

「画像処理基礎」

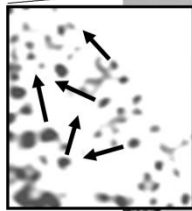
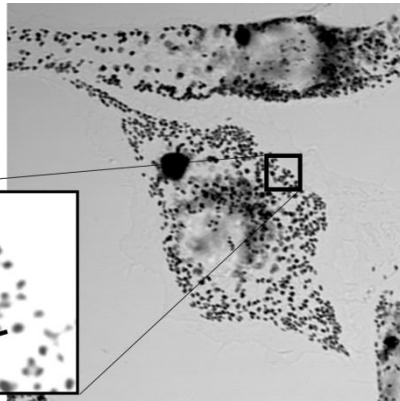
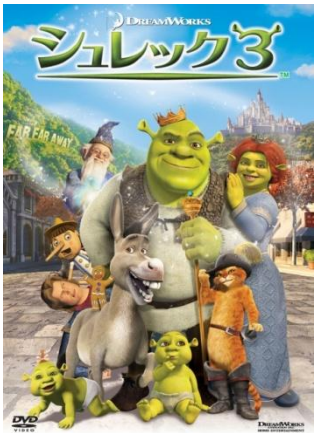
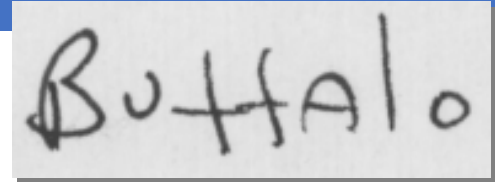
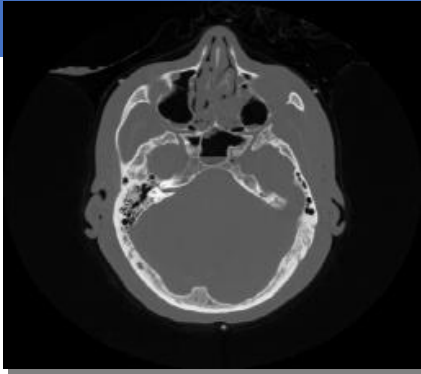
九州大学 大学院システム情報科学研究所
情報知能工学部門
データサイエンス実践特別講座
末廣大貴、Thomas Diego、正井克俊

画像解析概要

画像データ解析の流れ

画像データ (画像空間)

いろいろな画像



Does image data make sense?

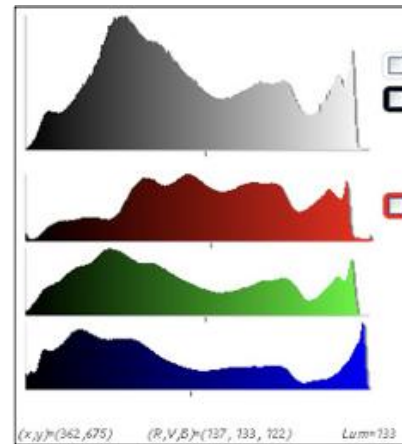
- The image is only pixel-by-pixel luminance value information



Meaningful information

A picture of the Edo period where a Japan woman wearing a kimono is playing the shamisen?

Computer:

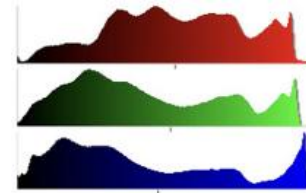


Extracting Information from Data

- People can extract various information from images that are only numerical information of pixels

Look into histograms

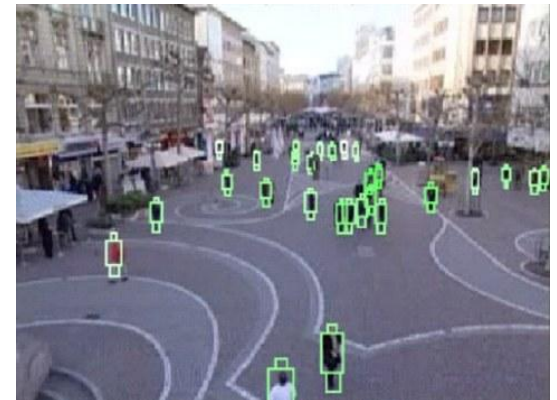
- What colors are most?



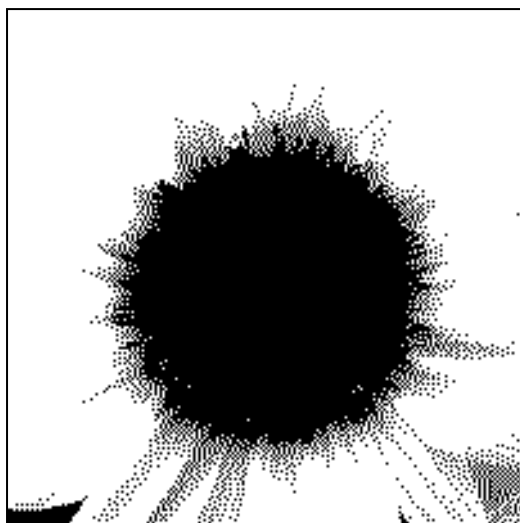
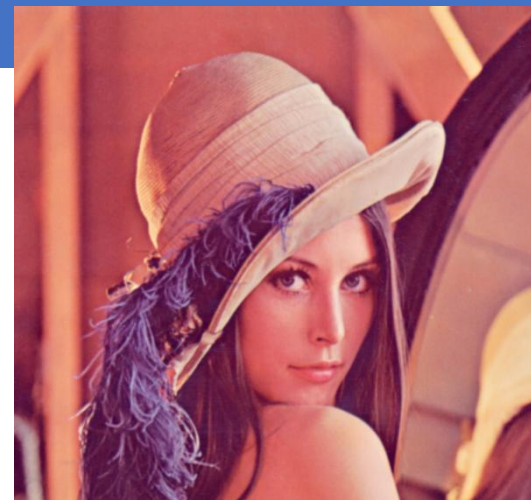
- Where is the edge?

- Where are the people?

- Where is the building located?



2値画像，濃淡画像，カラー画像



カラー画像



=



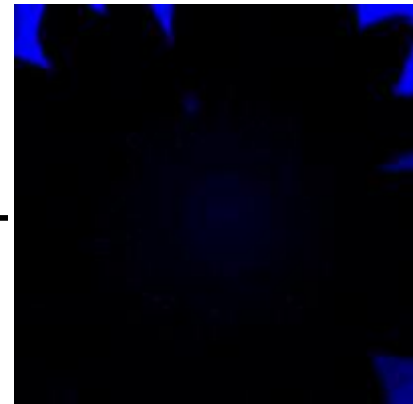
R成分

+



G成分

+



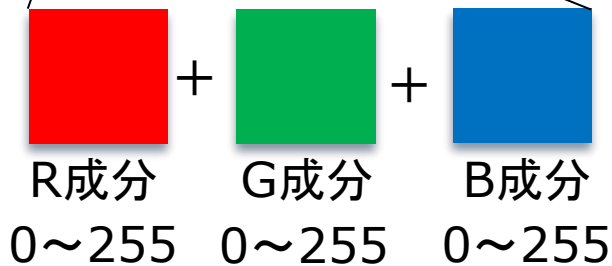
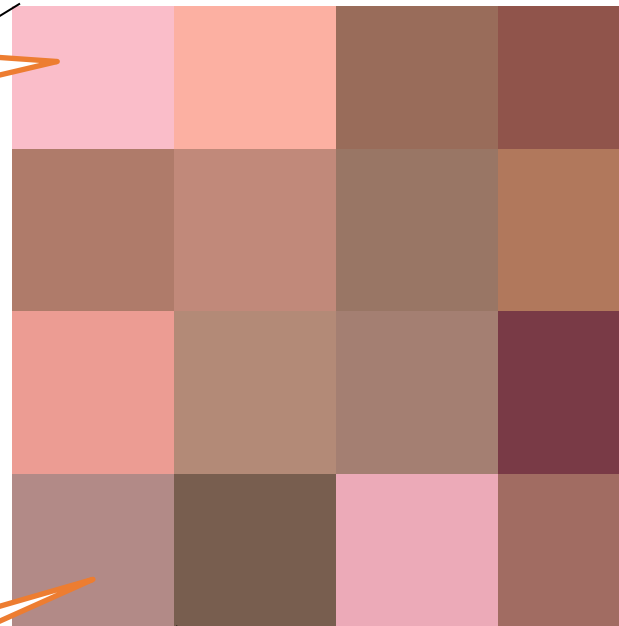
B成分

デジタル画像

A set of finite points (pixels)
that are regularly aligned



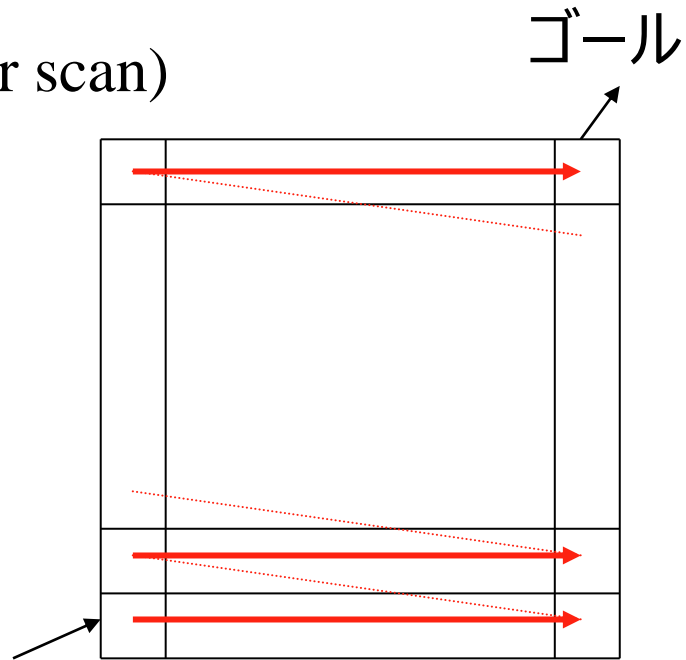
The color of each pixel is
represented by three
components: red, green, and blue





画像の走査 (スキャン)

- ラスタ走査 (raster scan)



スタート

すべての画素を1回ずつ通過



ラスタ走査 (つづき)

- Vector representation of images Enumerating Pixel Values in Raster Scanning Order

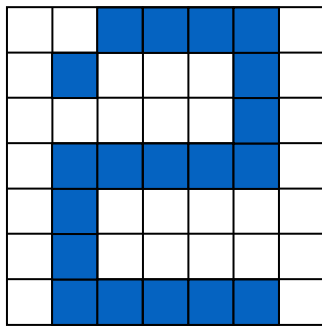
$$(f_{1,1}, \dots, f_{1,Y}, f_{2,1}, \dots, f_{x,y}, \dots, f_{X,Y})$$

XY次元ベクトル

- Any image = 1 point in XY dimensional space

画像のベクトル表現(vector representation of image)

- 2値画像画像の場合の例

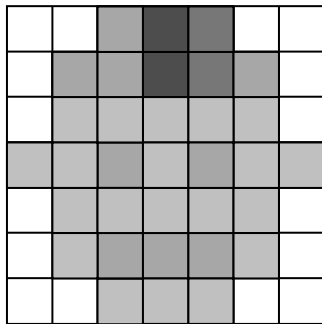


7×7画像

$$\Rightarrow x = \underbrace{(0, 0, 1, 1, 1, 1, 0, 0, 1, \dots, 1, 0)}^T$$

49次元ベクトル

- 多値画像画像の場合の例



7×7画像

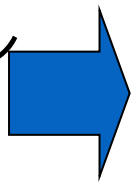
$$\Rightarrow x = \underbrace{(0, 0, 255, 213, 182, 0, 0, \dots, 0)}^T$$

49次元ベクトル

“画像空間” (image space)

$N \times N$ 画像

ゴール

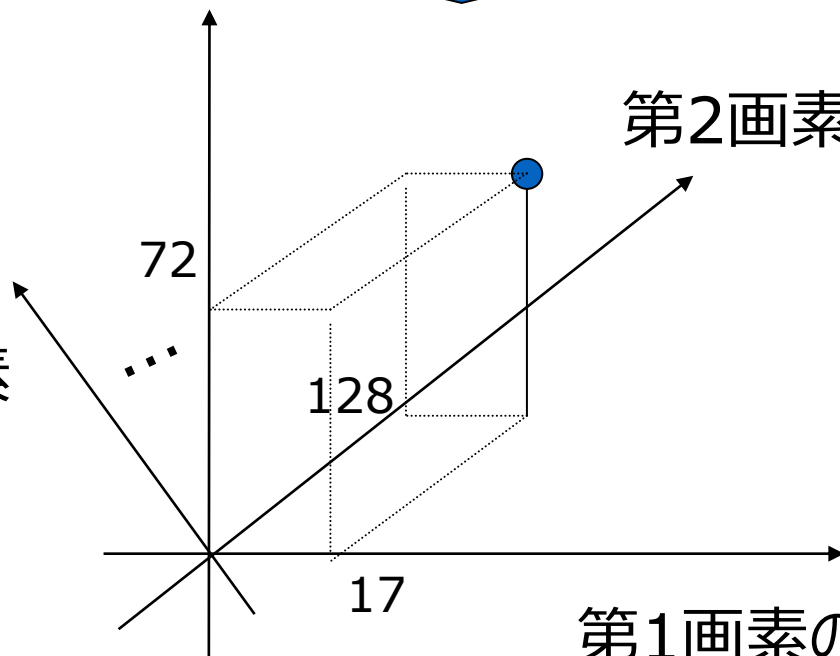


(17, 128, 72, ..., 153)

