图像处理 HW2：简易图像大小调整

李天宇　　2200013188　　信息科学技术学院

1　基本实现

OpenCV 给出的`resize()`函数定义如下：

```
cv.resize( src, dsize[, dst[, fx[, fy[, interpolation]]]] ) -> dst
```

其中，`interpolation`为插值方法，这里需要用到的是最近邻插值、双线性插值和双三次插值法。其枚举值如下：`cv.INTER_NEAREST`, `cv.INTER_LINEAR`, `cv.INTER_CUBIC`

2　基础功能实现

2.1　最近邻插值

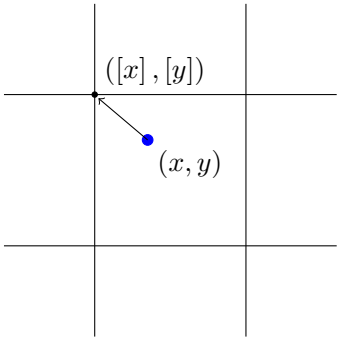


图 1: 最近邻插值

像素值映射的公式如下：

$$\begin{aligned} src_x &= round(dst_x/scale) \\ src_y &= round(dst_y/scale) \end{aligned} \tag{1}$$

需要注意的是，舍入时可能因为“入”而超出范围，所以需要对边缘数据做处理。

另外，`image.shape`的第一维是高度，第二维是宽度。

2.2 双线性插值

2.2.1 单次线性插值

如图所示：

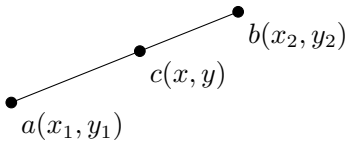


图 2: 单次线性插值

我们有：

$$\frac{f(x_1) - f(x))}{x_1 - x} = \frac{f(x_1) - f(x_2)}{x_1 - x_2} \tag{2}$$

整理得：

$$f(x) = \frac{x - x_2}{x_1 - x_2} f(x_1) + \frac{x_1 - x}{x_1 - x_2} f(x_2) \tag{3}$$

2.2.2 双线性插值法

双线性插值基于单次线性插值。

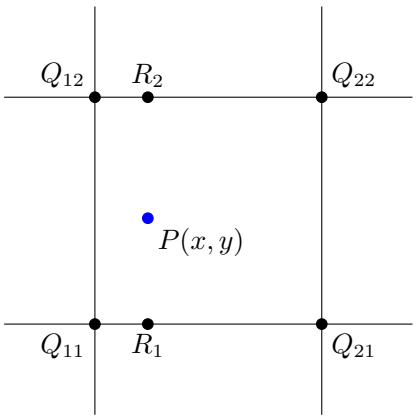


图 3: 双线性插值

要计算 $f(P)$，需要先用单次线性插值计算 $f(R_1)$，$f(R_2)$，再用 $f(R_1)$，$f(R_2)$ 计算 $f(P)$。具体计算公式如下：

$$\begin{aligned} f(R_1) &= \frac{x_2 - x}{x_2 - x_1} f(Q_{11}) + \frac{x - x_1}{x_2 - x_1} f(Q_{21}) \\ f(R_2) &= \frac{x_2 - x}{x_2 - x_1} f(Q_{12}) + \frac{x - x_1}{x_2 - x_1} f(Q_{22}) \\ f(P) &= \frac{y_2 - y}{y_2 - y_1} f(R_1) + \frac{y - y_1}{y_2 - y_1} f(R_2) \end{aligned} \tag{4}$$

整理得：

$$
\begin{aligned}
f(P) = &\frac{(x_2 - x)(y_2 - y)}{(x_2 - x_1)(y_2 - y_1)} f(Q_{11}) + \frac{(x - x_1)(y_2 - y)}{(x_2 - x_1)(y_2 - y_1)} f(Q_{21}) \\
&+ \frac{((x_2 - x)(y - y_1)}{(x_2 - x_1)(y_2 - y_1)} f(Q_{12}) + \frac{(x - x_1)(y - y_1)}{(x_2 - x_1)(y_2 - y_1)} f(Q_{22})
\end{aligned} \tag{5}
$$

这里，$(x_2 - x_1)(y_2 - y_1) = 1$，因而分母均为 1。

图3中，Q_{12} 的坐标为 $(\lfloor x \rfloor, \lfloor y \rfloor)$，其余易得，亦需注意边缘处理。

我查到的资料中提到，在 OpenCV 中，像素的映射公式有所改变：

$$
\begin{aligned}
src_x &= (dst_x + 0.5)/scale + 0.5 \\
src_y &= (dst_y + 0.5)/scale + 0.5
\end{aligned} \tag{6}
$$

这样处理没有显著变化，故保留原处理。

2.3 双三次插值

定义 $BiCubic$ 函数，用于计算某个点的权值，x 为该点到 P 点的距离：

$$
W(x) = \begin{cases}
(a+2)|x|^3 - (a+3)|x|^2 + 1, & for\ |x| \le 1 \\
a|x|^3 - 5a|x|^2 + 8a|x| - 4a, & for\ 1 < |x| \le 2 \\
0, & otherwise
\end{cases} \tag{7}
$$

