Q. 21 - 30

Q.21. ヒストグラム正規化

Histogram normalization

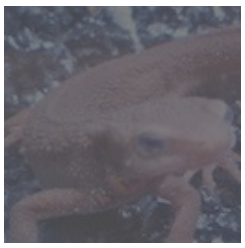
ヒストグラム正規化を実装せよ。

ヒストグラムは偏りを持っていることが伺える。 例えば、0に近い画素が多ければ画像は全体的に暗く、255に近い画素が多ければ画像は明るくなる。 ヒストグラムが局所的に偏っていることを**ダイナミックレンジが狭い**などと表現する。 そのため画像を人の目に見やすくするために、ヒストグラムを正規化したり平坦化したりなどの処理が必要である。

このヒストグラム正規化は**濃度階調変換(gray-scale transformation)** と呼ばれ、[c,d]の画素値を持つ画像を[a,b]のレンジに変換する場合は次式で実現できる。 今回は*imori_dark.jpg*を[0, 255]のレンジにそれぞれ変換する。

$$x_{out} = \begin{cases} a & if\ x_{in} < c \\ \frac{b-a}{d-c}(x_{in} - c) + a & else\ if\ c \leq x_{in} < d \\ b & else \end{cases}$$

入力 (imori_dark.jpg)	出力 (answers_image/answer_21_1.jpg)	ヒストグラム (answers_image/answer_21_2.png)



答え

- Python >> answers_py/answer_21.py
- C++ >> answers_cpp/answer_21.cpp

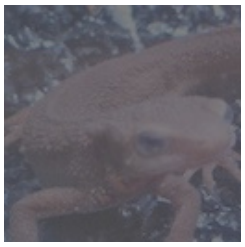
Q.22. ヒストグラム操作

ヒストグラムの平均値をm0=128、標準偏差をs0=52になるように操作せよ。

これはヒストグラムのダイナミックレンジを変更するのではなく、ヒストグラムを平坦に変更する操作である。

平均値m、標準偏差s、のヒストグラムを平均値m0, 標準偏差s0に変更するには、次式によって変換する。

$$x_{out} = \frac{s_0}{s}(x_{in} - m) + m_0$$

入力 (imori_dark.jpg)	出力 (answers_image/answer_22_1.jpg)	ヒストグラム (answers_image/answer_22_2.png)
		

答え

- Python >> answers_py/answer_22.py
- C++ >> answers_cpp/answer_22.cpp

Q.23. ヒストグラム平坦化

Histogram equalization

ヒストグラム平坦化を実装せよ。

ヒストグラム平坦化とはヒストグラムを平坦に変更する操作であり、上記の平均値や標準偏差などを必要とせず、ヒストグラム値を均衡にする操作である。

これは次式で定義される。 ただし、S ... 画素値の総数、Zmax ... 画素値の最大値、h(z) ... 濃度zの度数

$$Z' = \frac{Z_{max}}{S} \sum_{i=0}^{z} h(i)$$

入力 (imori.jpg)	出力 (answers_image/answer_23_1.jpg)	ヒストグラム (answers_image/answer_23_2.png)
		

答え

- Python >> answers_py/answer_23.py
- C++ >> answers_cpp/answer_23.cpp

Q.24. ガンマ補正

Gamma correction

*imori_gamma.jpg*に対してガンマ補正(c=1, g=2.2)を実行せよ。

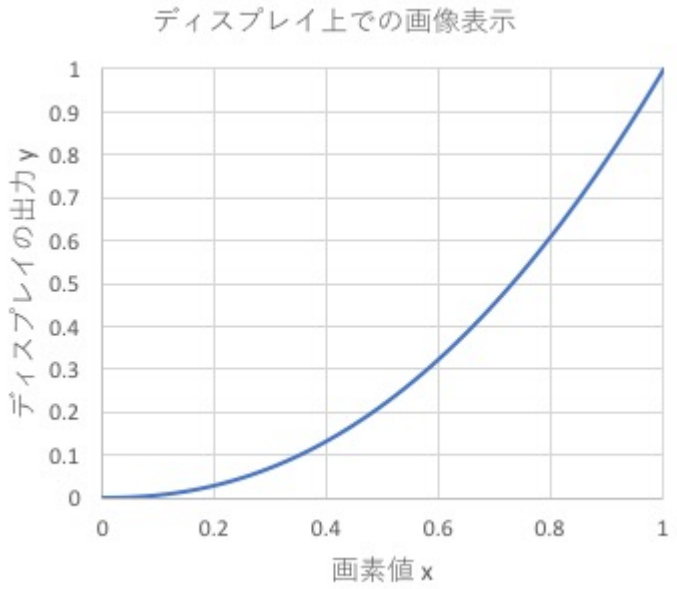
ガンマ補正とは、カメラなどの媒体の経由によって画素値が非線形的に変換された場合の補正である。 ディスプレイなどで画像をそのまま表示すると画面が暗くなってしまうため、RGBの値を予め大きくすることで、ディスプレイの特性を排除した画像表示を行うことがガンマ補正の目的である。