N^2 次元ベクトル

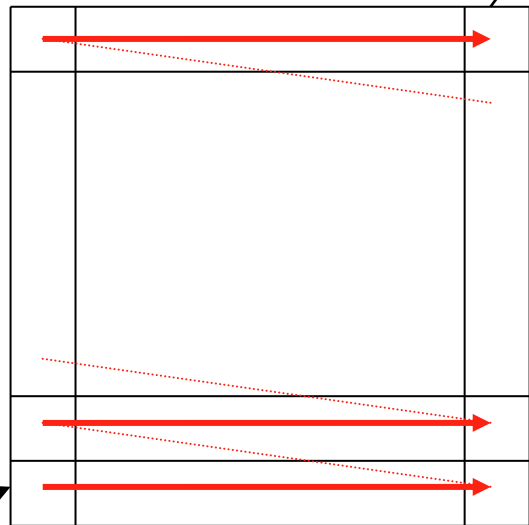


第2画素

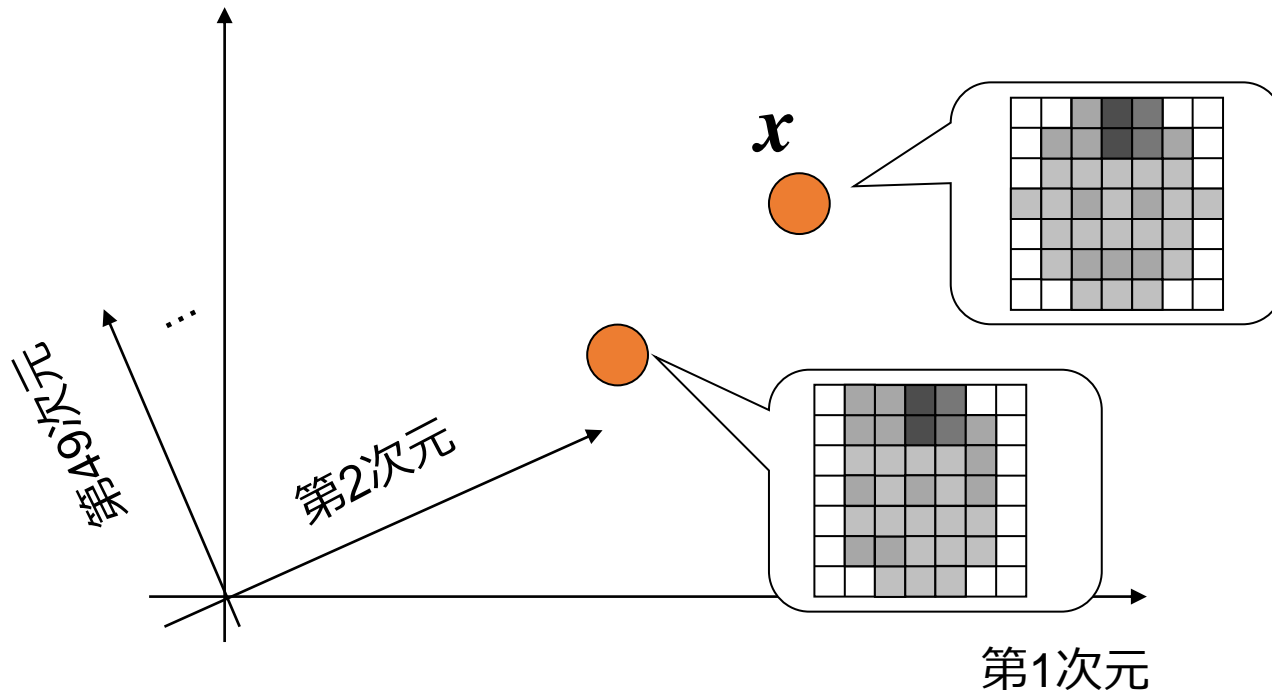


第 N^2 画素

スタート

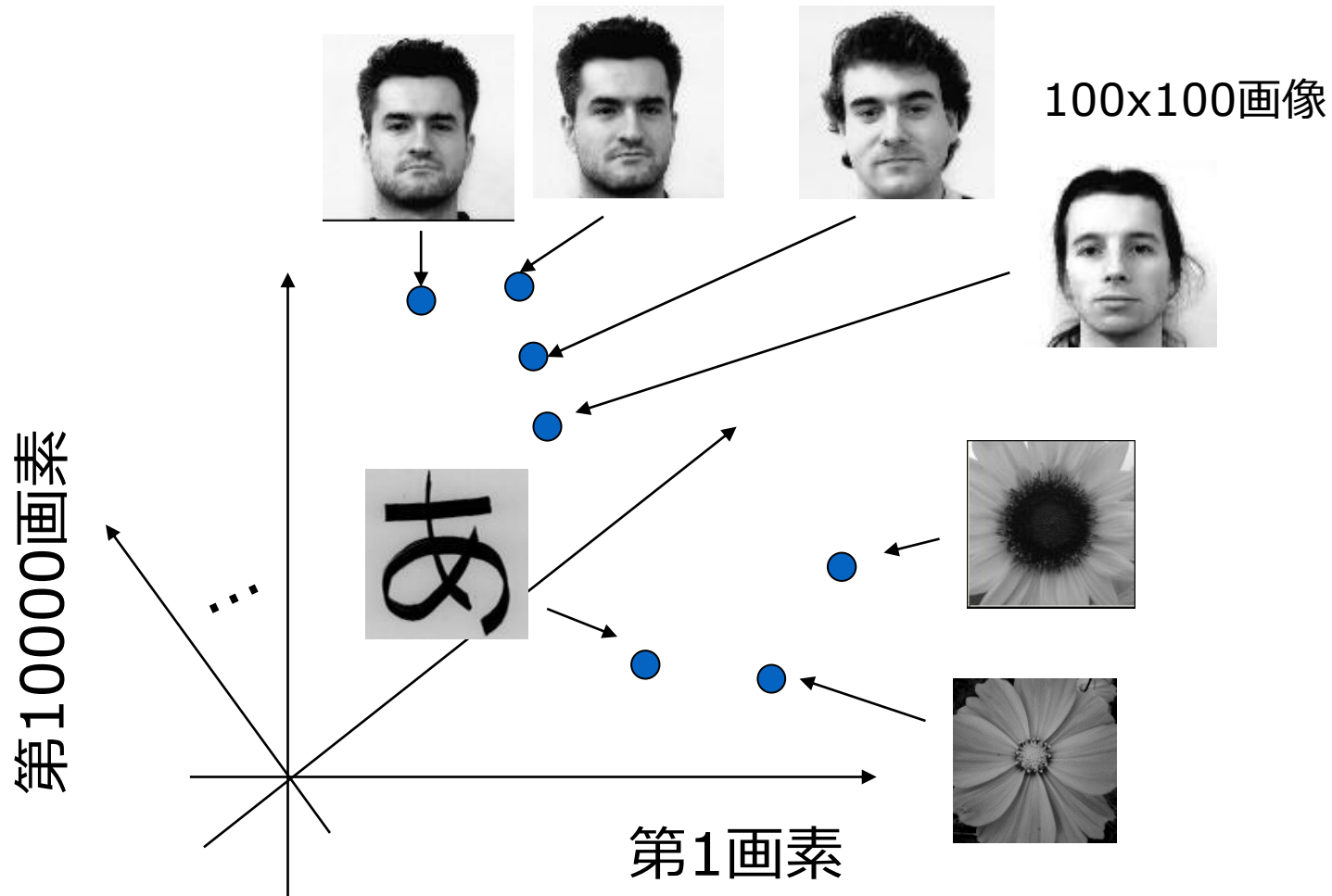


画像空間:任意の1点 = 1画像



A point in the image space corresponds to an image

画像空間

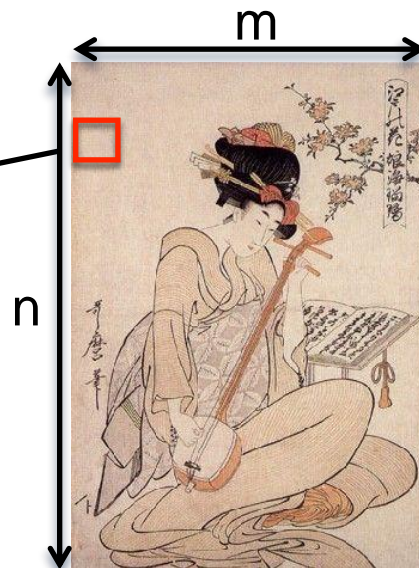
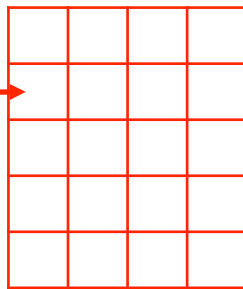


任意の100 x 100画像は10000次元空間で表せる！

画像データ

- カラー画像データは、ピクセルごとにRGBの3つのチャンネルの数字の組み合わせ

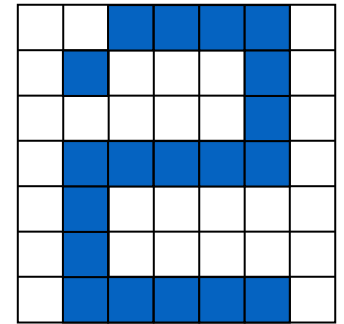
pixel = [red, green, blue]



- n 行 m 列の画像 : $n \times m$ ピクセルの数値
- カラー画像 : 1つのピクセルは3つのチャンネルの数値を持つ
 - red, green, blue.
- 例 : 100×100 のカラー画像
⇒ $3 \times 100 \times 100 = 30000$ の次元のベクトル

練習

- Numpyをインポートする
- ラスタ走査を使って “2”の画像をベクトルにする
白ピクセル = 0。青ピクセル = 255
- `numpy.array` を使ってデータ変換する
- `Reshape` を使ってベクトルから 7×7 の行列を求める
- 7×7 の行列をプリントする



Google Driveとの連携

準備: Google Driveへフォルダごとアップ

The screenshot shows the Google Drive web interface. On the left is a navigation sidebar with options: 'New', 'My Drive', 'Shared with me', 'Recent', 'Starred', 'Trash', and 'Storage' (showing 8.2 GB of 15 GB used). The main area is titled 'My Drive' and contains a notification: 'My Drive trash is changing. Starting October 13, items will be automatically deleted forever after they've been in the trash for 30 days.' Below this is a 'Quick Access' section with three items: 'basis_of_python_1.ipynb' (edited today), 'introduction.ipynb' (edited today), and 'imageprocessing1.ipynb' (uploaded today). At the bottom is a 'Folders' section with three folders: 'Colab Notebooks', 'Google Buzz', and 'ImgData'.

フォルダごとドラッグすれば、
フォルダの中身も全てアップロード可能

Google Driveとの接続

Colaboratory へようこそ
ファイル 編集 表示 挿入 ランタイム ツール ヘルプ 変更を保存できません

ドライブ

Colaboratory とは

- 環境構築が不要
- GPU への無料アクセス

Drive をマウント
をクリック

Colaboratory へようこそ
ファイル 編集 表示 挿入 ランタイム ツール ヘルプ 変更を保存できませんでした

ドライブにコピー

Colaboratory とは

Colaboratory (略称: Colab) は、ブラウザから Python

- 環境構築が不要
- GPU への無料アクセス
- 簡単に共有

Colab は、学生からデータサイエンティスト、AI リサー
ご覧ください。下のリンクからすぐに使ってみることも

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

Google ドライブをマウントするには、このセルを実行してください。

閉じる

セルが現れるので、実行

Google Driveとの接続

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

... Go to this URL in a browser: <https://accounts.google.com/o/oauth2/auth?c>

Enter your authorization code:

リンクをクリック



Google Driveのアカウントを選択



許可



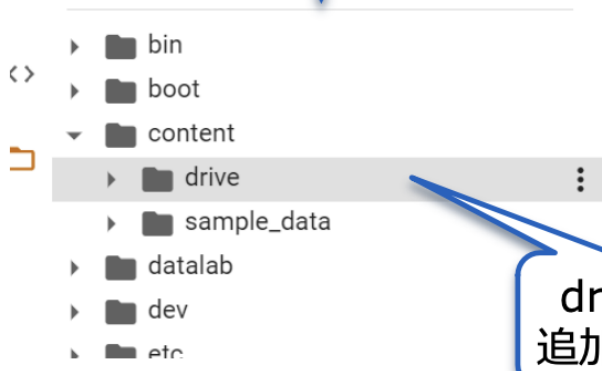
コピー

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

... Go to this URL in a browser: <https://accounts.google.com/o/oauth2/auth?c>

Enter your authorization code:

ペースト

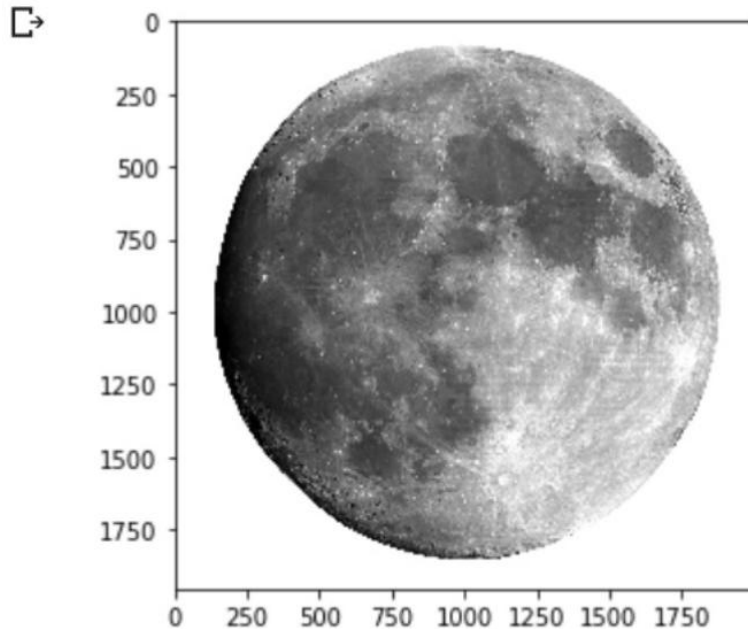


driveが追加された

Drive上のファイルを読めるかを確認

```
▶ from skimage import io
   from skimage import data
   #data_dir = './ImgData/'
   data_dir = './drive/My Drive/ImgData/'
   filename = data_dir + 'moon.png'
   moon = io.imread(filename)
   io.imshow(moon)
   io.show()
```

