2023 年秋季学期 图像处理 编程作业 01

姓名：梁乐彬 学号：[redacted]

问题 1-图像处理软件包比较

如果选择采用 MATLAB 完成编程作业，请同学们熟悉其图像处理工具软件包。如果选择采用 Python，请同学们熟悉 Pillow (https://pillow.readthedocs.io/en/stable/)，Scitkit-image (https://scikit-image.org/) or OpenCV (https://opencv.org/) 这三个软件包之一。
从上述四个软件包中选择至少三个进行比较。针对每个软件包，
请回答以下问题：
- 该软件包支撑什么图像类型和格式?
- 使用该软件包如何读取，显示和写一个图像?
- 该软件包提供哪些类型的图像处理功能?

Pillow:

1、图像类型和格式：支持灰度图像、RGB 图像和 RGBA 图像。支持的图像格式如 JPEG、PNG、GIF、BMP 等。
2、读取、显示和写入图像：使用 Image.open()读取图像，使用 Image.show()显示图像，使用 Image.save()保存图像。
3、图像处理功能：提供调整大小、裁剪、旋转、滤镜效果、颜色调整等功能。如可以使用 Image.resize()调整图像大小，使用 Image.crop()裁剪图像，使用 Image.rotate()旋转图像，使用 ImageFilter 模块中的滤镜函数添加滤镜效果。

Scikit-image:

1、图像类型和格式：支持灰度图像和多通道彩色图像。支持常见的图像格式如 JPEG、PNG、TIFF 等。
2、读取、显示和写入图像：使用 skimage.io.imread()读取图像，使用 skimage.io.imshow()显示图像，使用 skimage.io.imsave()保存图像。
3、图像处理功能：提供滤波、边缘检测、图像分割、形态学操作等功能。可以使用 skimage.filters 模块中的函数进行滤波操作，使用 skimage.feature 模块进行边缘检测，使用 skimage.segmentation 模块进行图像分割，使用 skimage.morphology 模块进行形态学操作等。

OpenCV:

1、图像类型和格式：支持多通道彩色图像、灰度图像和特殊类型的图像。支持常见的图像格式如 JPEG、PNG、TIFF 等。
2、读取、显示和写入图像：使用 cv2.imread()读取图像，使用 cv2.imshow()显示图像，使用 cv2.imwrite()保存图像。
3、图像处理功能：提供图像滤波、边缘检测、图像变换、特征提取等功能。可以使用 cv2.filter2D()函数进行图像滤波，使用 cv2.Canny()函数进行边缘检测，使用 cv2.warpAffine()函数进行图像变换，使用各种特征检测算法如 SIFT、SURF 等进行特征提取。

问题 2 黑白图像灰度扫描 （task2.py）

实现一个函数 s = scanLine4e(f, I, loc)，其中 f 是一个灰度图像，I 是一个整数，loc 是一个字符串。当 loc 为'row'时，I 代表行数。当 loc 为'column'时，I 代表列数。输出 s 是对应的相关行或者列的像素灰度值矢量。 调用该函数，提取 cameraman.tif 和 einstein.tif 的中心行和中心列的像素灰度矢量并将扫描得到的灰度序列绘制成图。

实现步骤：

1．导入函数库

2．定义了三个函数:

scanLine4e(f, I, loc): 该函数用于提取图像数组 f 中指定行或列的像素值。根据 I 的类型，判断是提取单行/列还是中间两行/列的像素值，并返回提取行或列像素的均值。

```
def scanLine4e(f,I,loc):
    try:
        if type(I) is int:          # 判断I的类型，当行/列是奇数时，I为INT类型，返回该行/列像素的均值
            if loc == "row":
                sOut = f[I,:]
            elif loc == "column":
                sOut = f[:,I]

        else:                       # 判断I的类型，当行/列是偶数时，I为list类型，返回中间两行像素的均值
            if loc == "row":
                sOut = np.sum(f[I[0]:I[1]+1,:],axis=0) / 2
                sOut = np.int16(sOut)
            elif loc == "column":
                sOut = np.sum(f[:,I[0]:I[1]+1],axis=1) / 2
                sOut = np.int16(sOut)

        return sOut

    except:
        print("ERROR01:参数值错误")
```

getCenterIndex(length): 该函数根据图像的宽度或长度，返回中心行或中心列的索引。如果图像的宽度或长度为偶数，则返回包含两个整数的列表，否则返回一个整数。

```
def getCenterIndex(length):
    if length % 2 == 0:
        index = [int(length / 2) - 1, int(length / 2)]  #偶数长度取中间两行/列的索引
    else:
        index = int(length / 2)  #奇数长度取中间行/列的索引
    return index
```

extractCenterLinePixels(image, axis): 该函数根据输入的图像数组和参数，返回图像中心行或中心列的像素值。根据参数 axis 确定要提取的是中心行还是中心列的像素值。

