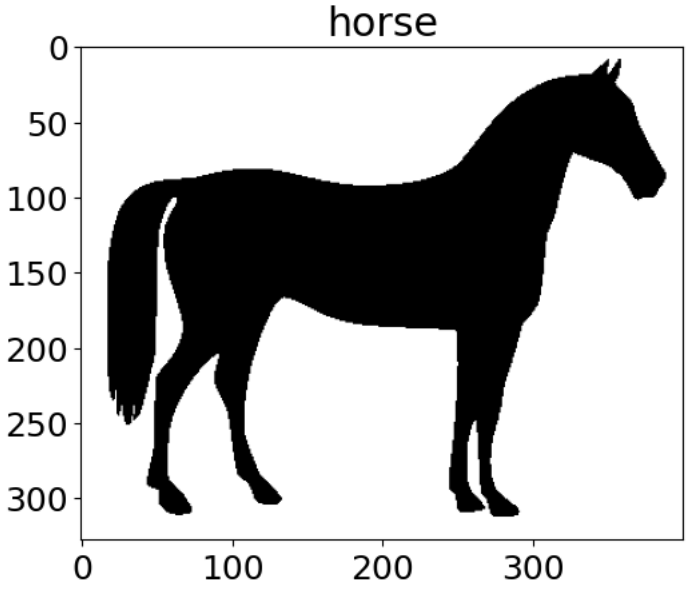

coffee
0
100
200
300
0
100
200
300
400
500

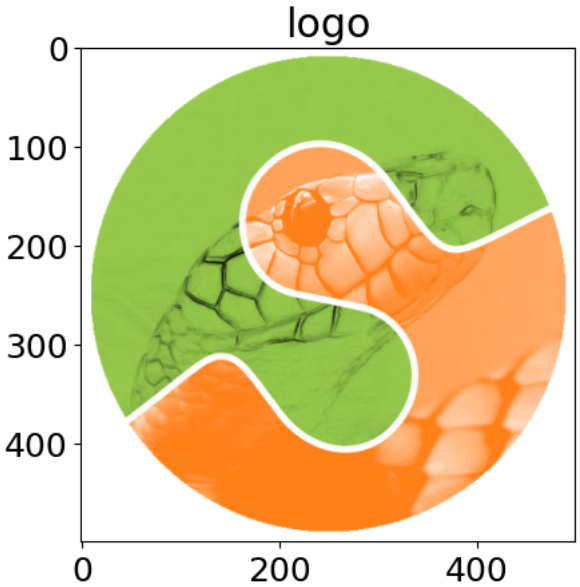

coins
0
50
100
150
200
250
300
0
100
200
300

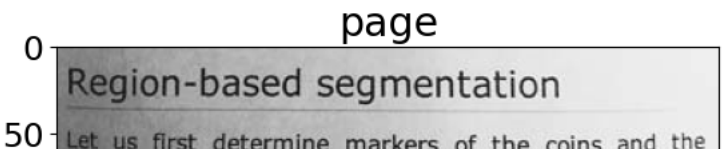

grass
0
100
200


300
400
500
0
200
400


gravel
0
100
200
300
400
500
0
200
400

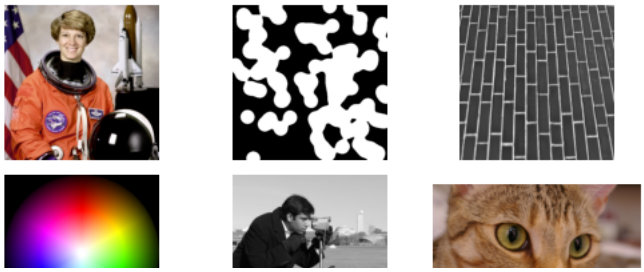

horse
0
50
100
150
200
250
300
0
100
200
300


logo
0
100
200
300
400
0
200
400



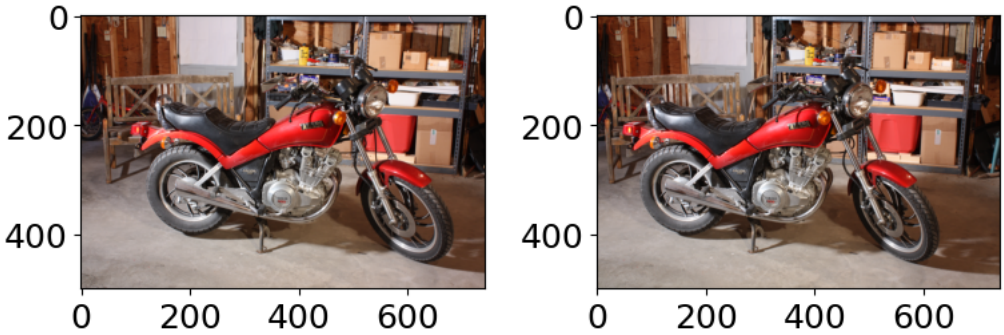
Thumbnail image for the gallery

```
fig, axs = plt.subplots(nrows=3, ncols=3)
for ax in axs.flat:
    ax.axis("off")
axs[0, 0].imshow(data.astronaut())
axs[0, 1].imshow(data.binary_blobs(), cmap=plt.cm.gray)
axs[0, 2].imshow(data.brick(), cmap=plt.cm.gray)
axs[1, 0].imshow(data.colorwheel())
axs[1, 1].imshow(data.camera(), cmap=plt.cm.gray)
axs[1, 2].imshow(data.cat())
axs[2, 0].imshow(data.checkerboard(), cmap=plt.cm.gray)
axs[2, 1].imshow(data.clock(), cmap=plt.cm.gray)
further_img = np.full((300, 300), 255)
for xpos in [100, 150, 200]:
    further_img[150 - 10 : 150 + 10, xpos - 10 : xpos + 10] = 0
axs[2, 2].imshow(further_img, cmap=plt.cm.gray)
plt.subplots_adjust(wspace=0.1, hspace=0.1)
```



Specific images

```
import matplotlib.pyplot as plt
import matplotlib

from skimage import data

matplotlib.rcParams["font.size"] = 18

fig, axes = plt.subplots(1, 2, figsize=(8, 4))
ax = axes.ravel()

cycle_images = data.stereo_motorcycle()
ax[0].imshow(cycle_images[0])
ax[1].imshow(cycle_images[1])

fig.tight_layout()
plt.show()
```