Google Drive上の
ファイルのパスを記載



画像解析に便利なライブラリ

OpenCV / scikit-image

OpenCV

If interested, try at home ☺

- <http://opencv.org>
- CV = “Computer Vision”
- 様々な画像解析ライブラリ（関数群）が提供されている
- Python用にライブラリが整備されている
- 注) アナコンダにはデフォルトでは入っていないため、インストールが別途必要。

http://docs.opencv.org/3.0-beta/doc/py_tutorials/py_tutorials.html

scikit-image

- Python用画像処理ライブラリ
- Numpy arraysを利用
 - グレースケール画像は2次元配列(array)で表現
- <http://scikit-image.org>

参考: <http://www.scipy-lectures.org/packages/scikit-image/>

Data Retrieval

- Loading pre-prepared images in skimage.data

```
from skimage import data  
coins = data.coins()
```



- Load image from file: skimage.io.imread()

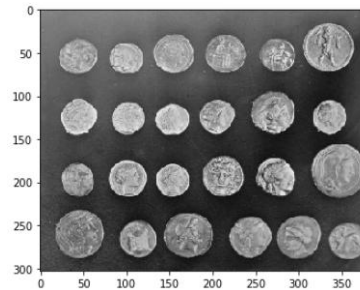
```
from skimage import io  
filename = 'moon.png'  
moon = io.imread(filename)
```



画像の可視化

- 画像の表示 : `skimage.io.imshow()`

```
io.imshow(coins)
```



- 画像サイズ

```
print("Dimensions of the coin image: ", coins.shape )  
print("Dimensions of the moon image: ", moon.shape )
```

```
Dimensions of the coin image: (303, 384)  
Dimensions of the moon image: (1955, 2000, 4)
```

- 輝度値の最大、最小、平均

```
print("Minimum value for image coins: ", coins.min() )  
print("Maximum value for image coins: ", coins.max() )  
print("Average value for image coins: ", coins.mean() )
```

```
Minimum value for image coins: 1  
Maximum value for image coins: 252  
Average value for image coins: 96.8555160204
```

練習1

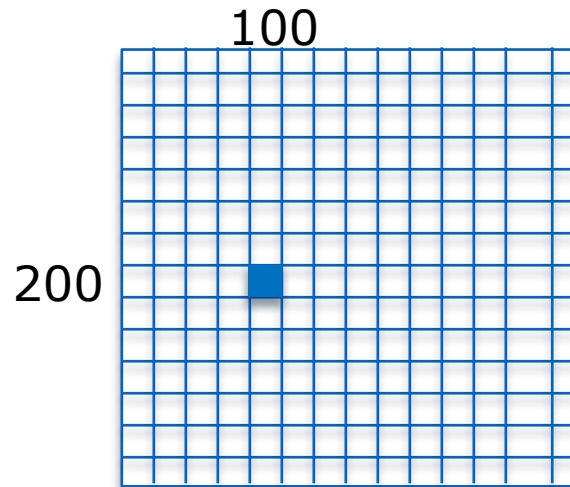
- skimage から io をインポートする
- “coffee.tiff” を読み込んで、可視化する
- 画像のサイズとピクセルの寸法を求める



画像と配列，画素レベルの処理

画素値へのアクセス

- 画素値へのアクセス



```
print("pixel value of image coins at location (200,100): ", coins[200,100] )
print("pixel value of image moon at location (100,300): ", moon[100,300] )
```

```
coins[200,100] = 0
print("New value of image coins at piel (200,100): ", coins[200,100] )
```

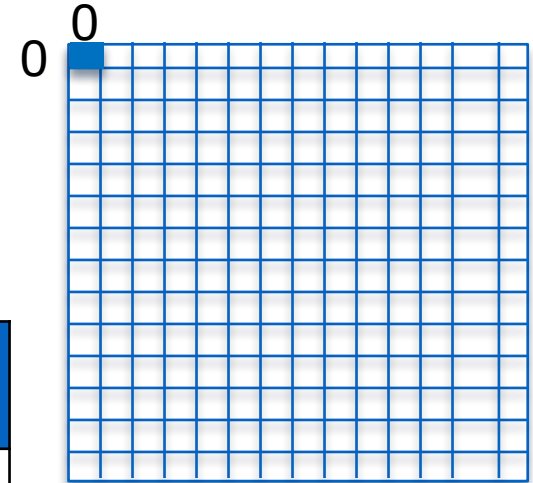
```
pixel value of image coins at location (200,100): 165
pixel value of image moon at location (100,300): [255 255 255  0]
New value of image coins at piel (200,100): 0
```

“moon” は 4 つの値を持つ! : red, green, blue, alpha (透過度)

画素値へのアクセス

- (0,0) : 画像の左上
- ※) pythonでは、0から始まる

Image type	coordinates
2D grayscale	(row, col)
2D multichannel (RGB)	(row, col, ch)
3D grayscale	(pln, row, col)
3D multichannel	(pln, row, col, ch)



row : 列
col : 行

カラー画像

- 1画素につき、(Red, Green, Blue) の3チャンネルの数値
- データ名.shape を使って確認しよう
- それぞれのチャンネルに、0 から 255 までの値が入っている
 - 例：赤 (255, 0, 0)
- Pythonで、それぞれのチャンネルにアクセスするには
[row, col, 0 or 1 or 2]

Red

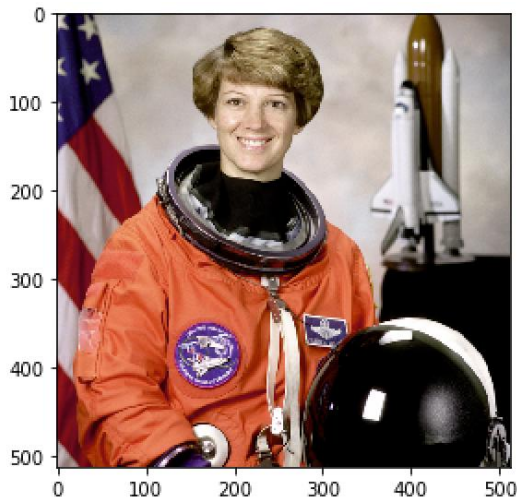
Green

Blue

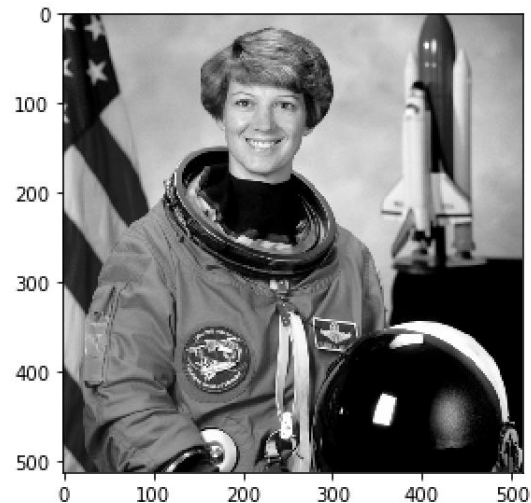
カラー画像⇔グレースケール画像変換

- (red, green, blue) ⇒ gray value
- 画像変換 : rgb2gray

```
from skimage.color import rgb2gray # Load the function for conversion
from skimage import data
from skimage import io
img = data.astronaut() # load the input image
img_gray = rgb2gray(img) # convert to gray image
io.imshow(img)
io.imshow(img_gray)
```



Input color image

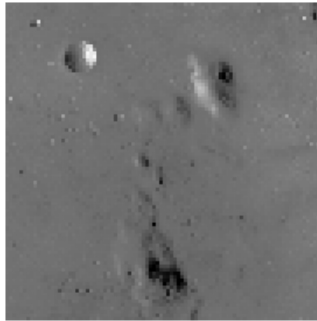


Output gray image

Contrast enhancement

- Pixel value: (Generally) $0 \sim 255$
- In fact, there are many images that have value only in a narrow area of the image
 - In the image below, there are only values of $[50, 100]$
 - Low contrast

Low contrast image



- `skimage.exposure`: A function is provided to enhance the contrast of the image (widen the width of the pixel value used).

Contrast enhancement technique

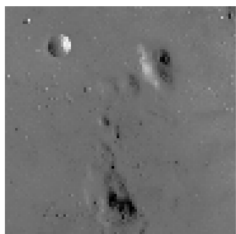
- Gamma correction
 - Lift (brighten) dark pixels
- Contrast correction using luminance histogram.
 - Use the luminance distribution to compensate for luminance values and enhance contrast
- Linear concentration conversion
 - Expand the distribution range of luminance values to $[0, 255]$

Histogram equalization

- “Histogram_Equalization.py” を利用



Low contrast image



元画像

Contrast stretching



線形変換

Histogram equalization



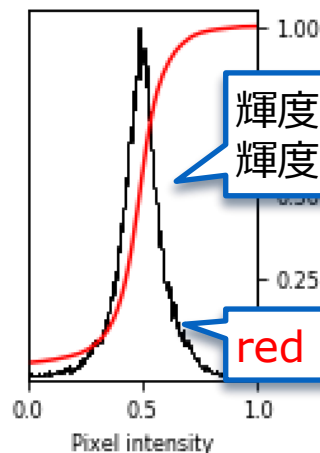
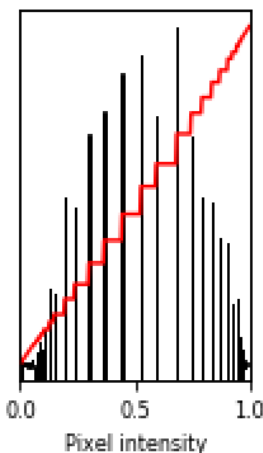
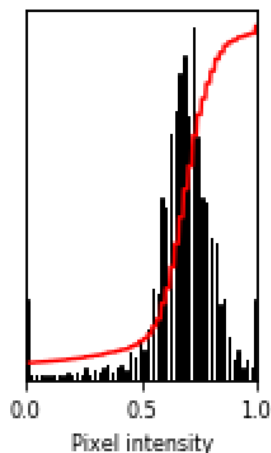
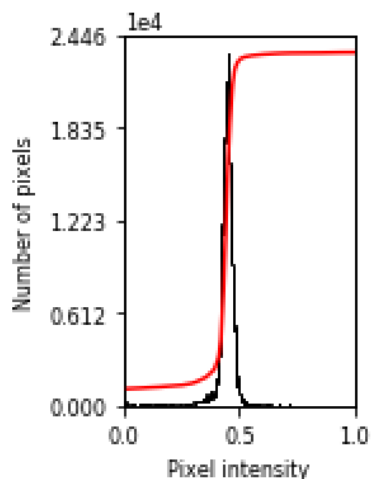
ヒストグラム
均一化

Adaptive equalization



ヒストグラム
適応均一化

Output images

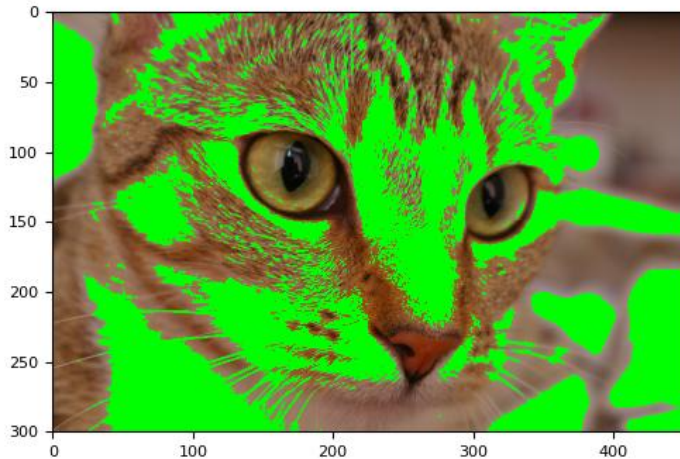


輝度値のヒストグラム
輝度範囲： [0: 1]

red curve : 輝度の積算

練習2

- skimage から data をインポートする
- "chelsea"を読み込んで、red channel > 160 となっている画素を green [0, 255, 0]に上書きして、表示



```
cat = data.chelsea() # load the cat image  
reddish = cat[:, :, 0] > 160
```

- Note on data type

Data type	Range
uint8	0 to 255
uint16	0 to 65535
uint32	0 to 2^{32}
float	-1 to 1 or 0 to 1
int8	-128 to 127
int16	-32768 to 32767
int32	-2^{31} to $2^{31} - 1$