获取图像的长度或宽度（根据 axis 确定），调用 getCenterIndex 函数获取中心行或中心列的索引，调用 scanLine4e 函数提取中心行或中心列的像素值。

```
def extractCenterLinePixels(image, axis):
    try:
        length = image.shape[0] if axis == "row" else image.shape[1]  #获取图像的长度或宽度
        index = getCenterIndex(length)  #获取中心行或中心列的索
        index_out = scanLine4e(image,index,axis)
        return index_out
    except:
        print("ERROR02:参数值错误")
```

3．使用 PIL 库加载 cameraman.tif 和 einstein.tif 图像，并分别获取它们的尺寸。

4．调用 extractCenterLinePixels 函数提取 cameraman.tif 图像的中心行和中心列的像素值。分别使用 scanLine4e 函数提取中心行和中心列的像素值，并将结果转换为 1xW1 的数组。

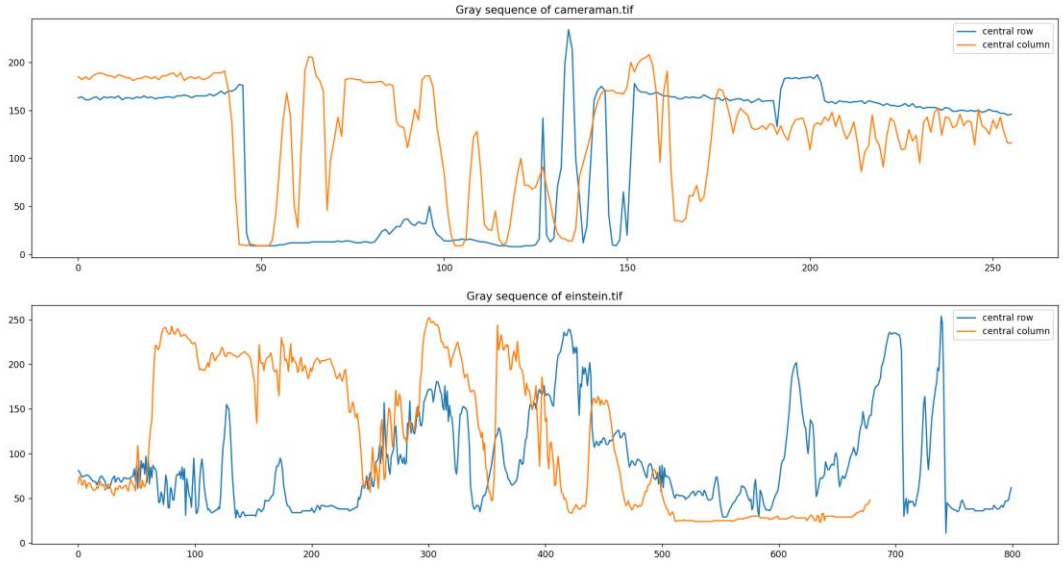
5 .调用 extractCenterLinePixels 函数提取 einstein.tif 图像的中心行和中心列的像素值。同样，使用 scanLine4e 函数提取中心行和中心列的像素值，并将结果转换为 1xW2 和 1xH2 的数组。

```
cameraman_row = extractCenterLinePixels(cameraman,"row")#列提取
cameraman_row = cameraman_row.reshape(1,W1) # 一维转二维 方便显示
cameraman_column = extractCenterLinePixels(cameraman,"column") #列提取
cameraman_column = cameraman_column.reshape(1,H1)  # 一维转二维

einstein_row = extractCenterLinePixels(einstein,"row")#列提取
einstein_row = einstein_row.reshape(1,W2)  # 一维转二维
einstein_column = extractCenterLinePixels(einstein,"column")#列提取
einstein_column = einstein_column.reshape(1,H2)  # 一维转二维
```

6. 绘制两个子图，展示 cameraman.tif 和 einstein.tif 的中心行和中心列的灰度序列。

```
plt.figure(figsize=(30,10))  # 创建显示窗口尺寸

plt.subplot(2,1,1)
plt.title("Gray sequence of cameraman.tif")
plt.plot(cameraman_row[0,:],label = 'central row')
plt.plot(cameraman_column[0,:],label = 'central column')
plt.legend()

plt.subplot(2,1,2)
plt.title("Gray sequence of einstein.tif")
plt.plot(einstein_row[0,:],label = 'central row')
plt.plot(einstein_column[0,:],label = 'central column')
plt.legend()
plt.show()
```

