华中科技大学

图 像 处 理 作 业

图像插值方法

院　　系: 人工智能与自动化学院

专 业 班 级: 自动化 1903 班

学 生 姓 名: 李子奥

学 生 学 号: U201914629

指 导 教 师: 谭山

2022 年 6 月 5 日

目　　录

1　图像插值方法

图像插值算法是一种常用的算法，在图像放大、旋转等多种变换中都有用到。由于原始图像进行某种变换后产生的新图像的像素并非与原始图像的像素完全一一对应，新图像中会出现很多“空穴”，这时就需要对这些“空穴”进行填补。而插值算法就是填补的方式。图象插值方法包括最近邻插值，双线性插值，双三次内插，边缘导向插值等。本文介绍了最近邻插值、双线性插值和双三次内插，并通过实验对比这三种插值方法的性能。

1.1　图像插值方法原理

1.1.1　最近邻插值

最近邻插值是最简单的插值方法，它的主要思想是将找到最相邻的整数点，作为将最近邻点的像素值作为插值后的输出。如图1.1所示，距离 P 点最近的**整数**像素点为 $A1$，所以 P 点通过最近邻插值得到的像素值与 $A1$ 点像素值相等。最近邻插值算法思想简单，运算快速，实现方便，但是与其他方法相比效果较差。

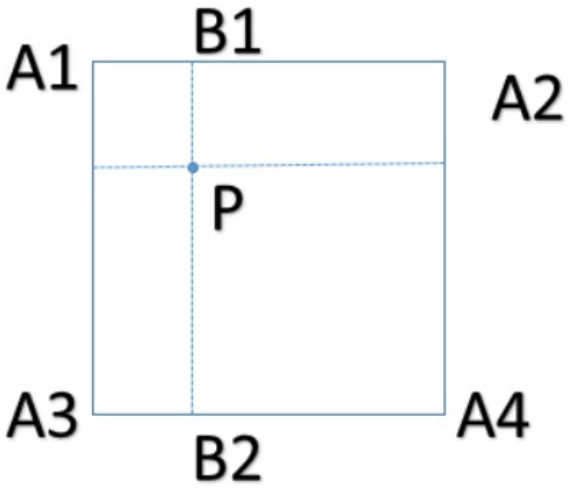

图 1.1　插值算法示意图

1.1.2 双线性插值

双线性插值算法在最近邻插值算法的基础上进行改进，P 点的像素值由距离它最近的四个**整数**点 $A1$，$A2$，$A3$，$A4$ 一同决定。双线性插值假设像素值在相邻像素间以线性的方式均匀变换，因此可以先算出 $B1$，$B2$ 点的像素值为：

$$\begin{aligned} r(B1) &= \frac{x_{B1}-x_{A1}}{x_{A2}-x_{A1}} \cdot r(A2) + \frac{x_{A2}-x_{B1}}{x_{A2}-x_{A1}} \cdot r(A1) \\ r(B2) &= \frac{x_{B2}-x_{A3}}{x_{A4}-x_{A3}} \cdot r(A4) + \frac{x_{A4}-x_{B2}}{x_{A4}-x_{A3}} \cdot r(A3) \end{aligned} \tag{1.1}$$

其中 $r(\cdot)$ 一点的像素值。通过 $B1$ 和 $B2$ 点的像素值能够得到 P 点的像素值为：

$$r(P) = \frac{y_P - y_{B1}}{y_{B2}-y_{B1}} \cdot r(B2) + \frac{y_{B2}-y_P}{y_{B2}-y_{B1}} \cdot r(B1) \tag{1.2}$$