画素レベルの処理

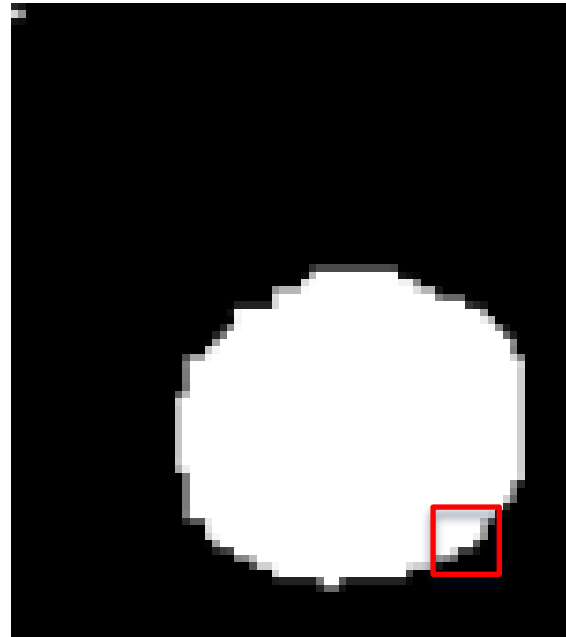
セグメンテーション

- ピクセルごとに背景か前景かをラベル付け

元画像



セグメンテーション画像



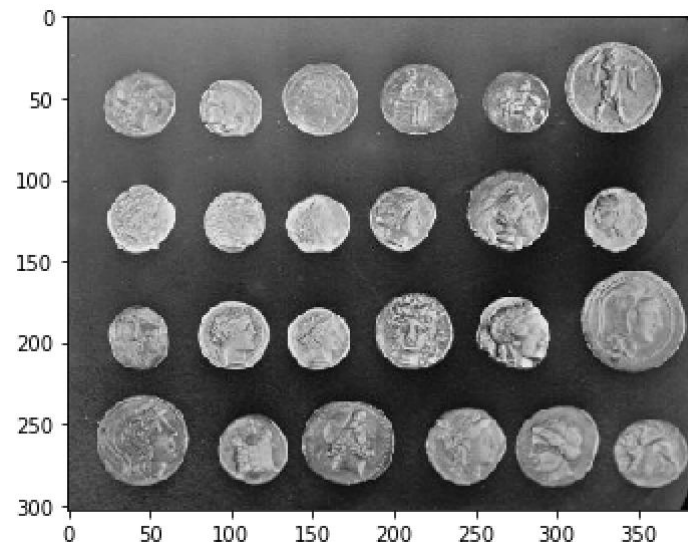
ピクセル
前景 : 1
背景 : 0

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

Load the coin image

- skimage.data

```
import numpy as np
from skimage import data, io
coins = data.coins()
io.imshow(coins)
```

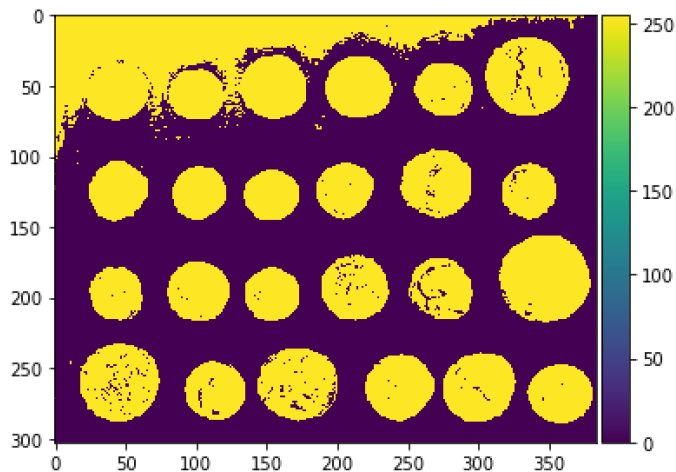


Segmentation by threshold

- Threshold method: If the pixels are higher than the threshold, they are in the foreground, if they are lower, they are in the background
- Example:

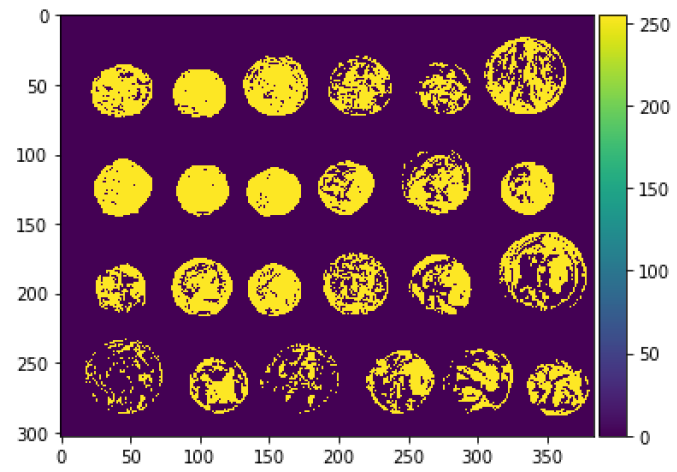
閾値 : 100

```
segmented_img = np.zeros(coins.shape)
mask = coins[:, :] > 100
segmented_img[mask] = 255
io.imshow(segmented_img)
```

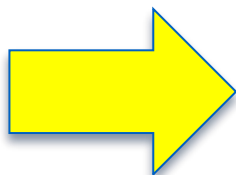
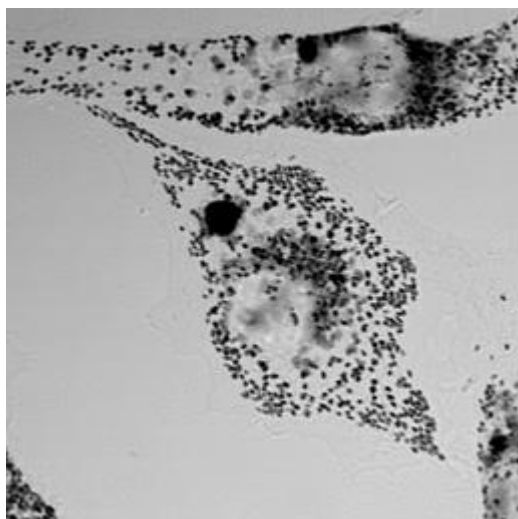


閾値 : 150

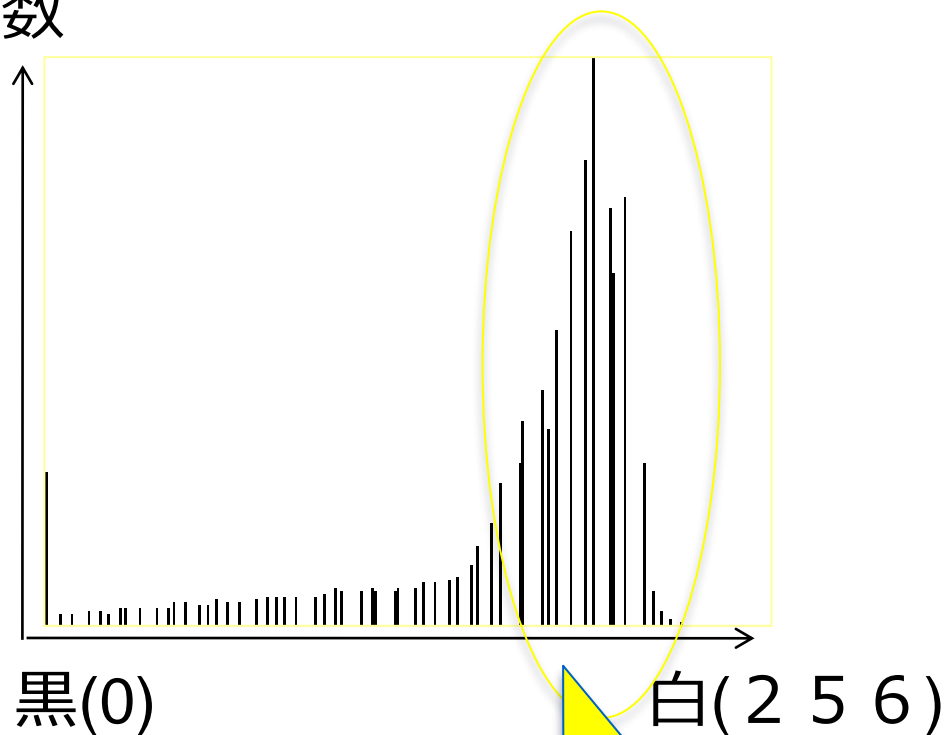
```
segmented_img = np.zeros(coins.shape)
mask = coins[:, :] > 150
segmented_img[mask] = 255
io.imshow(segmented_img)
```



Binary clues: histograms

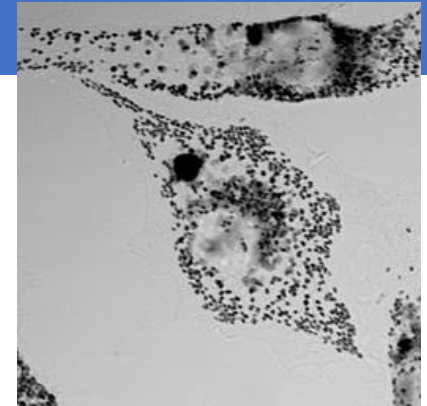


画素数



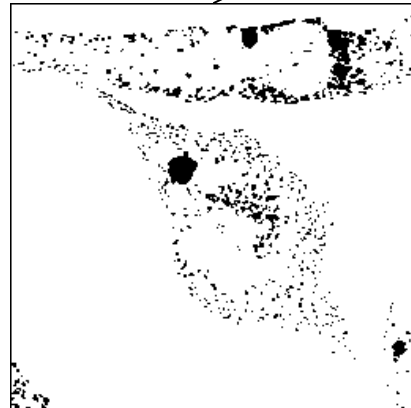
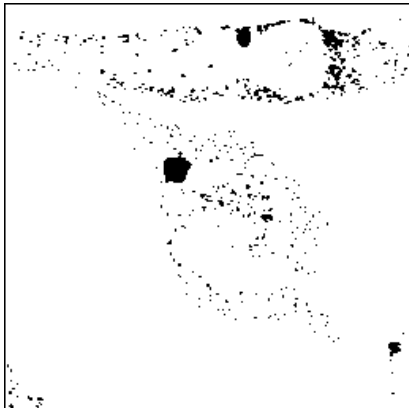
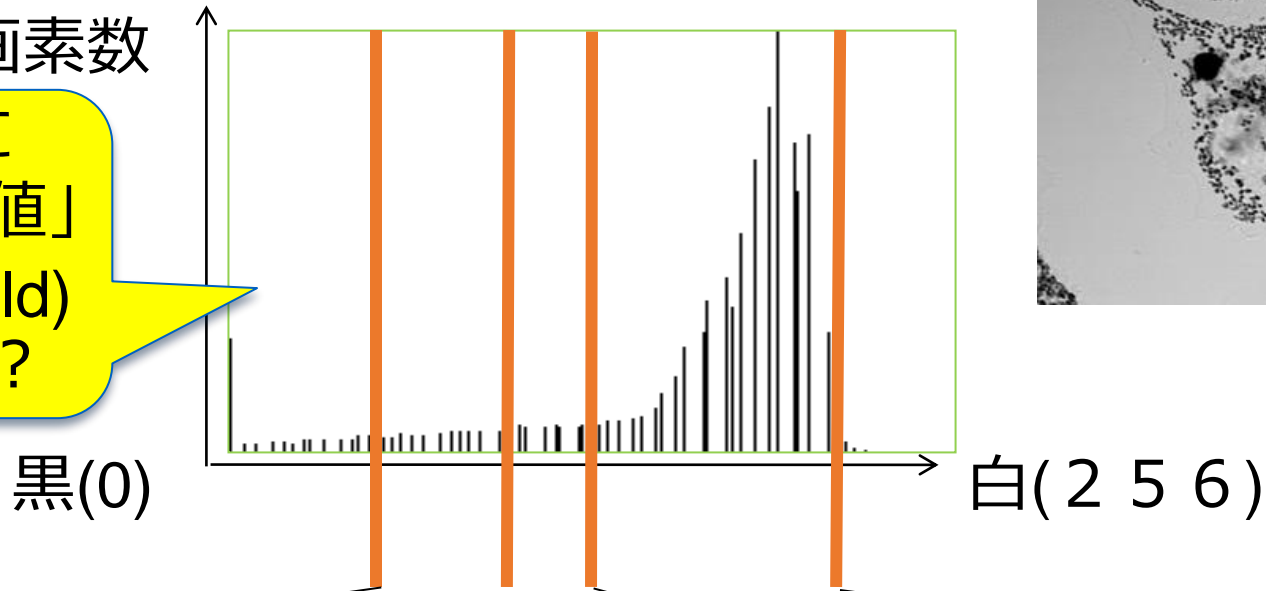
どうもこの辺が
「背景」?