运行效果：



问题 3 彩色图像转换为黑白图像（task3.py）

图像处理中的一个常见问题是将彩色 RGB 图像转换成单色灰度图像，第一种常用的方法是 取三个元素 R，G，B 的均值。第二种常用的方式，又称为 NTSC 标准，考虑了人类的彩色感知体验，对于 R,G,B 三通道分别采用了不同的加权系数，分别是 R 通道 0.2989，G 通道 0.5870，B 通道 0.1140. 实现一个函数 g = rgb1gray(f, method). 函数功能是将一幅 24 位的 RGB 图像, f, 转换成灰度图像, g. 参数 method 是一个字符串，当其值为'average' 时，采用第一种转换方法，当其值为'NTSC'时，采用第二种转换方法。将'NTSC'做为缺省方式。

调用该函数，将提供的图像 mandril_color.tif 和 lena512color.tiff 用上述两种方法转换成单色灰度图像，对于两种方法的结果进行简短比较和讨论。

实现步骤：

1．导入函数库

2．定义了一个函数:

rgb1gray() 函数根据给定的转换方法将彩色图像 f 转换为灰度图像 g。使用 if 和 elif 条件语句来确定转换方法，并根据方法选择相应的转换操作。当 method 的值为 'average' 时，使用 np.mean() 计算 f 数组沿着第三个维度的平均值。当 method 的值为 'NTSC' 时，创建权重数组 weights，然后使用 np.dot() 计算 f 数组和权重数组之间的点积。如果 method 的值既不是 'average' 也不是 'NTSC'，则引发异常

```python
def rgb1gray(f, method='NTSC'):
    """
    将彩色 RGB 图像转换为灰度图像。

    参数:
        f: numpy 数组，表示彩色图像，形状为 (H, W, 3)。
        method: 字符串，指定转换方法。可选值为 'average' 和 'NTSC'，默认为 'NTSC'。

    返回:
        gray: numpy 数组，表示灰度图像，形状为 (H, W)。
    """
    try:

        if method == 'average':
            gray = np.mean(f, axis=2, keepdims=True) # 取平均值方法
        elif method == 'NTSC':
            weights = np.array([0.2989, 0.5870, 0.1140]) # NTSC 标准方法
            gray = np.dot(f, weights) # 矩阵乘法
        else:
            raise ValueError("ERROR02:错误参数。支持的参数为 'average'和 'NTSC'")

    except ValueError as e:
        print("ERROR02: 参数值错误")
        raise e

    return gray.astype(np.uint8)
```

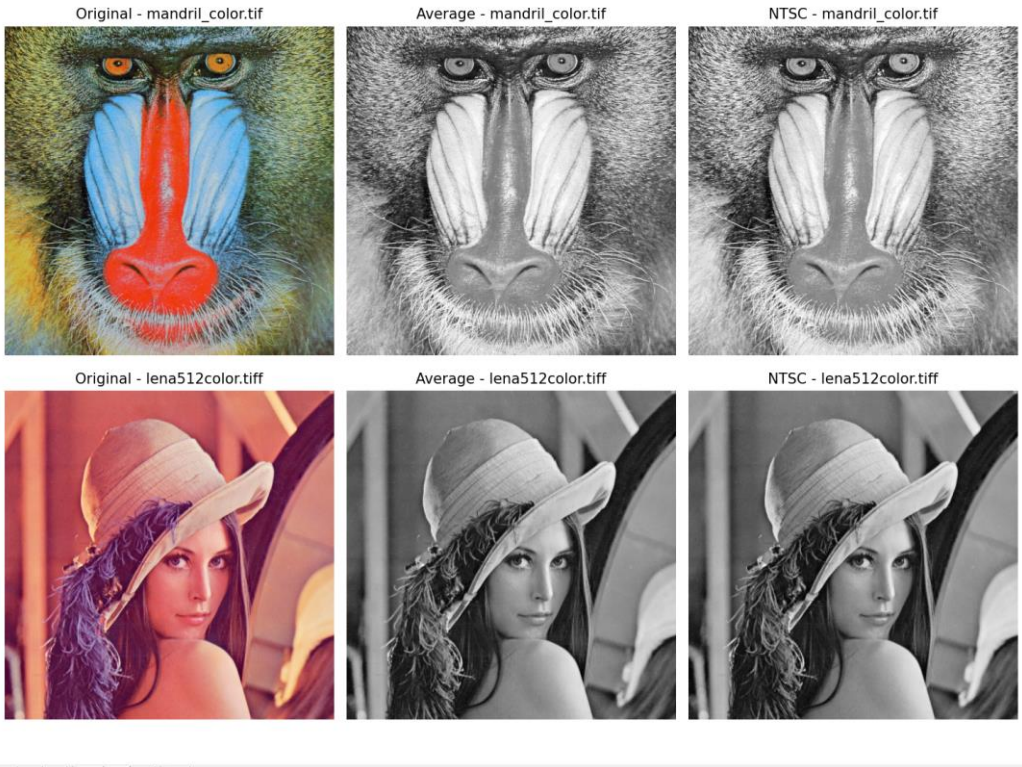
3．加载图像并调用 rgb1gray 将彩色图像转换为黑白图像

```python
# 加载图像
mandril_color = np.array(Image.open('mandril_color.tif'))
lena512color = np.array(Image.open('lena512color.tiff'))

# 转换为灰度图像
mandril_gray_average = rgb1gray(mandril_color, method='average')
mandril_gray_NTSC = rgb1gray(mandril_color, method='NTSC')

lena_gray_average = rgb1gray(lena512color, method='average')
lena_gray_NTSC = rgb1gray(lena512color, method='NTSC')
```