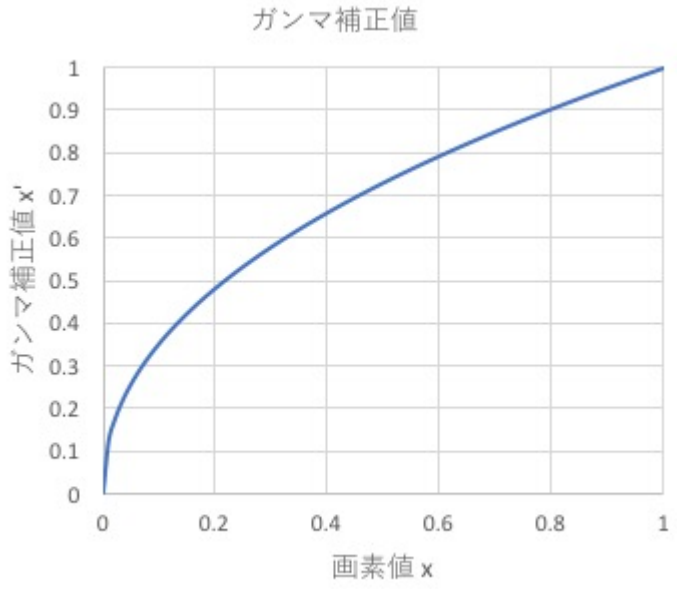
非線形変換は次式で起こるとされる。 ただしxは[0,1]に正規化されている。 c ... 定数、g ... ガンマ特性(通常は2.2)

$$x' = cI_{in}^{g}$$

そこで、ガンマ補正は次式で行われる。

$$I_{out} = (\frac{1}{c}I_{in})^{\frac{1}{g}}$$





入力 (imori_gamma.jpg)　出力 (answers_image/answer_24.jpg)

入力 (imori_gamma.jpg)	出力 (answers_image/answer_24.jpg)

答え

- Python >> answers_py/answer_24.py
- C++ >> answers_cpp/answer_24.cpp

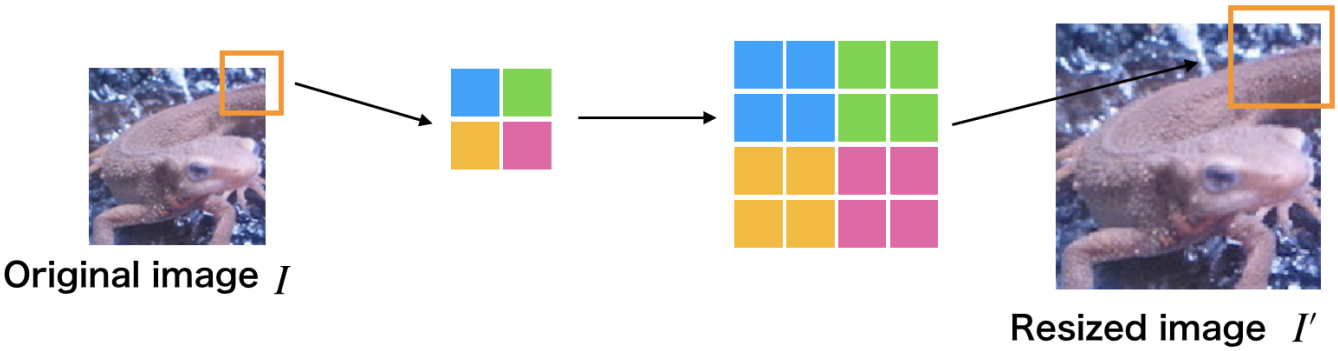
Q.25. 最近傍補間

最近傍補間により画像を1.5倍に拡大せよ。

最近傍補間(Nearest Neighbor)は画像の拡大時に最近傍にある画素をそのまま使う手法である。 シンプルで処理速度が速いが、画質の劣化は著しい。

次式で補間される。 □ ... 四捨五入

$$I'(x, y) = I(round(\frac{x}{ax}), round(\frac{y}{ay}))$$



入力 (imori.jpg)	出力 (answers_image/answer_25.jpg)

