暨南大学计算机科学系

《数字图像处理》课程设计

成绩：

题　　目：	频率域图像处理
姓　　名：	杨博
学　　号：	2021101228
专　　业：	网络工程
课程类别：	专业选修课
任课老师：	彭玉青
提交日期：	2023 年 6 月 16 日

1. 选题和内容

1.1 选题和内容

选题：菌落数目的监测

内容：生物图像中，经常要检测菌落的数目是多少，自行实现傅里叶变换程序进行边缘检测， 得出图像中的菌落数目。

2. 开发环境与库函数说明

2.1 开发环境

编程语言的选择：编程语言选择使用的是 python。Python 是一种高层次的结合了解释性、编 译性、互动性和面向对象的脚本语言。

在此次实验中使用的 Python 解释器为 base：3.8

2.2 库函数说明

在程序设计中使用到的库函数有：OpenCV、numpy

1.Opencv：

OpenCV 是一个基于 Apache2.0 许可（开源）发行的跨平台计算机视觉和机器学习软件库，可以运行在 Linux、Windows、Android 和 Mac OS 操作系统上。 它轻量级而且高效——由一系列 C 函数和少量 C++ 类构成，同时提供了 Python、Ruby、MATLAB 等语言的接口，实现了图像处理和计算机视觉方面的很多通用算法。

2.numpy：

NumPy（Numerical Python）是 Python 的一种开源的数值计算扩展。这种工具可用来存储和处理大型矩阵，比 Python 自身的嵌套列表（nested list structure）结构要高效的多（该结构也可以用来表示矩阵（matrix）），支持大量的维度数组与矩阵运算，此外也针对数组运算提供大量的数学函数库。

3. 课程设计编程思想介绍

3.1 面向对象编程

面向对象编程（Object-Oriented Programming，OOP）是一种编程思想，它将数据和操作数据的方法封装在一起，以对象的形式表示出来。这种编程思想是基于现实世界中的对象和它们之间的相互作用来构建程序的。面向对象编程中，每个对象都有自己的属性和方法，而且可以与其他对象进行交互。面向对象编程有三个主要特征：封装、继承和多态。其中，封装是指将数据和方法封装在一起，以便于控制访问；继承是指一个类可以继承另一个类的属性和方法；多态是指同一个方法可以在不同的类中有不同的实现方式。

4. 程序设计过程

4.1 基本原理

4.1.1 高斯滤波器

（1）高斯滤波器是一种线性平滑滤波器，它对图像中的噪声进行平滑处理，并

且可以模糊图像以减少噪点和细节。高斯滤波器将每个像素周围的像素值加权平均，其中距离较远的像素具有更小的权重，距离较近的像素具有更大的权重
（2）二维高斯函数公式：

$$f(x,y)=f(x)f(y)=\frac{1}{\sqrt{2\pi}\sigma_x}e^{-\frac{(x-u_x)^2}{2\sigma_x^2}}\frac{1}{\sqrt{2\pi}\sigma_y}e^{-\frac{(x-u_y)^2}{2\sigma_y^2}}$$

（3）高斯核是一个二维函数，通常在中心点处有最大值，然后随着距离中心点的增加，函数值逐渐降低。高斯核的大小和方差控制了滤波器的平滑程度。

4.1.2 傅里叶变换

（1）傅里叶变换（Discrete Fourier Transform，DFT）是一种将信号从时域（时间域）转换到频域（频率域）的数学工具。它可以将一个复杂的信号分解成若干个简单的正弦波（或余弦波）的叠加，每个正弦波的振幅、频率都可以从频谱图中得到。这样，在频域中，我们可以更清楚、更直观地看到信号的频率和能量分布情况。
（2）傅里叶变换的主要作用是在频域中描述信号的特征。通过傅里叶变换，我们可以得到信号的频谱分布，包括频率和能量分布情况，同时也可以反过来将频域图像进行反变换，重构出原始的信号。
（3）频率过滤作用主要是滤除图像信号中的某些频率分量，从而实现对图像的噪声去除、特征提取和增强等目的。在频域中，对于图像的频率分量，可以根据其出现的位置和强度进行分类。常见的频率分量包括低频分量、高频分量和中频分量。功能：

- 去除图像中的噪声：通常图像的噪声分布在高频区域，如果使用低通滤波器，即滤除高频分量，可以有效地去除图像中的噪声和杂点等。
- 保留图像的边缘信息：图像的边缘信息分布在高频区域，如果使用高通滤波器，即滤除低频分量，可以保留图像的边缘信息，从而实现对图像的边缘检测和轮廓提取。
- 提取图像的特征信息：频率过滤可以实现对图像的特征提取，例如可以对图像进行带通滤波，滤除低频和高频分量，只保留中频分量，从而实现对图像的纹理特征提取等。
- 图像增强：通过对图像进行频率过滤，可以改变原始图像的灰度分布和亮度对比度等特征，从而实现对图像的增强和优化。