4．通过画布显示原图像和两种不同的转换方法得到的灰度图像

```
# 显示原图和灰度图像
plt.figure(figsize=(12, 10))

# 原图像1
plt.subplot(2, 3, 1)
plt.imshow(mandril_color)
plt.title('Original - mandril_color.tif')
plt.axis('off')

# 灰度图像1
plt.subplot(2, 3, 2)
plt.imshow(mandril_gray_average[:, :, 0], cmap='gray')
plt.title('Average - mandril_color.tif')
plt.axis('off')

# 灰度图像2
plt.subplot(2, 3, 3)
plt.imshow(mandril_gray_NTSC[:, :, 0], cmap='gray')
plt.title('NTSC - mandril_color.tif')
plt.axis('off')

# 原图像2
plt.subplot(2, 3, 4)
plt.imshow(lena512color)
plt.title('Original - lena512color.tiff')
plt.axis('off')

# 灰度图像3
plt.subplot(2, 3, 5)
plt.imshow(lena_gray_average[:, :, 0], cmap='gray')
plt.title('Average - lena512color.tiff')
plt.axis('off')
```

运行效果：



比较和讨论：

对于 mandril_color.tif 和 lena512color.tiff 两幅图像，分别显示了原始图像和使用两种转换方法得到的灰度图像。可以观察到，使用 'average' 方法将彩色图像转换为灰度图像时，图像的亮度较为均匀。而使用 'NTSC' 方法时，考虑了人类的彩色感知体验，图像的亮度相对更加符合人眼的感知。具体选择哪种方法取决于应用的需求和预期的效果。

问题 4 图像二维卷积函数 （task4.py）

实现一个函数 g = twodConv(f, w)，其中 f 是一个灰度源图像，w 是一个矩形卷积核。要求输 出图像 g 与源图像 f 大小（也就是像素的行数和列数）一致。请注意，为满足这一要求，对 于源图像 f 需要进行边界像素填补(padding)。这里请实现两种方案。第一种方案是像素复制， 对应的选项定义为'replicate'，填补的像素拷贝与其最近的图像边界像素灰度。第二种方案是 补零，对应的选项定义为'zero'，填补的像素灰度为 0. 将第二种方案设置为缺省选择。

实现步骤：

1．导入函数库

2. ***twodConv(f, w, padding='zero')*** 实现了二维卷积。计算卷积核的填补宽度 pad_width，通过 (w.shape[0] - 1) // 2 来确定。这里假设卷积核的高度和宽度都是奇数。使用 pad_image 函数对输入图像 f 进行填补，填补宽度为 pad_width，填补方式由 padding 参数指定。填补后的图像赋值给 f_padded。创建一个与输入图像 f 相同大小的全零数组 g，用于存储卷积结果。使用双重循环遍历输入图像 f 的每个像素位置 (i, j)：通过索引 i - 1 到 i + w.shape[0] - 1 和 j - 1 到 j + w.shape[1] - 1，从填补后的图像 f_padded 中提取与卷积核大小相同的图像块。将图像块与卷积核 w 逐元素相乘，并对乘积结果求和。将求和结果赋值给卷积结果 g 的对应位置 (i - 1, j - 1)。循环结束后，返回卷积结果 g。

```
def twodConv(f, w, padding='zero'):
    """
    二维卷积函数

    参数:
    - f: 输入图像，一个二维 NumPy 数组
    - w: 卷积核，一个二维 NumPy 数组
    - padding: 填补选项，可选值为 'zero'（默认）和 'replicate'

    返回:
    - g: 卷积结果，一个二维 NumPy 数组
    """
    pad_width = (w.shape[0] - 1) // 2  # 填补宽度

    f_padded = pad_image(f, pad_width, padding) #调用自定义pad函数

    g = np.zeros_like(f) #输出图像与输入图像尺寸一样

    for i in range(1, f.shape[0] + 1):  #遍历图像元素进行卷积操作
        for j in range(1, f.shape[1] + 1):
            g[i - 1, j - 1] = np.sum(f_padded[i - 1:i + w.shape[0] - 1, j - 1:j + w.shape[1] - 1] * w) #根据卷积核大小来提取子图

    return g
```