双线性插值算法比最近邻插值算法效果好，但也带来了计算量的增加。

1.1.3 双三次内插

双三次内插进一步增加了领域内像素点的个数：P 点的像素值由距离它最近的十六个**整数**像素点一同决定，这使得双三次内插法生成的图像更加平滑。双三次内插法的插值平面表示为：

$$p(x,y) = \sum_{i=0}^{3}\sum_{j=0}^{3} a_{ij} x^i y^j \tag{1.3}$$

将所有 a_{ij} 解出，从而得到插值平面，进而得到目标点的像素值的方法被称为**双三次样条插值**。解出一个这样含有 16 个参数的方程的过程较为复杂，但通过在 x 方向与 y 方向都进行卷积可以相似的效果，这种方法被称为**双三次卷积插值**。对于双三次卷积插值，P 点的像素值表示为：

$$r(P) = \Sigma_{x_i}\Sigma_{y_j}\alpha_{x_i,y_j} r(x_i,y_j) \tag{1.4}$$

其中 $r(x_i,y_j)$ 表示 (x_i,y_j) 点的像素值，α_{x_i,y_j} 为权重系数。

$$\alpha_{x_i,y_j} = w(x_P - x_i) \times w(y_P - y_j) \tag{1.5}$$

卷积核 $w(\cdot)$ 的为：

$$w(z) = \begin{cases} (a+2)|z|^3 - (a+3)|z|^2 + 1 & |z| \le 1 \\ a|z|^3 - 5a|z|^2 + 8a|z| - 4a & 1 < |z| < 2 \\ 0 & \text{其他} \end{cases} \tag{1.6}$$

通常情况下，根据经验，当 $a = -0.5$ 插值效果较好。

1.2 图像插值方法实验

在我们测试的数据库中，共有 5 张尺寸为 512×512 的图片。对于一幅图像，首先使用下二采样（每隔一个像素取一个值），缩小为 256×256 的图片，再通过插值方法恢复成 512×512 的图像。为了比较插值方法的优劣，可以将插值后得到的图片与原图片进行逐像素的对比。在本文中，峰值信噪比被用作比较的指标：

$$\mathrm{PSNR}(x, y) = 10 \log_{10} \left(\frac{255^2}{\mathrm{MSE}(x, y)} \right) \tag{1.7}$$

图 1.2　barb 原始图像

图 1.3　barb 最近邻插值图像

图 1.4　barb 双线性插值图像

图 1.5　barb 双三次内插图像

表1.1对比 5 张通过不同插值方法还原后的图片的峰值信噪比。能够看出，在

所有图片上，最近邻插值的效果均差于双线性插值和双三次内插，而双线性插值与双三次插值的效果十分相近，但在这 5 张图片组成的数据集中，双线性插值的平均效果略好于双三次内插。

	最近邻插值	双线性插值	双三次内插
barb	22.42	**24.48**	23.64
boat	26.44	29.15	**29.20**
lena	28.69	32.18	**32.37**
mandrill	20.53	**22.67**	22.20
peppers-bw	26.96	**30.93**	30.66
平均值	25.01	**27.88**	27.61

表 1.1　图像插值方法峰值信噪比对比（我的实现）

	最近邻插值	双线性插值	双三次内插
barb	22.42	**23.91**	23.34
boat	26.44	**28.15**	28.04
lena	28.69	30.72	**30.74**
mandrill	20.53	**21.93**	21.58
peppers-bw	26.96	**28.87**	28.59
平均值	25.01	**26.72**	26.46

表 1.2　图像插值方法峰值信噪比对比（PIL 库实现）

为了确保代码的正确性与实验结果的正确性，我通过 `difftest` 的方法将我写的图像插值代码的运行结果与 `PIL` 库提供的图像插值代码的运行结果相对比，其中 `PIL` 库的运行结果如表1.2所示。

`PIL` 库的运行显示出的结论与我的代码的结论一致：即最近邻插值方法效果最差，双线性插值方法与双三次内插的效果接近，但在这 5 张图片的测试中双线性插值的效果略好。值得注意的是，在最近邻插值这一指标上，我的代码的运行结果与 `PIL` 库的运行结果完全一致；在双线性插值与双三次内插上，我的代码的运行结果与 `PIL` 库的运行结果都不相同，且我的代码的效果均**明显好于** `PIL` 库的效果。我猜测，这可能是因为：