2値化の手がかり：ヒストグラム



画素数

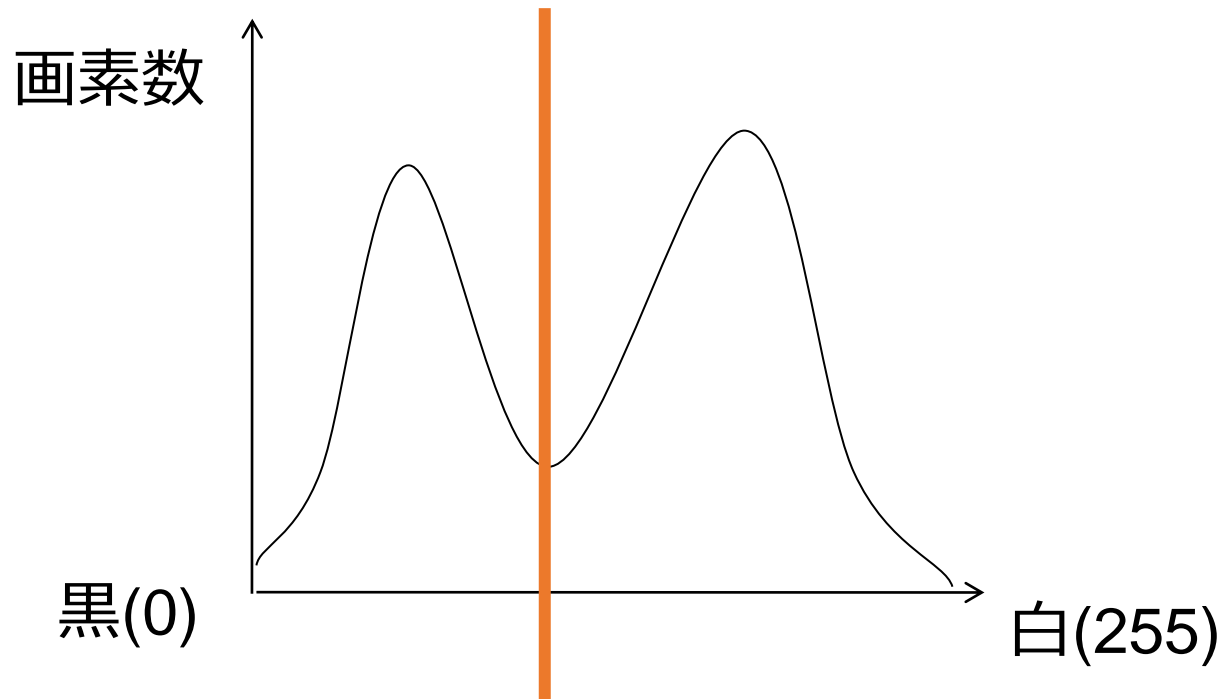
どこに
「しきい値」
(threshold)
を設定？





モード法

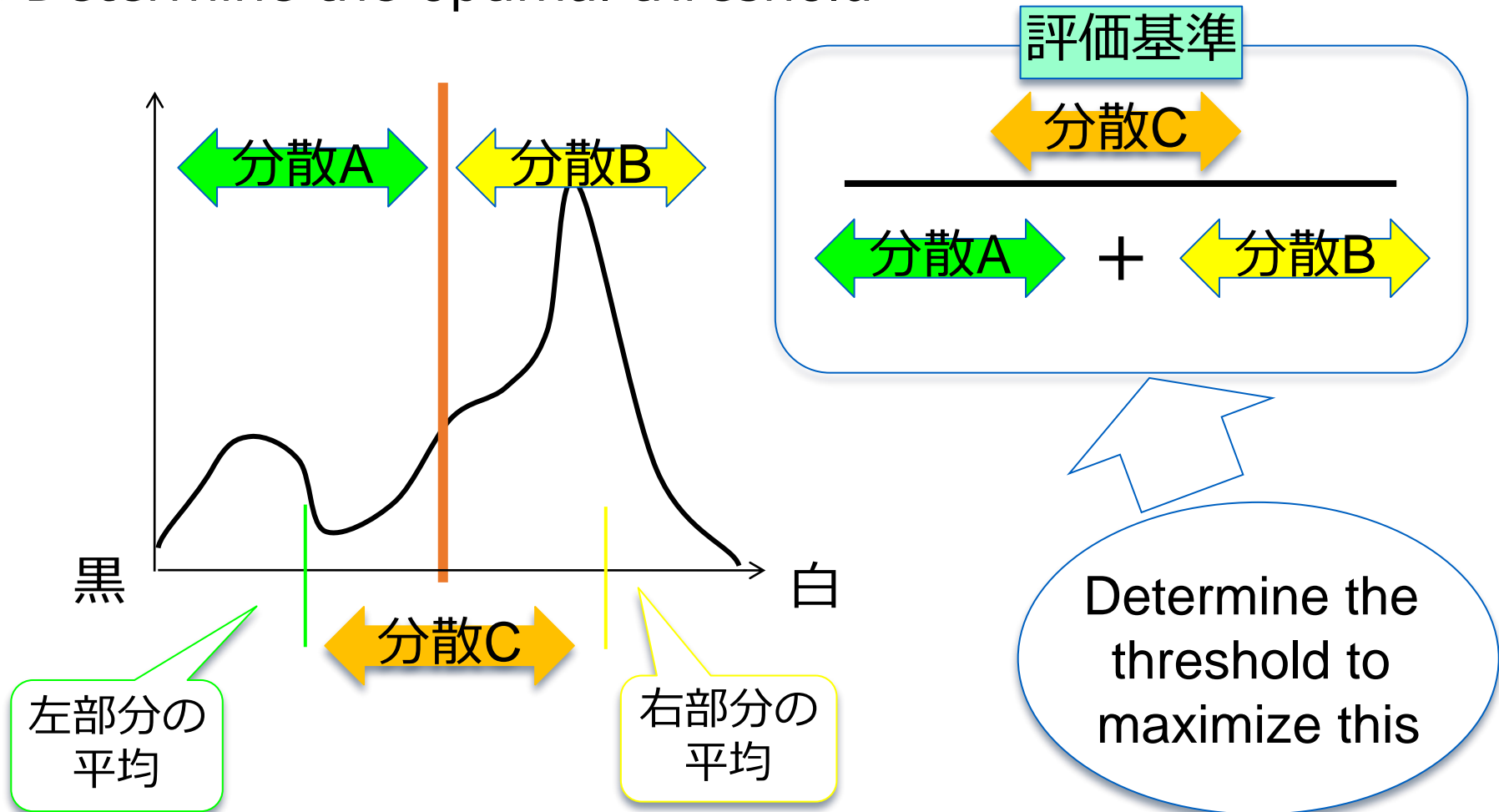
- ヒストグラムの谷 = しきい値





Binary of Otsu

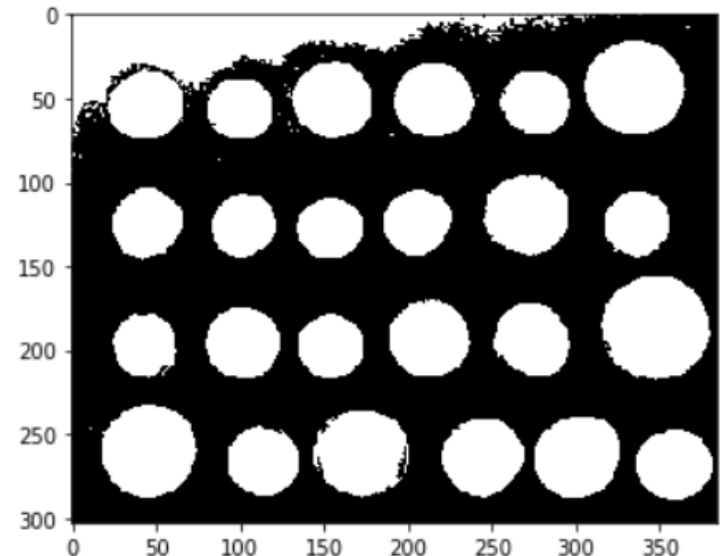
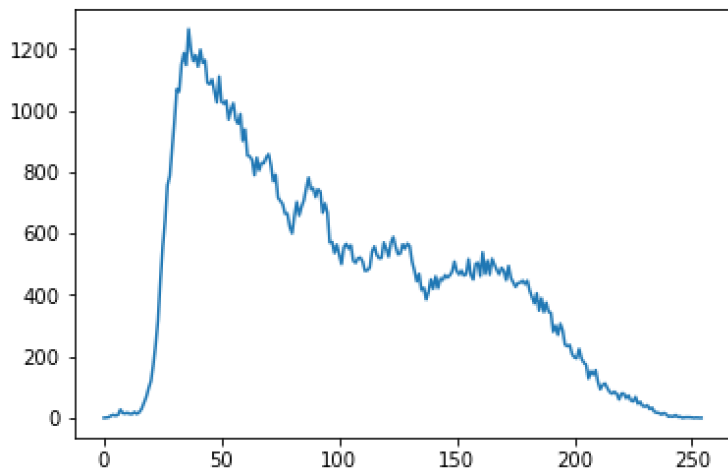
- Determine the optimal threshold



Threshold Limits

- The distribution of brightness values is not always neatly divided between foreground and background
 - Even in the background, there are pixels with a higher brightness value than the foreground (or vice versa)
- A threshold that is too high will miss the foreground, and a threshold that is too low will extract too much foreground

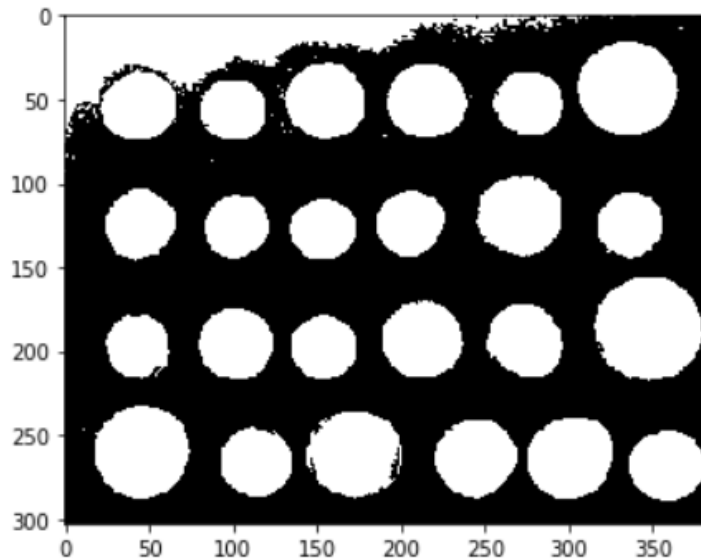
```
import matplotlib.pyplot as plt
histo = np.histogram(coins, bins=np.arange(0, 256))
plt.plot(histo[1][0:255], histo[0])
```



Filling the holes

- By coloring the inside of the boundary, the area of the coin is extracted
- `ndi.binary_fill_holes()`を利用

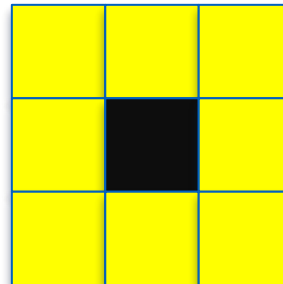
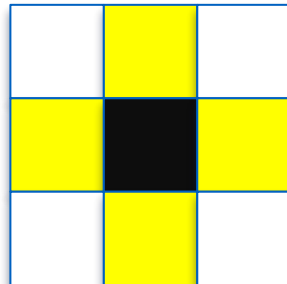
```
from scipy import ndimage as ndi
fill_coins = ndi.binary_fill_holes(edges)
io.imshow(fill_coins)
```



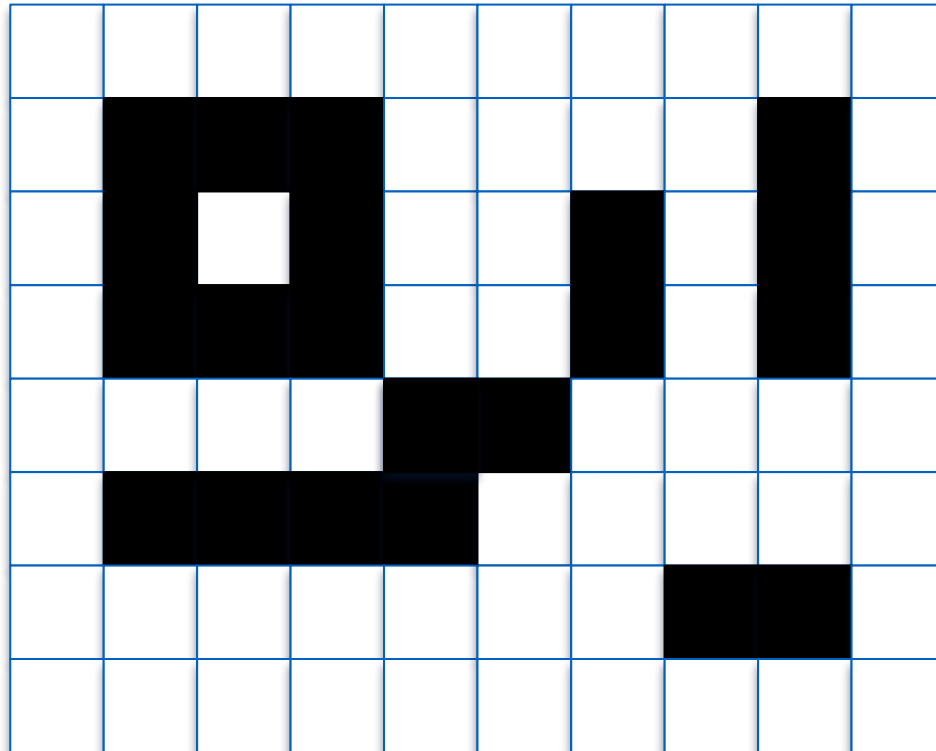


Analysis of linked components

- Connected component = a “cluster” of black pixels (or white pixels)
- 4-connectivity and 8-connectivity



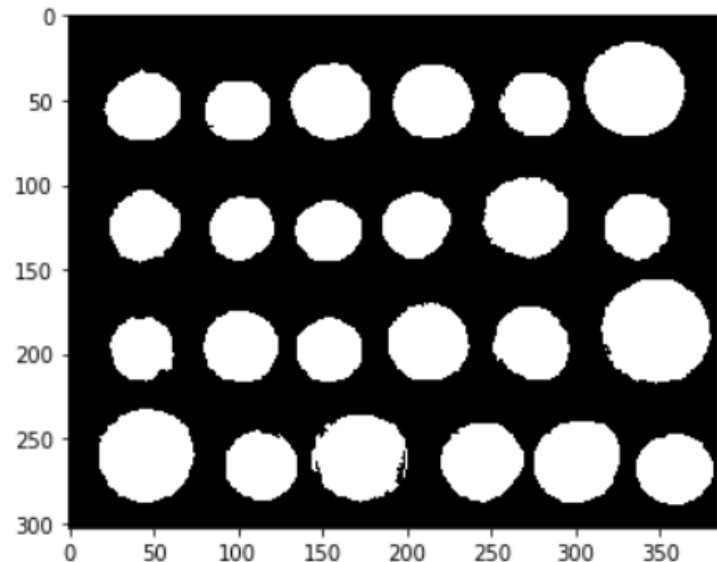
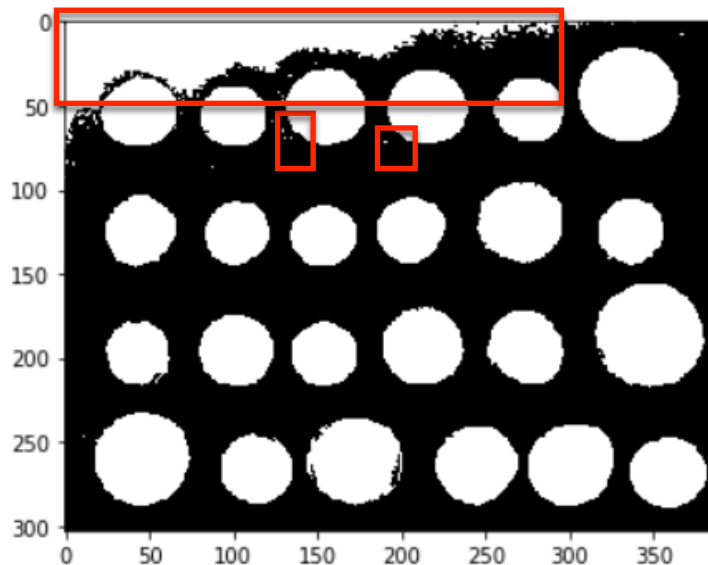
4 連結だと何個の連結成分？ 8連結では？



