ディジタルメディア処理

担当: 井尻 敬

フィルタ処理：トーンカーブ，線形フィルタ

達成目標

- 画素ごとの変換である**トーンカーブ**の機能と効果を説明できる
- **線形フィルタ**の機能と効果を説明できる

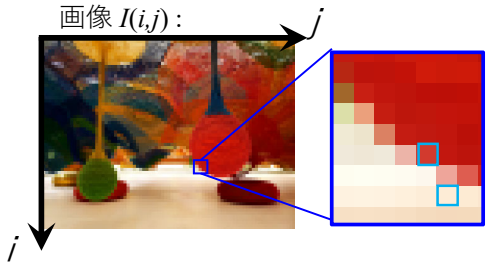
Contents

- トーンカーブ
  - 反転, 二値化, ポスタリゼーション, ソラリゼーション, ガンマ変換, カラー画像
- 線形フィルタ
  - 平滑化フィルタ, ソーベルフィルタ，ガウシアンフィルタ，ラプラシアンフィルタ

デジタル画像のフィルタリング

デジタル画像：カラー画像

- 離散値を持つ画素が格子状に並んだデータ
- 画素：pixel= picture + element
- 例 24bit bitmap :各pixelが(R,G,B)毎に8bit[0,255]の整数値を持つ



※原点位置は左下のことも

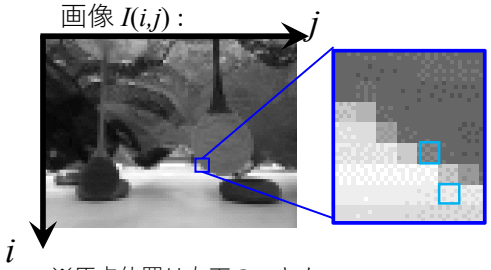
$$I(i,j) = \begin{pmatrix} R:210 \\ G:58 \\ B:46 \end{pmatrix}$$

$$I(i+2,j+1) = \begin{pmatrix} R:253 \\ G:236 \\ B:214 \end{pmatrix}$$

※ $I($ 縦軸, 横軸$)$ の順なので注意してください

デジタル画像：グレースケール画像

- 離散値を持つ画素が格子状に並んだデータ
- 画素：pixel= picture + element
- 例 8bit bitmap :各pixelが8bit [0,255]の整数値を持つ