1. **对于边界点的处理方式不同**。如双三次内插要求目标点附近有 16 个点，当目标点处于边界上时，它的邻域内没有 16 个点，因此有不同的处理方法。
2. **运算加速**。`PIL` 库的运算速度远快于我的代码，因此PIL库可能为了算法的运

行速度舍弃了一些性能。

3. **参数不同**。双三次内插需要提供一个超参数 α，当该参数不同时，插值的结果也会不同。

由于时间原因，我未对 `PIL` 库对插值方法的具体实现进行探究，如果以后有机会的话，可以尝试分析一下。

最后，本文从定性的角度分析这三种插值方式的优劣。由于篇幅原因，这里以 barb 图像为例，其余四张图像的对比放在附录B中。图1.2为 barb 原始图像的局部放大图，图1.3为 barb 图像下采样后通过最近邻插值法恢复的图像的局部放大图，1.4为双线性插值法恢复的图像的局部放大图，1.5为双三次内插法恢复的图像的局部放大图。最近邻插值还原的图片有严重的锯齿块形状；双线性插值还原出的图像对比度较小，线条较为模糊，图像色彩整体偏灰；双三次内插还原出的图像色彩与原图最为接近。但值得注意的是，这三种插值方法都没有正确恢复出原图像的条纹。

附录 A　程序代码

Listing A.1　main.py

```
from math import ceil
import os
from matplotlib.image import imsave
import matplotlib.pyplot as plt
import numpy as np
import PIL.Image as Img

L = 256
OutputDir = "output"
ALL_FILES = []
CURRENT_FILE = ""
IF_SHOW = False
Suffix = ".png"
USE_LIB = True

EnlargeEdge = 100
EnlargeArea = {'barb': [150, 220],
               'boat': [210, 300],
               'lena': [180, 200],
               'mandrill': [380, 230],
               'peppers-bw': [210, 240]}

def show_pic(pic1: np.array, str1=''):
    global CURRENT_FILE

    if CURRENT_FILE not in EnlargeArea.keys():
        raise NotImplementedError

    ea = EnlargeArea[CURRENT_FILE]
    imsave(os.path.join(OutputDir, 'E_'+str1+'_'+CURRENT_FILE+'.png'), pic1[ea[0]:ea[0]+EnlargeEdge,
        ea[1]:ea[1]+EnlargeEdge], cmap=plt.get_cmap('gray'),vmin=0,vmax=255)
    imsave(os.path.join(OutputDir, str1+'_'+CURRENT_FILE+'.png'), pic1, cmap=plt.get_cmap('gray'),vmin
        =0,vmax=255)

    if IF_SHOW:
        plt.imshow(pic1, cmap=plt.get_cmap('gray'),vmin=0,vmax=255)
        plt.xticks([])
        plt.yticks([])
        plt.title(CURRENT_FILE)
        plt.show()

def map_point(FromShape, ToShape, Point):
    return (Point / (FromShape - [1, 1]) * (ToShape - [1, 1]))

def shrink_image(pic: np.array):
    H, W = pic.shape
    assert(H % 2 == 0)
    assert(W % 2 == 0)

    H, W = int(H/2), int(W/2)
    new_pic = np.zeros((H, W), dtype=int)
    for h in range(H):
        for w in range(W):
            new_h = h*2 if h < H/2 else h*2+1
            new_w = w*2 if w < W/2 else w*2+1
            new_pic[h, w] = pic[new_h, new_w]
```

```
55
56      return new_pic
57
58  def NearestInterpolation(old_pic: np.array):
59      H, W = old_pic.shape
60      H, W = H*2, W*2
61      if USE_LIB:
62          new_pic = Img.fromarray(old_pic).resize((H, W), resample=Img.NEAREST)
63          return np.array(new_pic)
64
65      new_pic = np.zeros((H, W), dtype=np.int32)
66
67      pic_shape, new_pic_shape = np.array(old_pic.shape), np.array(new_pic.shape)
68      for h in range(H):
69          for w in range(W):