3. ***pad_image(image, pad_width, padding='zero')*** 实现了对图像进行填补的操作。获取输入图像 image 的行数和列数，分别赋值给 rows 和 cols。创建一个全零数组 padded_image，大小为 (rows + 2 * pad_width, cols + 2 * pad_width)，数据类型与输入图像相同。将输入图像 image 复制到 padded_image 的内部区域，即 padded_image[pad_width:pad_width + rows, pad_width:pad_width + cols] = image，完成图像的中心填补。如果 padding 参数为 'replicate'：在填补区域的上方和下方复制原始图像的第一行和最后一行，以复制边界像素。在填补区域的左侧和右侧复制原始图像的第一列和最后一列，以复制边界像素。返回填补后的图像 padded_image。

```python
def pad_image(image, pad_width, padding='zero'):
    """
    对图像进行填补

    参数:
    - image: 输入图像，一个二维 NumPy 数组
    - pad_width: 填补宽度
    - padding: 填补选项，可选值为 'zero'（默认）和 'replicate'

    返回:
    - padded_image: 填补后的图像，一个二维 NumPy 数组
    """
    rows, cols = image.shape  # 获取图像尺寸
    padded_image = np.zeros((rows + 2 * pad_width, cols + 2 * pad_width), dtype=image.dtype) # 根据填补宽度，创建一个全零数组
    padded_image[pad_width:pad_width + rows, pad_width:pad_width + cols] = image # 将原图像复制到中心区域

    if padding == 'replicate': #如果是复制边界
        # 复制边界像素
        # 复制行
        for i in range(pad_width):
            padded_image[i, pad_width:pad_width + cols] = image[0, :]
            padded_image[-i - 1, pad_width:pad_width + cols] = image[-1, :]

        # 复制列
        for j in range(pad_width):
            padded_image[pad_width:pad_width + rows, j] = image[:, 0]
            padded_image[pad_width:pad_width + rows, -j - 1] = image[:, -1]

    elif padding != 'zero':
        raise ValueError("Invalid padding option. Supported options are 'replicate' and 'zero'.")

    return padded_image
```

运行效果：

```
原数组:
[[1 2 3]
 [4 5 6]
 [7 8 9]]
【replicate】pad后数组:
[[0 1 2 3 0]
 [1 1 2 3 3]
 [4 4 5 6 6]
 [7 7 8 9 9]
 [0 7 8 9 0]]
卷积后后数组:
[[ 9 13 17]
 [21 25 29]
 [33 37 41]]
【zero】pad后数组:
[[0 0 0 0 0]
 [0 1 2 3 0]
 [0 4 5 6 0]
 [0 7 8 9 0]
 [0 0 0 0 0]]
卷积后后数组:
[[ 7 11 11]
 [17 25 23]
 [19 29 23]]
```

问题 5 归一化二维高斯滤波核函数 （task5.py）

实现一个高斯滤波核函数 w = gaussKernel(sig, m), 其中 sig 对应于高斯函数定义中的 σ,w 的大小为 m×m。请注意，这里如果 m 没有提供，需要进行计算确定。如果 m 已提供但过小， 应给出警告信息提示。w 要求归一化，即全部元素加起来和为 1。

实现步骤：

1. 导入函数库
2. ***gaussKernel(sig, m=None)*** 用于生成高斯滤波核。如果未提供 m 的值，则根据给定的标准差 sig 计算 m。计算方式为 m = 2 * math.ceil(3 * sig) + 1。这里使用了一个经验公式来估计滤波核的大小。如果提供了 m 的值，并且 m 是偶数，则打印警告信息，并将 m 的值增加 1，以确保滤波核具有正确的中心位置。创建一个大小为 (m, m) 的全零数组 w，用于存储滤波核。计算滤波核的中心坐标，即 (m - 1) / 2。使用双重循环遍历滤波核 w 的每个元素位置 (i, j)：计算当前位置相对于滤波核中心的偏移量 x 和 y。根据高斯函数的公式 math.exp(-(x**2 + y**2) / (2 * sig**2))，计算当前位置的值，并将其赋值给滤波核 w 的对应位置 (i, j)。对滤波核进行归一化，即将滤波核的所有元素除以它们的总和，以确保滤波核的权重之和为 1。返回生成的高斯滤波核 w。

```python
def gaussKernel(sig, m=None):
    """
    高斯滤波核函数

    参数:
    - sig: 高斯函数的标准差
    - m: 高斯滤波核的大小（可选）

    返回:
    - w: 高斯滤波核，一个二维 NumPy 数组
    """
    if m is None:
        # 根据 sigma 计算 m
        m = 2 * math.ceil(3 * sig) + 1
        print("警告：未提供 m 的值。根据 sigma 的计算结果为 m 分配了默认值:", m)
    elif m % 2 == 0:
        print("警告：提供的 m 值为偶数，请使用奇数值以获得正确的中心位置。")
        m += 1
        print("已自动调整 m 的值为:", m)

    # 创建空白的滤波核
    w = np.zeros((m, m))

    # 计算滤波核的中心坐标
    center = (m - 1) / 2

    # 计算高斯滤波核的每个元素的值
    for i in range(m):
        for j in range(m):
            x = i - center
            y = j - center
            w[i, j] = math.exp(-(x**2 + y**2) / (2 * sig**2))

    # 归一化滤波核
    w /= np.sum(w)

    return w
```