答え

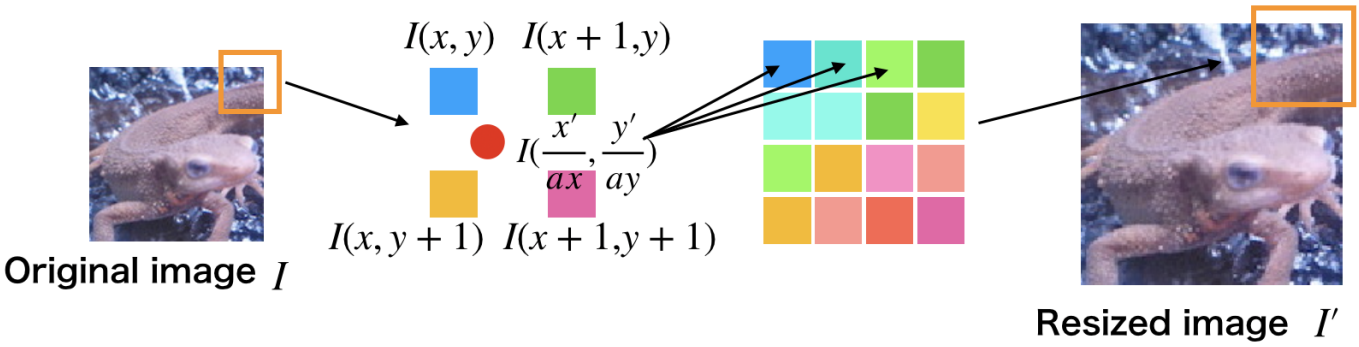
- Python >> answers_py/answer_25.py
- C++ >> answers_cpp/answer_25.cpp

Q.26. Bi-linear補間

Bi-linear補間により画像を1.5倍に拡大せよ。

Bi-linear補間とは周辺の４画素に距離に応じた重みをつけることで補完する手法である。 計算量が多いだけ処理時間がかかるが、画質の劣化を抑えることができる。

1. 拡大画像の座標(x', y')を拡大率aで割り、floor(x'/a, y'/a)を求める。
2. 元画像の(x'/a, y'/a)の周囲4画素、I(x,y), I(x+1,y), I(x,y+1), I(x+1, y+1)を求める



3. それぞれの画素と(x'/a, y'/a)との距離dを求め、重み付けする。 w = d / Sum d
4. 次式によって拡大画像の画素(x',y')を求める。 dx = x'/a - x , dy = y'/a - y

$$I'(x', y') = (1 - dx)(1 - dy)\, I(x, y) + dx\,(1 - dy)\, I(x + 1, y) + (1 - dx)\, dy\, I(x, y + 1) + dx\, dy\, I(x + 1, y + 1)$$

入力 (imori.jpg)	出力 (answers_image/answer_26.jpg)
	

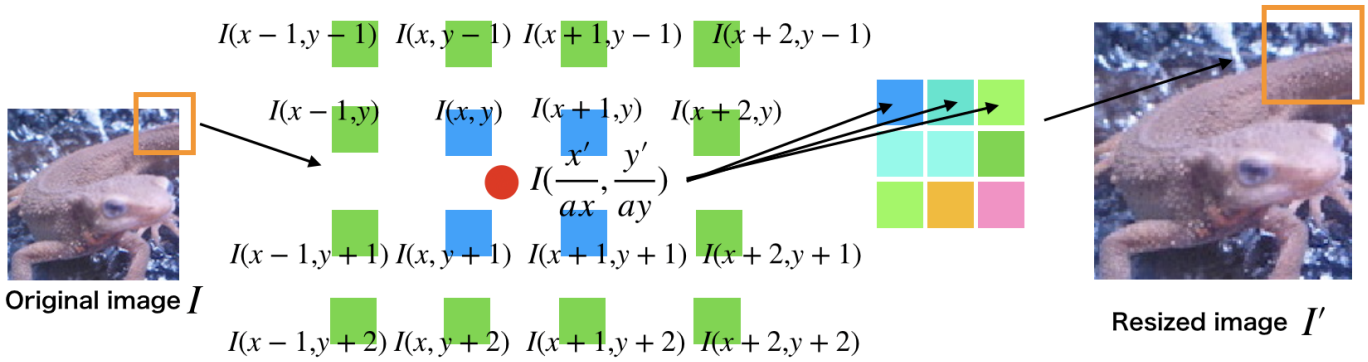
答え

- Python >> answers_py/answer_26.py
- C++ >> answers_cpp/answer_26.cpp

Q.27. Bi-cubic補間

Bi-cubic補間により画像を1.5倍に拡大せよ。

Bi-cubic補間とはBi-linear補間の拡張であり、周辺の16画素から補間を行う。



それぞれの画素との距離は次式の様に決定される。

$$d_{x1} = \left|\frac{x'}{ax} - (x-1)\right| \quad d_{x2} = \left|\frac{x'}{ax} - x\right| \quad d_{x3} = \left|\frac{x'}{ax} - (x+1)\right| \quad d_{x4} = \left|\frac{x'}{ax} - (x+2)\right|$$

$$d_{y1} = \left|\frac{y'}{ay} - (y-1)\right| \quad d_{y2} = \left|\frac{y'}{ay} - y\right| \quad d_{y3} = \left|\frac{y'}{ay} - (y+1)\right| \quad d_{y4} = \left|\frac{y'}{ay} - (y+2)\right|$$