70              mapped_point = map_point(new_pic_shape, pic_shape, [h, w])
71              mapped_point = round(mapped_point[0]), round(mapped_point[1])
72              new_pic[h, w] = old_pic[mapped_point[0], mapped_point[1]]
73
74      return new_pic
75
76  def BilinearInterpolation(old_pic: np.array):
77      H, W = old_pic.shape
78      H, W = H*2, W*2
79      if USE_LIB:
80          new_pic = Img.fromarray(old_pic).resize((H, W), resample=Img.BILINEAR)
81          return np.array(new_pic)
82      new_pic = np.zeros((H, W), dtype=np.int32)
83
84      pic_shape, new_pic_shape = np.array(old_pic.shape), np.array(new_pic.shape)
85      for h in range(H):
86          for w in range(W):
87              mapped_point = map_point(new_pic_shape, pic_shape, [h, w])
88              h_map, w_map = mapped_point[0], mapped_point[1]
89              h_min, w_min = int(mapped_point[0]), int(mapped_point[1])
90              h_max, w_max = ceil(mapped_point[0]), ceil(mapped_point[1])
91
92              h_min = 0 if h_min < 0 else h_min
93              w_min = 0 if w_min < 0 else w_min
94              h_max = H-1 if h_max > H-1 else h_max
95              w_max = W-1 if w_max > W-1 else w_max
96
97              assert(h_max-h_min in [0, 1])
98              assert(w_max-w_min in [0, 1])
99
100             s1 = (h_max-h_map) / (h_max-h_min) if h_max-h_min == 1 else 1
101             s2 = (w_max-w_map) / (w_max-w_min) if w_max-w_min == 1 else 1
102
103             f1  = s1 * old_pic[h_min, w_min] + (1-s1) * old_pic[h_max, w_min]
104             f2  = (s1) * old_pic[h_min, w_max] + (1-s1) * old_pic[h_max, w_max]
105
106             f   = (s2) * f1 + (1-s2) * f2
107
108             new_pic[h, w] = f
109
110     return new_pic
111
112 def BicubicInterpolate(old_pic: np.array):
113     H, W = old_pic.shape
114     H, W = H*2, W*2
115     if USE_LIB:
116         new_pic = Img.fromarray(old_pic).resize((H, W), resample=Img.BICUBIC)
117         return np.array(new_pic)
118     new_pic = np.zeros((H, W), dtype=np.int32)
119
120     pic_shape, new_pic_shape = np.array(old_pic.shape), np.array(new_pic.shape)
121     for h in range(H):
```

```
122         for w in range(W):
123             mapped_point = map_point(new_pic_shape, pic_shape, [h, w])
124             h_map, w_map = mapped_point[0], mapped_point[1]
125             h_min, w_min = int(mapped_point[0]), int(mapped_point[1])
126             h_max, w_max = ceil(mapped_point[0]), ceil(mapped_point[1])
127 
128             assert(h_max - h_min in [0, 1])
129             assert(w_max - w_min in [0, 1])
130 
131             h_min = h_min-1 if h_max-h_min == 0 else h_min
