

问题二十一至问题三十

问题二十一：直方图归一化（Histogram Normalization）

关于直方图的几个操作的中文翻译一直都十分混乱（成龙抓头.jpg）。下面是我查了资料做的一个对照表，仅供参考：

中文	English	日本語	具体作用
直方图匹配 (规定化)	Histogram Matching (Specification)	ヒストグラムマッチング	将两张图像的累积分布函数调为一样
直方图均衡化	Histogram Equalization	ヒストグラム均等化 (平坦化)	拉开灰度差别，增强对比度
直方图归一化	Histogram Normalization	ヒストグラム正規化	将直方图所有分量限制在一定范围

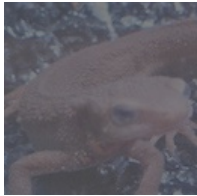

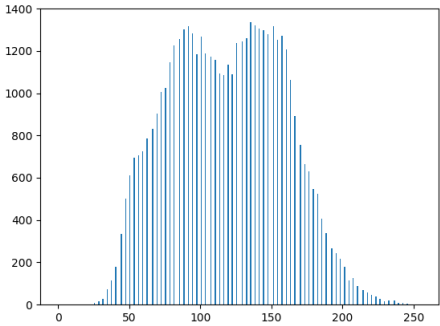
——gZR

来归一化直方图吧！

有时直方图会存在偏差。比如说，数据集中在0处（左侧）的图像全体会偏暗，数据集中在255处（右侧）的图像会偏亮。如果直方图有所偏向，那么其动态范围（dynamic range）就会较低。为了使人们能更清楚地看见图片，让直方图归一化、平坦化是十分必要的。

这种归一化直方图的操作被称作灰度变换（Grayscale Transformation）。像素点取值范围从 $[c, d]$ 转换到 $[a, b]$ 的过程由下式定义。这回我们将 `imori_dark.jpg` 的灰度扩展到 $[0, 255]$ 范围：

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

输入 (imori_dark.jpg)	输出 (answers/answer_21_1.jpg)	直方图(answers_image/answer_21_2.png)
		

答案

- Python >> [answers_py/answer_21.py](#)
- C++ >> [answers_cpp/answer_21.cpp](#)

问题二十二：直方图操作

让直方图的平均值 $m_0 = 128$ ，标准差 $s_0 = 52$ 吧！

这里并不是变更直方图的动态范围，而是让直方图变得平坦。

可以使用下式将平均值为 m 标准差为 s 的直方图变成平均值为 m_0 标准差为 s_0 的直方图：

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

输入 (imori_dark.jpg)	输出 (answers_image/answer_22_1.jpg)	直方图(answers_image//answer_22_2.png)
		

答案

- Python >> [answers_py/answer_22.py](#)
- C++ >> [answers_cpp/answer_22.cpp](#)

问题二十三：直方图均衡化（Histogram Equalization）

来让均匀化直方图吧！

直方图均衡化是使直方图变得平坦的操作，是不需要计算上面的问题中的平均值、标准差等数据使直方图的值变得均衡的操作。

均衡化操作由以下式子定义。 S 是总的像素数； Z_{max} 是像素点的最大取值（在这里是255）； $h(z)$ 表示取值为 z 的累积分布函数：

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

输入 (imori.jpg)	输出 (answers/answer_23_1.jpg)	直方图(answers_image/answer_23_2.png)
		

答案

- Python >> [answers_py/answer_23.py](#)
- C++ >> [answers_cpp/answer_23.cpp](#)

问题二十四：伽玛校正 (Gamma Correction)

[这里](#)是一篇写伽马校正比较好的文章，我觉得可以作为背景知识补充。

——gZR

对 `imori_gamma.jpg` 进行伽马校正 ($c = 1, g = 2.2$) 吧!

