



# 画像音声認識-3,4

## - フィルタ処理

4/17,20/2023

上條浩一



# 小テストルール

- 資料やwebで調べても構いません
- ただし、人に聞いたり、人の答えをカンニングするのはNGです
  - あくまで、自分の力で解いて下さい
- 制限時間を過ぎて提出したものはNGです
  - 直ぐ答え合わせをするので
- Class-B/Aの人はClass-A/Bの人に問題、答えを絶対に教えないでください
- Class-A/Bの人はClass-B/Aの人に問題、答えを絶対に聞かないでください
  - 自分にメリットが無いですし、教えたことが発覚した場合、教えた人、教えられた人の両方の全ての小テストを0点とします
  - AとBで異なるテストを行うので、明らかに他のclassの人のコピーと思われる場合、0点とします
- 小テスト、（授業内）演習に関しては、採点結果の報告やコメントは原則しません
- ただし、宿題に関しては採点結果の報告やコメントは原則します

# 小テストの答

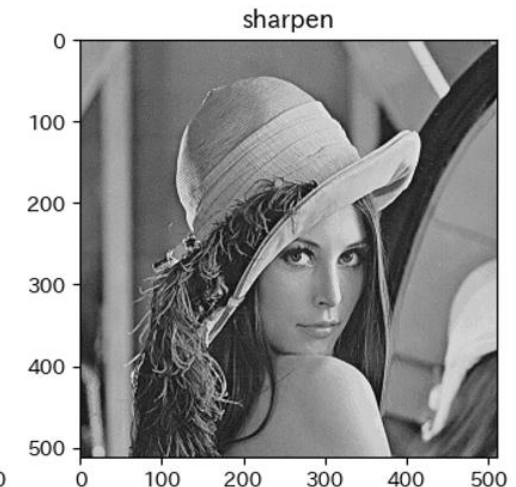
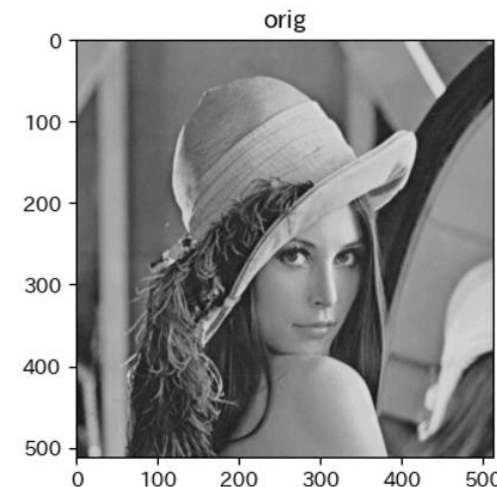
- 問1 (3点) 電磁波や光の波長で、左から短い順に並んでいるものを選び
- 答 紫外線-可視光線-赤外線
  
- 問2 (2点) 減法混色でシアンとマゼンタを混ぜた場合の色として正しいものを選び
- 答 Blue
  
- 問3 (2点)以下の画素数の中で、同じ画像で一番きれいに見えるものを1つ選び。ただし、量子化のbit数や表示するサイズ全て同じとする
- 答 256x256
  
- 問4 (3点)問4. カメラのレンズの公式において、レンズから被写体の実物までの距離が300mm、レンズから写像までの距離が600mmの場合、焦点距離は何mmか
- 答  $1/a + 1/b = 1/f \rightarrow f = 1/(1/a + 1/b) = 1/(1/600 + 1/300) = 200$

# 前回の復習 -画像処理とは、画像入出力、色変換

- 画像とは
  - 写真、絵、図面等を電子化したもの
- 画像の入出力
  - 入力：カメラ、センサ、スキャナ
  - 出力：テレビ、プリンタ
- 解像度と量子化
  - 解像度：画像の細かさ
  - 量子化：色、輝度の階調
- 光とは？
  - 電磁波の1種
  - 量子化：色、輝度の階調
- 画像処理と画像認識
  - 画像処理：画像を扱う信号処理全般
  - 画像認識：画像に写る内容を理解すること
- 加法混色と減法混色
- HSV色空間

# 本日のゴール

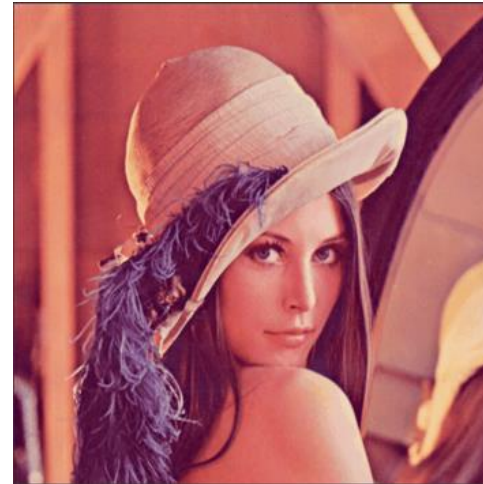
- フィルタ処理
    - 画像の見やすさの向上
    - 画像の特徴の強調
- ↓
- 平滑化フィルタ
    - 画像を滑らかにする
  - エッジ検出
    - 画像のエッジを検出する
  - 鮮鋭化フィルタ
    - 画像を鋭くする



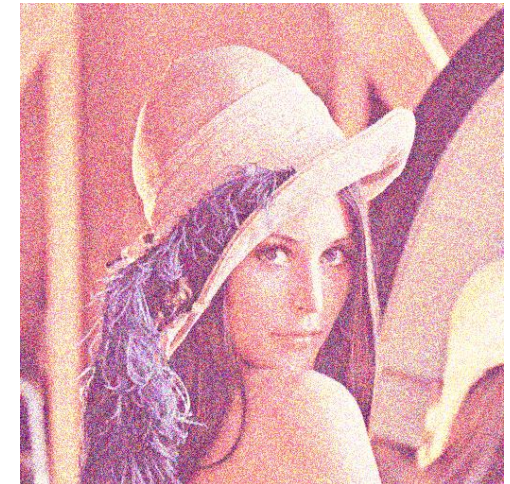


- 画像をカメラなどで撮影しても、ノイズが乗ってしまうことがある

- ガウスノイズ：正規分布に等しい確率の強さのノイズ。照明が不十分だったり、カメラのセンサや回路で付加されることがある



ガウス  
ノイズ



- インパルスノイズ：カメラの素子の異常や、画像処理の失敗（画素が最大値、最小値を振り切ってしまう）等で起きることがある



インパルス  
ノイズ



- ノイズを軽減したい！

- ノイズが乗ってなくても、画像を滑らかな感じにしたい！

➡ **平滑化**

- メディアン(median)処理
  - 極端な値の除去
- ブラー(blur)処理
  - 平均化
- ガウシアン(Gaussian)処理
  - 近くの画素に重みを付けた平均化

# メディアン(median)処理

- 画像の平滑化を行う（主に、極端な値を除くノイズ除去）
- 処理対象画素（ピクセル）の実行結果は、指定したカーネルサイズ  
の領域の平均値になる

画像中の3x3  
の画素値

2	3	2
1	<b>6</b>	5
4	1	1



2	3	2
1	<b>2</b>	5
4	1	1



カーネルサイズ：3x3の例


3x3の値を大きい順に並べた真ん中  
(6,5,4,3,**2**,2,1,1,1)



- 画像の平滑化を行う
- 処理対象画素（ピクセル）の実行結果は、指定したカーネルサイズ領域の平均値になる
- カラー画像の場合、各色（R,G,B）で処理を行う

2	3	2
1	<b>6</b>	5
4	1	1

カーネルサイズ：3x3の例


$$\frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

2	3	2
1	<b>3</b>	5
4	1	1

$$(2+3+2+5+1+1+4+1+6)/9 = 2.78 \rightarrow 3$$

- 近くの画素の影響を大きくしてフィルタをかけたい
- 画像の平滑化を行う (Gaussian Filterによるmasking)
- 処理対象画素 (ピクセル) の実行結果は、ガウシアンカーネルで計算した値になる

$$\frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

3x3のガウシアン  
カーネルの例

×

(画像の画素値)

2	3	2
1	6	5
4	1	1



2	3	2
1	3	5
4	1	1

$$(2 \times 1 + 3 \times 2 + 2 \times 1 + 5 \times 2 + 1 \times 1 + 1 \times 2 + 4 \times 1 + 1 \times 2 + 6 \times 4) / 16 = 3.3125$$