取 $a = -1$。

其 $x > 0$ 部分图像如下：

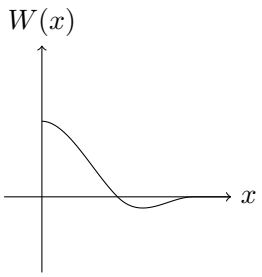


图 4: $W(x)$

以下是双三次插值需要的 16 个点：

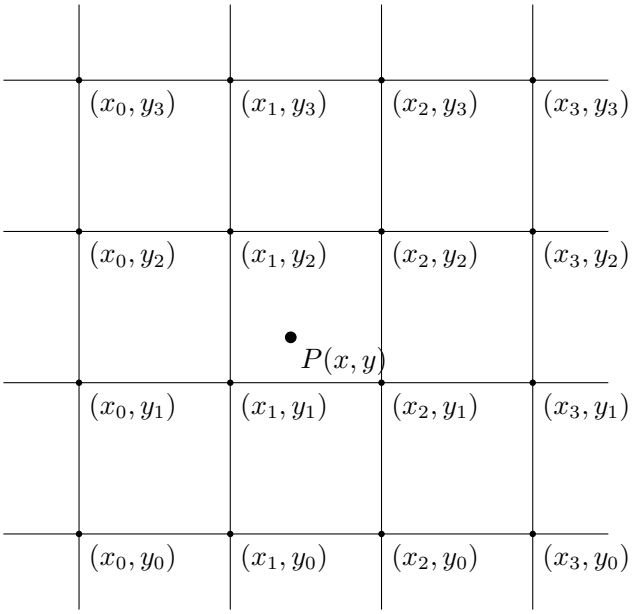


图 5: 双三次插值

如上图，P 点的值由这 16 个点加权而来，同样需要注意边缘处理。其中，$x_1 = \lfloor x \rfloor$，$y_2 = \lfloor y \rfloor$，其余点易得。

$$f(x,y) = \sum_{i=0}^{3}\sum_{j=0}^{3} f(x_i, y_j) W(x - x_i) W(y - y_j) \tag{8}$$

上述方法与使用卷积核进行卷积等价。另外，加权后得到的像素值可能超出`uint8`的表示范围，需要对其进行限制。

3 进阶功能实现

3.1 Lanczos 插值算法

Lanczos 插值算法在 OpenCV 也给出了实现，将`interpolation`设为`cv.LANCZOS4`即可。它将在 8×8 邻域上进行 Lanczos 插值。

Lanczos 插值算法基于以下的 Lanczos kernel：

$$L(x) = \begin{cases} sinc(x)sinc(x/a) & if \ -a < x < a, \\ 0 & otherwise. \end{cases} \tag{9}$$

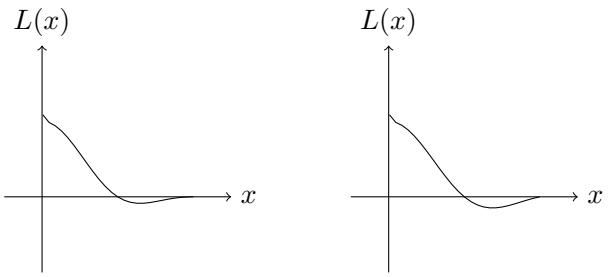
其中，$sinc(x)$ 定义如下：

$$sinc(x) = \begin{cases} \dfrac{\sin(\pi x)}{\pi x} & if \ x \neq 0, \\ 1 & if \ x = 0 \end{cases} \tag{10}$$

而 a 决定 kernel 的宽度，通常取 2 或 3。通常来讲，$a = 2$ 时，算法适用于图像缩小插值，$a = 3$ 时，算法适用于放大插值。

Lanczos kernel 在 $x > 0$ 上的图像于图6中给出。

这里的计算与双三次插值类似，见公式11。



(a) $a = 2$ (b) $a = 3$

图 6: Lanczos kernel

$$S(x) = \sum_{i=\lfloor x \rfloor - a + 1}^{\lfloor x \rfloor + a} s_i L(x - i) \tag{11}$$

3.2 差异图

由于算法的差异，我的图像与 OpenCV 给出的实现在大小上有时会出现 1 像素长的误差，所以在做差时，需要取小者的形状计算。

3.3 图像几何变换

3.3.1 仿射变换

仿射变换将矩阵 M 以如下方式应用于图像：

$$\texttt{dst}(x, y) = \texttt{src}(M_{11}x + M_{12}y + M_{13}, M_{21}x + M_{22}y + M_{23}) \tag{12}$$