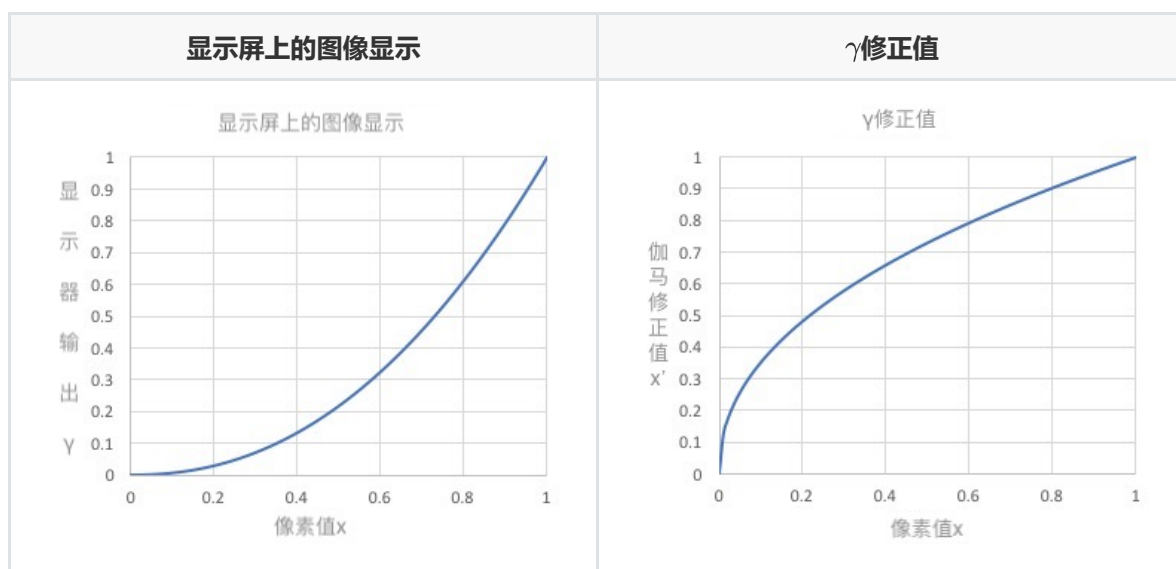
伽马校正用来对照相机等电子设备传感器的非线性光电转换特性进行校正。如果图像原样显示在显示器等上，画面就会显得很暗。伽马校正通过预先增大 RGB 的值来排除显示器的影响，达到对图像修正的目的。



由于下式引起非线性变换，在该式中， x 被归一化，限定在 $[0, 1]$ 范围内。 c 是常数， g 为伽马变量（通常取 2.2）：

$$x' = c \cdot I_{in}^g$$

因此，使用下面的式子进行伽马校正：

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



输入 (imori_gamma.jpg)	输出 (answers_image/answer_24.jpg)
	

答案

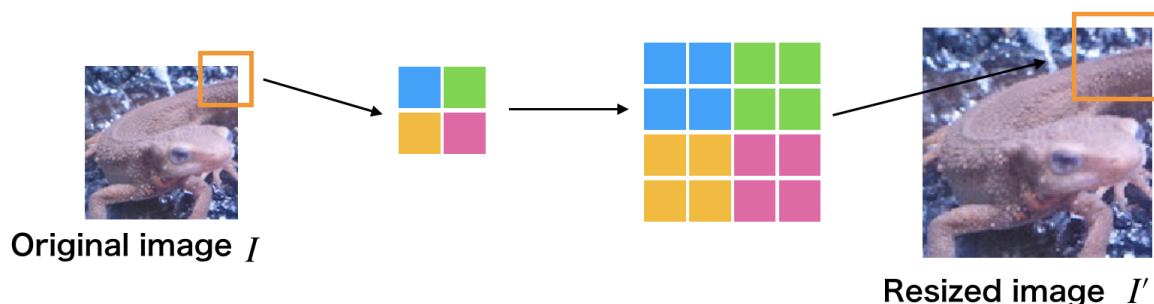
- Python >> [answers_py/answer_24.py](#)
- C++ >> [answers_cpp/answer_24.cpp](#)

问题二十五：最邻近插值（Nearest-neighbor Interpolation）

使用最邻近插值将图像放大1.5倍吧！

最近邻插值在图像放大时补充的像素取最临近的像素的值。由于方法简单，所以处理速度很快，但是放大图像画质劣化明显。

使用下面的公式放大图像吧！ I' 为放大后图像， I 为放大前图像， a 为放大率，方括号是四舍五入取证操作：



$$I'(x, y) = I\left(\left[\frac{x}{a}\right], \left[\frac{y}{a}\right]\right)$$

输入 (imori.jpg)	输出 (answers_image/answer_25.jpg)
	