※原点位置は左下のことも

■ $I(\ i\ ,\ j)\ = 130$

□ $I(i+2,j+1) = 235$

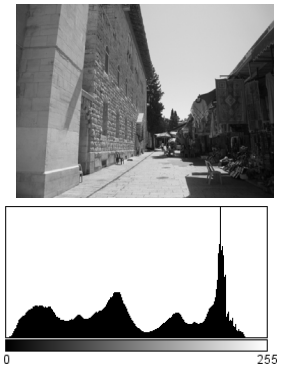
※ $I($ 縦軸, 横軸$)$ の順なので注意してください

『頻度表（ヒストグラム）』とは

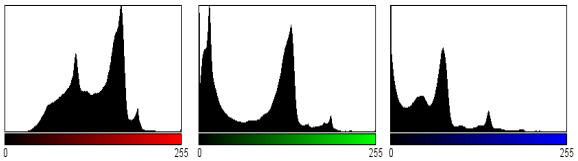
各階調の画素数を数えた表のこと

回転や平行移動に依存しない特徴量 → 画像処理に頻出

グレースケール画像

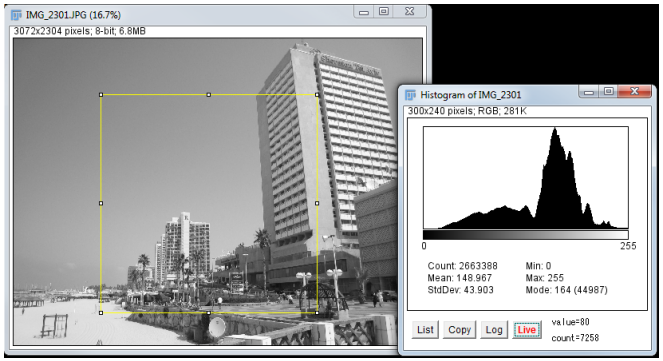


RGBカラー画像



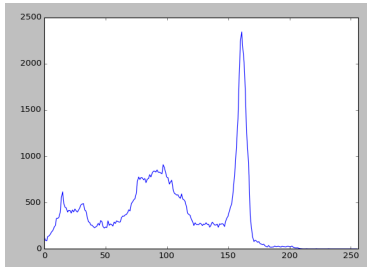
ImageJでヒストグラムを確認してみる

1. ImageJ 起動
2. 画像読み込み
3. Menu > analyze > histogram
4. LiveをOnにすると矩形選択した領域のヒストグラムを確認可能

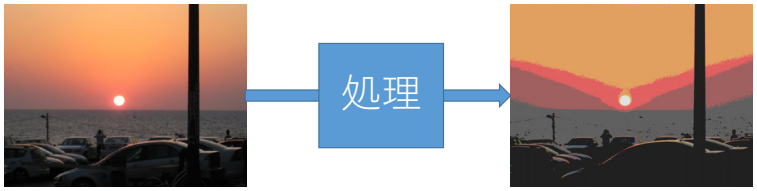



```
import numpy as np
import pylab as plt
import cv2
import itertools
#画像読み込み & グレースケール化
img     = cv2.imread("imgs/sample.png")
img_gry = cv2.cvtColor( img,
cv2.COLOR_BGR2GRAY )

#histogram生成
hist = np.zeros(256)
for y in range(img_gry.shape[0]):
    for x in range(img_gry.shape[1]):
        hist[ img_gry[y,x] ] += 1

#windowを生成して画像を表示
cv2.imshow("Image", img_gry )

#histをmatplotlibで表示
plt.plot(hist)
plt.xlim([0,256])
plt.show()
```

ヒストグラムの計算：histograpm.py



デジタル画像のフィルタリング



入力画像に対し計算処理を施し，所望の画像に変換する

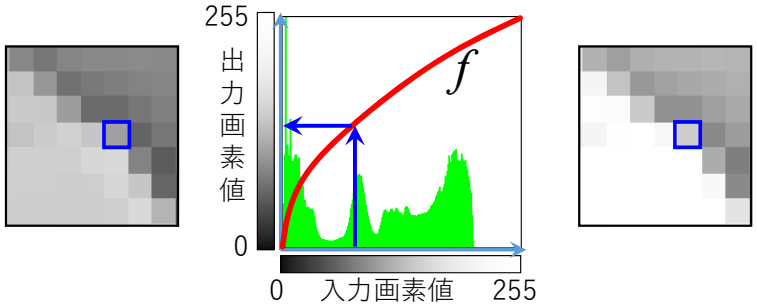
- 特定の周波数を強調する・捨てる（ノイズ除去）
- 画像処理（ステレオ視・領域分割・識別器）の前処理として利用する
- アーティスティックな効果を得る

トーンカーブ

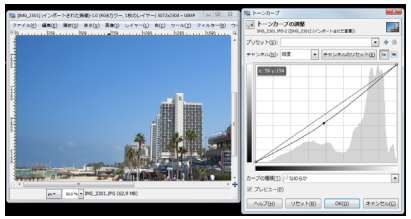
トーンカーブ

ToneCurve.exe (C++)
Image>Adjust>Window/Level (ImageJ)

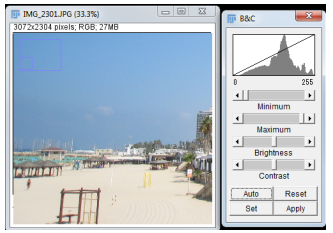
- 画像は8bit グレースケールとする
- 各画素の値を異なる値に変換する**階調変換関数**を考える
  - 新しい画素値 =f(画素値)
- 階調変換関数を曲線で表現したものを**トーンカーブ**と呼ぶ



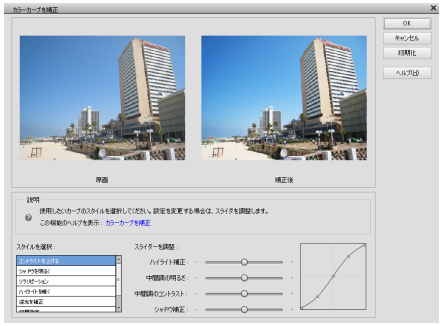
トーンカーブは写真編集の基本ツール



GIMP

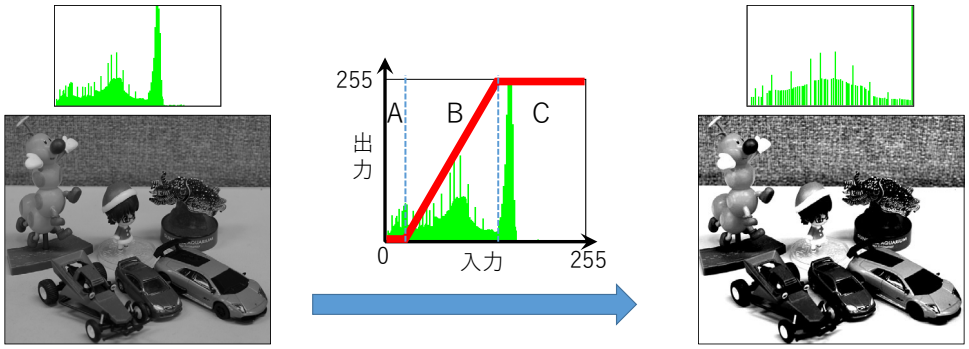


ImageJ: 自由編集でないのでちょっと違うけど



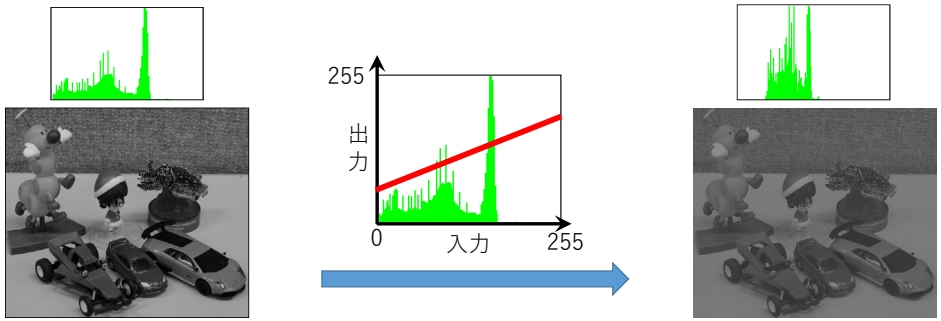
Photoshop Elements カラーカーブ
使いやすいように自由度の限定されたトーンカーブのようなもの
Photoshop CSにはトーンカーブがある（あった）