不过，要先使用`invertAffineTransform`进行反转，然后在上述公式中替代 M。

3.3.2 图像旋转

想做出 360 度的旋转很麻烦，主要是要找基准点。我画了图算了坐标，但最后发现标准不一样，于是作罢。

3.4 斜切

调整仿射变换的矩阵 M 即可实现斜切，这里只做了一个方向的演示。

4 实验结果

下面是对图像处理的过程。

这次作业涉及的内容实在太多，没法做到尽善尽美，很多方面也没能完全做好。OpenCV 的很多标准也实在很令人迷惑。

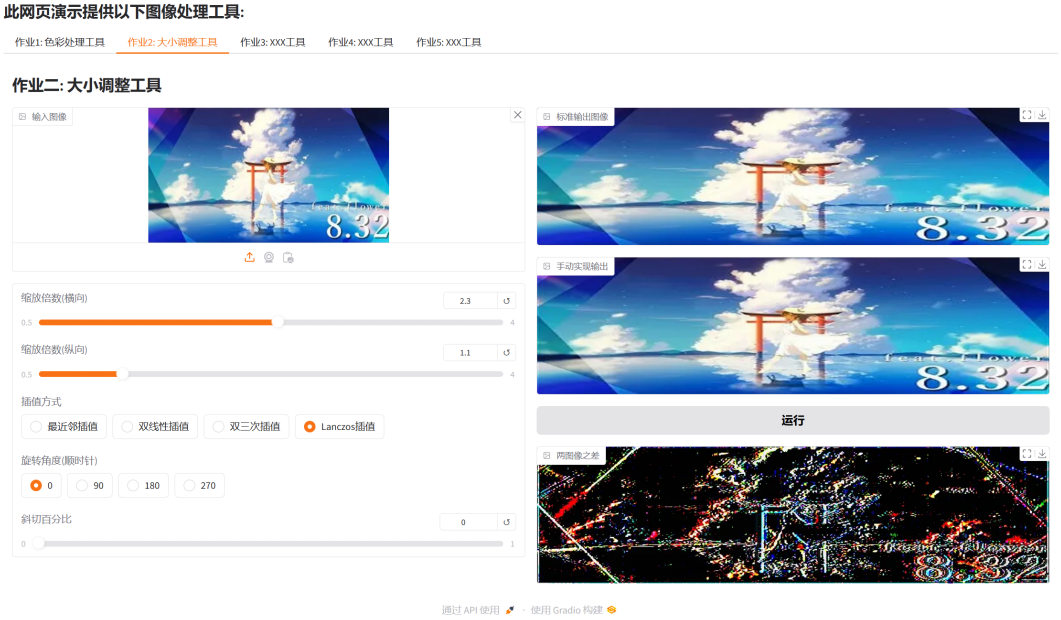


图 7: Lanczos 插值

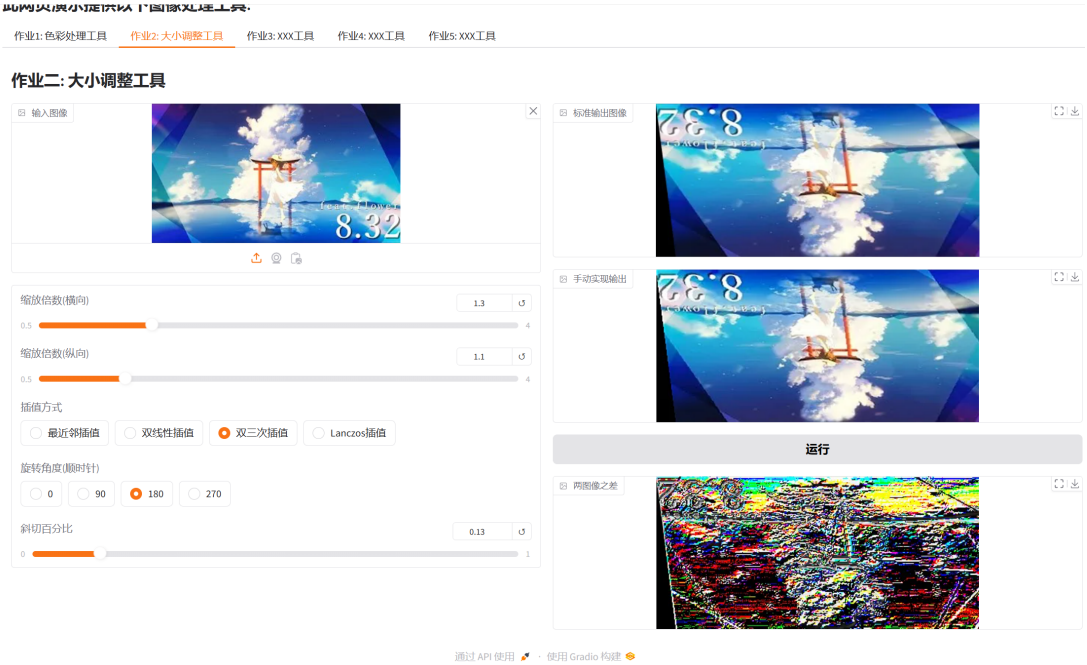


图 8: 综合测试