Removing small objects

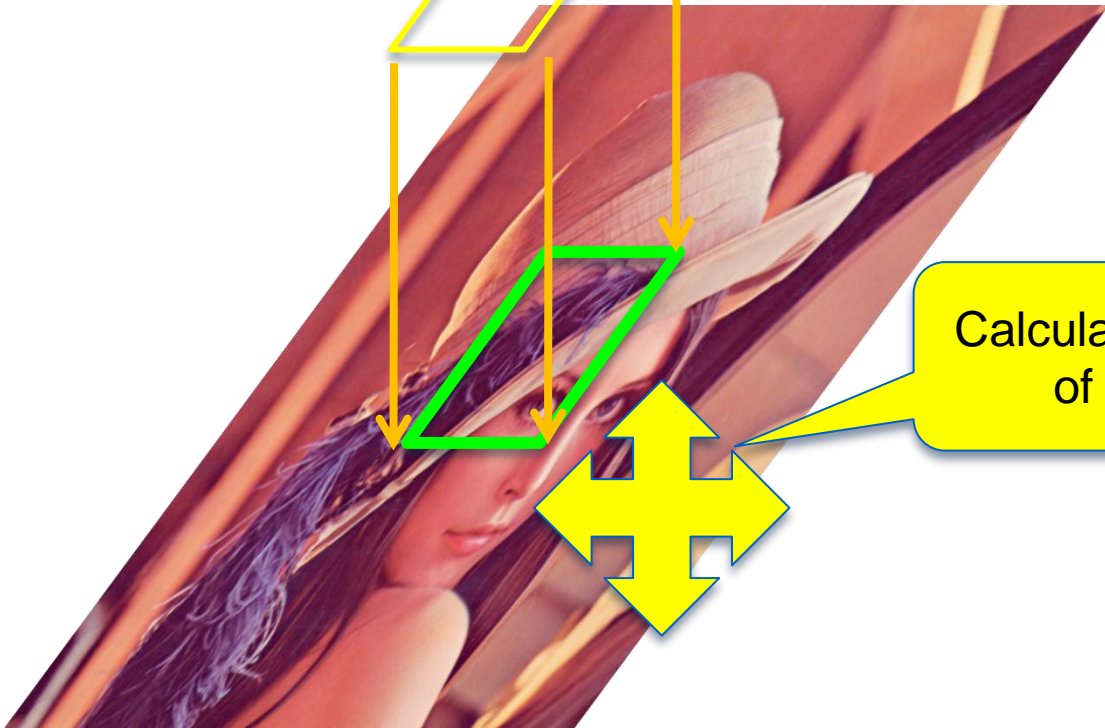
- Labeled objects that are too small or too large are likely extracted incorrectly, so check the size and delete
- `ndi.label` function を利用

```
label_objects, nb_labels = ndi.label(fill_coins)  
sizes = np.bincount(label_objects.ravel())  
mask_sizes = sizes > 20  
mask_sizes[0] = 0  
coins_cleaned = mask_sizes[label_objects]  
io.imshow(coins_cleaned)
```



filtering

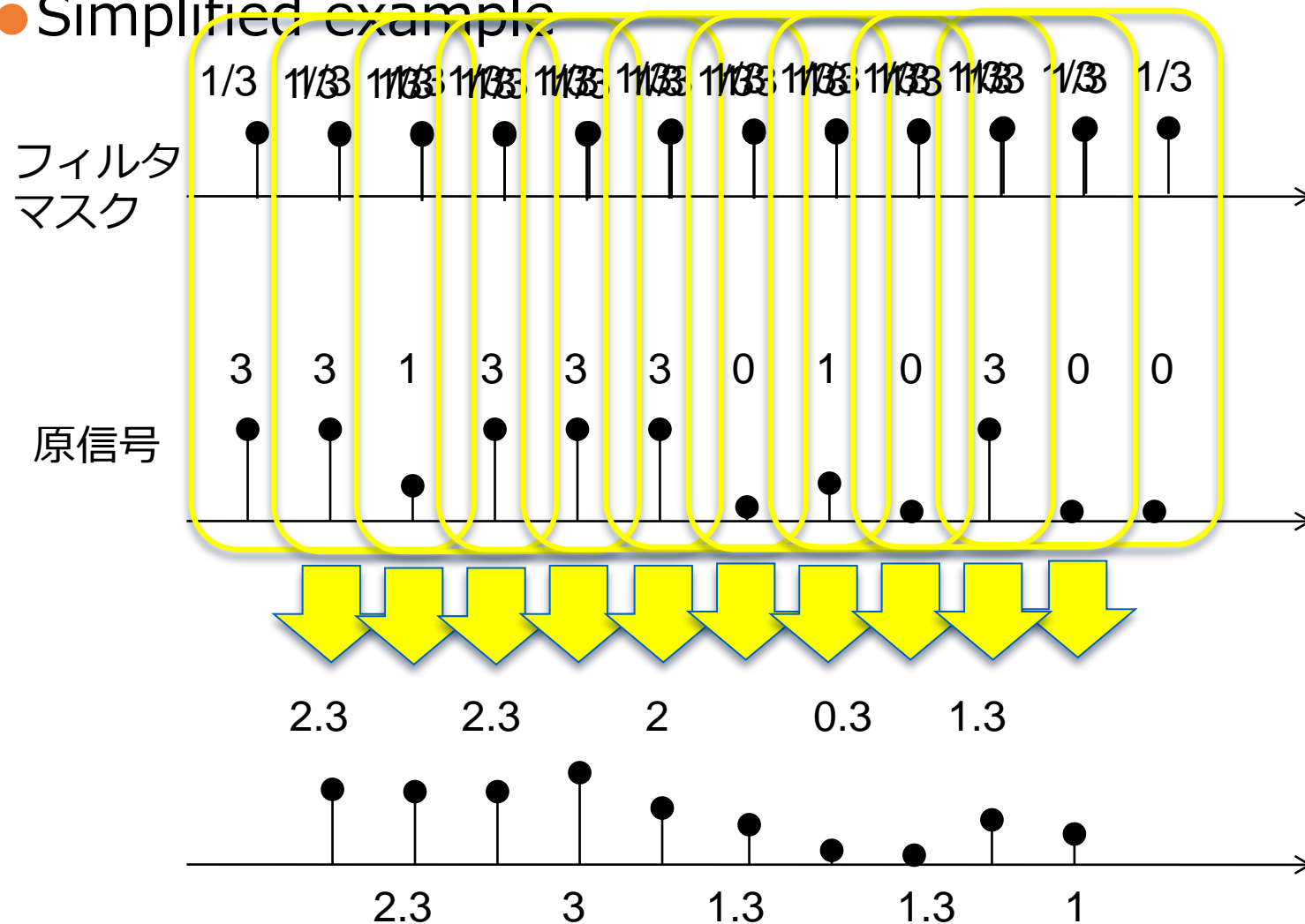
Some calculation results



Calculating all pixels of an image

The most basic filter: smoothing

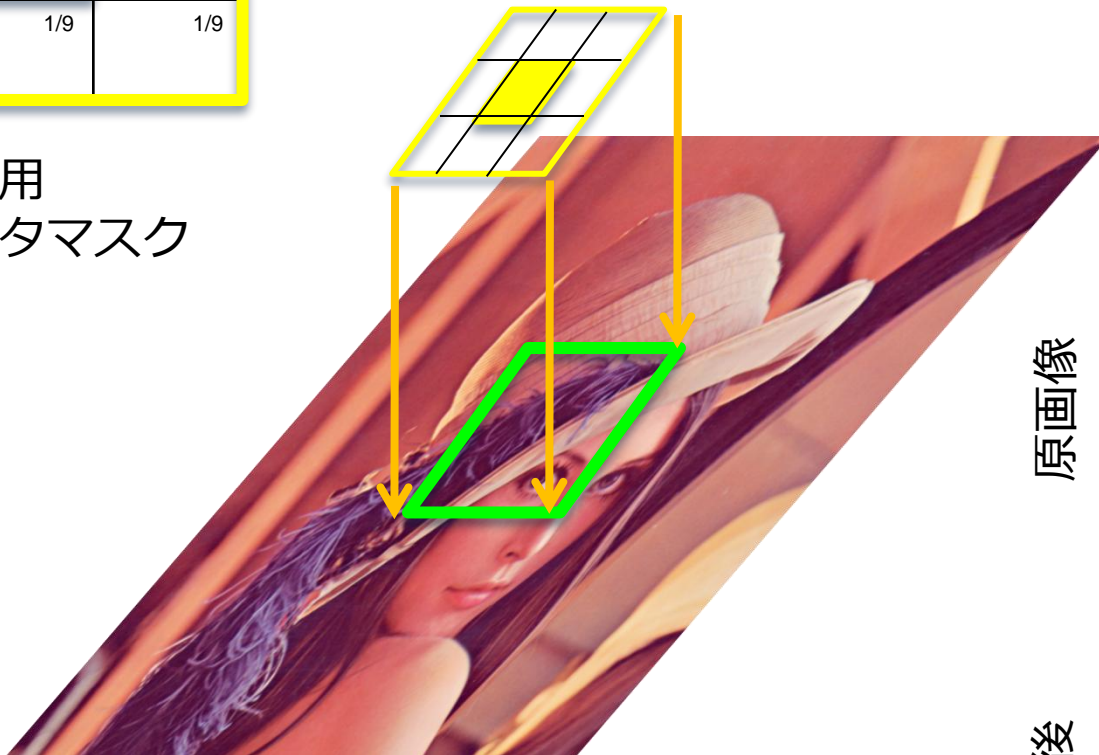
● Simplified example



最も基本的なフィルタ：平滑化

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

平滑化用
フィルタマスク
(3x3)



マスク

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9



内積

原画像

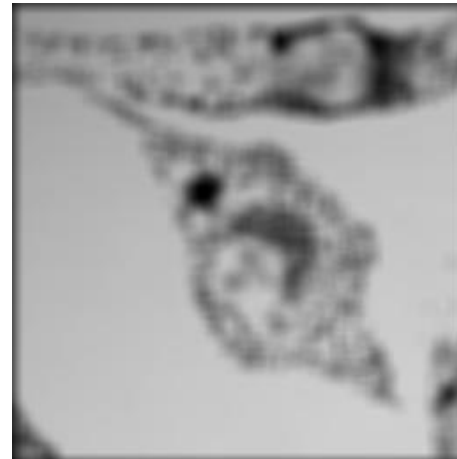
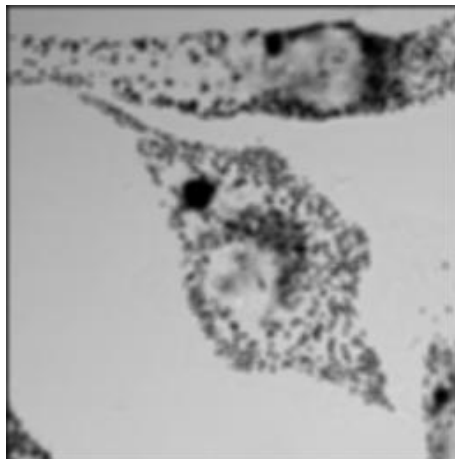
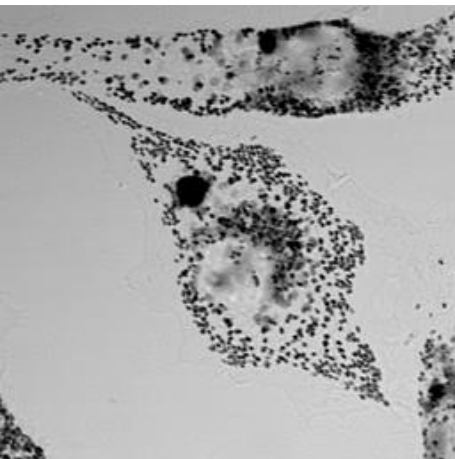
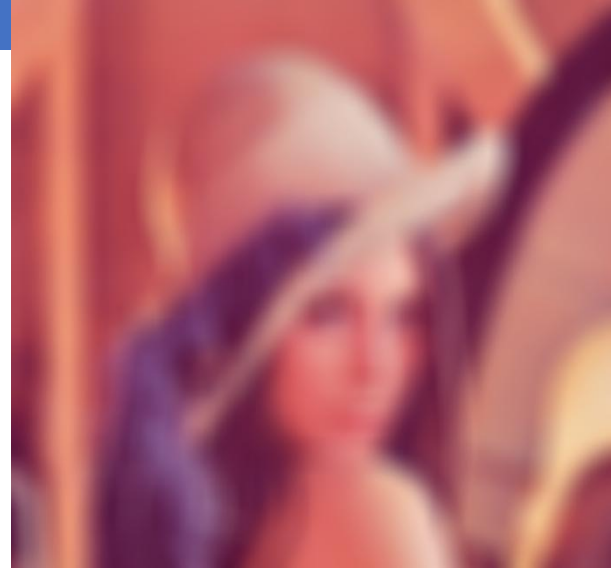
10	10	10
10	20	10
10	10	10



平滑化後

10	10	10
10	11	10
10	10	10

最も基本的なフィルタ：平滑化



平滑化用のフィルタマスク

0	0	0
0	1	0
0	0	0

無処理のオペレータ
(そのまま出力)