トーンカーブ: コントラストを上げる



- 領域A：出力画素値0となり黒つぶれ
- 領域C：出力画素値255となり白飛び
- 領域B：傾きが1より大きいため、画素値の取り得る範囲が広がりコントラストが上がる
  画素値は離散値であるため出力ヒストグラムは飛び飛びに

トーンカーブ: コントラストをさげる



- 傾きが1より小さいため、出力画素値の取り得る範囲が縮まり、コントラストが下がる

トーンカーブ：特殊効果

元画像

ネガポジ反転

ポスタリゼーション

出力の色数を
極端に減らす

二値化

閾値より下は0
閾値以上は255

ソラリゼーション

上記のような
曲線を指定

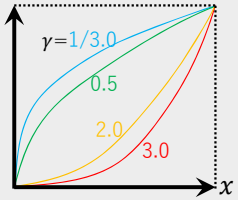
※実装が間に合わず手書きで曲線を与えました。
※本来は関数で与えるべき

トーンカーブ：ガンマ変換（ガンマ補正）

次のトーンカーブを利用した濃淡変換をガンマ変換と呼ぶ

$$y = 255\left(\frac{x}{255}\right)^{\gamma}$$

x：入力値 [0,255]
y：出力値 [0,255]
γ：パラメータ（>0）



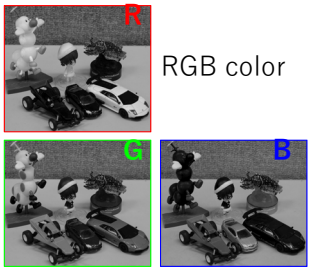
※ RGB各チャンネルにガンマ補正を適用

※ 画像出力デバイスには『**出力値** = (**入力値**)$^{\gamma}$』と言う関係があり、この特性を補正する目的で上記の関数が用いられていた。これを画像の補正に利用したのがガンマ変換

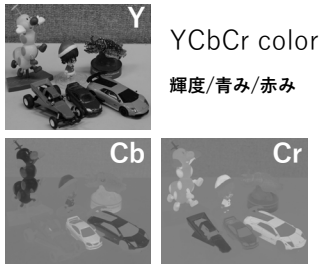
トーンカーブ：カラー画像への適用

カラー画像をトーンカーブで編集するとき …

- RGBの各チャンネルにトーンカーブの画素値変換を適用
- YCbCr Colorに変換し輝度値成分（Y）のみに変換を適用
- その他



RGB color

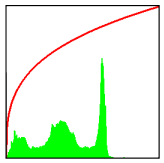


YCbCr color

輝度/青み/赤み

トーンカーブ：カラー画像への適用

入力画像



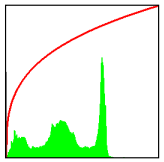
γ =0.3
のガンマ変換

RGB各チャンネル

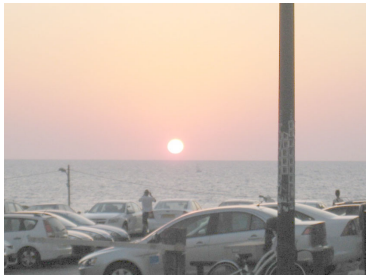
YCbCrの輝度Yのみ

トーンカーブ：カラー画像への適用

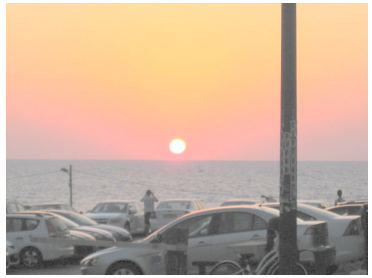
入力画像



γ =0.3
のガンマ変換

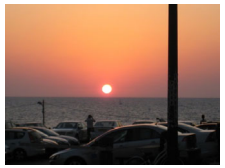


RGB各チャンネル

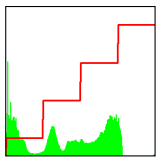


YCbCrの輝度Yのみ

トーンカーブ：カラー画像への適用



入力画像



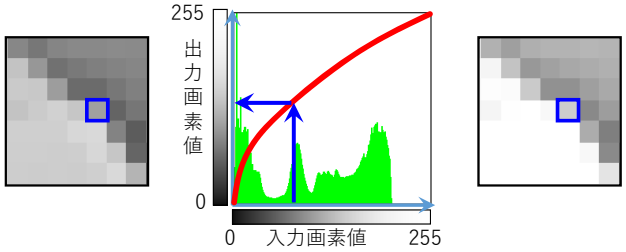
ポスタリゼーション

RGB各チャンネル

YCbCrの輝度Yのみ
（Cb・Crの階調数は減らない）

トーンカーブ：まとめ

- トーンカーブ：各画素の輝度値・色を変換する**階調変換関数**
- 画像の見栄えの編集に利用される
- キーワード: コントラスト変換・ネガポジ反転・ポスタリゼーション・ソラリゼーション・2値化・ガンマ補正



