信号与系统实践作业

523031910204 韩米硕

1. 实验一：离散信号的卷积

1.1. 离散一维卷积

离散卷积公式：

$$(f * g)[n] = \sum_{m=-\infty}^{\infty} f[m]g[n-m]$$

实验中实现零填充（zero-padding）的有限长信号卷积。

1.1.1. pytorch 库内置一维卷积

```
def conv1d(signal, kernel):
    kernel = kernel[::-1]  # 核反转
    m, n = len(kernel), len(signal)
    padded = np.pad(signal, (m-1, m-1), 'constant')
    return np.array([
        np.sum(padded[i:i+m] * kernel)
        for i in range(n + m - 1)
    ])
```

1.1.2. 手动实现一维卷积

```
def conv1d(signal, kernel):
    # 反转卷积核
    kernel = kernel[::-1]
    m, n = len(kernel), len(signal)
    # 输出长度 = 输入长度 + 核长度 - 1
    output_length = n + m - 1
    result = np.zeros(output_length)
    # 零填充输入信号
    padded_signal = np.concatenate([np.zeros(m - 1), signal, np.zeros(m - 1)])
    # 滑动窗口计算卷积
    for i in range(output_length):
        window = padded_signal[i:i + m]
        # 点积计算
        result[i] = np.sum(window * kernel)
    return result
```

1.2. 离散二维卷积

二维离散卷积：

$$(f * g)[i,j] = \sum_{m}\sum_{n} f[m,n]g[i-m,j-n]$$

1.2.1. pytorch 库内置二维卷积

```
def conv2d(image, kernel):
    kernel = np.flipud(np.fliplr(kernel))  # 二维反转
    h, w = image.shape
    kh, kw = kernel.shape
    padded = np.pad(image, ((kh-1,kh-1),(kw-1,kw-1)))
    return np.array([
        [np.sum(padded[i:i+kh, j:j+kw] * kernel)
         for j in range(w + kw -1)]
        for i in range(h + kh -1)
    ])
```

1.2.2. 手动实现二维卷积

```
def conv2d(image, kernel):
    # 获取图像和卷积核尺寸
    img_h, img_w = len(image), len(image[0])
    ker_h, ker_w = len(kernel), len(kernel[0])
    # 反转卷积核（水平和垂直方向）
    kernel = [row[::-1] for row in kernel][::-1]
    # 计算输出尺寸
    out_h = img_h + ker_h - 1
    out_w = img_w + ker_w - 1
```

```
    # 创建输出矩阵并初始化
    result = [[0 for _ in range(out_w)]
    for _ in range(out_h)]
    # 零填充输入图像
    padded_img = []
    pad_h = ker_h - 1
    pad_w = ker_w - 1
    # 顶部填充
    padded_img.extend([[0] * (img_w + 2 *
     pad_w) for _ in range(pad_h)])
    # 中间行（左右填充）
    for row in image:
        padded_row = [0] * pad_w + row +
    [0] * pad_w
        padded_img.append(padded_row)
    # 底部填充
    padded_img.extend([[0] * (img_w + 2 *
     pad_w) for _ in range(pad_h)])
    # 执行卷积
    for i in range(out_h):
        for j in range(out_w):
            # 提取当前窗口
            window = [
                row[j:j + ker_w]
                for row in padded_img[i:i
     + ker_h]
            ]
            # 计算点积
            conv_sum = 0
            for x in range(ker_h):
                for y in range(ker_w):
                    conv_sum += window[x
    ][y] * kernel[x][y]
            result[i][j] = conv_sum
    return result
```

2. 实验二：语音与图像的卷积实践

2.1. 一维语音信号处理

2.1.1. 混响

```
def reverb_kernel(duration=0.3, decay
    =0.5, sample_rate=44100):
    """生成指数衰减的混响核"""
    length = int(duration * sample_rate)
    t = np.linspace(0, duration, length)
    kernel = np.exp(-t * decay) * np.sin
    (2 * np.pi * 5 * t)  # 带震荡的衰减
    return normalize(kernel)
```