132             w_min = w_min-1 if w_max-w_min == 0 else w_min
133 
134             total_weights = 0
135             total_value = 0
136             for i in range(h_min-1,h_max+2):
137                 for j in range(w_min-1, w_max+2):
138                     if i < 0 or j < 0 or \
139                        i >= old_pic.shape[0] or \
140                        j >= old_pic.shape[1]:
141                         continue
142                     ww = BiCubic(i-h_map) * BiCubic(j-w_map)
143                     total_weights += ww
144                     total_value += (ww * old_pic[i, j])
145 
146             new_pic[h, w] = total_value / total_weights
147 
148     return new_pic
149 
150 def BiCubic(x, alpha=-0.75):
151     if x < 0:
152         x = -x
153     if x >= 2:
154         return 0
155     elif x <= 1:
156         return (alpha+2) * np.power(x, 3) - (alpha+3)*np.power(x, 2) + 1
157     else:
158         return alpha*np.power(x, 3) - 5*alpha*np.power(x, 2) + 8*alpha*x - 4*alpha
159 
160 def get_pic_loss(pic1, pic2):
161     assert(pic1.shape == pic2.shape)
162     mse = np.mean(np.power(pic1 - pic2, 2))
163     psnr = 10 * np.log10(255 ** 2 / mse)
164     return mse, psnr
165 
166 def main():
167     global CURRENT_FILE
168 
169     _, _, ALL_FILES = list(os.walk("./data"))[0]
170     print(ALL_FILES)
171 
172     os.makedirs(OutputDir, exist_ok=True)
173     print('USE_LIB=', USE_LIB)
174 
175     for file in ALL_FILES:
176         CURRENT_FILE = os.path.splitext(os.path.basename(file))[0]
177 
178         pic = plt.imread(os.path.join("data", file))
179 
180         shrink_pic = shrink_image(pic)
181 
182         nearest_interpolate_pic = NearestInterpolation(shrink_pic)
183         nearest_loss = get_pic_loss(pic, nearest_interpolate_pic)
184 
185         bilinear_interpolate_pic = BilinearInterpolation(shrink_pic)
186         bilinear_loss = get_pic_loss(pic, bilinear_interpolate_pic)
187 
188         bicubic_interpolate_pic = BicubicInterpolate(shrink_pic)
```

```
189         bicubic_loss = get_pic_loss(pic, bicubic_interpolate_pic)
190 
191         show_pic(pic, "Original")
192         show_pic(nearest_interpolate_pic, "Nearest")
193         show_pic(bilinear_interpolate_pic, "Bilinear")
194         show_pic(bicubic_interpolate_pic, "Bicubic")
195 
196         print('-'*20 + CURRENT_FILE + '-'*20)
197         print('Nearest (MSE, PSNR): ', nearest_loss)
198         print('Bilinear (MSE, PSNR): ', bilinear_loss)
199         print('Bicubic (MSE, PSNR): ', bicubic_loss)
200 
201 if __name__ == "__main__":
202     main()
```