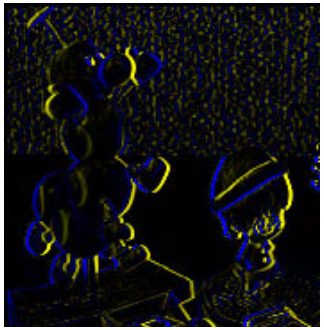
線形フィルタ

線形フィルタの例

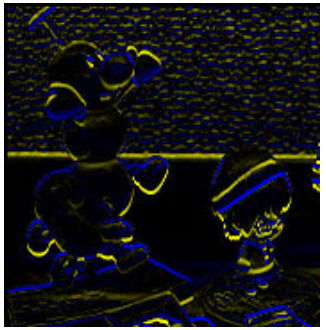
ぼかす

鮮鋭化

線形フィルタの例：エッジ抽出



縦方向



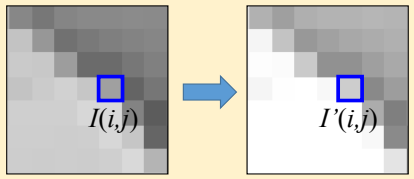
横方向

空間フィルタとは

- 空間フィルタとは周囲の情報を利用して画素値を決めるフィルタ
- 空間フィルタは、線形フィルタと非線形フィルタに分けられる

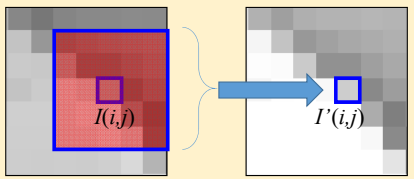
トーンカーブ:
出力画素 $I'(i,j)$ を求めるのに
入力画素 $I(i,j)$のみを利用

入力画像 : $I(i,j)$　　出力画像 : $I'(i,j)$



空間フィルタ:
出力画素 $I'(i,j)$ を求めるのに
入力画素 $I(i,j)$の周囲画素も利用

入力画像 : $I(i,j)$　　出力画像 : $I'(i,j)$

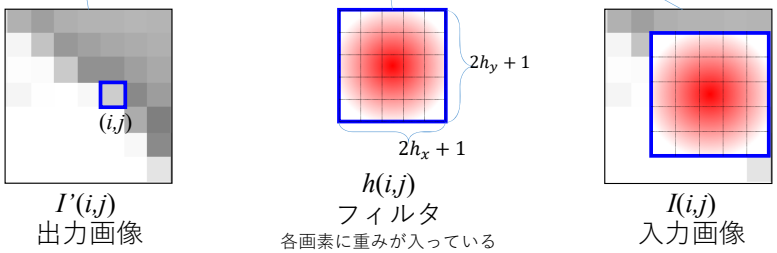


※簡単に説明して後で帰ってくる

2次元 線形フィルタ とは

出力画素値を，入力画像の周囲画素の重み付和で計算する

$$I'(i,j) = \sum_{m=-h_y}^{h_y} \sum_{n=-h_x}^{h_x} h(m,n)\ I(i+m, j+n)$$

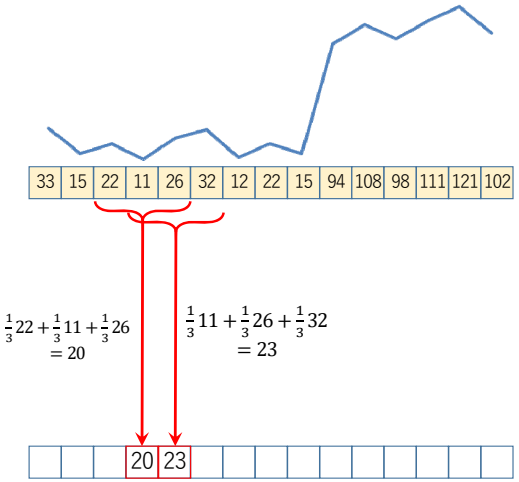


$I'(i,j)$
出力画像

$h(i,j)$
フィルタ
各画素に重みが入っている

$I(i,j)$
入力画像

線形フィルタの例 1D

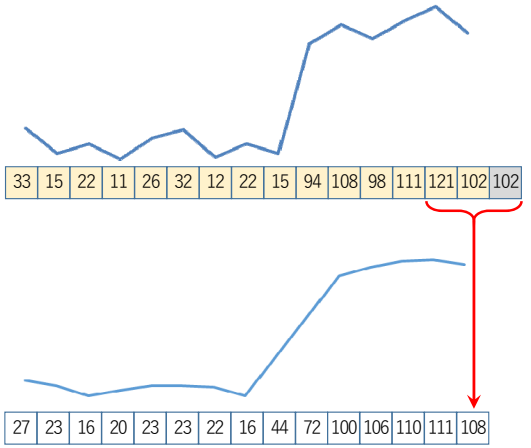


平滑化したい!