答案

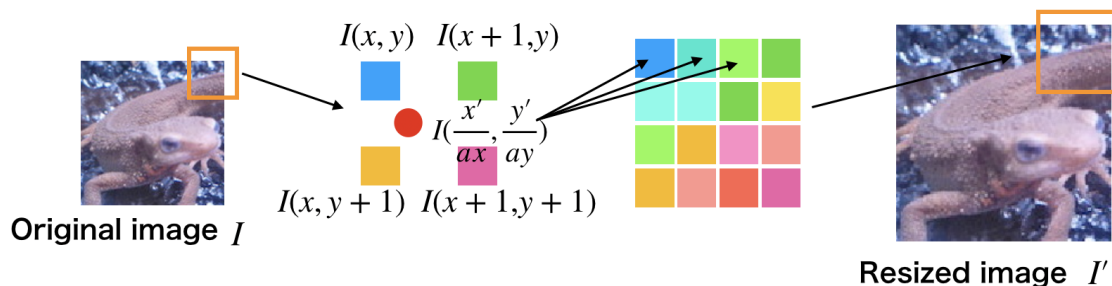
- Python >> [answers_py/answer_25.py](#)
- C++ >> [answers_cpp/answer_25.cpp](#)

问题二十六：双线性插值（Bilinear Interpolation）

使用双线性插值将图像放大1.5倍吧！

双线性插值考察4邻域的像素点，并根据距离设置权重。虽然计算量增大使得处理时间变长，但是可以有效抑制画质劣化。

1. 放大后图像的坐标 (x', y') 除以放大率 a ，可以得到对应原图像的坐标 $(\lfloor \frac{x'}{a} \rfloor, \lfloor \frac{y'}{a} \rfloor)$ 。
2. 求原图像的坐标 $(\lfloor \frac{x'}{a} \rfloor, \lfloor \frac{y'}{a} \rfloor)$ 周围4邻域的坐标 $I(x, y)$, $I(x + 1, y)$, $I(x, y + 1)$, $I(x + 1, y + 1)$ ：



3. 分别求这4个点与 $(\frac{x'}{a}, \frac{y'}{a})$ 的距离，根据距离设置权重： $w = \frac{d}{\sum d}$
4. 根据下式求得放大后图像 (x', y') 处的像素值：

$$d_x = \frac{x'}{a} - x$$

$$d_y = \frac{y'}{a} - y$$

$$I'(x', y') = (1 - d_x) \cdot (1 - d_y) \cdot I(x, y) + d_x \cdot (1 - d_y) \cdot I(x + 1, y) + (1 - d_x) \cdot d_y \cdot I(x, y + 1) + d_x \cdot d_y \cdot I(x + 1, y + 1)$$

输入 (imori.jpg)	输出 (answers_image/answer_26.jpg)
	

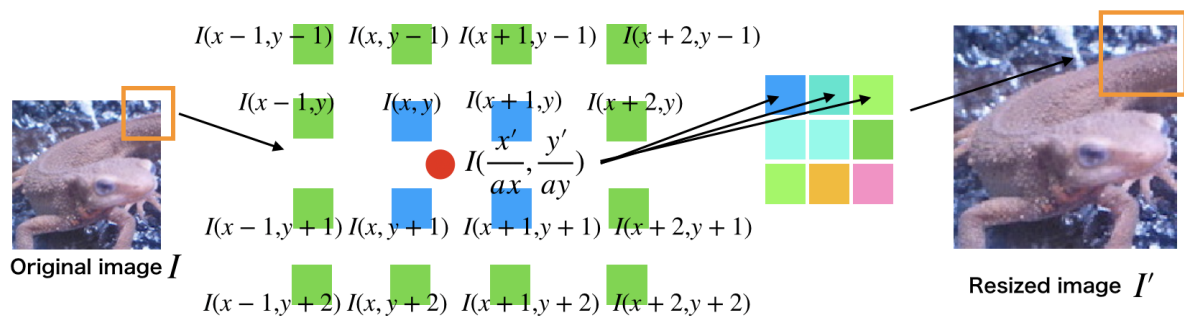
答案

- Python >> [answers_py/answer_26.py](#)
- C++ >> [answers_cpp/answer_26.cpp](#)

问题二十七：双三次插值（Bicubic Interpolation）

使用双三次插值将图像放大1.5倍吧！

双三次插值是双线性插值的扩展，使用邻域16像素进行插值。



各自像素间的距离由下式决定：

$$d_{x_1} = \left| \frac{x'}{a \cdot x} - (x-1) \right| \quad d_{x_2} = \left| \frac{x'}{a \cdot x} - x \right| \quad d_{x_3} = \left| \frac{x'}{a \cdot x} - (x+1) \right| \quad d_{x_4} = \left| \frac{x'}{a \cdot x} - (x+2) \right|$$

$$d_{y_1} = \left| \frac{x'}{a \cdot y} - (y-1) \right| \quad d_{y_2} = \left| \frac{x'}{a \cdot y} - y \right| \quad d_{y_3} = \left| \frac{x'}{a \cdot y} - (y+1) \right| \quad d_{y_4} = \left| \frac{x'}{a \cdot y} - (y+2) \right|$$