2.1.2. 锐化

高频增强，将语音信号的边缘锐化，增强中心语音信息的功能比重。

```
def sharpen_kernel():
    """高频增强核（边缘检测变体）"""
    return normalize(np.array([-0.5, 1,
    -0.5]))
```

2.1.3. 去噪

首先对原音频添加高斯白噪声，在原音频基准上添加随机扰动。然后采用均值滤波的办法，使用均值去噪核，平均噪声的过多干扰

```
    # 添加高斯白噪声
    noise = np.random.normal(0, 0.05, len
    (audio))
    noisy_audio = audio + noise
    # 去噪核（均值滤波）
    denoise_kernel = np.ones(5)/5  # 5点
    移动平均
    clean_audio = conv1d(noisy_audio,
    denoise_kernel)
```

2.1.4. 低通

基于 *sinc* 函数的低通滤波器核，用于音频信号处理中的低频增强。

```
def bass_boost_kernel(cutoff=200,
    sample_rate=44100):
    """简易低通滤波器核"""
    freq_ratio = cutoff / sample_rate
    n = int(5 / freq_ratio)  # 核长度与截
    止频率相关
    t = np.linspace(-n//2, n//2, n)
    kernel = np.sinc(2 * freq_ratio * t)
     # sinc函数实现低通
    return normalize(kernel)
```

2.2. 二维图像信号处理

我们对若干图片进行处理，将三色图转化为灰度图，便于后续操作。

2.2.1. 高斯模糊处理

应用高斯模糊卷积核，对图片处理。高斯核通过加权平均模糊图像，权重服从二维正态分布，中心像素影响最大，能有效抑制高频噪声。

```
def gaussian_kernel(size=3, sigma=1.0):
    """生成高斯核"""
    kernel = [[0] * size for _ in range(
    size)]
    center = size // 2
    total = 0
    for i in range(size):
        for j in range(size):
            x, y = i - center, j - center
            kernel[i][j] = (1 / (2 *
    3.1416 * sigma ** 2)) * 2.718 ** (-(x
    ** 2 + y ** 2) / (2 * sigma ** 2))
            total += kernel[i][j]
    # 归一化
    return [[val / total for val in row]
    for row in kernel]
```

示例如图6。

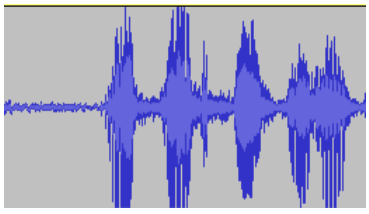

图 1: 图片标题

2.2.2. 边缘检测

Sobel 核通过突出水平/垂直方向的像素值突变，正负权重组合能增强边缘响应。

```
def sobel_kernel():
    """Sobel边缘检测核"""
    return [
        [-1, 0, 1],
        [-2, 0, 2],
        [-1, 0, 1]
    ]
```

示例如图6。

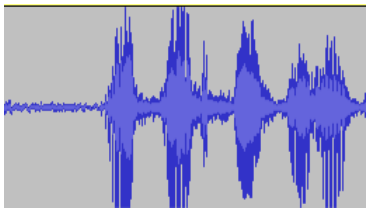

图 2: 图片标题

2.2.3. 图像锐化

中心正权重 (5) 增强原像素，周围负权重 (-1) 抑制邻域像素，通过差分放大高频细节。

```
def sharpen_kernel():
    """图像锐化核"""
    return [
        [ 0, -1,  0],
        [-1,  5, -1],
        [ 0, -1,  0]
    ]
```

2.2.4. 添加椒盐噪声

随机像素变为黑白极值

```
def add_noise(matrix, prob=0.05):
    noisy = [row.copy() for row in matrix
    ]
    for i in range(len(noisy)):
        for j in range(len(noisy[0])):
            if random.random() < prob:
                noisy[i][j] = 0 if random
    .random() < 0.5 else 255
    return noisy
```