1/3	1/3	1/3

周囲 3 ピクセル
の平均を取る

線形フィルタの例 1D



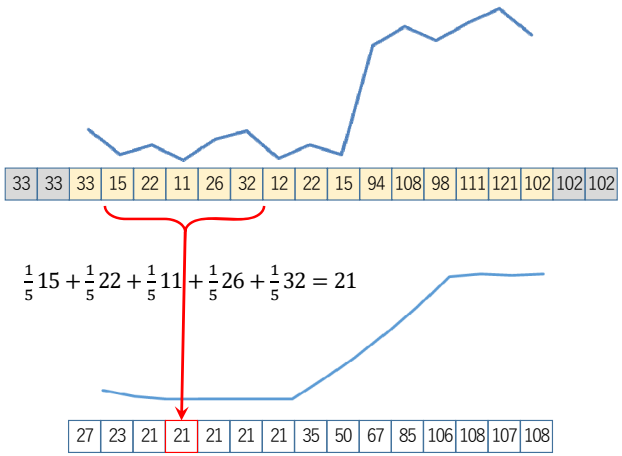
※端ははみ出すので値をコピー（ほかの方法もある）

平滑化したい!

1/3	1/3	1/3

周囲 3 ピクセル
の平均を取る

線形フィルタの例 1D

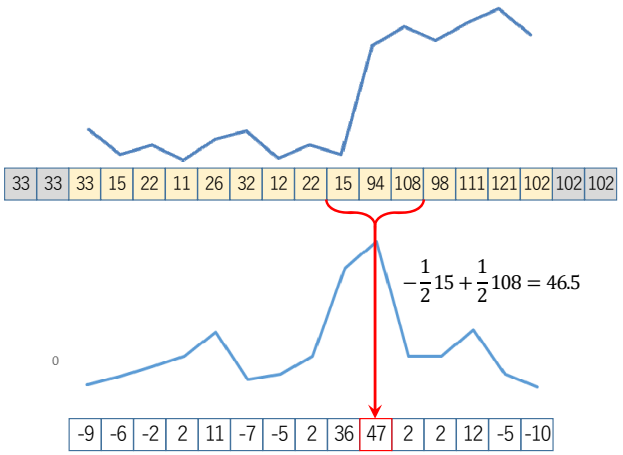


もっと
平滑化したい!

1/5	1/5	1/5	1/5	1/5

周囲5ピクセル
の平均を取る

線形フィルタの例 1D



※端ははみ出すので値をコピー（ほかの方法もある）

エッジ
（変化の大きい部分）
を検出したい

-0.5	0	0.5

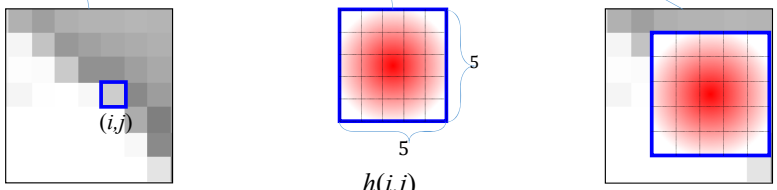
右と左のピクセルの
差をとる

線形フィルタとは

出力画素値を，入力画像の周囲画素の重み付和で計算する

$$I'(i,j)=\sum_{m=-2}^{2}\sum_{n=-2}^{2}h(m,n)\ I(i+m,j+n)$$

サイズ 5x5
の例



$I'(i,j)$
出力画像

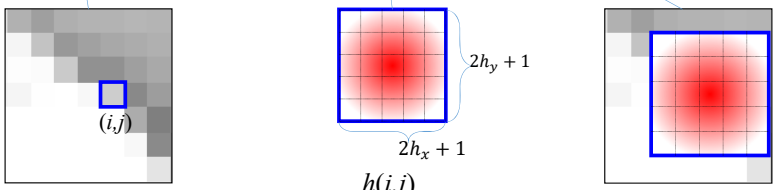
$h(i,j)$
5x5 フィルタ
各画素に重みが入っている

$I(i,j)$
入力画像

線形フィルタとは

出力画素値を，入力画像の周囲画素の重み付和で計算する

$$I'(i,j)=\sum_{m=-h_y}^{h_y}\sum_{n=-h_x}^{h_x}h(m,n)\ I(i+m,j+n)$$



$I'(i,j)$
出力画像

$h(i,j)$
フィルタ
各画素に重みが入っている

$I(i,j)$
入力画像

線形フィルタ：平滑化

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

1/25	1/25	1/25	1/25	1/25
1/25	1/25	1/25	1/25	1/25
1/25	1/25	1/25	1/25	1/25
1/25	1/25	1/25	1/25	1/25
1/25	1/25	1/25	1/25	1/25



LinaerFilter.exe (C++)
convolution1.py (python)
Process>Filters>Convolve (ImageJ)