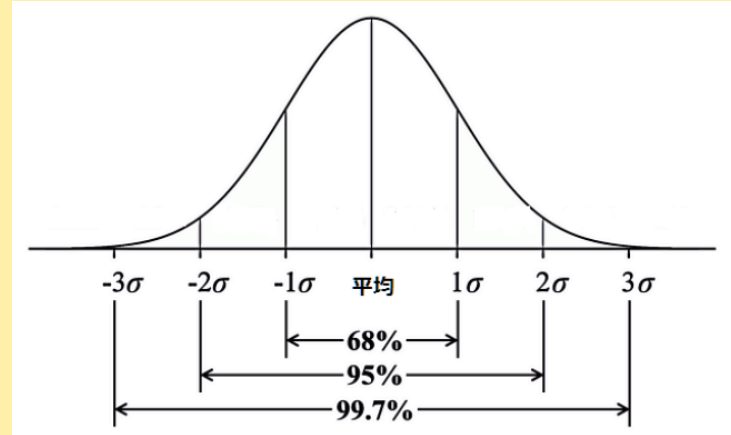
フィルタ一処置後

- ガウシアンカーネル(mask)は、ガウス分布関数計算される  
➤正規分布と同じ式

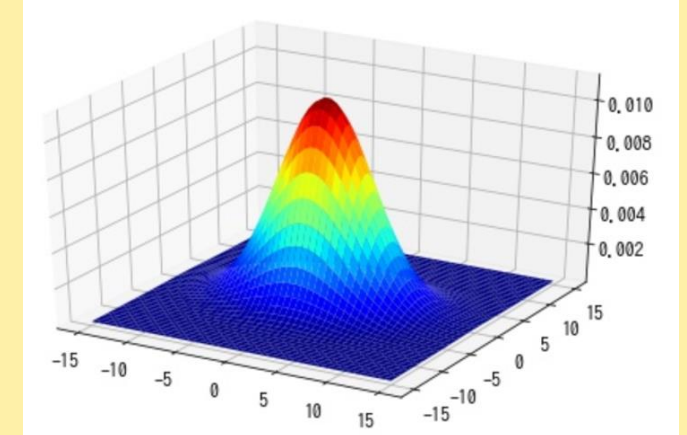
$$f(x, y) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right)$$

ガウス分布関数

ブルー(blur)処理



1次元



2次元

	X		
	$f(-1,-1)/s$	$f(0,-1)/s$	$f(1,-1)/s$
	$f(-1,0)/s$	$f(0,0)/s$	$f(1,0)/s$
↓ y	$f(-1,1)/s$	$f(0,1)/s$	$f(1,1)/s$



$$\frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

(0,0)を中心にした座標,  $\sigma=0.8$

# 演習3-0

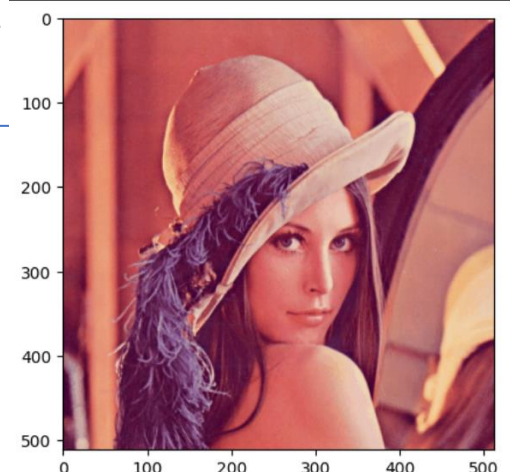
- g23\_3\_9.pyを実行し、正しくlenaが表示されることを確認してください。
  - エラーが起きないこと、色が変わっていないこと、を確認してください
  - 今後の演習で重要です

```
#Google colabの場合
"""
from google.colab import files
from google.colab.patches import cv2_imshow
"""

#Local PCの場合。Google colabはcomment out. bmpの置き場は適当に変えてください
img =
cv2.imread("c:/pythonwork/gazoonsei/img2/lena.bmp",
cv2.IMREAD_UNCHANGED)
```

```
#Google colabの場合
"""
uploaded = files.upload()    #2回目以降は不要
img = cv2.imread('lena.bmp', cv2.IMREAD_UNCHANGED)
"""
```

- Local PCの場合、
  - c:/pythonwork/gazoonsei/img2/lena.bmpの部分は、自分のdirectoryにしてください
- Google colabの場合
  - """で囲まれている部分を外して、img=cv2.imread"#local PCの場合”、下のimg=cv2.imread(“…”)はコメントアウトしてください
  - uploaded = files.upload() の部分は、uploadに時間がかかる場合のみ利用して、それ以外はコメントアウトしてください





# 演習3-1(LMS提出)

- 左下の画素値を持つ画像に対して、median処理、blur処理、Gaussian処理、をかけた場合の真ん中の画素の画素値がいくつになるか求め、結果を1., 2., 3の順にLMSに記述しなさい (blur処理、Gaussian処理は、前2頁と同じ3x3のカーネルを利用)

1.median処理

5	3	7
2	20	16
4	8	10



5	3	7
2	?	16
4	8	10

3.blur処理

5	3	7
2	?	16
4	8	10

2.Gaussian処理

5	3	7
2	?	16
4	8	10

# 演習3-1 解答

- 左下の画素値を持つ画像に対して、median処理、blur処理、Gaussian処理、をかけた場合の真ん中の画素の画素値がいくつになるか求め、結果を1., 2., 3の順にLMSに記述しなさい (blur処理、Gaussian処理は、前2頁と同じ3x3のカーネルを利用)

5	3	7
2	20	16
4	8	10



## 1.median処理

2,3,4,5,6,**7**,8  
,10,16,20

5	3	7
2	<b>7</b>	16
4	8	10

## 3.blur処理

5	3	7
2	<b>8</b>	16
4	8	10

## 2.Gaussian処理

5	3	7
2	<b>10</b>	16
4	8	10

$$(5 \times 1 + 3 \times 2 + 7 \times 1 + 2 \times 2 + 20 \times 4 + 16 \times 2 + 4 \times 1 + 8 \times 2 + 10 \times 1) / 16 = 10.25$$

$$(5 + 3 + 7 + 2 + 20 + 16 + 4 + 8 + 10) / 9 = 8.33$$

## 演習3-2(LMS提出)

- g23\_3\_0.pyの”to be filled in”(37行目)の部分をコーディングして、カーネルサイズ3x3のblur filterを実現して、出力してください
  - noisy\_lena.bmpはLMSからdownloadして、12行目は自分のdirにしてください
  - このあと説明する、 `cv2.blur` 関数はまだ使わないでください
  - 37行目以外の方法でもよいです
- 37行目のコーディング部分のみを、LMSに貼り付けてください

original



Blur filter-0



ヒント：imgRGBの(x,y)の画素と(x+1,y)の画素の平均をimgRGB2の(x,y)に置き換える場合、

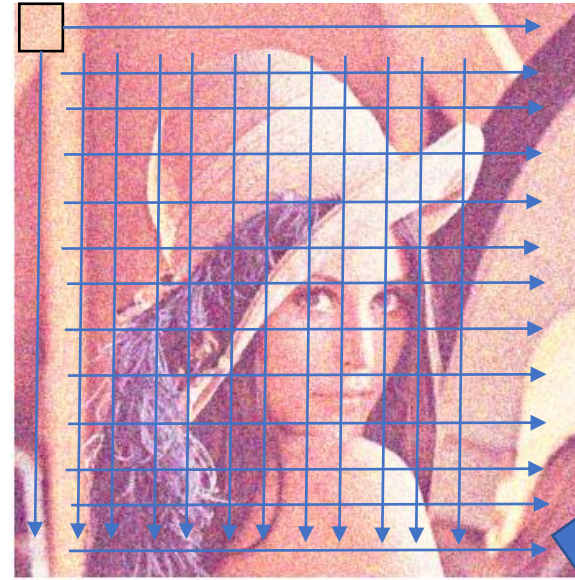
```
imgRGB2[y][x][c] =  
round((int(imgRGB[y][x][c])+  
int(imgRGB[y][x+1][c]))/2)
```

とします(この問題では、2でなく9で割ります)  
intを付けないと、足した値が255以下に制限されます

round : 四捨五入

# 注意

- フィルタは、全画面に満遍なくかけられています。
- つまり、 $3 \times 3$ 等のカーネルが、1ピクセルずつ動いてフィルタ処理されます
- フィルタ処理をかけた画像に更にフィルタ処理をかけないように、フィルタ処理は常にoriginal画像に対して施されます



例えば $3 \times 3$ カーネルで、画像サイズが $256 \times 256$ の場合、フィルタ処理は、縦、横各々254回、計 $254 \times 254$ 回（カラーの場合は、更に $\times 3$ 回）行われる