重みは距離によって次の関数により決定される。 a は多くの場合-1をとる。だいたい図の青色のピクセルは距離|t| <=1、緑色が1<|t|<=2の重みとなる。

$$h(t) = \begin{cases} (a+2)|t|^3 - (a+3)|t|^2 + 1 & when\ |t| \le 1 \\ a|t|^3 - 5a|t|^2 + 8a|t| - 4a & when\ 1 < |t| \le 2 \\ 0 & else \end{cases}$$

これら画素と重みを用いて、次式で拡大画像の画素が計算される。 それぞれの画素と重みを掛けた和を重みの和で割る。

$$I'(x',y') = \frac{1}{\sum_{j=1}^{4}\sum_{i=1}^{4} h(d_{xi})h(d_{yj})} \sum_{j=1}^{4}\sum_{i=1}^{4} I(x+i-2,y+j-2)h(d_{xi})h(d_{yi})$$

入力 (imori.jpg) **出力 (answers_image/answer_27.jpg)**

答え

- Python >> answers_py/answer_27.py
- C++ >> answers_cpp/answer_27.cpp

Q.28. アフィン変換(平行移動)

アフィン変換を利用して画像をx方向に+30、y方向に-30だけ平行移動させよ。

アフィン変換とは3x3の行列を用いて画像の変換を行う操作である。

変換は(1)平行移動(Q.28) (2)拡大縮小(Q.29) (3)回転(Q.30) (4)スキュー(Q.31) がある。

元画像を(x,y)、変換後の画像を(x',y')とする。 画像の拡大縮小は、次式で表される。

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$

一方、平行移動は次式となる。

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} tx \\ ty \end{pmatrix}$$

以上を一つの式にまとめると、次式になり、これがアフィン変換である。

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} a & b & tx \\ c & d & ty \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

しかし実装する時は、元画像に対して１ピクセルずつ行うと、処理後の画像で値が割り当てられない可能性がでてきてしまう。よって、処理後画像の各ピクセルに対してAffine変換の逆変換を行い、値をあ割り当てる元画像の座標を取得する必要がある。Affine変換の逆操作は次式となる。

$$\begin{pmatrix} x \\ y \end{pmatrix} = \frac{1}{ad - bc} \begin{pmatrix} d & -b \\ -c & a \end{pmatrix} \begin{pmatrix} x' \\ y' \end{pmatrix} - \begin{pmatrix} tx \\ ty \end{pmatrix}$$

今回の平行移動では次式を用いる。tx, tyが平行移動のピクセルの移動距離となる。

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & tx \\ 0 & 1 & ty \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

入力 (imori.jpg)	出力 (answers_image/answer_28.jpg)
	

答え

- Python >> answers_py/answer_28.py

- C++ >> answers_cpp/answer_28.cpp

Q.29. アフィン変換(拡大縮小)

アフィン変換を用いて、(1)x方向に1.3倍、y方向に0.8倍にリサイズせよ。

また、(2) (1)の条件に加えて、x方向に+30、y方向に-30だけ平行移動を同時に実現せよ。

入力 (imori.jpg)	出力 (1) (answers_image/answer_29_1.jpg)	出力 (2) (answers_image/answer_29_2.jpg)
		

答え

- Python >> answers_py/answer_29.py
- C++ >> answers_cpp/answer_29.cpp

Q.30. アフィン変換(回転)

(1)アフィン変換を用いて、反時計方向に30度回転させよ。

(2) アフィン変換を用いて、反時計方向に30度回転した画像で中心座標を固定することで、なるべく黒い領域がなくなるように画像を作成せよ。 （ただし、単純なアフィン変換を行うと画像が切れてしまうので、工夫を要する。）

アフィン変換において、反時計方向にA度回転させる時は、次式となる。

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} cos(A) & -sin(A) & tx \\ sin(A) & cos(A) & ty \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

入力 (imori.jpg)	出力 (1) (answers_image/answer_30_1.jpg)	出力 (2) (answers_image/answer_30_2.jpg)
		

答え >>

答え

- Python >> answers_py/answer_30.py ,
- C++ >> answers_cpp/answer_30.cpp