線形フィルタ：ガウシアンフィルタ

係数をガウス分布に近づけ
中央ほど強い重みに

1/16	2/16	1/16
2/16	4/16	2/16
1/16	2/16	1/16

$\frac{1}{256}$

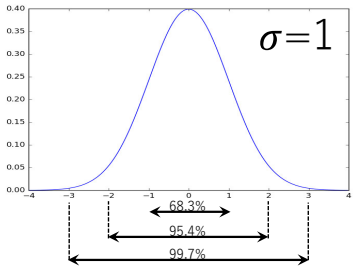
1	4	6	4	1
4	16	24	16	4
6	24	36	24	6
4	16	24	16	4
1	4	6	4	1



線形フィルタ：ガウシアンフィルタ

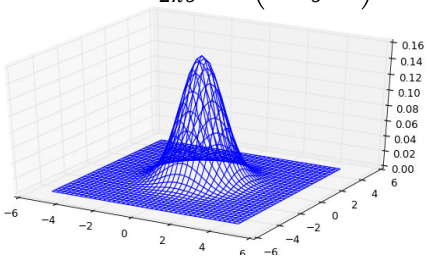
ガウス関数1D

$$g_\sigma(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{x^2}{\sigma^2}\right)$$

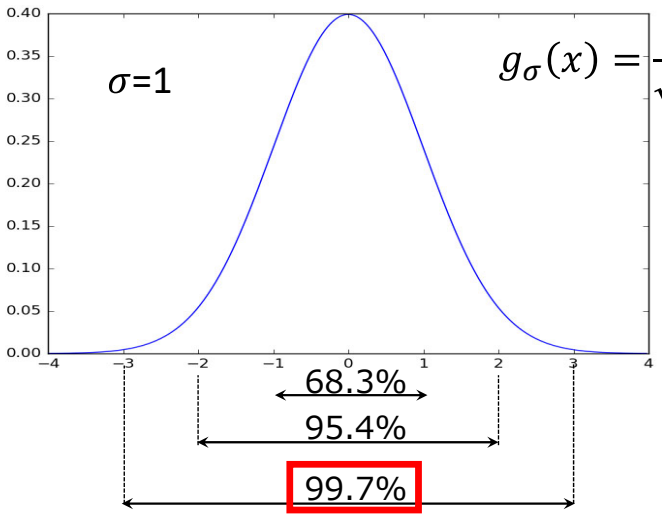


ガウス関数2D

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



これを重みにして線形フィルタをしたい
さすがに3x3は精度が悪くない？？

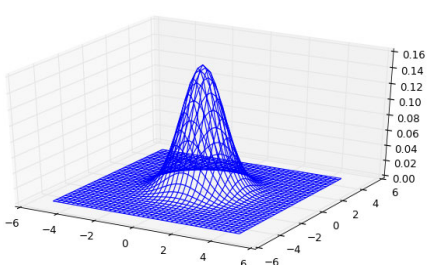


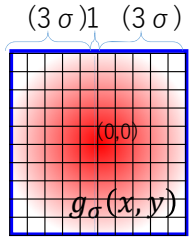
$$g_\sigma(x) = \frac{1}{\sqrt{2\pi\sigma^2}} exp\left(-\frac{x^2}{\sigma^2}\right)$$

線形フィルタ：ガウシアンフィルタ

課題によっては『3×3』や『5×5』の窓では精度が悪い

→精度を出すには窓の半径を 3 σ程度にすべき
（計算時間はかかる）





例）σ = 5 pixelの
ガウシアンフィルタ
↓
サイズは31×31が適当

復習：微分 と 勾配

関数 $f(x,y)$ のx軸, y軸方向の偏微分は以下の通り定義される

$$\frac{\partial f(x,y)}{\partial x} = \lim_{h\to 0} \frac{f(x+h,y)-f(x-h,y)}{2h}$$

$$\frac{\partial f(x,y)}{\partial y} = \lim_{h\to 0} \frac{f(x,y+h)-f(x,y-h)}{2h}$$

※$\frac{\partial f(x,y)}{\partial x}$ は、x軸方向に微小量動いた際のfの変化量

$f(x,y)$の勾配 $\nabla f(x,y)$は2次元ベクトルであり，

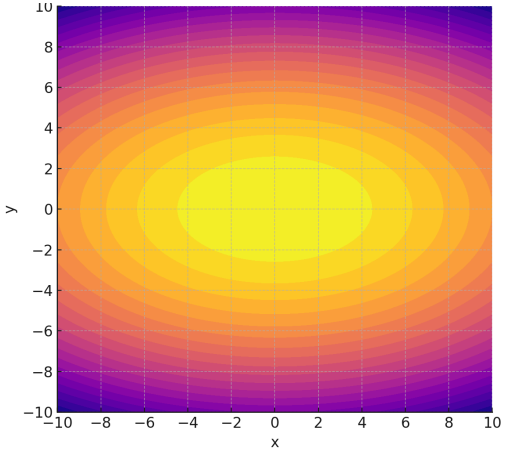
$$\nabla f(x,y) = \begin{pmatrix} \frac{\partial f(x,y)}{\partial x} \\ \frac{\partial f(x,y)}{\partial y} \end{pmatrix}$$