$M \times M$ カーネルで、画像サイズが $N \times N$ の場合、フィルタ処理は、縦、横各々 $(N - M + 1)$ 回、計 $(N - M + 1) \times (N - M + 1)$ 回（カラーの場合は、更に $\times 3$ 回）行われる



```
total = int(imgRGB[i-1][j-1][k])+int(imgRGB[i][j-1][k])+imgRGB[i+1][j-1][k]+  
imgRGB[i-1][j][k]+int(imgRGB[i][j][k])+int(imgRGB[i+1][j+1][k])+  
int(imgRGB[i-1][j+1][k])+int(imgRGB[i][j+1][k])+int(imgRGB[i+1][j+1][k])
```

```
#      こちらでも正解です  
      ""  
      total = 0  
      for ii in range(i-1,i+2):  
          for jj in range(j-1,j+2):  
              total += int(imgRGB[ii][jj][k])  
  
      imgRGB2[i][j][k] = round(total/9)  
      ""
```

# フィルタ適用画像例-2

g23\_3\_1.py



```
dst =  
cv2.medianBlur(img,3):  
Median処理、カーネルサイズ3x3
```

```
dst =  
cv2.GaussianBlur(img,(3,3),0)  
Gaussian処理、カーネルサイズ3x3
```

```
dst = cv2.blur(img,(3,3)):  
Blur処理、カーネルサイズ3x3
```



`dst = cv2.medianBlur(img,3)` : Median処理

`dst = cv2.blur(img,(3,3))` : blur処理

`dst = cv2.GaussianBlur(img,(3,3),0)` : Gaussian処理

上記は全てカーネルサイズ(ksize)が 3x3 の例だが、3の所を他の数字にすれば、そのカーネルサイズとなる

GaussianBlurの最後の引数は  $\sigma$  を表す。但し、0の場合、 $\sigma = 0.3 * (ksize / 2 - 1) + 0.8$  (この場合、 $0.3 * (3 / 2 - 1) + 0.8 = 0.95$ )

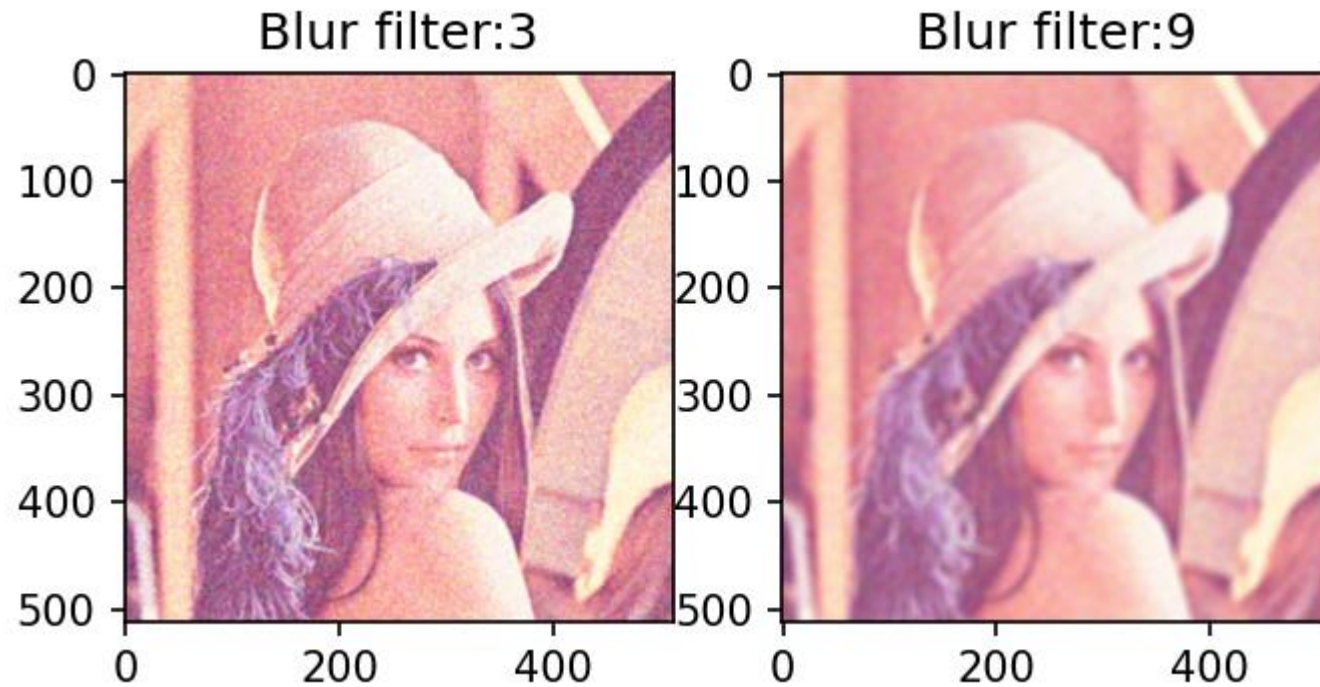
1. Blur処理で、カーネルサイズを、3x3と9x9で適用し、その画像をLMSに貼ってください。また、何故そのような画像になるか、理由を考えてください（理由はLMSに書かなくてよいです）。
  2. Gaussian処理で、カーネルサイズを、9x9に固定して、 $\sigma$ の値を、1,6に変えて、画像をLMSに貼ってください。また、何故そのような画像になるか、理由を考えてください（理由はLMSに書かなくてよいです）
- 各々、g23\_3\_2.py, g23\_3\_3.pyを参考にしてください( to be filled in の所をコーディングして、実行してください)
  - 画像は、noisy\_lena.bmpを使ってください



# 演習3-3 解答例-1

```
i=1
for s in [3,9]:
    ax = fig.add_subplot(1,3,i)
    ax.set_title("Blur filter:"+str(s), loc='center')
    dst = cv2.blur(imgRGB,(s,s))
    plt.imshow(dst)
    i += 1
```

g23\_3\_2\_ans.py

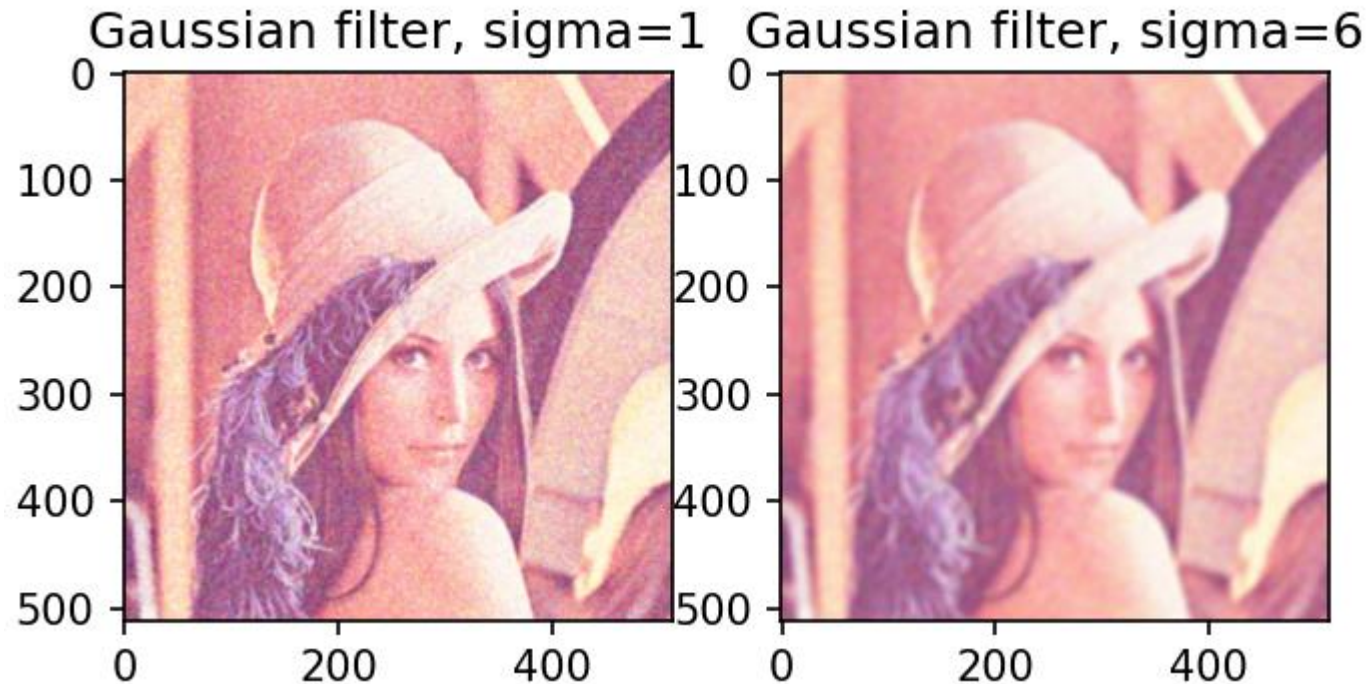


理由：カーネルサイズが大きくなることにより、平均化する範囲が大きくなり、ノイズが平均化されて軽減された一方で、元画像自体もピンぼけしたような感じになった。

## 演習3-3 解答例-2

```
i = 1
for s in [1,6]:
    ax = fig.add_subplot(1,3,i)
    ax.set_title("Gaussian filter, sigma="+str(s), loc='center')
    dst = cv2.GaussianBlur(imgRGB,(9,9),s)
    plt.imshow(dst)
    i += 1
plt.show()
```