Q.1 Find the following images from Data using scikit-image libabry

'lily','camera','logo'

'cat','bricks', 'coffee'

'skin', 'cell','clock'

colormap should be gray

arrange into Thumbnail of 3X3 matrix

Using simple NumPy operations for manipulating images

Table Of Contents
Installation of Required Libraries
Importing the Required Libraries
Opening an Image
Details of an image
Saving ndarray as Image
Rotating an Image
Negative of an Image
Padding Black Spaces
Visualizing RGB Channels
Colour Reduction
Trim Image
Pasting With Slice
Binarize Image
Flip Image
An alternate way to Flip an Image
Blending Two Images
Masking Images
Histogram For Pixel Intensity

```
import numpy as np
from skimage import data
import matplotlib.pyplot as plt

camera = data.camera()
print(camera.shape)
camera[:10] = 0
mask = camera < 5
camera[mask] = 255
inds_x = np.arange(len(camera))
inds_y = (4 * inds_x) % len(camera)
camera[inds_x, inds_y] = 0

l_x, l_y = camera.shape[0], camera.shape[1]
X, Y = np.ogrid[:l_x, :l_y]
outer_disk_mask = (X - l_x / 2)**2 + (Y - l_y / 2)**2 > (l_x / 2)**2
camera[outer_disk_mask] = 0

plt.figure(figsize=(4, 4))
plt.imshow(camera, cmap='gray')
plt.axis('off')
plt.show()
```

```
(512, 512)
```

skimage.transform

This module includes tools to transform images and volumetric data.

Geometric transformation:

These transforms change the shape or position of an image. They are useful for tasks such as image registration, alignment, and geometric correction. Examples: AffineTransform, ProjectiveTransform, EuclideanTransform.