运行效果：

```
输入sigma: 1, m=3
高斯滤波核:
[[0.07511361 0.1238414  0.07511361]
 [0.1238414  0.20417996 0.1238414 ]
 [0.07511361 0.1238414  0.07511361]]
```

问题 6.1 灰度图像的高斯滤波 （task6_1.py）

调用上面实现的函数，对于问题 1 和 2 中的灰度图像（cameraman, einstein, 以及 lena512color 和 mandril_color 对应的 NTSC 转换后的灰度图像）进行高斯滤波，采用 σ=1, 2，3，5。任 选一种像素填补方案。

实现步骤：

1. 导入函数库
2. 加载前面问题的函数后
3. 导入图片
4. 将彩色图片使用 NTSC 方法灰度化后
5. 设置预设高斯滤波参数
6. 设置像素填补类型为 replicate
7. 设置显示画布

```
# 加载图像
cameraman_g = np.array(Image.open('cameraman.tif'))
einstein_g = np.array(Image.open('einstein.tif'))
mandril_color = np.array(Image.open('mandril_color.tif'))
lena512color = np.array(Image.open('lena512color.tiff'))


# NTSC灰度化
mandril_color_g = rgb1gray(mandril_color, method='NTSC')
lena512color_g = rgb1gray(lena512color, method='NTSC')

# 高斯滤波参数
sigmas = [1, 2, 3, 5]

# 像素填补选项
padding = 'replicate'

# 创建一个子图网格
num_rows = len(sigmas)  # 网格的行数
num_cols = 4  # 网格的列数（用于显示4张图片）
```

8. 遍历不同的 sigma 值，并根据每个 sigma 值生成对应的高斯滤波核，然后将滤波核应用于不同的图像上进行滤波操作，并将滤波后的图像绘制出来。
9. 使用 enumerate 函数遍历 sigmas 列表中的每个 sigma 值，并使用 i 记录当前的索引。
10. 对于每个 sigma 值，调用 gaussKernel 函数生成对应的高斯滤波核 w。
11. 使用 twodConv 函数将生成的高斯滤波核 w 应用到不同的图像上进行滤波操作，并将滤波后的图像分别存储在 cameraman_filtered、einstein_filtered、lena512color_filtered 和 mandril_color_filtered 变量中。
12. 使用 imshow 函数将滤波后的图像绘制在子图中，其中 axs[i, 0]、axs[i, 1]、axs[i, 2] 和 axs[i, 3] 分别对应于不同图像的子图，使用 cmap='gray' 参数指定绘制灰度图。

```python
# 遍历不同的 sigma 值并绘制图片
for i, sigma in enumerate(sigmas):
    # 生成高斯滤波核
    w = gaussKernel(sigma)

    # 对图像应用高斯滤波
    cameraman_filtered = twodConv(cameraman_g, w, padding)
    einstein_filtered = twodConv(einstein_g, w, padding)
    lena512color_filtered = twodConv(lena512color_g, w, padding)
    mandril_color_filtered = twodConv(mandril_color_g, w, padding)

    # 绘制滤波后的图片
    axs[i, 0].imshow(cameraman_filtered, cmap='gray')
    axs[i, 0].set_title('Cameraman (σ={})'.format(sigma))

    axs[i, 1].imshow(einstein_filtered, cmap='gray')
    axs[i, 1].set_title('Einstein (σ={})'.format(sigma))

    axs[i, 2].imshow(lena512color_filtered, cmap='gray')
    axs[i, 2].set_title('Lena512color (σ={})'.format(sigma))

    axs[i, 3].imshow(mandril_color_filtered, cmap='gray')
    axs[i, 3].set_title('Mandril_color (σ={})'.format(sigma))

# 移除空的子图
if num_rows * num_cols > len(sigmas) * 4:
    for j in range(len(sigmas) * 4, num_rows * num_cols):
        fig.delaxes(axs[j // num_cols, j % num_cols])

# 调整子图之间的间距
plt.tight_layout()

# 显示绘图
plt.show()
```