4.1.2 反傅里叶变换

反傅里叶变换将频域信号转换回时域信号，从而实现对信号的还原和恢复。在图像处理中，经过傅里叶变换和频率过滤等处理后，得到的是经过频域滤波之后的图像，我们需要将其变换回时域，才能得到完整的图像。

4.1.3 二值化处理

进行二值化处理的作用是将图像转换为黑白两种颜色进行表示，即将灰度图像中的每个像素点设置为黑色或白色。具体来说，将图像中的每个像素点的灰度值与一个阈值进行比较，如果该像素点的灰度值大于阈值，则将其设置为白色，否则设置为黑色。

4.1.4 形态学操作

进行形态学操作的作用是对图像中的目标进行形态学变换，以提取、增强、修复或去除目标特征。形态学操作基于数学形态学理论，利用结构元素对图像进行腐蚀、膨胀、开运算、闭运算等操作。

具体：

- 腐蚀操作：可以减小或消除图像中的边界，使得目标轮廓更加精细明显，便于目标分割和提取；
- 膨胀操作：可以增大或扩张图像中的目标区域，使得目标连接更加紧密，便于目标检测和识别；
- 开运算操作：先进行腐蚀操作再进行膨胀操作，可以去除小的噪声和非主要目标，同时保留主要目标的形状和大小；
- 闭运算操作：先进行膨胀操作再进行腐蚀操作，可以填补小的空洞和断裂，同时保留目标的整体形状和大小。
- Morphological Gradient：获得类似物体的轮廓。

4.2 程序设计思路

(1) 读取图像，并将其转换成灰度图像。
(2) 对图像进行高斯滤波以平滑图像并减少噪音。
(3) 对高斯滤波后的图像进行傅里叶变换。
(4) 将傅里叶变换的结果进行频率过滤，抑制低频成分，增强高频成分，以强化边缘信息。
(5) 对过滤后的频率域数据进行反傅里叶变换以得到边缘信息。
(6) 对边缘信息进行二值化处理，并通过形态学操作去除噪点。
(7) 输出结果计算菌落数量。

4.3 流程图

图1 流程图

4.4 程序代码

```
import cv2
import numpy as np


def gaussian_kernel(size, sigma=1.5):
    """生成高斯卷积核"""
    kernel = np.zeros((size, size))
    center = size // 2
    s = 2 * (sigma ** 2)
    for x in range(-center, center+1):
        for y in range(-center, center+1):
            r = np.sqrt(x**2 + y**2)
            kernel[x+center, y+center] = (np.exp(-(r**2)/s)) / (np.pi * s)
    return kernel / np.sum(kernel)


def my_filter2D(src, kernel):
    """二维卷积函数"""
    if len(src.shape) == 2:
        rows, cols = src.shape
        dst = np.zeros_like(src)
        k_rows, k_cols = kernel.shape
        center_x, center_y = k_rows//2, k_cols//2
        for i in range(center_x, rows-center_x):
            for j in range(center_y, cols-center_y):
                dst[i,j] = np.sum(src[i-center_x:i+center_x+1,
j-center_y:j+center_y+1]*kernel)
        return dst
    else:
        b, g, r = cv2.split(src)
        b_dst = my_filter2D(b, kernel)
```

```
        g_dst = my_filter2D(g, kernel)
        r_dst = my_filter2D(r, kernel)
        return cv2.merge((b_dst, g_dst, r_dst))
#
def fft2(img):
    """二维快速傅里叶变换"""
    rows, cols = img.shape
    f = np.zeros((rows, cols), dtype=np.complex64)
    for u in range(rows):
        for v in range(cols):
            sum = 0
            for x in range(rows):
                for y in range(cols):
                    sum += img[x, y] * np.exp(-2j * np.pi * ((u * x) / rows + (v * y) / cols))
            f[u, v] = sum
    return f

def ifft2(f):
    """二维快速傅里叶逆变换"""
    rows, cols = f.shape
    img = np.zeros((rows, cols), dtype=np.float32)
    for x in range(rows):
        for y in range(cols):
            sum = 0
            for u in range(rows):
                for v in range(cols):
                    sum += f[u, v] * np.exp(2j * np.pi * ((u * x) / rows + (v * y) / cols))
            img[x, y] = sum.real / (rows * cols)
    return img
```

```
# 读取图像并转换成灰度图像
img = cv2.imread('image3.jpg')
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# 高斯滤波
kernel_size = 9
sigma = 1.0
gaussian_kernel = gaussian_kernel(kernel_size, sigma)
blurred_img = my_filter2D(gray, gaussian_kernel)

# 傅里叶变换
f = fft2(blurred_img)

# 频率过滤
rows, cols = gray.shape
crow, ccol = rows // 2, cols // 2
f[crow - 10:crow + 10, ccol - 10:ccol + 10] = 0

# 反傅里叶变换
img_back = ifft2(f)

# 二值化处理
threshold = 82
ret, binary_img = cv2.threshold(img_back.astype(np.uint8), threshold, 255, cv2.THRESH_BINARY)

# 形态学操作
kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (3, 3))
morphed_img = cv2.morphologyEx(binary_img, cv2.MORPH_OPEN, kernel)

# 计算连通区域
```

```
count, labels = cv2.connectedComponents(morphed_img.astype(np.uint8))

# 为每个连通区域标记颜色及数量
output = img.copy()
for i in range(1, count):
    mask = labels == i
    output[mask] = (0, 0, 255)
    pos = np.round(cv2.mean(np.stack([np.where(mask, gray, 0)]*3, axis=-1))).astype(int)
    cv2.putText(output, str(i), tuple((pos[1], pos[0])), cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 255,
255), 2)

# 显示并保存结果
cv2.imshow('Result', output)
cv2.imwrite('result.jpg', output)
print("菌落数量：  %d" % (count - 1))
```