Image resizing and rescaling:

These transforms change the size or resolution of an image. They are useful for tasks such as down-sampling an image to reduce its size or up-sampling an image to increase its resolution. Examples: resize(), rescale().

Feature detection and extraction:

These transforms identify and extract specific features or patterns in an image. They are useful for tasks such as object detection, image segmentation, and feature matching. Examples: hough_circle(), pyramid_expand(), radon().

Image transformation:

These transforms change the appearance of an image without changing its content. They are useful for tasks such a creating image mosaics, applying artistic effects, and visualizing image data. Examples: warp(), iradon().

- Rescale, resize, and downscale

Rescale operation resizes an image by a given scaling factor.

The scaling factor can either be a single floating point value, or multiple values - one along each axis.

Resize serves the same purpose, but allows to specify an output image shape instead of a scaling factor.

Downscale serves the purpose of down-sampling an n-dimensional image by integer factors using the local mean on the elements of each block of the size factors given as a parameter to the function.

```
import numpy as np
from skimage import data
import matplotlib.pyplot as plt
from skimage.transform import rescale, resize, downscale_local_mean
a = np.arange(15).reshape(3, 5)
```

```
a
```

```
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14]])
```

```
downscale_local_mean(a, (2, 3))
```

```
array([[3.5, 4. ],
       [5.5, 4.5]])
```

```
import numpy as np

arr = np.array([[1, 2, 3], [4, 5, 6]])

print(arr.shape)
```

```
(2, 3)
```

```
downscale_local_mean(arr,factors=(3,3))
```

```
array([[2.33333333]])
```

```
image_rescaled = rescale(arr, 0.25, anti_aliasing=False)
image_resized = resize(arr, (image.shape[0] // 4, image.shape[1] // 4
                       anti_aliasing=True)
image_downscaled = downscale_local_mean(arr, (4, 3))

image_rescaled
```

```
array([[3.7947076e-19]])
```

```
image_resized
```

```
array([[3.21449003e-19, 3.18907905e-19, 3.16366806e-19, ...,
        4.37492517e-19, 4.34951418e-19, 4.32410320e-19],
       [3.16366806e-19, 3.13825707e-19, 3.11284608e-19, ...,
        4.32410320e-19, 4.29869221e-19, 4.27328122e-19],
       [3.11284608e-19, 3.08743509e-19, 3.06202410e-19, ...,
        4.27328122e-19, 4.24787023e-19, 4.22245924e-19],
       ...,
       [3.36695597e-19, 3.34154498e-19, 3.31613399e-19, ...,
        4.52739110e-19, 4.50198011e-19, 4.47656913e-19],
       [3.31613399e-19, 3.29072300e-19, 3.26531201e-19, ...,
        4.47656913e-19, 4.45115814e-19, 4.42574715e-19],
       [3.26531201e-19, 3.23990102e-19, 3.21449003e-19, ...,
        4.42574715e-19, 4.40033616e-19, 4.37492517e-19]])
```

```
image_downscaled
```

```
array([[1.75]])
```

```
import matplotlib.pyplot as plt

from skimage import data, color
from skimage.transform import rescale, resize, downscale_local_mean

image = color.rgb2gray(data.astronaut())

image_rescaled = rescale(image, 0.25, anti_aliasing=False)
image_resized = resize(image, (image.shape[0] // 4, image.shape[1] //
                       anti_aliasing=True)
image_downscaled = downscale_local_mean(image, (4, 3))

fig, axes = plt.subplots(nrows=2, ncols=2)

ax = axes.ravel()

ax[0].imshow(image, cmap='gray')
ax[0].set_title("Original image")

ax[1].imshow(image_rescaled, cmap='gray')
ax[1].set_title("Rescaled image (aliasing)")

ax[2].imshow(image_resized, cmap='gray')
ax[2].set_title("Resized image (no aliasing)")

ax[3].imshow(image_downscaled, cmap='gray')
ax[3].set_title("Downscaled image (no aliasing)")

ax[0].set_xlim(0, 512)
ax[0].set_ylim(512, 0)
plt.tight_layout()
plt.show()
```