[g23\\_3\\_3\\_ans.py](#)



# 演習3-3 解答例-2 続き

$\sigma$ が大きくなると、大きくなる

$$f(x, y) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right)$$

ガウス分布関数

$\sigma$ が大きくなると、小さくなる

理由： $\sigma$ が大きくなると、 $x^2+y^2$ の値に関わらず $f(x,y)$ の値が近くなり、結果的に、カーネル内全ての画素の平均化に近くなるため、平均化されたような画像になる

$\sigma$	$x^2+y^2 \rightarrow$	0	1	2	3	4
	1	0.159	0.097	0.059	0.036	0.022
	5	0.147	0.152	0.155	0.156	0.157
	9	0.148	0.148	0.147	0.147	0.147

#画像音声処理3,4回目

```
import cv2
import matplotlib.pyplot as plt
```

#Google colabの場合

```
#from google.colab import files
#uploaded = files.upload() #2回目以降は不要. drag & dropでできればそもそも不要
#img = cv2.imread('lena.bmp', cv2.IMREAD_UNCHANGED)
```

#Local PCの場合。Google colabはcomment out. bmpの置き場は適当に変えてください

```
img = cv2.imread("c:/pythonwork/gazoonsei/img2/lena.bmp", cv2.IMREAD_UNCHANGED)
```

```
imgRGB=cv2.cvtColor(img,cv2.COLOR_BGR2RGB)
```

```
fig = plt.figure(dpi=100, figsize=(8,8))
```

#3-2 Gaussian filter

```
i = 1
for s in [1,6]:
    ax = fig.add_subplot(1,3,i)
    ax.set_title("Gaussian filter, sigma="+str(s), loc='center')
    dst = cv2.GaussianBlur(imgRGB,(9,9),s)
    plt.imshow(dst)
    i += 1
```

ここは、自分のPCにfileを入れたdirectoryに置き換えてください

opencv (cv2)だとBGRの順序なので、RGBの順序に置き換えます  
(Google Colab PCも同じ)



# 補足2 – Google Colab

#画像音声処理3,4回

```
import cv2
```

```
import matplotlib.pyplot as plt
```

#Google colabの場合

```
from google.colab import files
```

```
uploaded = files.upload() #2回目以降は不要. drug & dropでできればそもそも不要
```

```
img = cv2.imread('lena.bmp', cv2.IMREAD_UNCHANGED)
```

#Local PCの場合。Google colabはcomment out. bmpの置き場は適当に変えてください

```
#img = cv2.imread("c:/pythonwork/gazoonsei/img2/lena.bmp", cv2.IMREAD_UNCHANGED)
```

```
imgRGB=cv2.cvtColor(img,cv2.COLOR_BGR2RGB)
```

```
fig = plt.figure(dpi=100, figsize=(8,8))
```

```
#3-2 Gaussian filter
```

```
i = 1
```

```
for s in [1,6]:
```

```
    ax = fig.add_subplot(1,3,i)
```

```
    ax.set_title("Gaussian filter, sigma="+str(s), loc='center')
```

```
    dst = cv2.GaussianBlur(imgRGB,(9,9),s)
```

```
    plt.imshow(dst)
```

```
    i += 1
```

g23\_3\_3\_ans\_gc.py

中身はg23\_3\_2\_ans.py

と同じ(コメントアウトの場所のみ違う)

ここのコメントアウトは外してください。

最初の2行は、2回目以降や、**drug & drop**でできればそもそも不要

ここは、#でコメントアウト

opencv (cv2)だとBGRの順序なので、RGBの順序に置き換えます  
(local PCも同じ)

- 縦・横方向の簡単なエッジ検出
- Sobelフィルタ
- Laplacianフィルタ
- Cannyエッジ検出

# エッジ検出とは？

- 画像中の輝度が大きく変化するところを検出する
  - 画像の変わり目やオブジェクトが検出できる
- エッジ検出は、RGBをYUV (Y Cr Cb) という、輝度成分と色成分に分けて、その輝度成分で検出することが多い（3つの成分で行うと、エッジが分散することがある）
- そのため、基本、モノクロ画像（のちにカラー画像）を進めます



エッジ検出



# エッジ検出の直感的な理解

- 輝度が大きく変わる場所を見つければよい。



- 例えば、縦方向の線の場合、輝度が右のように急激に変化していることが考えられる。
- どうすればこれを見つけられる？



- X方向に、隣の輝度との差を計算



- $(x,y)$  と  $(x+1,y)$  の輝度の差を計算

2	50	49
2	50	55
4	54	61
2	55	62
2	58	59
4	59	61



- 縦方法のエッジを強調したい
- 縦に垂直な方向（つまり、横）方向の、画素値の変化を調べばよい

2	3	2
2	<b>6</b>	15
4	11	1

(真ん中)  $15 - 6 = 9$



$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & -1 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$

1	-1	-
4	<b>9</b>	-
7	-10	-

$\begin{bmatrix} 0 & 0 & 0 \\ -1 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$  や  $\begin{bmatrix} 0 & 0 & 0 \\ -1 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}$  等もあります



- 横方法のエッジを強調したい
- 横に垂直な方向（つまり、縦）方向の、画素値の変化を調べればよい

2	3	2
2	<b>6</b>	5
4	11	1

(真ん中)  $11 - 6 = 5$



$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

0	3	3
2	<b>5</b>	-4
-	-	-

$\begin{bmatrix} 0 & -1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$  や  $\begin{bmatrix} 0 & -1 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$  等もあります

# これらは一次微分フィルタともいう

- これらのフィルタを、1次微分フィルタ、と呼ぶ

$\Delta I_x(x, y) = I(x + 1, y) - I(x, y)$  x方向微分(縦成分)

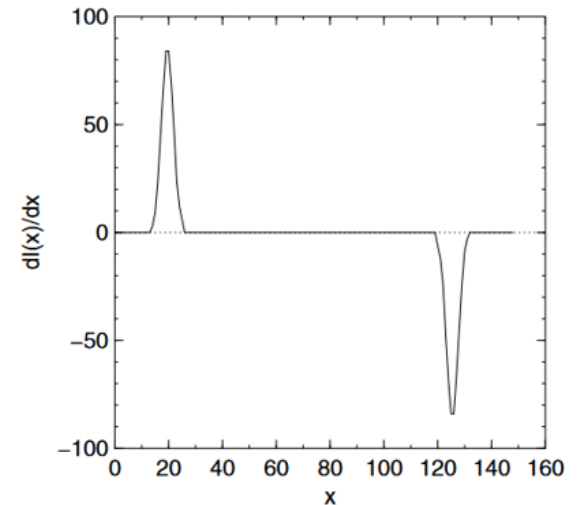
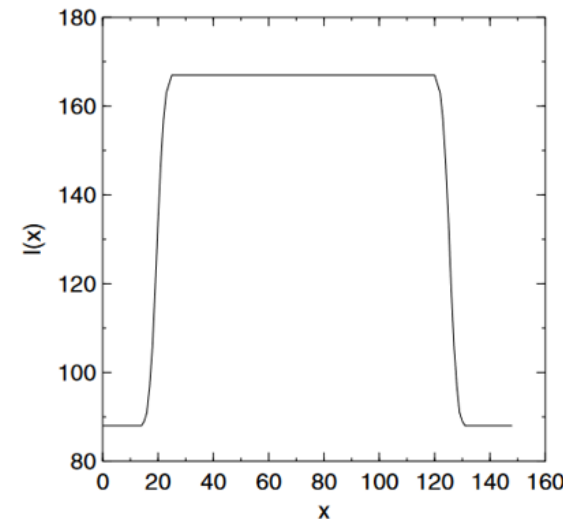
$\Delta I_y(x, y) = I(x, y + 1) - I(x, y)$  y方向微分(横成分)

$I(x, y)$ :元画像の(x,y)の画素値  
 $\Delta I_x(x, y), \Delta I_y(x, y)$ :フィルタ処理後の(x,y)の画素値