运行效果：

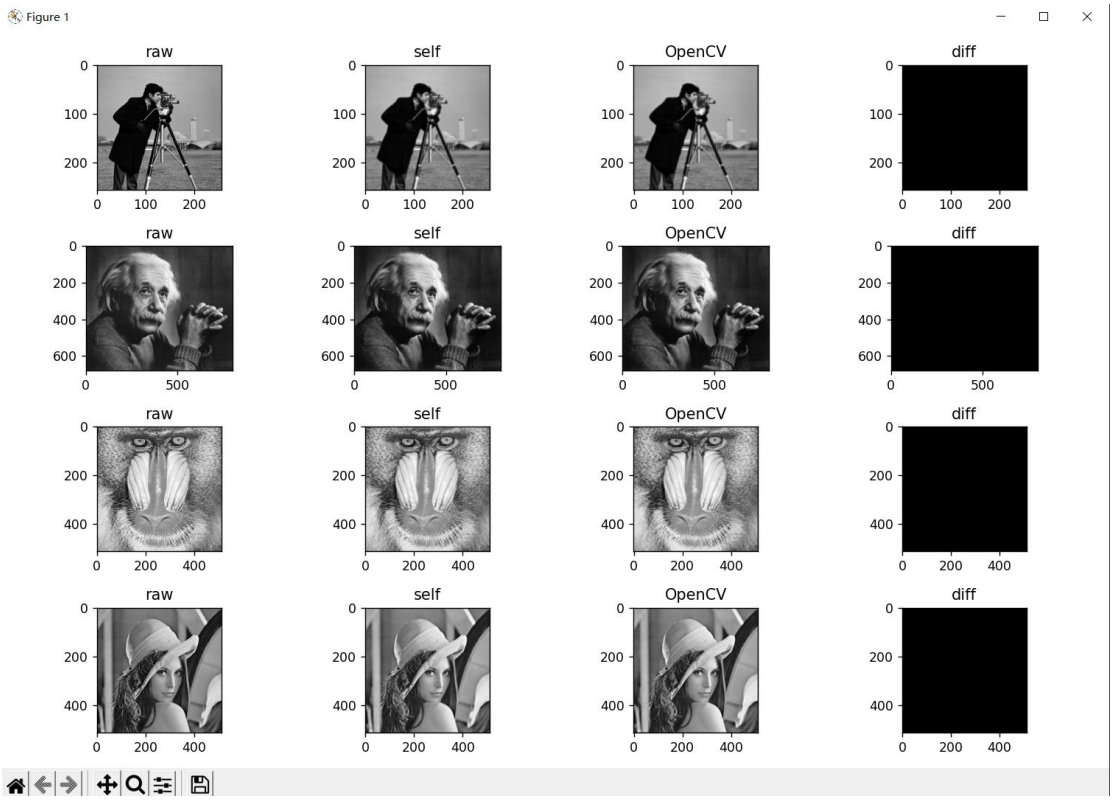
问题 6.2 灰度图像的高斯滤波（task6_2.py）

对于 σ=1 下的结果，与直接调用 MATLAB 或者 Python 软件包相关函数的结果进行比较（可以简单计算差值图像）。

实现步骤：

1. 首先定义了一个 sigma 值，用于控制高斯滤波的模糊程度。
2. 使用 OpenCV 加载了四张图像，分别是'cameraman.tif'、'einstein.tif'、'mandril_color.tif'和'lena512color.tiff'。加载后的图像以灰度方式存储在 images 列表中。
3. 使用自定义函数进行高斯模糊。对于每张图像，使用 gaussKernel(sigma)函数生成高斯滤波器的核，然后使用自定义的 twodConv()函数对图像进行滤波。滤波结果存储在 filtered_images_custom 列表中。
4. 使用 OpenCV 的 cv2.getGaussianKernel 函数获取卷积核。
5. 使用 np.outer 获得二维卷积核后使用 cv2.filter2D(image.astype(np.double), -1, kernel_2d, borderType=cv2.BORDER_CONSTANT)，使用补零方法为 pad 方法，对图像进行卷积操作。
5. 计算自定义滤波结果和 OpenCV 滤波结果之间的差异图像。对于每对滤波结果，使用 np.abs()函数计算绝对差异，并将差异图像存储在 diff_images 列表中。

```
# 使用OpenCV加载图像
image_filenames = ['cameraman.tif', 'einstein.tif', 'mandril_color.tif', 'lena512color.tiff']
images = [cv2.imread(filename, cv2.IMREAD_GRAYSCALE) for filename in image_filenames]

# 得到自定义卷积核
kernel = gaussKernel(sigma)
#print()

# 使用自定义函数进行高斯
filtered_images_custom = []
for image in images:
    image_double = image.astype(np.double)
    filtered_image_custom = twodConv(image_double, kernel)
    filtered_images_custom.append(filtered_image_custom)

# 使用OpenCV进行高斯卷积
kernel_size = (7, 7)  # 卷积核大小 m = 2 * ceil(3 * sigma) + 1
gaussian_kernel = cv2.getGaussianKernel(kernel_size[0], sigma)  # 获取高斯滤波器的核
kernel_2d = np.outer(gaussian_kernel, gaussian_kernel)
#print(kernel_2d)
filtered_images_opencv = [ cv2.filter2D(image.astype(np.double), -1, kernel_2d, borderType=cv2.BORDER_CONSTANT) for image in images]

# 计算差异图像
diff_images = [ (custom_img - opencv_img) for custom_img, opencv_img in zip(filtered_images_custom, filtered_images_opencv)]

# 创建画布用于显示图像
```