点(x,y)において$f(x,y)$の増加が一番大きくなる方向を示す

$$f(x,y) = -x^2 - 3\,y^2$$

上記の関数の(2,-2), (6,-2)
における勾配を計算し、
さらに図示せよ

復習 : 微分と勾配



$$f(x,y) = -x^2 - 3\,y^2$$

上記の関数の(2,-2), (6,-2)
における勾配を計算し、
さらに図示せよ

線形フィルタ：エッジ検出（**微分**）

2 次元関数 $f(x,y)$ のx方向偏微分

$$f_x = \frac{\partial f(x,y)}{\partial x} = \lim_{h\to 0}\frac{f(x+h,y)-f(x,y)}{h}$$
$$= \lim_{h\to 0}\frac{f(x,y)-f(x-h,y)}{h}$$
$$= \lim_{h\to 0}\frac{f(x+h,y)-f(x-h,y)}{2h}$$

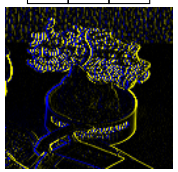
※h と近似することで画像の偏微分が得られる

画像 $I(i,j)$ の横方向偏微分 (近似)

$$I_j(i,j) \approx I(i,j+1) - I(i,j) \quad \ldots(a)$$
$$\approx I(i,j) - I(i,j-1) \quad \ldots(b)$$
$$\approx \frac{I(i,j+1)-I(i,j-1)}{2} \quad \ldots(c)$$

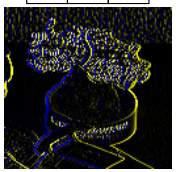
※ 正値:黄色 ,
負値:青 で可視化

0	0	0
0	-1	1
0	0	0



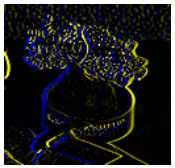
(a)

0	0	0
-1	1	0
0	0	0



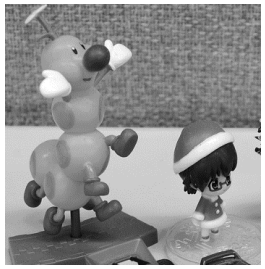
(b)

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

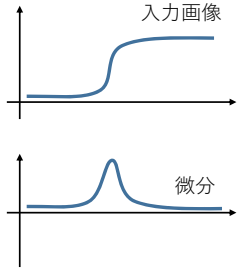


(c)

線形フィルタ：エッジ検出(微分)



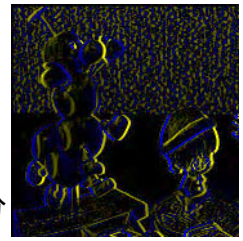
$I(i,j)$ 入力画像



微分フィルタには画像のエッジ
で強く応答する

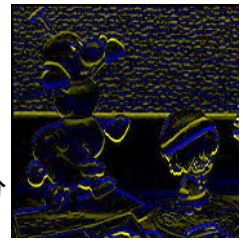
0	0	0
-1/2	0	1/2
0	0	0

$I_j(i,j)$
横方向微分



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

$I_i(i,j)$
縦方向微分



線形フィルタ :エッジ検出（**微分**）

- 前述の単純なフィルタはノイズにも鋭敏に反応する
- ノイズを押さえつつエッジを検出するフィルタが必要

横方向微分 ： 横方向微分 し 縦方向平滑化 する
縦方向微分 ： 縦方向微分 し 横方向平滑化 する

Prewitt filter

-1	0	1
-1	0	1
-1	0	1

-1	-1	-1
0	0	0
1	1	1

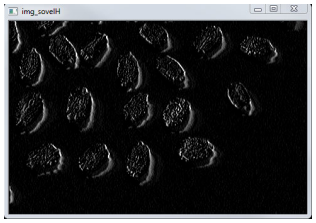
Sobel filter

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

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

元画像

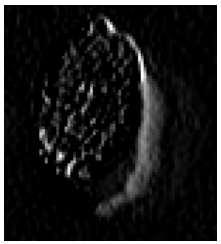
-1	0	1
-2	0	2
-1	0	1



0	0	0
-4	0	4
0	0	0

微分フィルタの正値を可視化

Sobelフィルタではノイズが
削減されているのが分かる



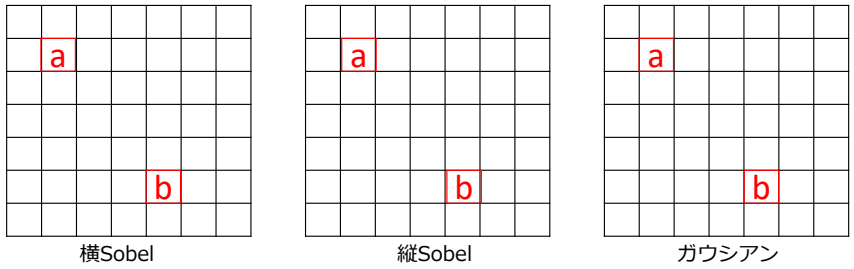
練習: フィルタ処理

- 右の7x7 画像に対して…
  1. 横方向Sobelフィルタを適用せよ
  2. 縦方向Sobelフィルタを適用せよ
  3. ガウシアンフィルタを適用せよ