- 微分の式：

$$f'(x) = \frac{f(x + \Delta x) - f(x)}{\Delta x}$$

上記は、 $\Delta x = 1$ の場合に該当



<http://www.osakac.ac.jp/labs/hild/IPslide7.pdf>

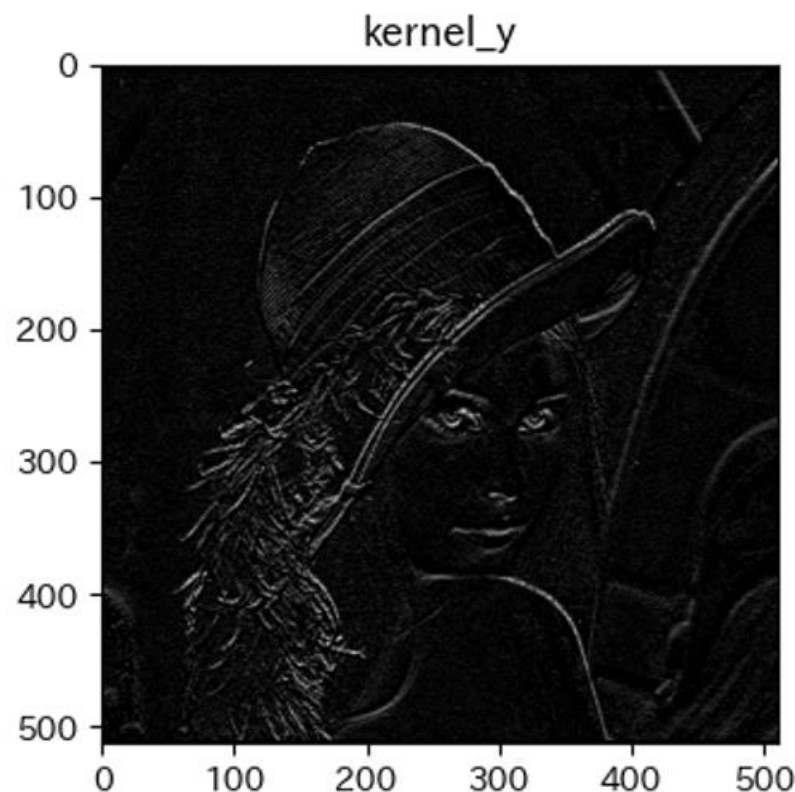
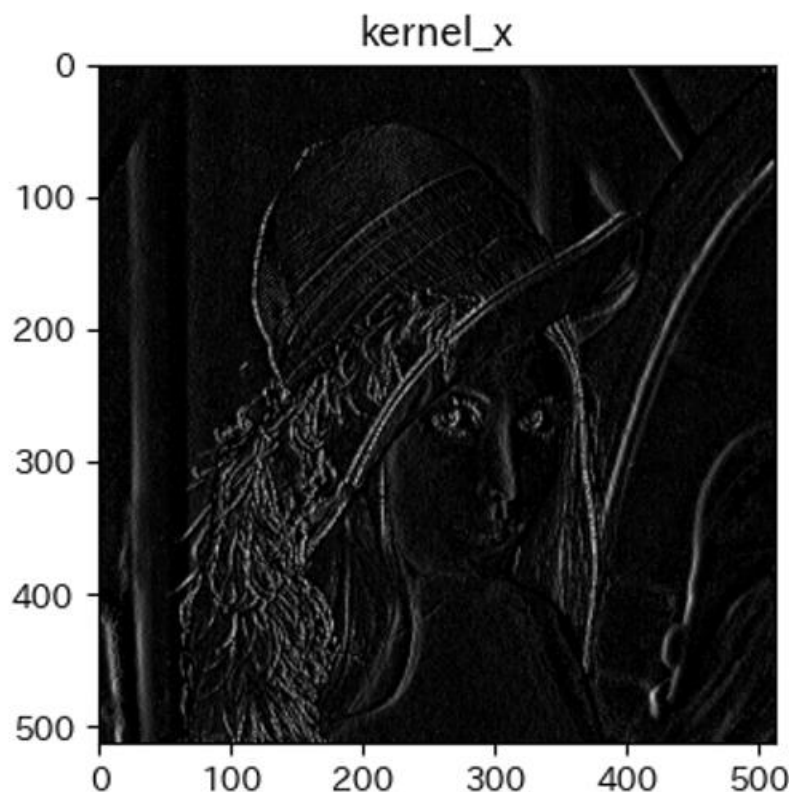
# エッジ検出 適用画像例

# 1. 垂直方向の輪郭検出

```
kernel_x = np.array([[0, 0, 0],  
                    [0, -1, 1],  
                    [0, 0, 0]])  
img_x = 5*cv2.filter2D(img, -1, kernel_x)
```

# 2. 水平方向の輪郭検出

```
kernel_y = np.array([[0, 0, 0],  
                    [0, -1, 0],  
                    [0, 1, 0]])  
img_y = 5*cv2.filter2D(img, -1, kernel_y)
```



[g23\\_3\\_4.py](#)

計算結果が0未満や255より大きい場合、各々0,255に丸め込まれる

g23\_3\_4.py

を実行し（fileの位置を自分のものに変換）、エッジ検出した画像が出力されることを確認してください

# コードの説明 (g23\_3\_4.py)

# 1. 垂直方向の輪郭検出

```
kernel_x = np.array([[0, 0, 0],  
                    [0, -1, 1],  
                    [0, 0, 0]])
```

```
ax = fig.add_subplot(1,2,1)
```

```
ax.set_title("kernel_x", loc='center')
```

```
img_x = 5*cv2.filter2D(img, -1, kernel_x)
```

```
plt.imshow(img_x, cmap='gray')
```

このカーネル(kernel\_x)で、  
フィルタをかける

5倍して、画像  
を濃くする

-1:入出力で同じ型 (基本-1)

これを入れないと、grayimageで出力されない



- 元画像にノイズが乗っている場合に、ノイズが乗りにくくなるように、左右、上下の画素を使って微分を行うフィルタ

2	4	7
2	<b>6</b>	5
4	1	1



$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

-	-	-
-	<b>8</b>	-
-	-	-

x方向：  
 $-(2+2*2+4)+(7+5*2+1)$   
 $=8$

$$\begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

-	-	-
-	<b>10</b>	-
-	-	-

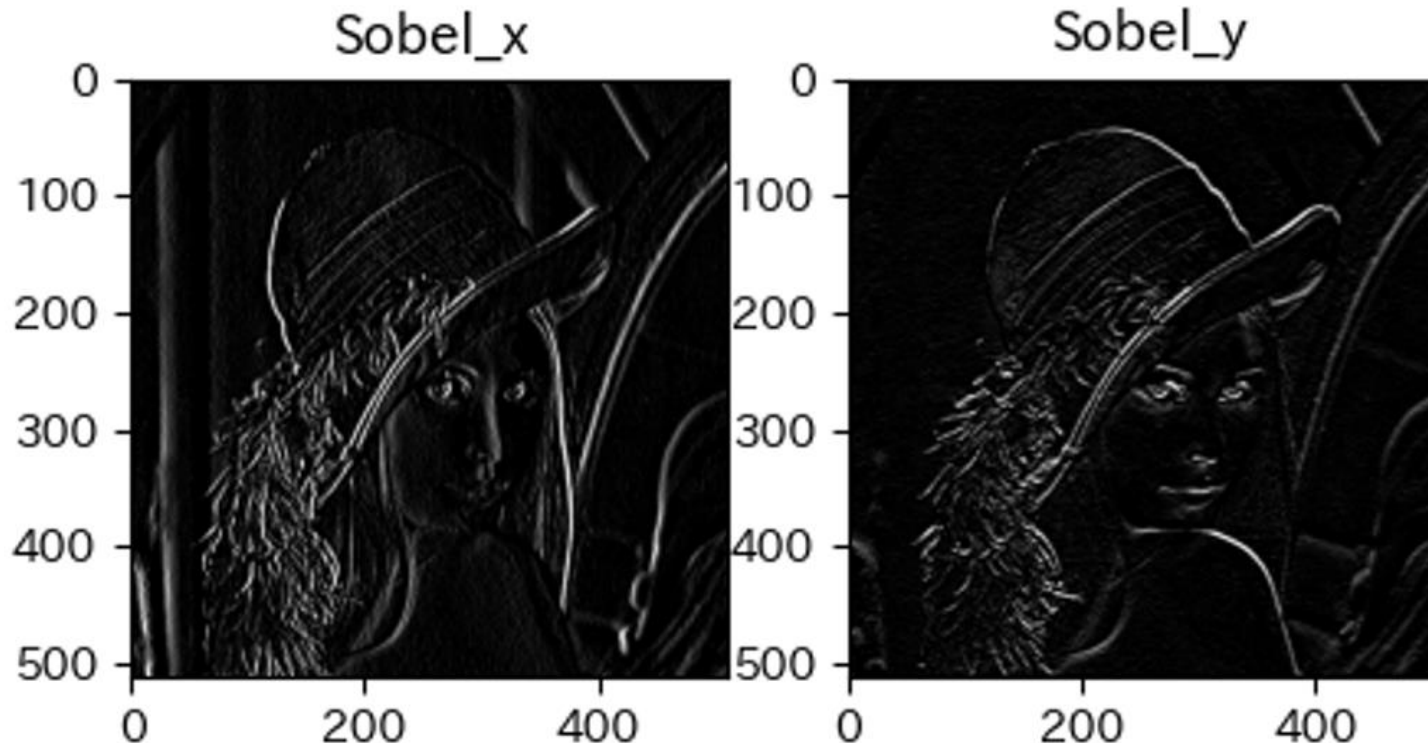
y方向：  
 $-(4+1*2+1)+(2+4*2+7)$   
 $=10$