附录 B　实验结果

图 B.1　boat 原始图像

图 B.2　boat 最近邻插值图像

图 B.3　boat 双线性插值图像

图 B.4　boat 双三次内插图像

图 B.5　lena 原始图像

图 B.6　lena 最近邻插值图像

图 B.7　lena 双线性插值图像

图 B.8　lena 双三次内插图像

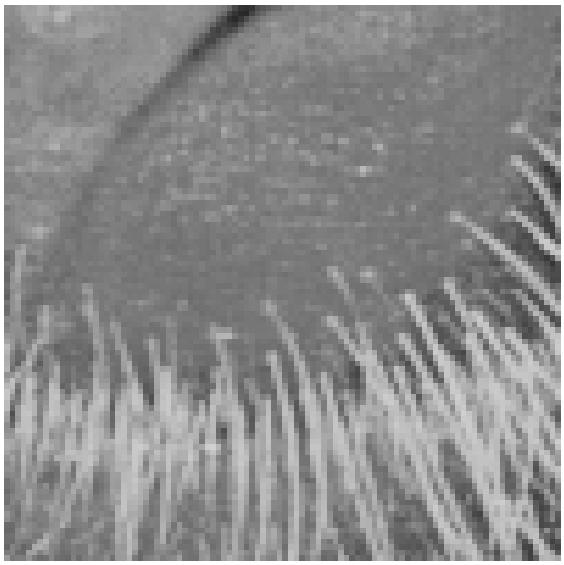

图 B.9　mandrill 原始图像

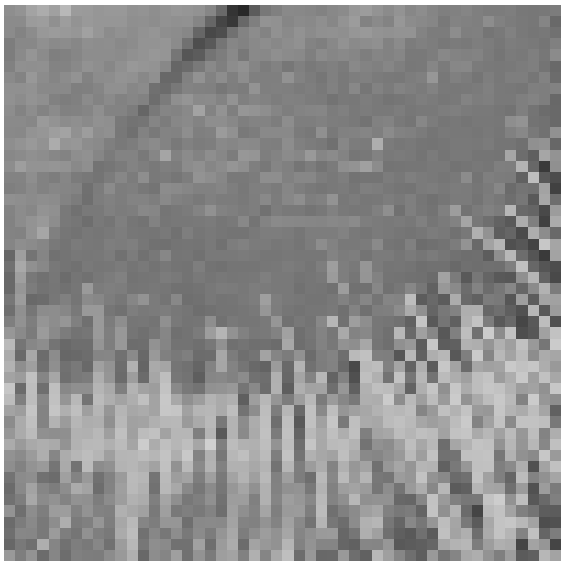

图 B.10　mandrill 最近邻插值图像

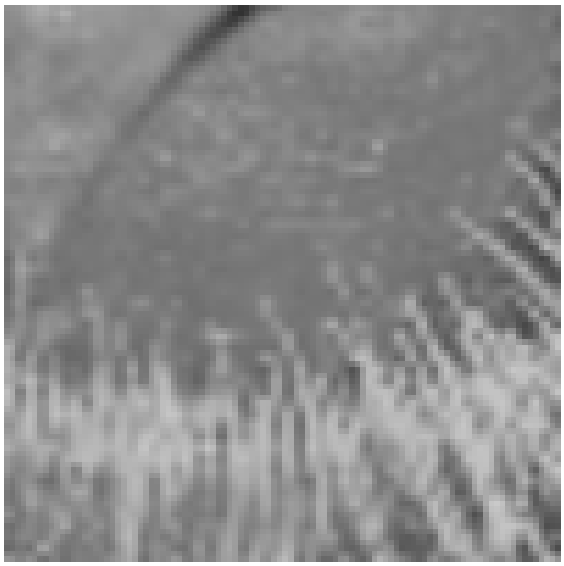

图 B.11　mandrill 双线性插值图像

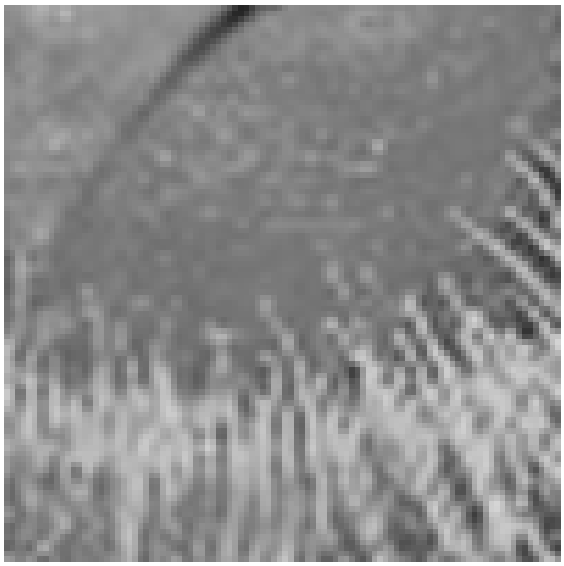

图 B.12　mandrill 双三次内插图像

图 B.13　peppers-bw 原始图像

图 B.14　peppers-bw 最近邻插值图像

图 B.15　peppers-bw 双线性插值图像

图 B.16　peppers-bw 双三次内插图像