5. 实验结果与分析

5.1 实验结果

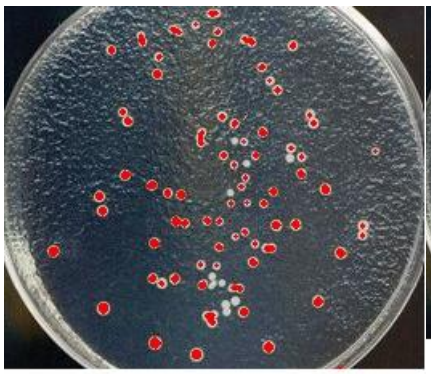

图2 实验结果

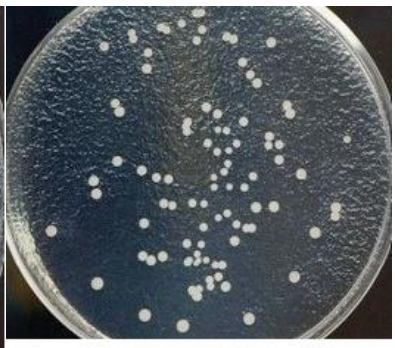

图3 原图

5.2 实验结果分析

5.2.1 参数选择与调整

在实验中经过反复多次试错实验，最终选择高斯核大小为 9，标准差为 1.0，在频率过滤时，将中心位置设置为 10x10 的矩形区域，二值化阈值设置为 82 时，在形态学操作中将核的大小设置为 3x3 时得到的效果最好。

5.2.2 结果分析

优点：观察实验结果我们可以发现对于这个复杂的图像我们成功的将大部分的噪声去除，包括培养皿边缘亮度高与菌群亮度相近，以及菌群周围存在大量的细小纹理对菌群造成的干扰

不足：不难发现有极少数的菌群没有被记录到，在多次试错实验中始终没有找到合适的高斯核大小与标准差、二值化阈值、形态学核、以及频率过滤几者之间最好的组合解决这个问题。将标准差缩小，保留细节的同时，实验中发现培养皿的边缘部分会出大量的点被记录。

6. 参考

1. https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_morphological_ops/py_morphological_ops.html#morphological-ops
2. https://blog.csdn.net/jgj123321/article/details/94448463
3. https://zhuanlan.zhihu.com/p/19763358
4. https://blog.csdn.net/xdg15294969271/article/details/119732470

7. 课程总结

数字图像处理是计算机科学中的一个重要领域，它研究如何使用计算机对数字图像进行处理、分析和识别。通过对数字图像课程的学习，我们学习到了如何可以使用多种算法和工具，如滤波、边缘检测等，同时也学习到了其背后的原理与逻辑。

数字图像处理涵盖了很多领域，如医学影像分析、计算机视觉、计算机图形学等，因此具有广泛的应用价值。比如，在医学领域，数字图像处理技术可以被用于 X 光图像的诊断和治疗；在安防领域，数字图像处理技术可以被用于人脸识别和视频监控等。

通过学习数字图像处理，我们掌握了数字图像处理的基本方法和技术，如空间域和频域的处理、图像增强和去噪、形态学处理和特征提取等；还需要学会如何使用编程语言和图像处理库进行实际操作。

数字图像处理为我们提供了一种强大的工具，可以帮助我们更好地理解和处

理图像信息，从而应用于各种实际场景中。