# Sobel フィルタ – 適用画像例

`img_sx = cv2.Sobel(img, -1, 1, 0, ksize = 3)`
 $\rightarrow \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$ 
 $\rightarrow$  縦方向のエッジの強調 (Sobel\_x)

出力=入力と同じdtype      x方向微分次数      y方向微分次数      カーネルサイズ

`img_sy = cv2.Sobel(img, -1, 0, 1, ksize = 3)`
 $\rightarrow \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$ 
 $\rightarrow$  横方向のエッジの強調 (Sobel\_y)



g23\_3\_5.py

g23\_3\_5.py

を実行し（fileの位置を自分のものに変換）、エッジ検出した画像が出力されることを確認してください

# Laplacian フィルタ-1

- 2次微分フィルタ

2	3	2
2	<b>6</b>	5
4	1	1

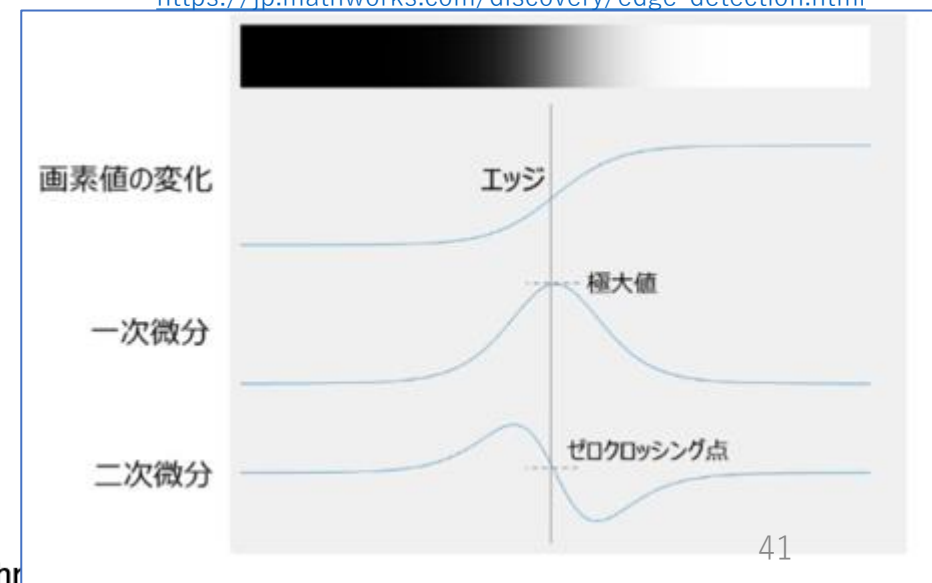


$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

2	3	2
2	<b>-14</b>	5
4	1	1

x方向：  
 $1+3+5+1-6*4=-14$

<https://jp.mathworks.com/discovery/edge-detection.html>



- 2次微分フィルタの考え方
  - 1次微分フィルタを、更に微分したフィルタ

$$\frac{\partial}{\partial x} I(x, y) = \Delta I_x(x, y) = I(x + 1, y) - I(x, y)$$

$$\begin{aligned} \frac{\partial^2}{\partial x^2} I(x, y) &= \frac{\partial}{\partial x} \left( \frac{\partial}{\partial x} I(x, y) \right) = I(x + 1, y) - I(x, y) - (I(x, y) - I(x - 1, y)) \\ &= I(x + 1, y) - 2I(x, y) - I(x - 1, y) \end{aligned}$$

同様に、

$$\frac{\partial^2}{\partial y^2} I(x, y) = I(x, y + 1) - 2I(x, y) - I(x, y - 1)$$

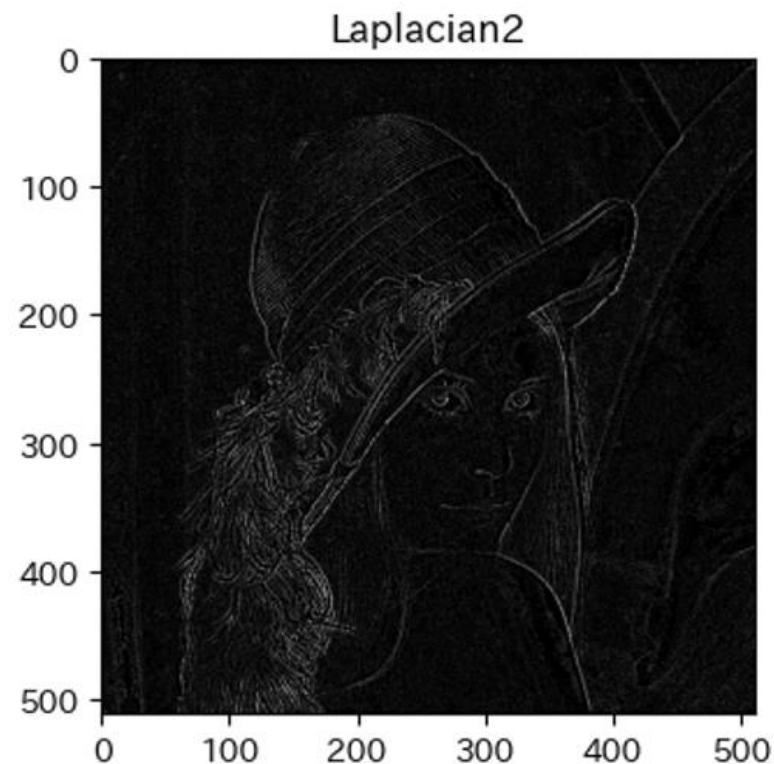
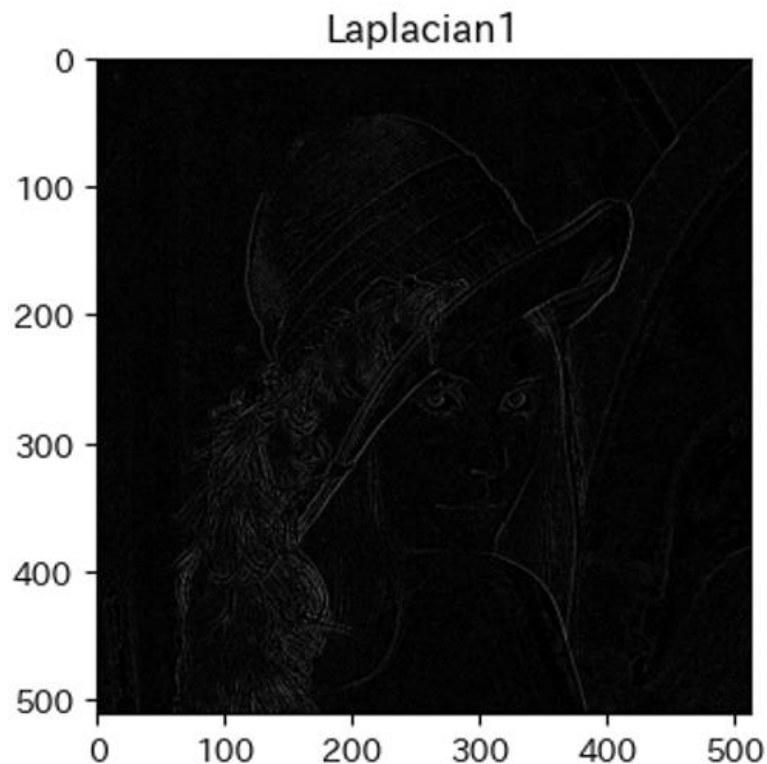
$$\frac{\partial^2}{\partial x^2} I(x, y) + \frac{\partial^2}{\partial y^2} I(x, y) = \underline{I(x, y + 1) + I(x, y - 1) + I(x + 1, y) + I(x - 1, y) - 4I(x, y)}$$

