图像处理作业——图像复原

刘坤鑫*

2019 年 12 月 5 日

摘要

本文采用 Python 编程，主要调包 skimage，以 Lenna 标准图像为研究对象，采用了 7 种噪声对其灰度图进行处理，高效地实现了课件中所有的 10 种滤波器，加上 skimage 自带的高斯模糊和均值滤波器 2 种滤波器，产生共 $10 * 12 = 120$ 张结果图，并进行了简要的对比分析。

1 Lenna 标准测试图像

图 1: Lenna

如图1所示，Lenna 或 Lena 是自 1973 年以来在图像处理领域广泛使用的标准测试图像的名称。可在 WiKi 上获取 512x512 的标准处理图像以及原图[1]。

*3017218061 软件工程一班

[1] https://en.wikipedia.org/wiki/Lenna

2 添加噪声

利用 python 的包 skimage 进行加噪处理，共包含以下噪声：

skimage.util.random_noise 中 mode 参数设置

```
mode = [
    'gaussian',
    'localvar',
    'poisson',
    'salt',
    'pepper',
    's&p',
    'speckle',
]
```

(a) Lenna with gaussian noise

(b) Lenna with salt and pepper noise

图 2: Lenna with noise

如图2为部分加噪效果图。为方便处理，加噪前已经将原图转为灰度图。完整结果参见 github[2]。

3 滤波器实现

本文实现了课件中所有的共 10 种滤波器，并添加了 skimage 自带的 2 种滤波器作为对比：

12 种滤波器对应函数名

```
filters = [
```

[2] https://github.com/qq734628996/image_processing/tree/master/homework2/

```
    arithmeticMeanFilter,
    geometricMeanFilter,
    harmonicMeanFilter,
    inverseHarmonicMeanFilter,
    medianFilter,
    maximumFilter,
    minimumFilter,
    medianRangeFilterFilter,
    improvedAlphaMeanFilter,
    AdaptiveMedianFilter,
    skimageGaussian,
    skimageMedian,
]
```

在实现非自适应滤波器时，考虑到所有滤波器都是共同的操作——卷积。理论上，卷积操作已经有了很好的计算优化。为了简单起见，又不失为充分考虑 Python 语言特性，并利用 numpy 进行计算加速。封装函数如下：

填充边界，存储掩模

```
def _paddingFilling(image, m=3, n=3):
    up, down = image[0], image[-1]
    for i in range(m // 2):
        image = np.vstack([up, image, down])
    left , right = image[:, [0]], image[:, [-1]]
    for i in range(n // 2):
        image = np.hstack([left, image, right])
    return image

def _imageSpliting(image, m=3, n=3):
    height, width = image.shape
    oldImage = _paddingFilling(image, m, n)
    oldImages = []
    for i in range(m):
        for j in range(n):
            oldImages.append(oldImage[i:i + height, j:j + width])
    oldImages = np.asarray(oldImages)
    return oldImages
```

上述代码实现了图像的边界处理，并且保存每个像素对应的掩码块。有了上述代码，非自适应滤波器的实现变得统一而简单。例如算术平均滤波器的实现如下：

算术平均滤波器

```
def arithmeticMeanFilter(image, m=3, n=3):
    oldImages = _imageSpliting(image, m=m, n=n)
    newImage = np.mean(oldImages, axis=0)
    return newImage
```

在笔者 i5 的笔记本上运行 512x512 的 Lenna 图片，只需要 0.01s。

剩下非自适应滤波器实现几乎同理，除了要注意不要出现除以 0 的情况，均无太大区别，且运行速度不差。对于自适应中值滤波器，网上代码大都冗长而低效，本文在此给出一个简

介而高效的 Python 实现：

自适应中值滤波器

```
def AdaptiveMedianFilter(image, SMax=7):
    height, width = image.shape
    newImage = image.copy()
    for i in range(height):
        for j in range(width):
            filterSize = 3
            z = image[i][j]
            while(filterSize <= SMax):
                S = filterSize//2
                tmp = image[max(0, i-S): i+S+1, max(0, j-S): j+S+1].reshape(-1)
                tmp.sort()
                zMin = tmp[0]
                zMax = tmp[-1]
                zMed = tmp[len(tmp)//2]
                if(zMin < zMed and zMed < zMax):
                    if(z == zMin or z == zMax):
                        newImage[i][j] = zMed
                    break
                else:
                    filterSize += 2
    return newImage
```

完整代码参见附录4

4 结果展示

如图3只展示了四种效果良好的滤波器，其余滤波器效果均不理想。对于高斯噪声，图像很难再复原到原图像的模样。其中，算术平均滤波器效果和逆谐波均值滤波器效果近似，中值滤波器减少了图像模糊，但留下了比较多的噪声。自适应均值滤波器笔者认为在细节上做的最好，在尽可能保留细节的情况下，尽可能地减少噪声。

如图4只展示了四种效果良好的滤波器，其余滤波器效果均不理想。其中，改进阿尔法均值滤波器很大程度上去除了了椒盐噪声，中值滤波器几乎去除了所有的椒盐噪声，自适应中值滤波器不仅去除了所有的椒盐噪声，还保留了图像细节（如帽子纹路），skimage 自带的中值滤波器和本文实现的中值滤波器略有区别，但整体效果几乎一样。