2.2.5. 中值滤波去噪

取邻域中值，有效消除孤立噪声点。

```
def median_filter(matrix, size=3):
    h, w = len(matrix), len(matrix[0])
    padded = [[0] * (w + size - 1) for _
    in range(h + size - 1)]
```

```
    for i in range(h):
        for j in range(w):
            padded[i + size // 2][j + size // 2] = matrix[i][j]
    result = [[0] * w for _ in range(h)]
    for i in range(h):
        for j in range(w):
            window = []
            for x in range(size):
                for y in range(size):
                    window.append(padded[i + x][j + y])
            window.sort()
            result[i][j] = window[size * size // 2]
    return result
```

3. 实验三：二维傅里叶变换频谱分析

3.1. 傅里叶变换与反傅里叶变换的频谱特性

计算振幅谱和相位谱，并绘制对应的傅里叶变换图像6。

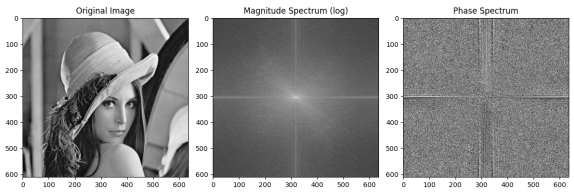

图 3: 图片标题

通过振幅或者相位进行反变换，重构出原图像6。

3.2. 结论

- 相较于振幅谱，相位谱对信号重建起决定性作用
- 频域滤波可实现比空域卷积更精确的频率控制
- 二维卷积需注意边界处理（padding 策略）

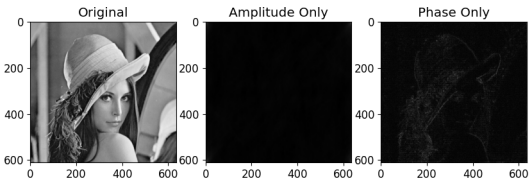

图 4: 图片标题

3.3. 频域滤波

通过滤波操作，得到与卷积相同的效果，低通滤波的效果类似高斯模糊，高通滤波检测边缘轮廓

```
rows, cols = img_array.shape
crow, ccol = rows // 2, cols // 2  # 中心点
# 创建低通掩码（保留中心低频）
mask_lowpass = np.zeros((rows, cols), np.uint8)
mask_lowpass[crow-30:crow+30, ccol-30:ccol+30] = 1  # 保留中心 60x60 区域
# 应用滤波
fft_filtered = fft_shifted * mask_lowpass
img_lowpass = np.fft.ifft2(np.fft.ifftshift(fft_filtered)).real

mask_highpass = 1 - mask_lowpass  # 去除中心低频
fft_filtered_high = fft_shifted * mask_highpass
img_highpass = np.fft.ifft2(np.fft.ifftshift(fft_filtered_high)).real
```

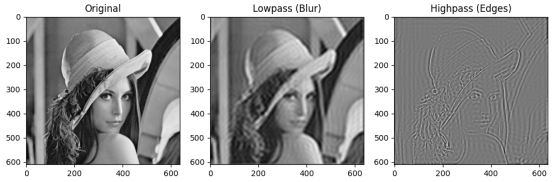

图 5: 图片标题

4. LaTeX 写作示例

本章提供使用 LaTeX 书写报告时可能使用的各种文档对象的使用示例。**请在报告写作完成后删除本章**。

4.1. 公式

示例如式1。

$$\pi = \frac{3.1415926}{1} \tag{1}$$

4.2. 图像

示例如图6。

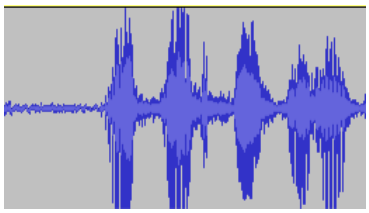

图 6: 图片标题

4.3. 表格

示例如表1。

表 1: 表标题

左对齐	居中对齐	右对齐
内容	内容	内容
内容	内容	内容

4.4. 代码

示例如下。

```
# 这是注释
def func(a, b):
    print("Hello, world!")
```