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$



# Laplacian フィルタ – 適用画像例

```
img_lap = cv2.Laplacian(img, -1)    →Laplacian1  
img_lap2 = 2*cv2.Laplacian(img, -1) →Laplacian2 →Laplacian1が薄いので、輝度を2倍
```



g23\_3\_6.py

# 演習3-6(LMS提出)

- 左下の画素値を持つ画像に対して、前頁までのSobel filter (Sobel\_x, Sobel\_y)とLaplacian filterを施した際の、フィルタの出力を求め、1., 2., 3.の順に、LMSに記述しなさい。注：値が0未満、255より大の場合、各々0, 255に丸め込みます

4	7	40
5	6	50
8	8	30



1.Sobel\_x

$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

2.Sobel\_y

$$\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

3.Laplacian

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

# 演習3-6 解答

- 左下の画素値を持つ画像に対して、前頁までのSobel filter (Sobel\_x, Sobel\_y)とLaplacian filterを施した際の、フィルタの出力を求め、1., 2., 3.の順に、LMSに記述しなさい

4	7	40
5	6	50
8	8	30



## 1.Sobel\_x

$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

$$40-4+2(50-5)+30-8= 148$$

## 2.Sobel\_y

$$\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

$$8-4+2(8-7)+30-40=-4$$

→0に丸め込む

## 3.Laplacian

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

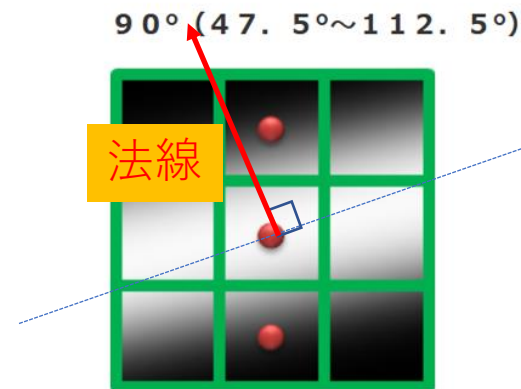
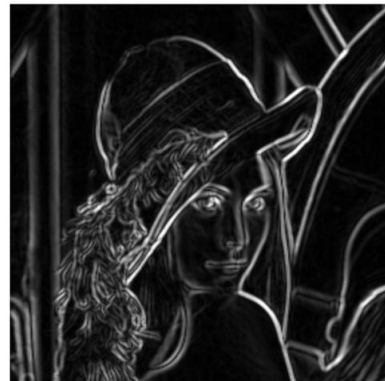
$$5+7+50+8-6 \times 4= 46$$

縦線 (Sobel\_x)が強く反応している

- 画像の輪郭（尾根）に沿ったエッジが検出できる John.F.Cannyが1986年に発表したエッジ検出法

- メディア処理実習で、実装だけ少しやりました
- 詳細は、以下等を参照

- ✓ <https://towardsdatascience.com/canny-edge-detection-step-by-step-in-python-computer-vision-b49c3a2d8123>
- ✓ <https://imaging-solution.net/imaging/canny-edge-detector/>



<https://imaging-solution.net/imaging/canny-edge-detector/>

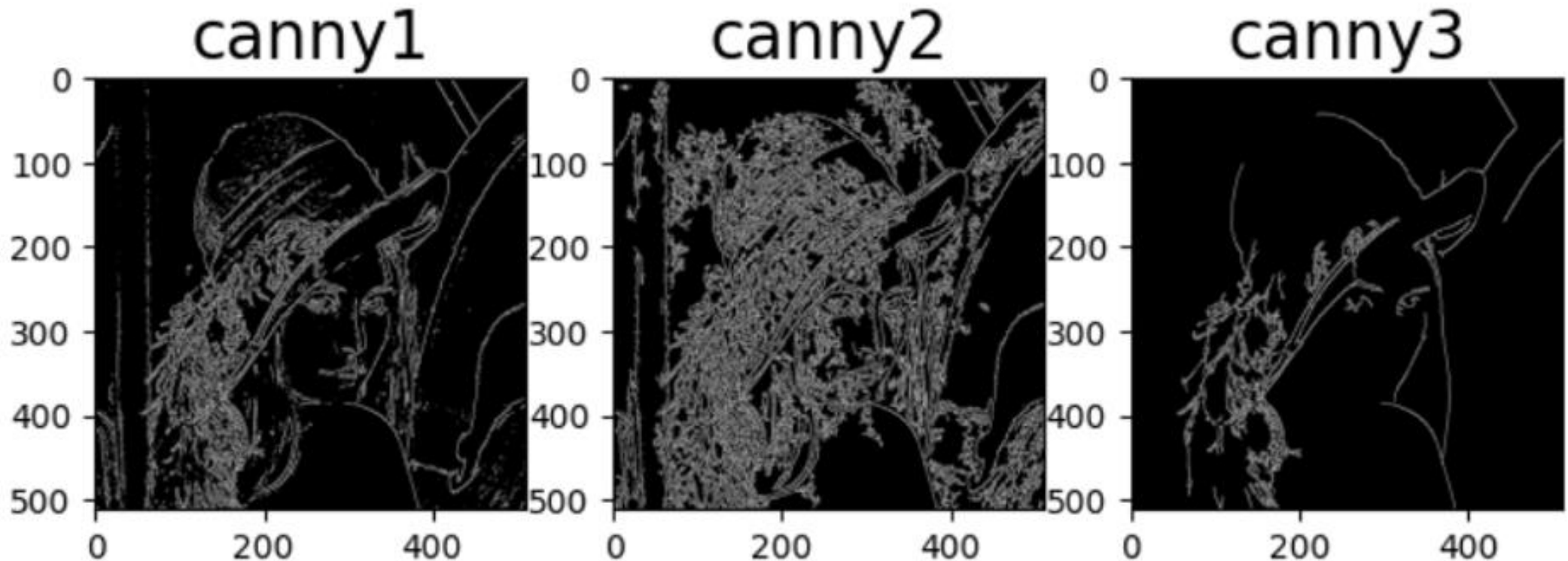
# Canny出力例

パラメータ 1 : エッジの  
スタートの閾値

パラメータ2 : エッジの  
終わりの閾値

```
imgGray_canny1 = cv2.Canny(img,100,100)  
imgGray_canny2 = cv2.Canny(img,10,100)  
imgGray_canny3 = cv2.Canny(img,100,500)
```

g23\_3\_8.py



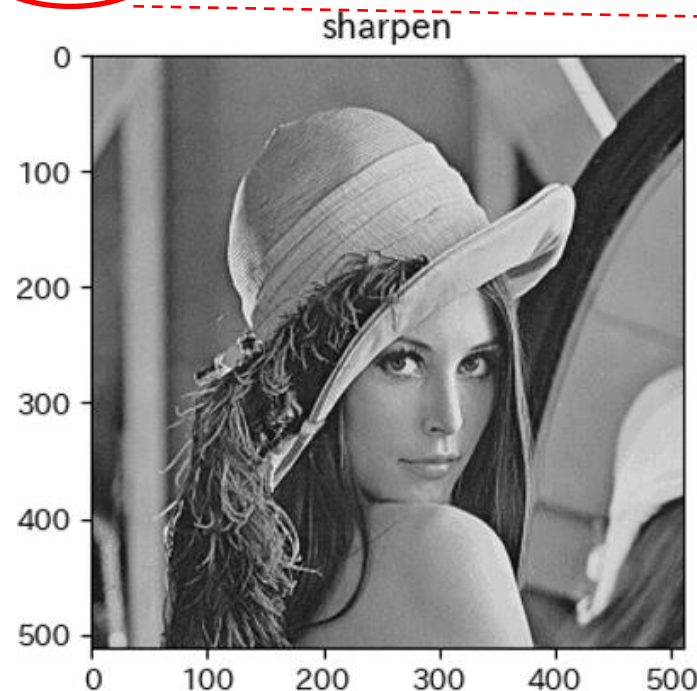
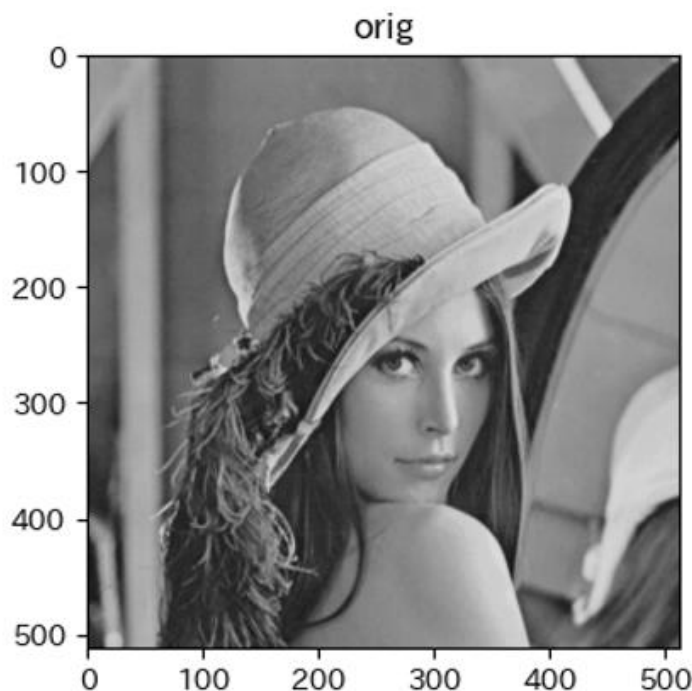


g23\_3\_8.py

を実行し（fileの位置を自分のものに変換）、エッジ検出した画像が出力されることを確認してください

- 鮮鋭化フィルタ = 元画像 - ラプラシアンフィルタ

```
kernel_x = np.array([[0, -1, 0],  
                    [-1, 5, -1],  
                    [0, -1, 0]])  
img_sharp = cv2.filter2D(img, cv2.CV_8U, kernel_x)
```