0.11	0.11	0.11
0.11	0.12	0.11
0.11	0.11	0.11

平均を求める
オペレータ
(平滑化)

0.05	0.05	0.05
0.05	0.60	0.05
0.05	0.05	0.05

中心重点の平均を
求めるオペレータ
(ぼけ過ぎない平滑化)

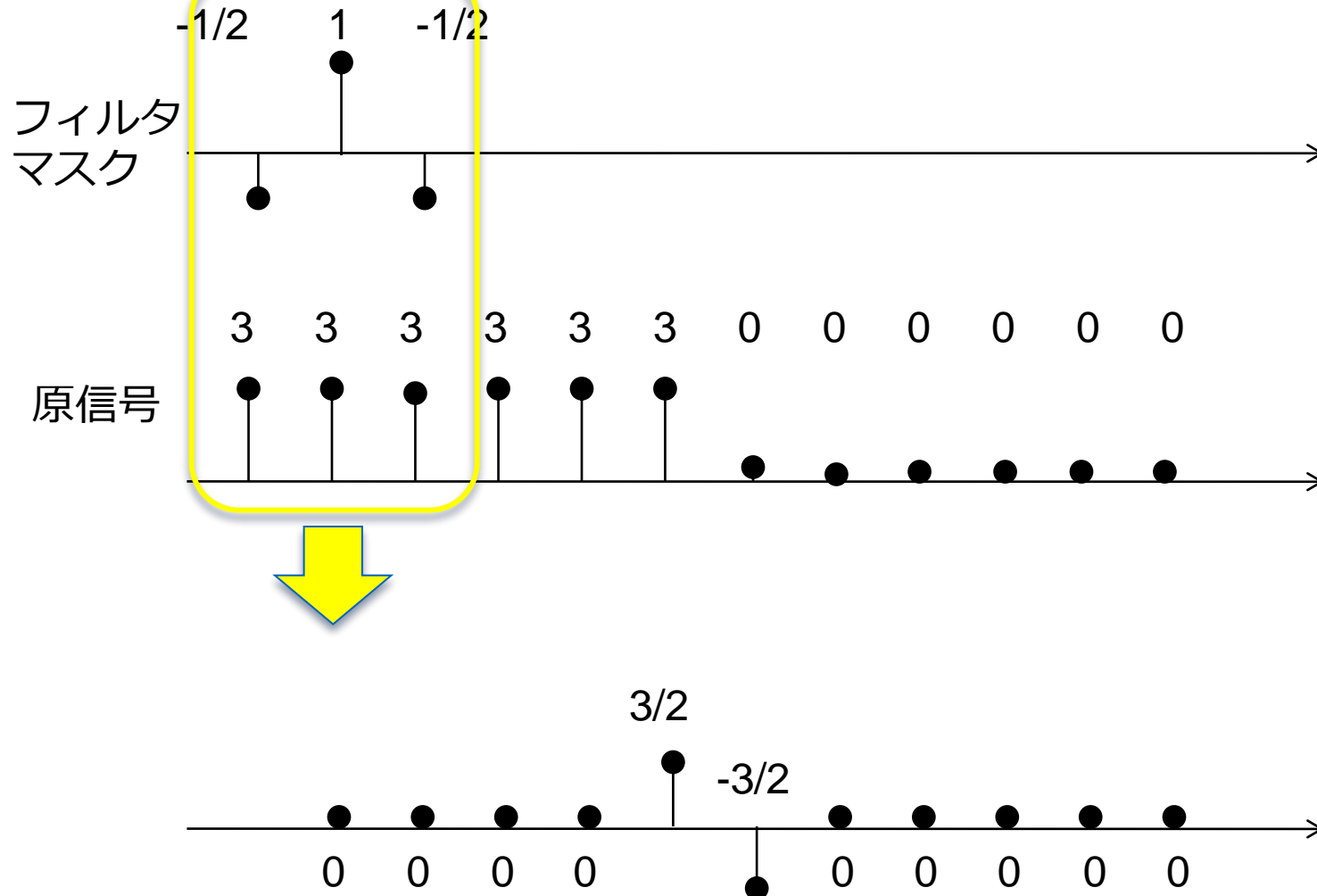


What is filtering

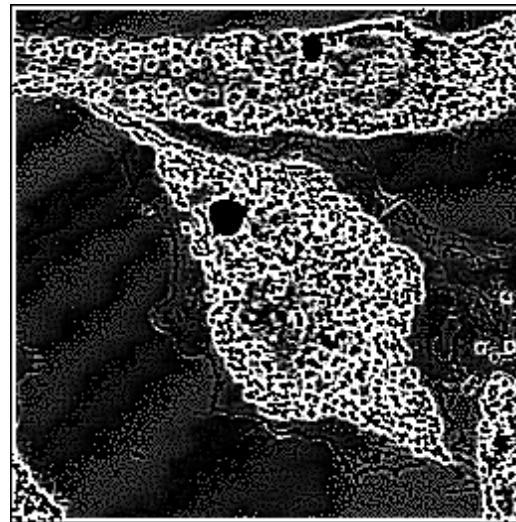
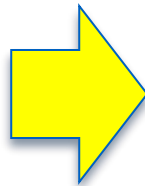
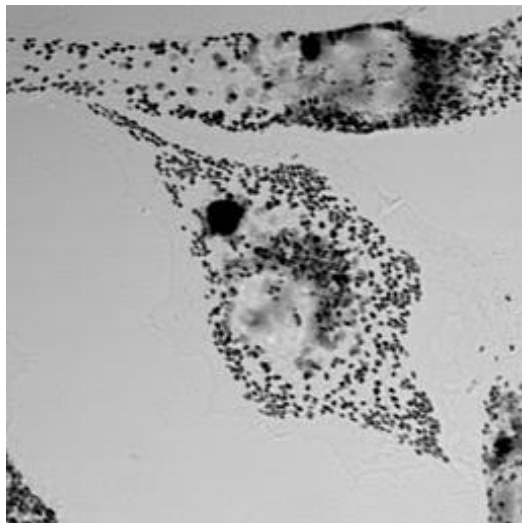
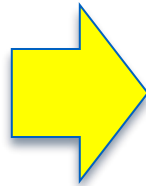
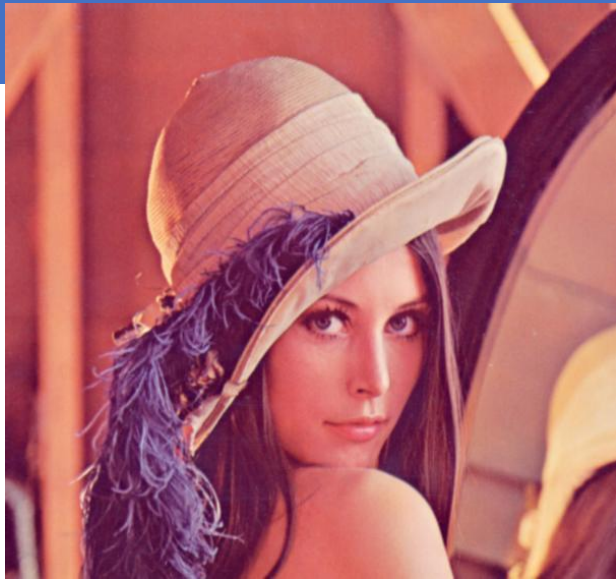
- Some calculation in the local area centered on each pixel on the image
- Aimed at improving image quality and extracting features

基本的なフィルタ：エッジ検出

● 単純化した例



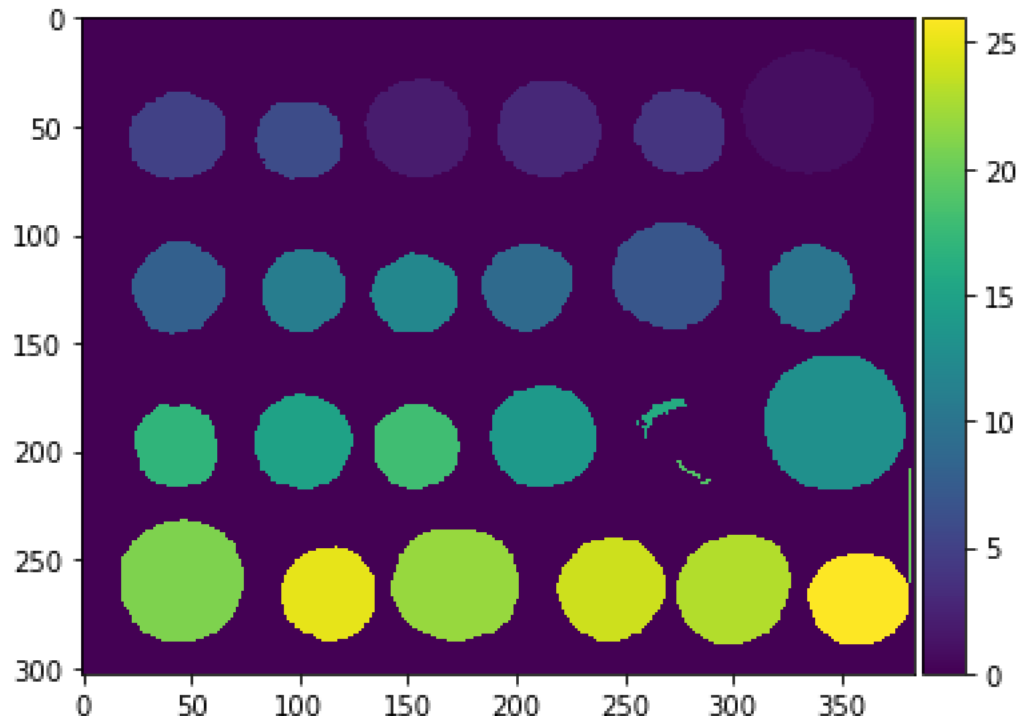
基本的なフィルタ：エッジ検出



Painting images with labels

- 一まとまりの領域ごとにID(label)をつける
- `ndi.label`

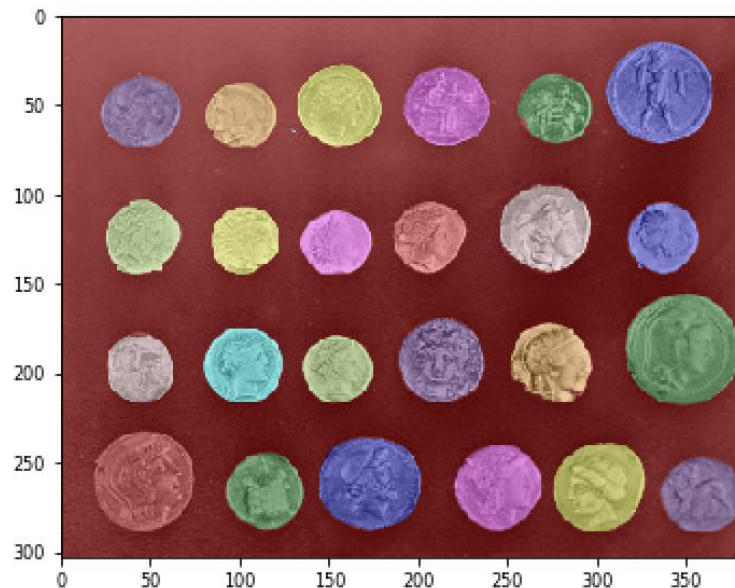
```
labeled_coins, _ = ndi.label(coins_cleaned) # Label all coins one by one  
io.imshow(labeled_coins)
```



Visualize results (1)

- `label2rgb()` でラベルごとに独立した色を与えて可視化.

```
from skimage.color import label2rgb
image_label_overlay = label2rgb(labeled_coins, image=coins)
io.imshow(image_label_overlay)
```

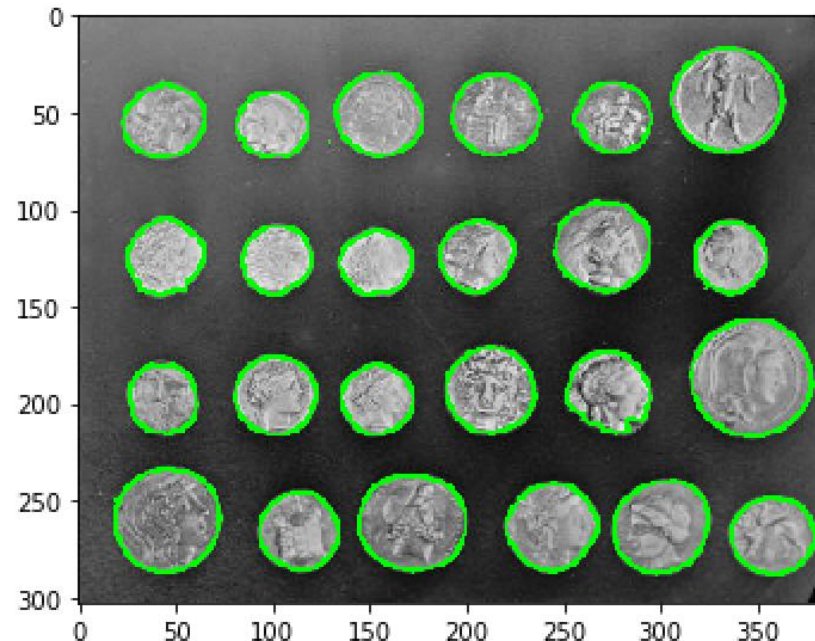


Visualize results (2)

- オブジェクトの境界を可視化
- `skimage.morphology.erosion`

```
contours = segmentation ^ ndi.binary_erosion(segmentation, np.ones((5,5)))
```