完整结果参见 github[3]。

[3]https://github.com/qq734628996/image_processing/tree/master/homework2/

(a) 算术平均滤波器

(b) 中值滤波器

(c) 逆谐波均值滤波器

(d) 自适应中值滤波器

图 3: 不同滤波器复原后的高斯噪声 Lenna 图

(a) 改进阿尔法均值滤波器

(b) 中值滤波器

(c) 自适应中值滤波器

(d) skimage 自带的中值滤波器

图 4: 不同滤波器复原后的椒盐噪声 Lenna 图

附录

完整 Python 代码如下：

filter.py

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-

from skimage import io
from PIL import Image
import skimage
import skimage.filters
import numpy as np
import matplotlib.pyplot as plt
import functools
import time
import os


def _paddingFilling(image, m=3, n=3):
    up, down = image[0], image[-1]
    for i in range(m // 2):
        image = np.vstack([up, image, down])
    left, right = image[:, [0]], image[:, [-1]]
    for i in range(n // 2):
        image = np.hstack([left, image, right])
    return image


def _imageSpliting(image, m=3, n=3):
    height, width = image.shape
    oldImage = _paddingFilling(image, m, n)
    oldImages = []
    for i in range(m):
        for j in range(n):
            oldImages.append(oldImage[i:i + height, j:j + width])
    oldImages = np.asarray(oldImages)
    return oldImages


def arithmeticMeanFilter(image, m=3, n=3):
    oldImages = _imageSpliting(image, m=m, n=n)
    newImage = np.mean(oldImages, axis=0)
    return newImage


def geometricMeanFilter(image, m=3, n=3):
    oldImages = _imageSpliting(np.log(image + 1e-6), m=m, n=n)
    newImage = np.exp(np.mean(oldImages, axis=0))
    return newImage


def harmonicMeanFilter(image, m=3, n=3):
    oldImages = _imageSpliting(1 / (image + 1e-6), m=m, n=n)
    newImage = (1 / np.mean(oldImages, axis=0))
    return newImage


def inverseHarmonicMeanFilter(image, m=3, n=3, Q=1):
```

```python
    oldImages1 = _imageSpliting((image + 1e-6) ** (Q + 1), m=m, n=n)
    oldImages2 = _imageSpliting((image + 1e-6) ** Q, m=m, n=n)
    return np.sum(oldImages1, axis=0) / np.sum(oldImages2, axis=0)


def medianFilter(image, m=3, n=3):
    oldImages = _imageSpliting(image, m=m, n=n)
    newImage = np.median(oldImages, axis=0)
    return newImage


def maximumFilter(image, m=3, n=3):
    oldImages = _imageSpliting(image, m=m, n=n)
    newImage = np.max(oldImages, axis=0)
    return newImage


def minimumFilter(image, m=3, n=3):
    oldImages = _imageSpliting(image, m=m, n=n)
    newImage = np.min(oldImages, axis=0)
    return newImage


def medianRangeFilterFilter(image, m=3, n=3):
    oldImages = _imageSpliting(image, m=m, n=n)
    newImage = (np.max(oldImages, axis=0) + np.min(oldImages, axis=0)) / 2
    return newImage


def improvedAlphaMeanFilter(image, m=3, n=3, d=2):
    d = d // 2
    oldImages = _imageSpliting(image, m=m, n=n)
    oldImages = np.sort(oldImages, axis=0)
    newImage = np.mean(oldImages[d:m * n - d], axis=0)
    return newImage


def AdaptiveMedianFilter(image, SMax=7):
    height, width = image.shape
    newImage = image.copy()
    for i in range(height):
        for j in range(width):
            filterSize = 3
            z = image[i][j]
            while( filterSize <= SMax):
                S = filterSize//2
                tmp = image[max(0, i-S): i+S+1, max(0, j-S): j+S+1].reshape(-1)
                tmp.sort()
                zMin = tmp[0]
                zMax = tmp[-1]
                zMed = tmp[len(tmp)//2]
                if(zMin < zMed and zMed < zMax):
                    if(z == zMin or z == zMax):
                        newImage[i][j] = zMed
                    break
                else:
                    filterSize += 2
    return newImage


def skimageGaussian(image, sigma=1):
```

```python
    newImage = skimage.filters.gaussian(image, sigma=sigma)
    return newImage


def skimageMedian(image):
    newImage = skimage.filters.median(image)
    return newImage


def main():
    basePath = 'img'
    imagePath = os.path.join(basePath, 'lena512.bmp')
    image = io.imread(imagePath)
    mode = [
        'gaussian',
        'localvar',
        'poisson',
        'salt',
        'pepper',
        's&p',
        'speckle',
    ]
    filters = [
        arithmeticMeanFilter,
        geometricMeanFilter,
        harmonicMeanFilter,
        inverseHarmonicMeanFilter,
        medianFilter,
        maximumFilter,
        minimumFilter,
        medianRangeFilterFilter,
        improvedAlphaMeanFilter,
        AdaptiveMedianFilter,
        skimageGaussian,
        skimageMedian,
    ]
    for m in mode:
        print(m)
        path = os.path.join(basePath, m)
        if not os.path.exists(path):
            os.mkdir(path)
        imageNoise = skimage.util.random_noise(image, mode=m)
        savePath = os.path.join(path, '{}.png'.format(m))
        io.imsave(savePath, imageNoise)
        for f in filters:
            savePath = os.path.join(path, '{}_{}.png'.format(m, f.__name__))
            io.imsave(savePath, f(imageNoise))


if __name__ == "__main__":
    main()
```