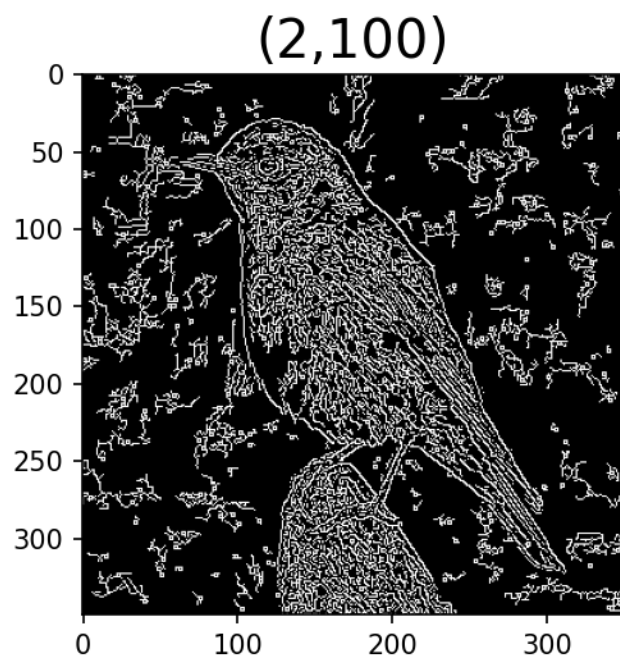
$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} - \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$


画素値を8bit unsigned  
(0-255)に収める

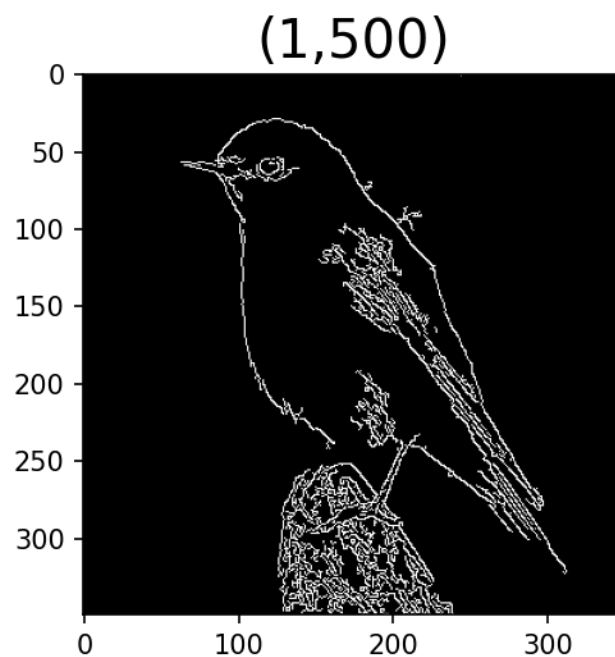
[g23\\_3\\_7.py](#)

1. g23\_3\_8.pyを参考にして、noisy\_bird.jpgを使って、パラメータ1,2が各々(2,100), (100,500)でcanny edge detectionをかけた場合の画像をLMSにuploadし、何故このような画像になったのかを簡単に説明して下さい。
2. g23\_3\_4.py, g23\_3\_7.pyを参考にして、Laplacian filterをcv2.filter2Dを使って実装し、そのコードをLMSに添付してください。
3. 提出期限：B=4/22(土), A=4/24(月), どちらも 9:00

## 宿題2-1 解答例



`cv2.Canny(img, 2, 100)`



`cv2.Canny(img, 1, 500)`

左：edgeのstart/endの閾値が低いため、noiseのedgeも拾い、edgeが長く続く

右：edgeのstartの閾値は低いが、endの敷居が高いため、noiseのedgeは拾わず、画像のedgeの成分の大きいもののみが残る

# 宿題2-2 解答例（重要な部分のみ）

# ラプラシアン

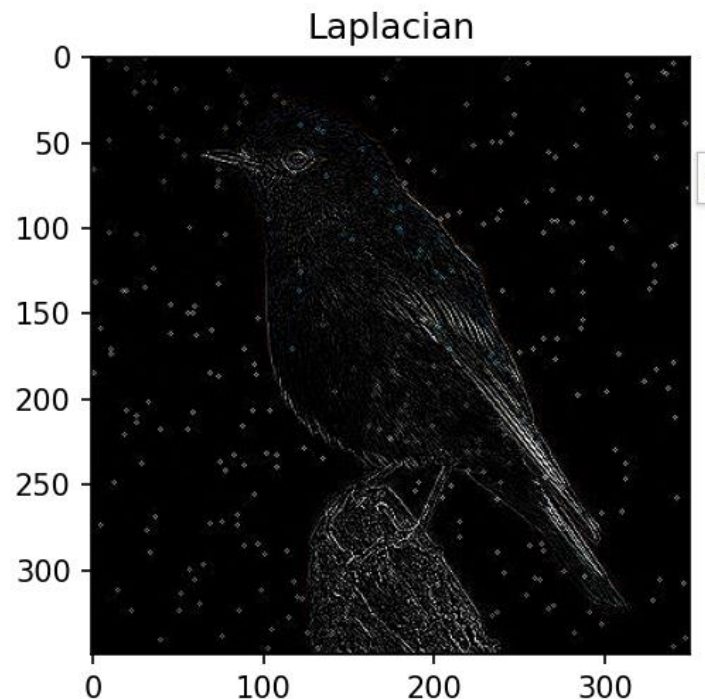
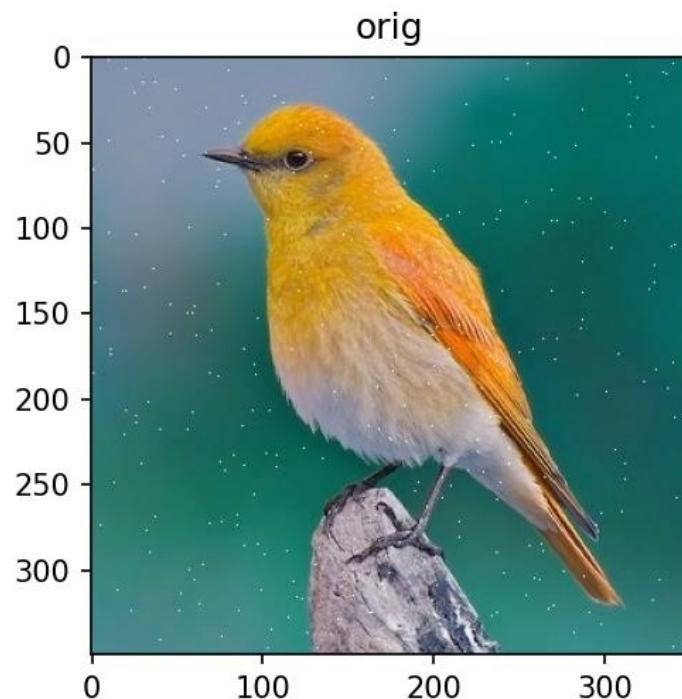
```
kernel_x = np.array([[0, 1, 0],  
                    [1, -4, 1],  
                    [0, 1, 0]])
```

```
img_laplacian = cv2.filter2D(img, cv2.CV_8U, kernel_x)
```

# ラプラシアン

```
np.array([[1, 1, 1],  
        [1, -8, 1],  
        [1, 1, 1]])
```

もOK(8 neighbor)





# 本日のまとめ

- フィルタ処理
  - 画像の見やすさの向上
  - 画像の特徴の強調



- 平滑化フィルタ

- 画像を滑らかにする



- エッジ検出

- 画像のエッジを検出する



- 鮮鋭化フィルタ

- 画像を鋭くする