相似度：计算方法为均方误差 MSE，一种简单的相似度度量方法，它计算差异图像中每个像素与对应像素的差值的平方，并对所有差值求平均。MSE 越小，表示两个图像越相似。可见卷积后的图像几乎完全一样。

```
警告：未提供 m 的值。根据 sigma 的计算结果为 m 分配了默认值：7
cameraman.tif:
1.8688066335325295e-27
einstein.tif:
1.0286830666803364e-27
mandril_color.tif:
2.07358707816894e-27
lena512color.tiff:
2.4107531004173303e-27
```

问题 6.3 灰度图像的高斯滤波（task6_3.py）

然后，任选两幅图像，比较其他参数条件不变的情况下像素复制和 补零下滤波结果在边界上的差别。

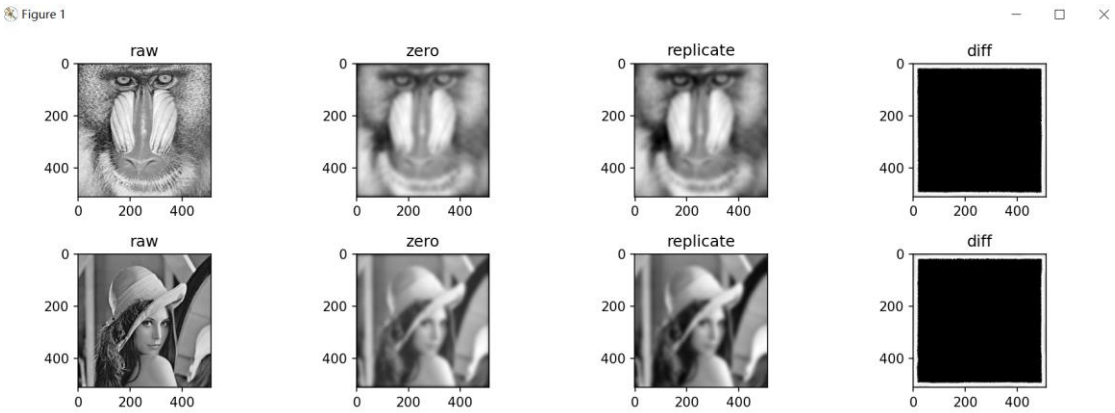
实现步骤：

1.采用 6.2 问题中的改版代码即可实现针对 mandril_color.tif 和 lena512color.tiff

```python
# 定义sigma值
sigma = 7

# 使用OpenCV加载图像
image_filenames = ['mandril_color.tif', 'lena512color.tiff']
images = [cv2.imread(filename, cv2.IMREAD_GRAYSCALE) for filename in image_filenames]

# 使用自定义函数进行高斯模糊
filtered_images_zero = []
filtered_images_replicate = []
for image in images:
    kernel = gaussKernel(sigma) #设置高斯核
    filtered_image_zero = twodConv(image, kernel,padding="zero") #补0
    filtered_image_replicate = twodConv(image, kernel,padding="replicate") #像素复制
    filtered_images_replicate.append(filtered_image_replicate) #处理后的图像进入队列
    filtered_images_zero.append(filtered_image_zero)

# 计算差异图像
diff_images = [np.abs(custom_img - opencv_img) for custom_img, opencv_img in zip(filtered_images_zero, filtered_images_replicate
```

3. 设置 sigma 值为 7
4. 画布显示对比图像

运行效果：



比较结果：

可以明显看出 zero 和 replicate 得出的 diff 图只有边缘位置差别比较大，当使用复制最近的像素点进行填补时，卷积之后的结果较好，并且边界比使用补零方式的边界亮。这是因为复制最近的像素点的填补方式可以保留边界的信息，而补零方式会引入额外的零值像素，导致边界变暗。补零方式会在边界周围添加黑色（零值）像素，而这些像素会对边界产生影响，使得边界区域的像素值降低，从而导致边界变暗。相比之下，复制最近的像素点填补方式可以利用原始图像中最近的像素值进行填充，从而保留了边界的亮度。