权重由基于距离的函数取得。 a 在大部分时候取 -1 。大体上说，图中蓝色像素的距离 $|t| \leq 1$ ，绿色像素的距离 $1 < |t| \leq 2$ ：

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

利用上面得到的权重，通过下面的式子扩大图像。将每个像素与权重的乘积之和除以权重的和。

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

输入 (imori.jpg)	输出 (answers_image/answer_27.jpg)

答案

- Python >> [answers_py/answer_27.py](#)
- C++ >> [answers_cpp/answer_27.cpp](#)

问题二十八：仿射变换（Affine Transformations）——平行移动

利用仿射变换让图像在 x 方向上 $+30$ ，在 y 方向上 -30 吧！

仿射变换利用 3×3 的矩阵来进行图像变换。

变换的方式有平行移动（问题二十八）、放大缩小（问题二十九）、旋转（问题三十）、倾斜（问题三十一）等。

原图像记为 (x, y) ，变换后的图像记为 (x', y') 。

图像放大缩小矩阵为下式：

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

另一方面，平行移动按照下面的式子计算：

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} t \cdot x \\ t \cdot y \end{pmatrix}$$

把上面两个式子盘成一个：

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} a & b & t \cdot x \\ c & d & t \cdot y \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

但是在实际操作的过程中，如果一个一个地计算原图像的像素的话，处理后的像素可能没有在原图像中有对应的坐标。

上面那句话原文是

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。


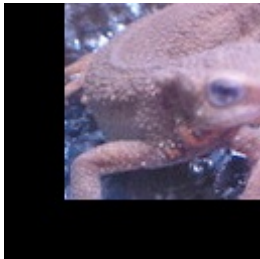
直译大概是“处理后的图像可能没有被分配到值。”我也不知道该怎么翻译才好.....你们看输出图像左下角黑色的那一块，就是这种没有被“分配”到的情况。

因此，我们有必要对处理后的图像中各个像素进行仿射变换逆变换，取得变换后图像中的像素在原图像中的坐标。仿射变换的逆变换如下：

$$\begin{pmatrix} x \\ y \end{pmatrix} = \frac{1}{a \cdot d - b \cdot c} \cdot \begin{pmatrix} d & -b \\ -c & a \end{pmatrix} \cdot \begin{pmatrix} x' \\ y' \end{pmatrix} - \begin{pmatrix} t \cdot x \\ t \cdot y \end{pmatrix}$$

这回的平行移动操作使用下面的式子计算。 t_x 和 t_y 是像素移动的距离。

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

输入 (imori.jpg)	输出 (answers_image/answer_28.jpg)
	



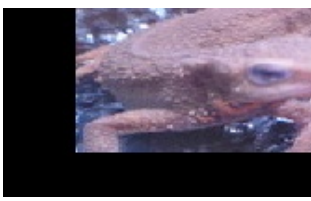
答案

- Python >> [answers_py/answer_28.py](#)
- C++ >> [answers_cpp/answer_28.cpp](#)

问题二十九：仿射变换（Affine Transformations）——放大缩小

1. 使用仿射变换，将图片在 x 方向上放大1.3倍，在 y 方向上缩小至原来的 $\frac{4}{5}$ 。

2. 在上面的条件下，同时在 x 方向上向右平移30（+30），在 y 方向上向上平移30（-30）。

输入 (imori.jpg)	输出 (1) (answers_image/answer_29_1.jpg)	输出 (2) (answers_image/answer_29_2.jpg)
		

答案

- Python >> [answers_py/answer_29.py](#)
- C++ >> [answers_cpp/answer_29.cpp](#)

问题三十：仿射变换（Affine Transformations）——旋转

1. 使用仿射变换，逆时针旋转30度。
2. 使用仿射变换，逆时针旋转30度并且能让全部图像显现（也就是说，单纯地做仿射变换会让图片边缘丢失，这一步中要让图像的边缘不丢失，需要耗费一些工夫）。

使用下面的式子进行逆时针方向旋转 A 度的仿射变换：

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} \cos(A) & -\sin(A) & t \cdot x \\ \sin(A) & \cos(A) & t \cdot y \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

输入 (imori.jpg)	输出 (1) (answers_image/answer_30_1.jpg)	输出 (2) (answers_image/answer_30_2.jpg)
		

答案

- Python >> [answers_py/answer_30.py](#) ,
- C++ >> [answers_cpp/answer_30.cpp](#)