※ a, bの画素のみ回答せよ

4	4	4	1	2	3	3
4	4	4	1	2	3	3
4	4	4	1	2	3	3
4	4	4	1	2	3	3
4	4	4	1	2	3	3
4	4	4	1	2	3	3
4	4	4	1	2	3	3

入力画像



横Sobel　縦Sobel　ガウシアン

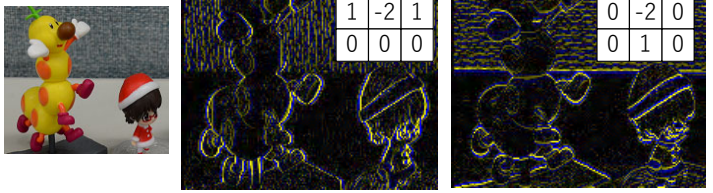
線形フィルタ：2階微分

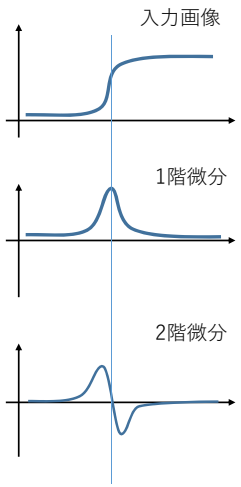
関数 $f(x,y)$ の2階偏微分は, 以下の通り定義される

$$f_{xx} = \frac{\partial^2 f(x,y)}{\partial x^2} = \lim_{h\to 0} \frac{f(x+h,y) - 2f(x,y) + f(x-h,y)}{h^2}$$

画像 $I(i,j)$ の2階偏微分の近似は…

$$I_{jj} = I(i,j+1) - 2I(i,j) + I(i,j-1)$$





線形フィルタ：ラプラシアンフィルタ

関数 $f(x,y)$ のラプラシアン

$$\Delta f(x,y) = \frac{\partial^2 f(x,y)}{\partial x^2} + \frac{\partial^2 f(x,y)}{\partial y^2}$$

画像 $I(i,j)$ のラプラシアン

$$I(i,j) = I_{ii}(i,j) + I_{jj}(i,j)$$

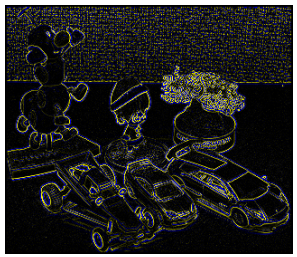
$\Delta I(i,j)$

$$\Delta I(i,j) = \begin{bmatrix} 0 & 1 & 0 \\ 0 & -2 & 0 \\ 0 & 1 & 0 \end{bmatrix} * \text{[image]} + \begin{bmatrix} 0 & 0 & 0 \\ 1 & -2 & 1 \\ 0 & 0 & 0 \end{bmatrix} * \text{[image]}$$

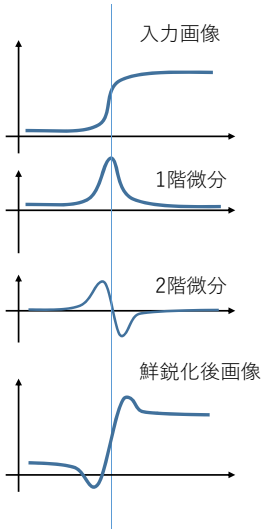
$$= \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} * \text{[image]}$$

ラプラシアンフィルタ

『＊』はフィルタ処理



方向に依存しないエッジが一度で得られる
エッジをまたぎ正負の対が現れる
白→黒 なら [0-+0]が現れる



線形フィルタ：鮮鋭化フィルタ

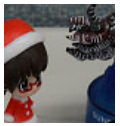
2回微分に関するラプラシアンフィルタを改良すると
画像のエッジを強調する鮮鋭化フィルタが設計できる

	?	

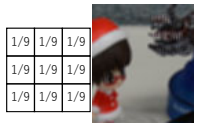
まとめ: 線形フィルタ

出力画素値を周囲画素の重み付和で計算するフィルタ

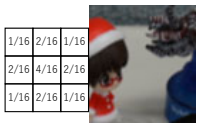
$$I'(i,j) = \sum_{m=-h_y}^{h_y} \sum_{n=-h_x}^{h_x} h(m,n)\ I(i+m, j+n)$$



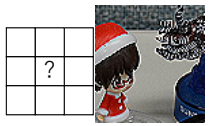
平滑化フィルタ

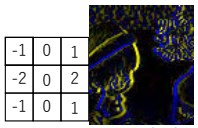


ガウシアンフィルタ

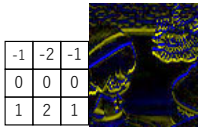


鮮鋭化フィルタ

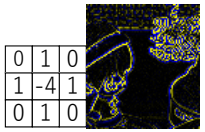




Sobelフィルタ(横)



Sobelフィルタ(縦)



ラプラシアンフィルタ