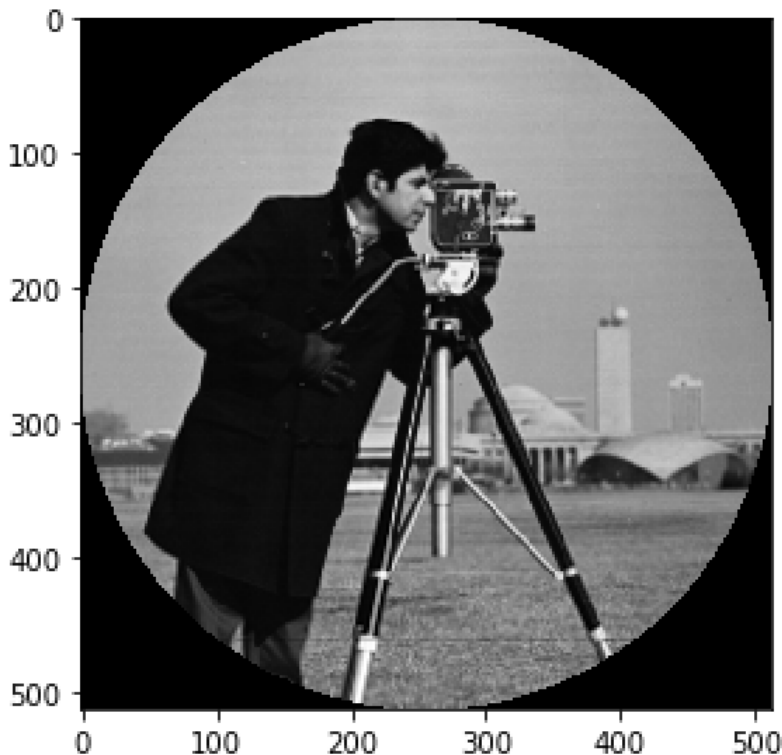
```
from skimage.color import grey2rgb  
coins_rgb = grey2rgb(coins)  
coins_rgb[contours] = [0,255,0]  
io.imshow(coins_rgb)
```



演習

演習 1

- “camera”画像を読み込み、下図のように円状（中心：画像の中心、半径R：画像の縦横のサイズの半分）の外の画素値を全て0とするプログラム



円の方程式

$$\{(i, j) | (i - c_i)^2 + (j - c_j)^2 < R^2\},$$

※)

(i, j) : ピクセルの座標

(c_i, c_j) : 円の中心座標

R : 円の半径

Answer

● Indexing with sets of indices

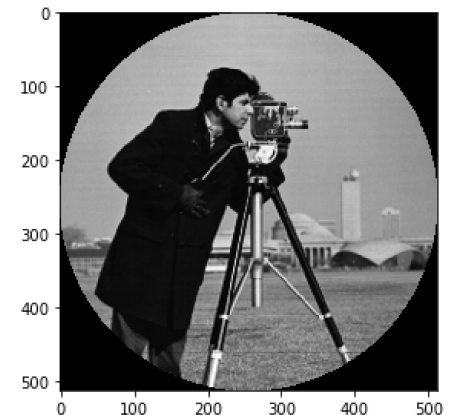
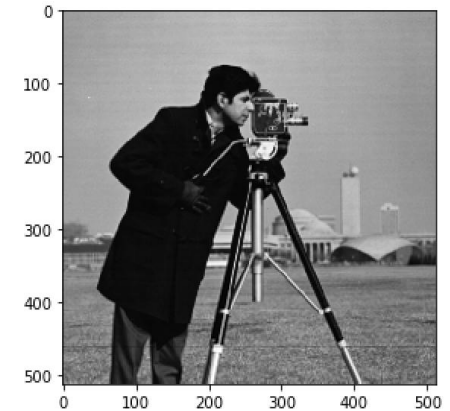
```
camera = data.camera() # load the image camera
io.imshow(camera)      # Display the image
```

```
import numpy as np
nrows, ncols = camera.shape # Get the dimension of the image
                                # (number of rows and number of columns)
row, col = np.ogrid[:nrows, :ncols] # Create a vectors of row indices
                                # and column indices

print( "Row indices: ", row )
print( "Column indices: ", col )
```

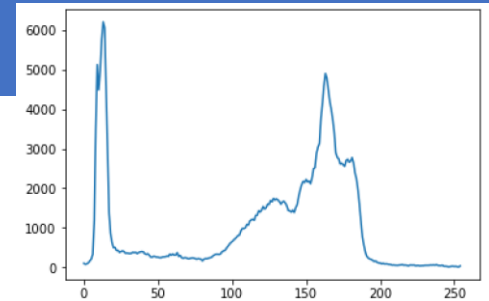
```
Row indices: [[ 0]
 [ 1]
 [ 2]
 ...,
 [509]
 [510]
 [511]]
Column indices: [[ 0  1  2 ..., 509 510 511]]
```

```
cnt_row, cnt_col = nrows / 2, ncols / 2 # Center of the disk
outer_disk_mask = ((row - cnt_row)**2 + (col - cnt_col)**2 > (nrows / 2)**2)
# outer_disk_mask has True values for pixel outside the disk
# and False values for pixels inside the disk
camera[outer_disk_mask] = 0 # Put all pixel with True values
                            # in outer_disk_mask to 0
io.imshow(camera)          # Display the image
```



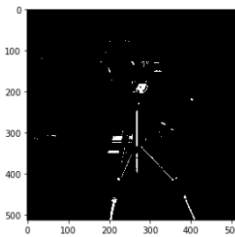
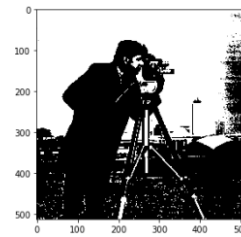
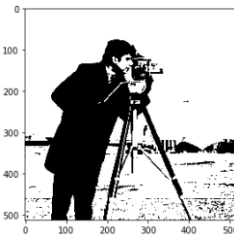
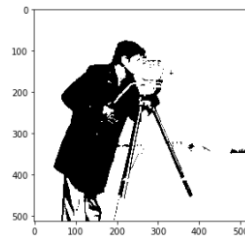
演習 2 (thresholding)

- "camera"画像の呼び出し
- 輝度値のヒストグラム可視化
- 複数の閾値を用いて、セグメンテーションの結果を比べよう！



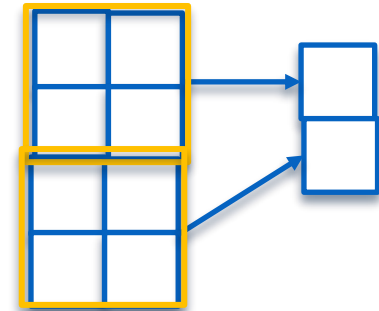
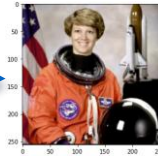
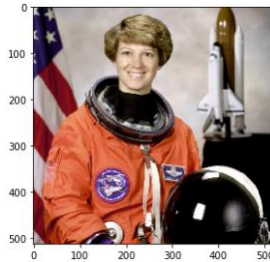
- ヒント:

- `skimage.data.camera()`
- `numpy.histogram()`



演習 3 (downsampling)

- “astronaut” 画像を読み込み, 画像を2分の1のサイズにダウンサンプリングしよ
 - 周辺の画素の平均値を求めて、新画像の1画素の値にする

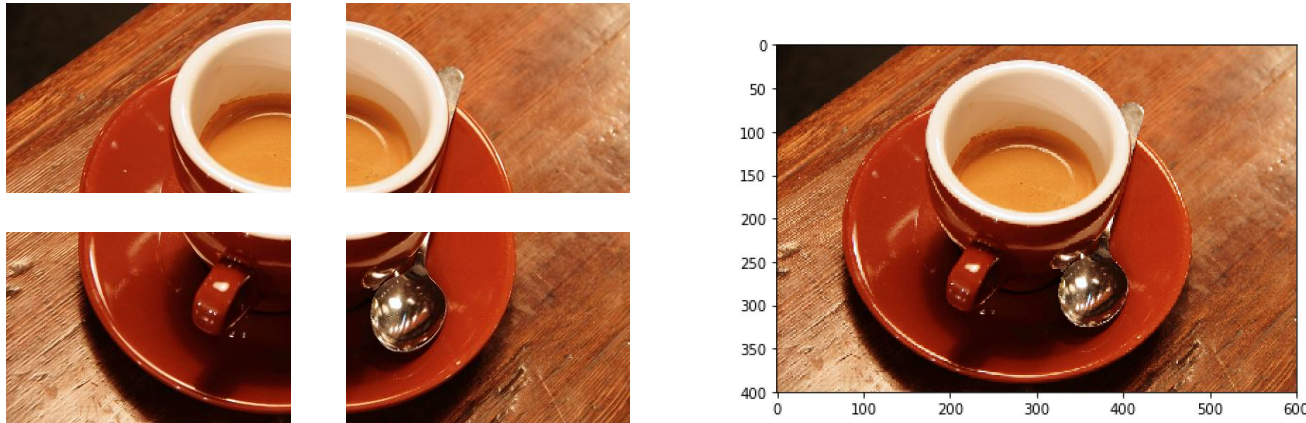


- ヒント

- float型に変換してデータを取得
 - `image = data.astronaut().astype(float)`
- 行と列の数の取得
 - `nrows, ncols, _ = image.shape` (color image は 3次元)
- **unsigned int8** 型に戻す
 - `io.imshow(result.astype(np.uint8))`

演習 4 (image stitching)

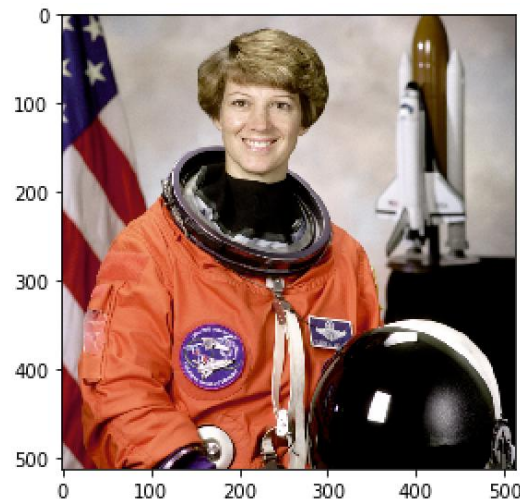
- 4つの画像(im1.tiff, im2.tiff, im3.tiff, im4.tiff)を読み込んで、つなげることで一つの画像を作って、保存しよう！



- ヒント:
 - 画像の読み込み : `io.imread`
 - 画像の大きさ : `shape` attribute
 - 画像の初期化 : `numpy.zeros` (specify `np.uint8` for the type to show nice results)
 - 画像の保存 `io.imsave`

演習 5

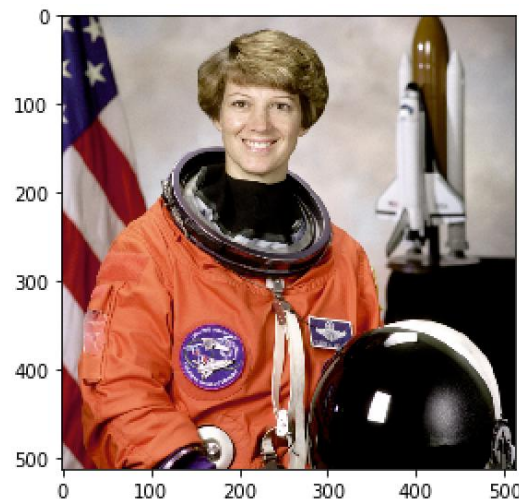
- “astronaut” 画像を読み込み、RGB成分それぞれの画像を表示してみよう



- ヒント:
 - `image = data.astronaut()`
 - `image.simage.shape: (512L, 512L, 3L)`

演習 6

- “astronaut” 画像を読み込み、円状（中心：画像の中心、半径 R ：画像の縦横のサイズの半分）の外の画素値を全て 0 とするプログラム



- ヒント:
 - 26ページのカラーバージョン

演習 7

- “LENNA.bmp” を読み込み、sobel filter でエッジ抽出してみよう



- ヒント:
 - 60ページ

演習 8

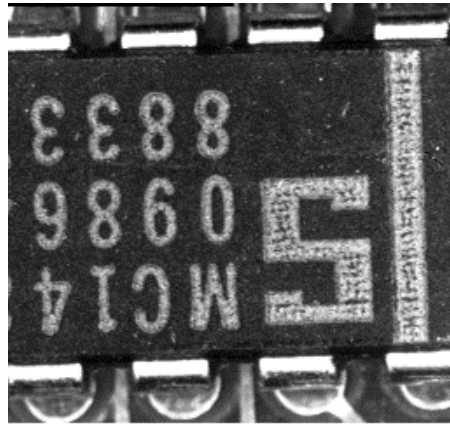
- “coins”画像を読み込み、色んなコントラスト強調を試してみよう



- ヒント:
 - from skimage import data, io
 - coins = data.coins()
 - 36ページ

演習 9

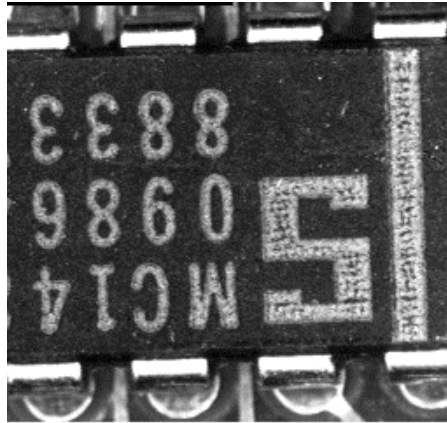
- “Text.bmp”を読み込み、テキスト領域を閾値を用いてセグメンテーションせよ
- 領域ごとにラベリングを行って、領域ごとに色をつけよ



- ヒント:
 - 平滑化してから、輝度値で閾値処理
 - 他の領域を含んでもO.K.

演習 10

- “Text.bmp”を読み込み、テキスト領域をwatershedを用いてセグメンテーション



- ヒント:
 - Sobel filter を用いてelevation_mapを作成
 - マーカー領域は、閾値でラフ（明らかに輝度が高い領域を抽出）に抽出
 - `from skimage.morphology import watershed`
 - 他の領域を含んでもO.K.